A DISCRETE UTTERANCE
VOICE RECOGNITION ALGORITHM

BY
WILLIAM DANIEL WIGGER
B.S., Oakland University, 1984

THESIS
Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1986

Urbana, Illinois
ACKNOWLEDGMENTS

Many people assisted me with this thesis. Special thanks are extended to Professor Ricardo Uribe, my thesis advisor, and Jim Kepler for their time, effort, and enthusiasm relating to the voice recognition system. I would also like to thank Jim Garceau for his contributions to the software.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>INTRODUCTION ..................................................</td>
</tr>
<tr>
<td>1.1</td>
<td>System Overview .............................................</td>
</tr>
<tr>
<td>2.</td>
<td>THE TRAINING PROGRAM ...........................................</td>
</tr>
<tr>
<td>2.1</td>
<td>Reception of Data ............................................</td>
</tr>
<tr>
<td>2.2</td>
<td>Storage of Data ................................................</td>
</tr>
<tr>
<td>2.3</td>
<td>How to Use the Program .......................................</td>
</tr>
<tr>
<td>3.</td>
<td>THE MATCHING PROGRAM ...........................................</td>
</tr>
<tr>
<td>3.1</td>
<td>Use of Word Files and the Reception of Data ...............</td>
</tr>
<tr>
<td>3.2</td>
<td>Time-warping Algorithm .......................................</td>
</tr>
<tr>
<td>3.3</td>
<td>Matching Algorithms Used ....................................</td>
</tr>
<tr>
<td>3.3.1</td>
<td>First Matching Algorithm ..................................</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Second Matching Algorithm ................................</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Third Matching Algorithm ...................................</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Fourth Matching Algorithm ................................</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Summary .......................................................</td>
</tr>
<tr>
<td>3.4</td>
<td>Pattern Averaging and Adding to the Library ............</td>
</tr>
<tr>
<td>3.5</td>
<td>How to Use the Program .......................................</td>
</tr>
<tr>
<td>4.</td>
<td>THE PURGE PROGRAM .............................................</td>
</tr>
<tr>
<td>5.</td>
<td>THE GRAPH PROGRAM ...............................................</td>
</tr>
<tr>
<td>6.</td>
<td>EXPERIMENTAL RESULTS .........................................</td>
</tr>
<tr>
<td>6.1</td>
<td>Experiment 1 ..................................................</td>
</tr>
<tr>
<td>6.2</td>
<td>Experiment 2 ..................................................</td>
</tr>
<tr>
<td>6.3</td>
<td>Experiment 3 ..................................................</td>
</tr>
<tr>
<td>6.4</td>
<td>Experiment 4 ..................................................</td>
</tr>
</tbody>
</table>
7. CONCLUSION ................................................................. 58
  7.1 Discussion ............................................................... 58
  7.2 Suggested Improvements ................................................. 59
  7.3 Concluding Remarks ...................................................... 60
APPENDIX A - RMATCH9 PROGRAM LISTING ............................. 62
APPENDIX B - TRAIN5 PROGRAM LISTING .................................. 94
APPENDIX C - PRGFLE PROGRAM LISTING .............................. 106
APPENDIX D - GRAPH PROGRAM LISTING .............................. 113
REFERENCES ................................................................. 119
Advanced Digital Systems Laboratory's (ADSL) research on voice recognition was initiated in 1983 by Thomas Liu, for his Master of Science degree [1]. Jim Kepler continued developing this system through 1984-85 for his Master of Science degree [2]. The author of this present thesis worked with Mr. Kepler in developing the voice recognition system. While Mr. Kepler greatly improved the hardware and the firmware (of the microprocessor controller board in the feature extractor), the author of this present thesis concentrated on the software development of the system.

The system is a discrete utterance voice recognition system. That is, when one word is said into the system, it will attempt to recognize this word. This is different from a continuous speech recognition system which tries to recognize many words, spoken in sentence form. The latter system may try to recognize key words, and then from the context and English grammar, it will attempt to recognize the entire sentence.
The motivation behind the development of the ADSL voice recognition system is described in Mr. Kepler's thesis [2], and will not be repeated here.

1.1 System Overview

Figure 1.1 shows the general system configuration. A word is said into the microphone, and the microphone signal is sent directly to the feature extractor. The feature extractor gathers some specific information about the signal, and passes this information to the host computer. The host computer tries then to determine what word was spoken using a library of word patterns which was previously built.

The frequency content of speech ranges approximately from 100 Hz to 3000 Hz. Seven bandpass filters are employed in the feature extractor to cover this range, each an octave apart. The filters are designed with center frequencies at 62.5 Hz, 125 Hz, 250 Hz, 500 Hz, 1000 Hz, 2000 Hz, and 4000 Hz. The design of these filters and their frequency responses are given in [2].

There are two types of information that the feature extractor obtains from the microphone signal. The first type of data obtained from the microphone signal is the peak detected values of the filters. The peak detected values are allowed to range from 0 to 5 volts (which is converted to a number between 0
and 255), and are sampled at a constant interval. An Analog-to-Digital converter is used to read and store these values.

The second type of information obtained from the microphone signal, by the feature extractor, consists of the zero crossing counts for each filter, and for the microphone. The outputs of each filter and of the microphone are connected to a zero crossing detection circuit. These zero crossing detection circuits count how many times the signal crosses zero for a given sampling period. Figure 1.2 shows a block diagram of the feature extractor.

Therefore, for every sampling period there are fifteen readings to be taken. These fifteen readings consist of the peak detected values for each of the seven bandpass filters, the number of zero crossings at each filter during the sampling period, and the number of zero crossings at the microphone during the sampling period.

These fifteen readings represent one frame, and may be thought of as a snapshot of the voiced signal at a given time. Frames are taken and successively stored during the reception of a word. The collection of all the frames for a given word is called the word's pattern.

This pattern is sent to the host computer through a serial line at 9600 baud.

The host computer matches this pattern against other patterns previously made (stored in what will be called a library file). Upon determining the best match, if any, the host
Figure 1.2 - Feature extractor
computer recognizes the spoken word as being the word to which it matched.

The development of the software for the host computer was the main emphasis of the present thesis.
CHAPTER 2
THE TRAINING PROGRAM

2.1 Reception of Data

To collect a word pattern, obtained from a spoken word, into the Z-151 personal computer (host computer), via the feature extractor box and the Z-151 serial port, the following protocol must take place (the feature extractor communicates at 9600 baud through a serial line). First, the ascii value for an '!' character must be sent to the feature extractor; this tells the feature extractor to start looking for a beginning of a word, and when found, to collect the pattern for that word. While the feature extractor is collecting the word pattern, it begins to transmit the pattern to the host computer. This tells the Z-151 that the beginning of a word has been found. In order to obtain a frame of data, the host computer must now send the ascii value for a New Line (OA hex) to the feature extractor. A frame of data is terminated by an NL character.

In more detail, the TRAINS program reads a frame of data via the assembly routine A872. The main routine calls the A872
subroutine and passes the address of an array where the collected frame is to be stored. This address is passed as a parameter on the stack. The A872 subroutine is written in 8088 assembly language, since that is the processor used by the Z-151. The first step of the subroutine is to manipulate the stack so as to put the address of the passed array in a register. It then reads the status word of the serial port to see if a byte has been sent. If a byte has been sent, it then reads it and discards it as garbage. It then sends an NL character to the feature extractor to tell it that it is ready to accept a frame of data. It then polls the port until a character is received and stores this character in the passed array. If the character is an NL (or CR) it then remanipulates the stack and returns. If not an NL, it waits to receive the next character, stores it in the array, and repeats this process until an NL is received.

A frame of data consist of fifteen readings -- the seven peak detected values of the seven bandpass filters, the seven zero-crossing values of the seven bandpass filters and the zero-crossing value of the microphone. The frames are collected successively by the feature extractor for the entire length of the word. Frames are collected at an interval of 10 milliseconds. These frames, put together, constitute the word pattern for the spoken word. For a more detailed explanation of this process, please refer to [2], pages 10-14.

After returning from the A872 subroutine, one frame of data is stored in the passed character array. Each of the fifteen readings is stored as two ascii characters and may range from 00-
FF (hex). The high order byte is stored in the lower memory location. The first two levels (characters) in the character array represent the peak detected value for the 62.5 Hz band, PD(62.5). After this are the peak detected values for the other six bands. The next two levels (15 and 16) in the character array represent the zero crossing value for the microphone, ZC(Microphone). The following two levels represent the zero crossing value for the 4000 Hz filter, ZC(4000Hz). After this are the zero crossing values for the remaining six filters. At the end of the array is the ascii value for an NL. The array, or frame of data, may be represented as follows: PD(62.5Hz), PD(125Hz), PD(250Hz), PD(500Hz), PD(1000Hz), PD(2000Hz), PD(4000Hz), ZC(Microphone), ZC(4000Hz), ZC(2000Hz), ZC(1000Hz), ZC(500Hz), ZC(250Hz), ZC(125Hz), ZC(62.5Hz), NL.

Frames of a word are gathered and stored until the end of a word is detected. Originally the end of a word was detected by the feature extractor. The feature extractor waited until all seven peak detected values were under their respective thresholds for a certain number of frames. Both the threshold values and the number of ending frames were user defined. But a problem was found with this method, the problem being that the beginning of a word was defined when any combinations of bands above their thresholds for a certain number of frames. This beginning frame count was again user defined and could be different from the ending frame count. But the thresholds used to declare the beginning and ending of a word were the same. It was found that these thresholds need to be slightly different for better word boundary detection. Therefore the detection of the end of a word.
is now done in software in the TRAINS program. The program, controlling the feature extractor, now allows the user to set the beginning as well as the ending thresholds and the ending silent period count (the number of frames in which all peak detected bandpass values must be below their respective thresholds to declare the end of a word), see Section 2.3. This allows the ending thresholds to be different from the beginning thresholds. The program then declares the end of a word when all peak detected bandpass values are below their respective ending thresholds for the silent period count.

Another slight change in detecting the end of a word was found to improve the system. Originally, when the word was first detected to begin, the end of word detection automatically began. This was found to cut off words that began with the letter 's' or 'p' too quickly. The beginning of a word was correctly detected, but because of the soft sound of the 's' or the initial burst of sound of the 'p', followed by a brief period of silence, the end of word was detected right after or during these sounds, thus ignoring the rest of the word. Manipulation of the thresholds did not seem to solve this problem. The best solution was to not start looking for the end of a word until a certain number of frames were taken, allowing enough time for these initial sounds.

When the program determines the end of word, it then sends out a reset signal to the feature extractor. The reset signal is an ascii '='. This is needed because the feature extractor, with its parameters set in the specified way, will still be looking for the end of a word. The reset signal tells it to stop sending
data and to go back into a state where it will accept commands from the host computer. Notice that since it is desired for the host computer to detect the end of a word, the parameters of the feature extractor are set up so that the feature extractor will not detect an end of a word (therefore the need for the reset signal to be sent).

The way in which the thresholds and parameters should be set to use this way of detecting the word boundaries is discussed in Section 2.3.

2.2 Storage of Data

A set of words is stored in a file. The storage of each word consists of three parts. The first part is the spelling of the word. The next part stored is a count for the word. The counter counts how many times this particular word pattern has been matched with another word that has the same spelling. Of course when the word is first said it has a count of zero. The third part of storing a word consists of the word pattern. This is a collection of all the frames for that particular word.

The program TRAINS receives these three parts in the following manner. The spelling of the word is supplied by the user before the word is said and is stored in a string variable. The count of a word is defaulted to zero and is written out to file when the time arises.

The reading in of the frames of a word is done as follows. As described in the previous section an ascii '!' is sent out to the feature extractor to tell it to start looking for a word
(this is done after the user supplies the spelling of the word which is going to be said). It then calls the assembly routine A872 to obtain one frame of data. After reception of a frame of data into a one-dimensional array (as described in the previous section), this array is stored in the nth level (row) of a two-dimensional array, where n is the nth frame received for this word. The program then detects the end of a word as described in the previous section. Next it stores the spelling of the word in a file that was previously specified by the user, followed by a Carriage Return (CR) and an NL. Then it stores a zero (the word count) followed by a CR and NL. And finally it stores the frames, in successive order, in the file with only an NL at the end of each frame. After the last frame an '*' followed by an NL and a '$' followed by an NL are stored to mark the end of a word. The next word is then stored in the exact same format right after the previous word. After the last word has been entered, the file is closed with no special marker used to denote the last word (besides the End-Of-File marker which the system inserts). When the word is stored, a user defined number of frames may be chopped off the end of a word; that is, the whole word will be stored except for the last x number of frames, where x is user defined, or defaulted to a certain value by the program. This is done because the ending part of the word is where all the peak detected values for the bandpass filters were below their thresholds and therefore the last few frames contain little information about the word. It is therefore of no use to take up room in the file to store these data.
2.3 How to Use the Program

The program TRAINS is to be used to make files of words. These files may consist of a set of words said once, or multiple times. A file may be used as an original library file, or as a comparison file. The files are formatted as discussed in the previous section. First will be a description of how files are made from a user's point of view, and then more detail about the uses of these files.

The program is easy to run, and with a brief explanation it has been found that someone not familiar with the system can use it without difficulties. First the operating system of the host computer (DOS) must be activated. To run the program, the user types the program name TRAINS. The first question asked by the program is the name of the file in which the user wants to store the data. This file name should not be the same as a previous file name because what was in the old file will be deleted.

Next will be a prompt which allows you to take one of four different actions. If an '*' is entered then the program will close the file which contains the said word patterns, and will return to the operating system (DOS). (If an '&' is entered then the program will allow the user to issue a command which will either warm boot or cold boot the feature extractor. The booting procedure is explained in [2], page 53. Under a normal operating procedure the feature extractor does not need to be rebooted). If a '?' is entered, then the program goes into a parameter changing routine, allowing the user to change many of the parameters. There are two main groups of parameters which can be
The first group consists of parameters which are used by the feature extractor. To change these parameters, type 'F' when the first menu is shown. The program will request (through the assembly routine A07P) from the feature extractor the current values for the parameters used, and will display them on the screen. It then prompts the user to enter one by one all the values for the parameters in a specified order.

The first parameter is the sampling period rate, SPER. This is the rate at which the voice signal (word) is to be sampled (frames gathered). The user must enter in a four digit hex number between 0000-FFFF. The larger the number the longer the sampling rate (more time between samples). A sampling rate of 10 milliseconds was found to work well, and that may be obtained by entering E800 for SPER. Other values for SPER are DC00 (15 msec sampling period) and CFFF (20 msec sampling period).

The next parameter is used as a settling time for the log amplifier. Since the log amplifier is no longer being used this value is superfluous, and a value of 01 will do quite well here.

The next seven parameters are the thresholds for the peak detected values for the seven bandpass filters. These thresholds are used to determine the beginning of a word as described in Section 2.1. The best settings for these thresholds found so far are: THR1=0E, THR2=0E, THR3=0A, THR4=0A, THR5=05, THR6=05, and THR7=02.

The next parameter to set is the silent period count, SPCC. This is the number of consecutive frames, in which the peak detected values for all the bands are below their respective thresholds, needed before the end of a word is declared. As
discussed in Section 2.1 the end of word detection is now done in software by the TRAIN5 program. Therefore it is not desired (because it may hang the system) that the feature extractor be allowed to detect the end of a word. Hence, a large value is entered for this parameter so that the feature extractor will not detect the end of a word before the program TRAIN5. A value of 30 for SPCC has worked without a problem during current experimentation.

The last parameter to be set is TSAM. This is the number of consecutive frames needed in which any one band in each frame is above its respective threshold (THR1-THR7) before the beginning of a word is declared. A value of 03 has worked well here.

In summary, these are the best settings for the feature extractor parameters found to date: SPER=EB00, STLTM=01, THR1=0E, THR2=0E, THR3=0A, THR4=0A, THR5=05, THR6=05, THR7=02, SPCC=30, and TSAM=03. Notice that all the leading zeros must be entered since each parameter is to be a certain number of bytes in length when sent to the feature extractor.

The assembly routine AB70 sends out these new parameters to the feature extractor. The feature extractor then echoes these parameters back (this is done one byte at a time) and the validity of the transmission is checked.

After the parameters of the feature extractor are changed, the program returns back to the main prompt. The second group of parameters that may be changed are the software thresholds. In order to change these thresholds a '?' should be entered when the main prompt appears and then an 'S' entered when
the menu is presented for changing parameters. The software thresholds will then be shown and the user allowed to change any combination of these thresholds.

The first seven thresholds the user may change are the peak detected values for the seven bandpass filters that are used to declare the end of a word as described in Section 2.1. The following values have been found to work well: THR1=255 (dec), THR2=10 (dec), THR3=10, THR4=10, THR5=10, THR6=10, and THR7=3. These are their current default values.

Another parameter that may be changed is the ending silent period count. This is the number of frames in which the peak detected values in all the bands must be below their respective thresholds before the end of word is declared. Currently a value of seven is being used, and is therefore the default value. This is discussed in more detail in Section 2.1.

The last parameter the user may change is the number of frames to be chopped off at the end of a word, before storing the word pattern in a file. The motivation for doing this was presented earlier. This value is currently defaulted to 7. A thorough understanding about how words are stored and the maximum number of frames that can be stored with respect to other programs which use these data (e.g., RMATCH9) is needed before changing this value.

If the user desires to change both the feature extractor parameters and the software parameters, then an 'A' should be typed when the menu for changing the parameters is shown.

After the parameters are set, a fourth option may be taken; this is the most widely used option when the main system prompt

15
appears. The user types in the word he or she is about to say into the microphone. The spelling of the word should be typed in, and it will be the spelling that the system uses for the next word pattern generated by the user.

The user will then be prompted to say the word and a period (\'.\') will appear on the next line. The user should then say the word. When a word is detected a period is added to the screen for every frame that is collected for the word. This shows the user how long the said word was, and that the system has detected the beginning of the word. Next the user is told how many frames long the word was (this is used mainly for developmental purposes). The user is then asked if he/she would like to store the word out to file (the file being the name supplied earlier by the user). If answered 'Y' then it is stored in the format described in Section 2.2. If answered 'N' (or anything not 'Y') then this word pattern is ignored. The user is then given the main prompt and allowed to take one of the four described options.

There are two uses for the word files. The first use is a library word file. That is, the words said and stored in this file will be used by the matching program, RMATCH9. The matching program will use this file as a library of word patterns, and will match a new word pattern obtained against the library word patterns. In this way if the new word pattern is found to correspond to a word pattern in the library, the word has been recognized.

The second use of word files is to use them to store word
patterns which will be matched against a library file later. This enables the user to always have the same word files to use as comparison words. Thus the user is able to try different matching algorithms on the exact same word patterns, making the comparison of matching algorithms much more accurate. In either application, the word files are created in the same way.
The matching program, RMATCH9, will receive word patterns and will compare these patterns, individually, against a library of word patterns. Through this comparison it will determine if the received word pattern matches any of the library words. If a match is found then the library word that produced the match is identified as the received word. If no match is found then the received word is said to not be in the library. If more than one close match is found then the received word is said to be ambiguous with respect to the current library. The RMATCH9 program then determines (either by asking the user, or by using the comparison file) if it recognized the word correctly. Upon finding this out, it then may add the word pattern to the library, average the word pattern with another word pattern in the library, or do nothing to change the state of the library. The program also keeps statistics as to how many times a word was recognized either correctly, incorrectly, ambiguously, or did not match.
3.1 Use of Word Files and the Reception of Data

The user is asked, at the beginning of the program, to supply the name of the desired library file. The library file is a set of words (and there may be multiple entries for the same word) that the program will use to compare incoming words (as described previously). The library words are read into a data structure which holds the spelling of the word, the count of the word (the number of times this word has been correctly matched to other words), the word pattern, and the number of frames in the word pattern.

The user is given two options in which the RMATCH9 program will receive incoming word patterns. The patterns may be obtained live, that is, spoken into the microphone and received from the feature extractor as described in Chapter 2; or the word patterns may be obtained from another word file whose name is supplied by the user. Up to ten word files may be entered by the user for processing. The second method offers many advantages for experimentation. The same word files can be used for comparison of words while trying different matching algorithms, or changing some of the parameters and thresholds of the matching algorithms. Also, it saves time to generate the word files (using TRAIN5) and to automatically process them later, since the user does not need to interact with the RMATCH9 program while the words are being processed.

If the first option is chosen the word pattern is obtained from the user via the feature extractor in the same way as the program TRAIN5 (described in Chapter 2) obtains the pattern. The
word pattern is read into a two-dimensional array, and will be compared against the word patterns in the library. The spelling of the said word is later obtained from the user after attempting to recognize the word.

If the second option is chosen the user is allowed to enter up to 10 different file names which contain words to be recognized (the word files being created by the TRAIN5 program). Words are read in and processed (matched with the library) in successive order, i.e., each word is read in and processed before the next word is read in. The program reads in a word from the word file, and stores the word pattern for this word in a two-dimensional array. The spelling of the word is read into a variable and will be used later to check if the word was recognized correctly. The word pattern is compared against the word patterns in the library.

3.2 Time-warping Algorithm

RMATCH9 uses four algorithms for comparing two word patterns. Two of the algorithms use another algorithm call 'dynamic time warping'. An understanding of this algorithm is needed in order to work with the program.

In order to explain the algorithm let us assume that there are two patterns, each represented by a series of integers between 0 and 255, which will be compared. It is desired that the comparison will yield a number which corresponds to the likeness of one pattern to the other. The lower this number the more similar the two patterns.
A simple way of doing this, assuming the two patterns to be of equal length, would be to subtract the value of one entry, or frame, in one pattern from the corresponding frame in the other pattern. The absolute value of this result is then added to the results from the other subtractions. In this way one number is obtained corresponding to the likeness of the two patterns. If the patterns are of unequal length then frames could be deleted from the longer pattern, added to the shorter pattern, or the frames in the shorter pattern could be compared to their relatively corresponding frames in the longer pattern. For an example of the latter, suppose one pattern had 20 frames and the other had 10 frames. Then frame 7 of the latter pattern would be compared (subtracted and the absolute value taken) to frame 14 of the longer pattern.

This algorithm appears to be a bit rigid. What if one pattern is identical to the other pattern but is skewed by one frame? Although it would be desirable to obtain a very low number for their likeness value, a higher number would result.

A way to loosen this algorithm is to allow one frame in one pattern to be compared to a number of frames in the other pattern. For example, if both patterns have 10 frames, then allow frame 4 of one pattern to be compared to frames 3, 4, and 5 of the other pattern. The comparison which produces the lowest score is then accepted. One restriction would then be needed for implementation; as a comparison between frame i of pattern x cannot be made with frame j of pattern y, if frame i-1 of pattern x matched with frame j+1 of pattern y. In our above example it would mean that if frame 4 of the first pattern were to match
with frame 5 of the second pattern, then frame 5 of the first pattern could not be allowed to match with frame 4 of the second pattern.

By using this algorithm, many different 'paths' can be taken with a score calculated for each path. By path, it is meant the way in which the frames were matched to each other, i.e., the frame number pairs which matched together between the two patterns. The path with the lowest score is then taken to be the best path and its score is the number which is said to correspond between the likeness of the two patterns. An example will be given shortly which should make this clear.

Notice that this algorithm must keep track of all the possible paths that could be taken and their corresponding scores. One parameter of this algorithm is how much flexibility is allowed in comparing frames. If the patterns are both 10 frames long, should frame 4 of the first pattern be compared to frames 3, 4 and 5 of the second pattern, or frames 2, 3, 4, 5, and 6 of the second pattern? RMATCH9 currently chooses the former restraint, but a provision is made to easily loosen or tighten this.

Next an example is given of comparing two patterns using the method, dynamic time-warping, just described. The first pattern is five frames in length and the second is six frames; they have the following values:
TABLE 3.2.1
PATTERNS TO BE COMPARED

<table>
<thead>
<tr>
<th>Frame number</th>
<th>Pattern 1 values</th>
<th>Pattern 2 values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>11</td>
</tr>
</tbody>
</table>

The program matches the shorter pattern onto the longer. Frame i of the shorter pattern is allowed to match onto a number of frames of the longer pattern. To determine which frames can be used to match with frame i of the shorter pattern, the following functions are used.

Let

- \( A \) = number of frames of the longer pattern
- \( B \) = number of frames of the shorter pattern
- \( M \) = flexibility variable (default value is 1)
- \( S \) = ratio between pattern lengths

Then \( S = \frac{A+M}{B+M} \)

The minimum frame number of the longer pattern to which frame i of the shorter pattern may be matched is given by: \( \text{ROUND}(S \times i) - M \)

The maximum frame number of the longer pattern to which frame i of the shorter pattern may be matched is given by: \( \text{ROUND}(S \times i) + M \)

Notice that by varying \( M \) (which is variable M_ADJ in the program) the flexibility of comparing frames is adjusted.

In the above example \( A = 6 \), \( B = 5 \), and \( M = 1 \) (the value which the program uses). Therefore, frame i of the shorter pattern may match to the following frames of the longer pattern.
By using this information a table will be built in order to determine the best path for matching these two patterns and that path’s corresponding score. The entry in row i, column j of the table is the score of the best path which uses the match of frame i of the shorter pattern to frame j of the longer pattern. If there is no entry in the table it means that frames i and j were not allowed to be compared according to the numbers generated in the previous table. The following table for comparing these two words follows:

**Table 3.2.3**

<table>
<thead>
<tr>
<th>TIME-WARPING OF THE TWO PATTERNS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>1  2  3  4  5  6</td>
</tr>
<tr>
<td>1  1  2  -  -  -</td>
</tr>
<tr>
<td>2  1  2  5  -  -</td>
</tr>
<tr>
<td>3  -  -  3  3  5</td>
</tr>
<tr>
<td>4  -  -  -  3  5  7</td>
</tr>
<tr>
<td>5  -  -  -  -  3  5</td>
</tr>
</tbody>
</table>

The lowest score is the minimum number in the final row, which in this example is 3. The final row is chosen as the ending place because only at this point, for every path, have all the frames in the shorter pattern been matched with frames in the longer pattern. The score returned by the program would be the rounded off value of 3 divided by 5 (5 being the number of frames of the shorter pattern), that is, 1. The path it took to achieve this was matching frame 1 of the shorter pattern to frame 1 of
the longer, frame 2 to frame 1, frame 3 to frame 3 (or 4), frame 4 to frame 4, and frame 5 to frame 5.

To further explain the example an explanation of how the number in row 4, column 4 was obtained will be given. First, the values for frame 4 of the shorter pattern and frame 4 of the longer pattern were subtracted with the absolute value returned: \( \text{ABS}(7-7)=0 \). Next the minimum entry in the table for row 3 was found. This minimum entry had to occur on or before column 4, because that is the frame number of the longer pattern which is currently being examined. Therefore the minimum between the numbers 3 and 3 was found, which of course is 3. These two numbers were then added \((3+0)\) to give the score of 3 in row 4, column 4 of the table.

### 3.3 Matching Algorithms Used

Up to four different matching algorithms may be used to identify a word. The ideal matching algorithm would be one that took no time to execute and was one hundred percent accurate. Usually, though, the faster the algorithm, the less accurate it tends to be. The algorithms which use the time-warping procedure described in the previous section were found to be the most accurate algorithms, but also took quite a while to execute. Therefore, it was desired to find an algorithm which would execute quickly and narrow down the words which were possible matches to the spoken word, without ruling out the word which should match correctly. Such an algorithm was found, and is therefore the first algorithm used. After this algorithm, two
other matching algorithms, which make use of the time-warping procedure and both the peak detected and zero-crossing values of the bandpass filters, are used. These two algorithms consider only those words in the library which have not been ruled out by the first algorithm. The best matching word is determined by the additive scores of the three previous algorithms. A match with the library word is usually determined at this point. That is, the word with the lowest score is the best matching word. A 'no match' condition may occur if all the library words have been ruled out by at least one of the previous matching algorithms. A very low percentage of the time there may be two library words at this point which both have low scores (close to each other). Therefore, a fourth matching algorithm is used to see if one of these two remaining words is a better match to the said word. If the word can still not be matched, both library words seem to match to this word equally, then the word is said to be ambiguous. A description of these four algorithms and how they interact follows.

3.3.1 First Matching Algorithm

The first algorithm should be thought of as a hashing function, or a coarse sieve. It quickly rules out most of the words so that the other algorithms have a small number of words to look at. This algorithm was found by trial and error, by looking at the data and finding ways of matching that yielded accurate results.

In this first algorithm, a word in the library is ruled out
if it is more than ten frames different than the spoken word.
Next, a score is computed for that word by comparing the average
peak detected values for the library word against those of the
spoken word. To carry out this comparison the patterns must
first be normalized to one another.

The goal behind normalization is to get the total amount of
energy, in the peak detected values of the two patterns being
compared, to be equal. This is needed because when the same word
is said twice, one may have more energy. Therefore, its pattern
may be the same, but it is offset at a higher energy level.

Since the first matching algorithm only compares the average
peak detected value in each band between patterns, it is
sufficient only to normalize the total amount of energy in each
band (note that 'energy' will be used to describe the peak
detected voltage readings of the bandpass filters) between
patterns. This is done by first finding the difference between
the total amount of energy in the two comparison patterns. The
total amount of energy in a pattern is given by the sum of all
the peak detected values of all the bandpass filters, except the
seventh (4000Hz) filter. The seventh filter is excluded because
it was found to give less consistent values for the peak detected
readings than the other filters. The following formulas are then
used to normalize the library word's pattern to the spoken word's
pattern:

\[
\begin{align*}
L_s & \quad \text{the total energy in the library pattern} \\
L_n & \quad \text{the number of frames of the library pattern} \\
L_j & \quad \text{the total energy in bandpass filter } j \text{ of the library pattern} \\
L[j] & \quad \text{the average peak detected value in band } j \text{ of the library pattern after normalization} \\
W_s & \quad \text{the total energy in the spoken pattern}
\end{align*}
\]
Wn – the number of frames of the spoken pattern
Wj – the total energy in bandpass filter j of the spoken pattern
W[j] – the average peak detected value in band j of the spoken pattern
DIFF – the summation of the differences between the average peak detected values of the two words for the first six filters

\[ T_1 = \frac{W_s}{W_n} - \frac{L_s}{L_n}; \]
\[ \text{FOR} \ j = 1 \text{ TO } 6 \text{ DO} \quad L[j] = \frac{L_j}{L_n} + T_1 \times \frac{L_j}{L_s}; \]
\[ \text{FOR} \ j = 1 \text{ TO } 6 \text{ DO} \quad W[j] = \frac{W_j}{W_n}; \]
\[ \text{FOR} \ j = 1 \text{ TO } 6 \text{ DO} \quad \text{DIFF} = \text{DIFF} + (L[j] - W[j]); \]

In order to calculate \( L[j] \), first the average peak detected value is calculated \( (L_j/L_n) \). Then added to this value is the difference between the total average energy, for a frame, of the two words, multiplied by the fraction of energy of band \( j \) compared to the total energy in the pattern.

DIFF is the score calculated by the first matching algorithm, and the lower the score, the better the match between the two patterns. Every library word that passed the first criterion (that of the frame difference described previously) is compared in this way to the spoken pattern, and a score is generated. Notice that although the library word is normalized to the spoken word, the pattern for the library word remains unchanged.

Next a routine is called which rules out all the patterns except those that produced the five lowest scores derived by the first matching algorithm. This greatly speeds up the matching process without loss of accuracy.

3.3.2 Second Matching Algorithm

The second matching algorithm uses the time-warping procedure described in Section 3.2. It time-warpers the peak
detected values of the two patterns (the library pattern versus the spoken word pattern) and returns a value which represents the likeness of the two patterns. The more alike the patterns, the lower this time-warped score. Before doing the time-warping, the algorithm first must decide if a library word should be considered as a possible match, or whether it should be ruled out, and the time-warping therefore is not necessary. There are two criteria for ruling the library word out. The first is that it has been ruled out by the previous algorithm as not being one of the best matching words. As was just said the first algorithm currently rules out all except the five best matching patterns. To rule a pattern out it is simply given a very high set score (255 is currently used). The second criteria for ruling a word out, is to check if its score in the first matching algorithm was above a certain threshold. Currently a threshold of 40 is used. Therefore, as the program now stands, the second matching algorithm will have to perform the time-warping procedure on at most five library words versus the spoken word (i.e., five different matches).

Before performing the time-warping between the peak detected values of the two patterns, the patterns have to be normalized to each other. As said before, normalizing is needed because when the same word is said multiple times, it may be said with different energies. Therefore, its pattern may be the same, but it is offset at a higher, or lower, energy level. A normalization very similar to the one described for the previous matching algorithm is used.

The philosophy behind the normalization used is to give each
pattern the same amount of total energy (energy will often be used here to describe the peak detected voltage readings of the bandpass filters). That is, the sums of all the peak detected values of the patterns are to be equal. The difference in total energy of the two patterns is calculated, and then it is distributed over the spoken word pattern. This means that the spoken word pattern will have energy added to it, or subtracted from it. The energy is distributed assuming the two patterns to be similar, but one is offset from the other.

The total energy difference is distributed over the seven bandpass filters' peak detected values, according to the proportion of energy in each filter with respect to the total energy in the pattern. Therefore, if there is an energy difference of 200 between the two patterns, and filter 1 of the spoken word pattern holds 10% of the total energy for that pattern, then we want to add 20 units to the first filter. Energy is added to (subtracted from) a filter by adding to (subtracting from) the individual frame values for that filter. Again, this is done proportional to how much energy a frame represents with respect to the total energy in the band. If it is desirable to add 10 units of energy to the first filter, and the first frame in this filter amounts to 10% of the total energy in the filter, then 1 unit of energy will be added to this frame.

The algorithm which accomplishes this follows.

\[ \text{Lt} \quad \text{the total energy in the library pattern} \]
\[ \text{Ln} \quad \text{the number of frames of the library pattern} \]
\[ \text{Wt} \quad \text{the total energy in the spoken pattern} \]
\[ \text{Wn} \quad \text{the number of frames of the spoken pattern} \]
\[ E[j] \quad \text{the amount of energy which should be added to band } j \text{ of the spoken word in order to normalize it to the} \]
library word (E[j] may be negative)

W[i,j] - the two-dimensional array which holds the spoken word pattern. Rows are the frame numbers, and columns the bandpass filter. For example, W[4,3] is the peak detected value of the third filter (250Hz) for the fourth frame.

WN[i,j] - the normalized pattern of W[i,j]

N_BPF - the number of bandpass filters, which is seven

T1 := Wt/Wn; average energy per frame for the spoken pattern
T2 := Lt/Ln; average energy per frame for the library pattern
FOR j := 1 TO N_BPF
   E[j] := (T2 - T1) * (Wj/Wt) / (Wn);
END;

FOR j := 1 TO Wn
   FOR k := 1 TO N_BPF
      BEGIN
         IF E[k] <> 0 THEN
            BEGIN
               IF WN[j,k] > 255 THEN WN[j,k] := 255; allow maximum value
               IF WN[j,k] < 0 THEN WN[j,k] := 0; allow minimum value
            END;
         ELSE FOR j := 1 TO Wn
            DO WN[j,k] := W[j,k];
      END;
END;

Once the normalization is completed then the time-warping algorithm is performed between the library word pattern and the normalized pattern of the spoken word. The time-warping is performed as described in Section 3.2, with one difference. In Section 3.2 it was described how two patterns are compared. But there are seven patterns: the peak detected values (pattern) for each of the seven filters, within each pattern of a word. Therefore it is desired to compare all seven patterns of one word to the corresponding seven patterns of the other word. To do this, a modification is made to the time-warping algorithm. As explained in Section 3.2, when two frames are compared, the absolute value of the difference is calculated and used to represent the difference in numeric form of the two frames. When comparing the seven peak detected values of the bandpass filters, each frame, instead of holding one value, now has seven values associated with it. Therefore band 1 of the one pattern is
subtracted from band 1 of the other pattern, with the absolute value returned. This is done for all seven bands. The maximum value from these seven subtractions is the number used to represent the difference between the two frames. For example, if frame $i$ of one pattern had the following readings for the seven peak detected values: 10,15,5,21,34,5,2 and frame $j$ of the other pattern had the following readings: 8,10,7,24,36,2,1, then the number returned to represent the difference between the two frames would be the maximum of 2,5,2,3,2,3,1. This would, of course, be 5. In this way, by using the time-warping algorithm a number is returned which represents the likeness of the two patterns. This number is then stored and used by the next matching algorithm.

Before the next matching algorithm is called, another routine is called. This routine performs two functions. First it looks at the scores for the second algorithm (time-warping all seven bandpass filter peak detected values) for all the library words when compared to the spoken word. Notice that all but five of the library word scores will be 255 (the highest score), since they were predetermined by algorithm 1 not to be a possible match. If any score is above a threshold then the word is given a score of 255, thus ruling out this word by the second algorithm. The current threshold used is 25. If a word scored less than 25, then the two scores from algorithm 1 and algorithm 2 are added and stored in an array. Otherwise, the score stored in this array is 255, showing that this library pattern has been ruled out. If at this point all the library patterns have been ruled out, then a 'no match' conclusion is drawn, meaning that
the spoken word does not match any of the library words. If there still are patterns which have not been ruled out, then algorithm 3 will be called.

3.3.3 Third Matching Algorithm

Algorithm 3 for matching patterns may be called in two ways. If the lowest scoring (cumulative score from algorithms 1 and 2) pattern is lower by a certain value (B is currently used) than the next lowest scoring word, then algorithm 3 may be called to only verify that this pattern is correct. To verify means that this pattern needs to score lower than, or equal to, a certain threshold when compared to the spoken pattern using algorithm 3. If the two lowest scoring patterns are close together, then algorithm 3 is called to hopefully determine which, if any, of the remaining patterns is a match.

Algorithm 3 also uses the time-warping procedure, but instead of comparing the peak detected values of the seven bandpass filters, it compares the zero-crossing values of 3 of the filters. The three filters it uses are the ones with their center frequencies at roughly 1000Hz, 2000Hz, and 4000Hz.

No normalization is required here, and the time-warping algorithm is implemented as described in Section 3.2. The zero-crossing pattern for the 1000Hz filter of the library pattern under question is time-warped with the zero-crossing pattern for the 1000Hz filter of the spoken pattern. This is also done for the 2000Hz and 4000Hz zero-crossing patterns. Then these three scores are added together and returned as a numeric
representation of the likeness of the two patterns. The lower the score, the greater the likeness.

If algorithm 3 is called only to verify that a library word is a match, then it only needs to compare this library pattern with the spoken pattern and return the result. If this result is less than or equal to a certain threshold (8 is currently used) then this library word is said to match the spoken word. If this result is not within the threshold, then algorithm 3 is called again, but this time to look at all the possible library patterns that could still be matches, i.e., patterns which have not been ruled out by the previous two matching algorithms. It then returns, for each comparison, a score obtained in the manner just described.

If the word was verified successfully then no further matching needs to be done; if however, the verify procedure was not called, or called and returned with too high a score, then further work needs to be done to produce a match.

After calling algorithm 3 the scores of each of the library patterns that were considered are looked at. If their scores for just algorithm 3 are greater than or equal to a certain threshold (8 is currently used) then this word is ruled out (it is given the high score of 255). If all words have been ruled out at this point, then a 'no match' conclusion is reached, meaning that the spoken word does not match any of the library words. If there are still words which have not been ruled out, then their scores for algorithms 1, 2, and 3 are added and stored.

If the lowest scoring word after algorithm 3 (the cumulative score of algorithms 1, 2, and 3) is lower by a certain amount (5
is currently used) than the next lowest scoring word, then this library word is said to match the spoken word. If after algorithm 3 the two lowest scoring are still very close (within 4) to each other, then another matching algorithm is called to try to pick one of these two words to be the correct matching word.

3.3.4 Fourth Matching Algorithm

The fourth algorithm only looks at the two previously lowest scoring patterns. First, the frame difference (the absolute value) between the two library patterns compared to the spoken pattern is calculated. If the previously lowest scoring pattern has a frame difference which is lower by a certain number (2 is currently used) than the frame difference of the second lowest scoring word, then this lowest scoring library word is said to match the spoken word.

If, after algorithm 4, a match can still not be determined, then the spoken word is said to be 'ambiguous'.

3.3.5 Summary

In summary, the four matching algorithms will be briefly described and their interaction shown.

Algorithm 1 - The frame value difference must be less than or equal to 10, and the comparison of the normalized average peak detected values of each filter must be less than 40. Otherwise this pattern is ruled out. Next, only the five lowest scoring patterns are kept; all other patterns are ruled out.
Algorithm 2 - Of the remaining patterns, the time-warped
difference of the peak detected values of the seven filters, all
time-warped together, is calculated. If the score for just the
time-warping is greater than 25, then the pattern is ruled out,
otherwise, the two scores from algorithms 1 and 2 are added and
stored. If all patterns have been ruled out, then a 'no match'
condition has occurred.

Algorithm 3 - If the lowest scoring word is lower by 8 than
the second lowest scoring word, then the word is to be verified.
To verify a word, the zero-crossing patterns for the 1000Hz,
2000Hz and 4000Hz filters are time-warped separately and the
three scores added together. If the cumulative score is less
than or equal to 8 then this pattern (word) is said to match with
the spoken pattern (word), and no more matching is needed.

If the verify score is too high, or the verify was never
called, then all the remaining library patterns are matched with
the spoken pattern. This is done, as just mentioned, by
performing the time warping procedure, between the library
pattern and the spoken pattern, on the zero-crossing patterns for
the 1000 Hz, 2000 Hz, and 4000 Hz filters. These three scores
are then added.

If this score is greater than or equal to a certain
threshold (8 is currently used), then this pattern is ruled out.
If, at this point, all patterns have been ruled out, then a 'no
match' condition has occurred. If there are still some patterns
left, then their cumulative scores for algorithms 1, 2, and 3 are
calculated. If the lowest scoring pattern is less than the
second lowest scoring pattern by a certain amount (5 is currently
used), then this lowest scoring library word is said to match the spoken word. If the two lowest scoring words are within 4 of each other, then algorithm 4 is called.

Algorithm 4 - The frame differences between the lowest scoring word and the spoken word, and the second lowest scoring word and the spoken word, are calculated. If the former frame difference is lower by at least 2 of the latter, then the lowest scoring library word is said to match the spoken word, otherwise, the spoken word is said to be 'ambiguous' with respect to the library.

3.4 Pattern Averaging and Adding to the Library

As shown in the previous section, a spoken word may match with a library word, may be ambiguous with respect to the library, or may have no match with respect to the library. If the spoken word matched with a library word, then it needs to be determined if it matched correctly or not. Since there are two ways to receive a word pattern for matching against the library, there are two ways to see if the match was correct. If the received word pattern was from a previously made data file (see Chapter 2) then the character string that represents this pattern is available in the file. The program only needs to compare this character string against the character string that represents the library word (which has already been read in). If the strings compare successfully then the word has been matched successfully, else it was an incorrect match. If the spoken pattern was received directly from the microphone (live data), then the
program tells the user the matched word, and asks if it was correct. The user then supplies the program with the appropriate answer.

Therefore, after the matching procedure is complete, one of four conclusions about the received word has been reached. It has been matched correctly, incorrectly, it is ambiguous, or it does not match.

If the word does not match, or has been matched incorrectly, then it may be added to the library, if the library is not full. The word is added at the end of the library in the manner discussed in Chapter 2. This new library word may then be used to match with words that will be received. The user has the option to configure the program to add words to the library, or to not add them. The next section (3.5) describes how to use the program.

If the word has been matched correctly then its pattern may be averaged with the library word pattern to which it matched. The motivation behind this is to get a better representation of the word in the library. In Chapter 5 the results are given for using the same data with library pattern averaging and without averaging. Also, the counter associated with the library word that matched is incremented by one.

The averaging is done in the following way. All the peak detected values and the zero-crossing values are averaged between the spoken pattern and the library pattern. It would not seem wise to do a one-to-one averaging, since the library pattern may have been averaged many times, and this would give too much.
influence to the word that was just received. It may be best to weight the library pattern according to how many times it has been averaged. Currently the library word is given a weight of four (80%) and the received word is given a weight of one (20%).

Finally, the boundaries need to be determined for averaging. This is done by centering the shorter pattern onto the longer pattern. The places where they overlap are averaged. At the ends, where only the longer pattern has frames, the longer pattern's frames are used. For example, if the library pattern had 30 frames and the received pattern had 34 frames, then the averaged pattern would be obtained as follows. Frames 1 and 2 would be frames 1 and 2 of the received pattern. Frames 3-32 would be the weighted average of the two patterns (frames 3-32 of the received pattern with frames 1-30 of the library pattern). And frames 33 and 34 would be frames 33 and 34 of the received pattern.

Therefore, if a pattern matches correctly it may be averaged with the library word it matched to, if the program is configured in this manner.

Finally, if a received word is ambiguous, then it may be ignored, averaged with a library word, or added to the library. If the program is configured so as not to allow averaging, or adding to the library, then the word will be ignored. If averaging is allowed then the received pattern will be averaged with the lowest scoring library word, if this library word was the same as the word that was said. If the lowest scoring library word is not the same as the received word, i.e., the word has been matched incorrectly, then the received word will be
added to the library. This addition is done provided that the library is not full, and the program has been configured to allow adding to the library.

3.5 How to Use the Program

To run the matching program, type RMATCH9. The first question it will ask is "What is the Library File Name?" The library file name, described in Section 3.1, is to be entered. The library is read in and each word is displayed with its current counter value.

Next the user is asked if received patterns should be averaged (this should not be confused with library wave averaging). Averaging received waves means that the value used for the peak detected value for frame i, of bandpass filter j, will be the average of the values obtained by the feature extractor for frame i-1 and i, of band j. This averaging was found neither to help nor hinder the matching procedure, and was used for the experiments.

After answering this question, the user will be asked how the library should be configured. The user may choose one of three options: 1) library averaging and adding allowed, 2) no library averaging, but adding is allowed, or 3) no library averaging or adding is allowed. Library averaging and adding were discussed in Section 3.4 and the user must choose one of these three options.

Next the user will be asked if the results should go to the print device. A sample of these results is given in figure
3.5.1, and are used for developmental purposes.

Figure 3.5.1 - Printed results of matching algorithms
The user is then given a list of thresholds and variables which may be changed. This list is given in the following format:

- (0) No change
- (1) FRAME THRESHOLD: 10
- (2) STAGE 1 THRESHOLD: 40
- (3) STAGE 2 THRESHOLD: 25
- (4) STAGE 2 DIFFERENCE: 8
- (5) STAGE 3 THRESHOLD: 8
- (6) STAGE 3 DIFFERENCE: 5
- (7) STAGE 4 DIFFERENCE: 2
- (8) AVERAGING WEIGHT FACTOR: 4
- (9) STAGE 1 KEEP: 5

The user is allowed to change any of the above values. The thresholds and variables shown above were described in Section 3.3, with the exception of the AVERAGING WEIGHT FACTOR, which was described in Section 3.4. Stage 1 refers to algorithm 1, stage 2 to algorithm 2 and so on. Whenever the above parameters are set, the user should enter '0' to proceed.

Next, the user must tell the program whether the spoken word patterns will be received live from the microphone (through the feature extractor), or from an already made data file. How to set up these data files is discussed in Section 3.1.

If the user decides to use some previously made data files then he/she will be asked to enter the names of the desired data files to use for comparison. Up to 10 files may be entered. When all the desired file names have been entered, then an '*' should be typed to denote that the user has finished. The program will now start matching the words in these data files to the library that was previously read in. The library will be updated according to how it was configured. When each word is processed, some intermediate results are printed out. Also, the current number of correctly matched, incorrectly matched,
ambiguous, and no match words, for the current set of files is printed. When completed, the library, if it has been updated at all, is written back out to disk. When the library is written back, each library word is printed with its current count value.

If the user decides to directly enter words through the microphone, then this option may be chosen. Everything works as described previously, except that instead of the program receiving words from another file, it prompts the user to say a word into the microphone. After obtaining the word pattern from the feature extractor, the pattern is matched against the library file that was previously read in. It should be noted that from this program the parameters of the feature extractor cannot be changed. Therefore, if other parameter values besides the default ones are desired they must be changed using the TRAIN5 program. This is easily accomplished by running the TRAIN5 program, changing the feature extractor parameter values, exiting the TRAIN5 program and running the RMATCH9 program. Note that the parameter values for the feature extractor will not change as long as the feature extractor is not turned off.

As noted before, the RMATCH9 program keeps statistics as to the number of words that were matched correctly, incorrectly, ambiguously, or did not match, for that particular running of the program. These statistics are printed out after every word, and also after a whole file has been processed.
The purge program, called PRGFLE, deletes from a file the words that have been matched a very small number of times. The motivation behind this is to delete from a library those word patterns which are seldom matched successfully, and thus make room for more patterns in the library. The program makes use of the counter variables, associated with each word in a file, to decide which words should be deleted.

Each word in a file has a counter variable which is incremented every time this word successfully matches to a received word. The purge program first sums up all the counts. If this sum is less than three times the number of words in the file, then the purge program exits automatically, with the conclusion that not enough matches have been made to rightfully determine what words should be deleted. If the total count is greater than three times the number of words in the file, then the program proceeds.

Next, the limit for a minimum count must be set. The
minimum count is the lowest number of times a word must match for keeping the word in the file, i.e., if a word has matched less than this number, then it will be deleted from the file. This number is the rounded off value of the total count divided by the number of words in the file, divided by three. The formula being:

\[ \text{THR2} = \text{ROUND}\left(\frac{\text{TCDUNT}/\text{NWORDS}}{3}\right) \]

The program then writes out to a new file the words which have counts above or equal to this threshold. The new file is given the same name as the old file, but with an extension of NEW.

To run the program, type PRGFLE. The program will first ask the user what is the file name of the file to be purged. After obtaining this, the file is read in and each word with its current count value is displayed.

The program will next add up all the count values. And if this number is too small, as described previously, it will exit. Otherwise, it will continue on.

The total count is then printed, followed by the minimum count for keeping a word. At this point the program stops and allows the user to survey which words are in the file, and which words will be kept for the new file. After the user presses the 'return', the program prints out each word, its count, and whether or not it will be added or not to the new file.

At this point the program stops and allows the user to either continue, or exit. If the user chooses to continue, then
the new file name is formed and displayed. This new file name will be the same as the one entered in previously, but with the extension of NEW. The user is then prompted to press 'return' to continue. The accepted words are then written out to the disk. While writing this new file, each word is displayed with its old count value.

Every word that is written out to file is given a new count value of zero. This is because new words will be added to the file, using RMATCH9, with a count value zero. Therefore, all counts should start at zero for a newly created library.
CHAPTER 5

THE GRAPH PROGRAM

The program called GRAPH will display word patterns. A display, or graph, of a word pattern will consist of a series of bar graphs of the peak detected values versus time, and the zero crossing values versus time, at each filter (also, the zero crossing values at the microphone are graphed). Figure 5.1 gives the graph of the 62.5 Hz filter for the word "black." And figure 5.2 gives the graph of the 2000 Hz filter for the same word.

The program first obtains the name of the file that contains the word patterns to be graphed. It then allows the user to graph the entire file, or to choose certain words. If the user chooses to graph the entire file, then it will be automatically graphed, with no more need for the user to interact with the program.

If the user wants only certain words graphed, then the program will start with the first word in the file and ask the user if a graph of this word is desired. If a graph is desired then the program will produce this graph, read in the next word,
Figure 5.1 - Response of 62 Hz filter for the word "black."
Figure 5.2 - Response of 2000 Hz filter for the word "black."
and again ask the user if a graph of this next word is desired. If a graph is not desired for the current word, then the program will go on to the next word and again prompt the user. In this way the user may graph certain words without having to graph the entire file. If the whole file is not being graphed, the user may exit the program at any time by entering in an '*' when the main prompt appears.

The program makes a graph in the following way. Each word in the file is read in. If a graph of this word's pattern is desired, every frame of the pattern is read in and converted from hexadecimal to decimal and stored. If a graph of this word is not desired, every frame is read in, but ignored, until the end of this word's pattern is reached. It is necessary for the program to read in even words which will not be graphed in order to move the file marker to the next word.

The peak detected values and the zero crossing values are displayed in bar graph form for each filter (beginning with the 62.5 Hz filter and ending with the 4000 Hz filter). The peak detected values are first divided by four before being displayed, in order to make sure all the readings will fit onto the graph. Finally, zero crossing values for the microphone are graphed.
6.1 Experiment 1

In the first experiment the following twelve words were used as a vocabulary: purple, blue, green, yellow, orange, red, brown, black, white, grey, gold, and pink. These words were said ten times, in the above order, on ten different days. An original library was created the first day, and contained one copy of each word. The resulting 1200 words were then compared to the library with the library being manipulated in four different ways. These four different ways are defined below.

Setup 1 - Adding to the library (until the library reached a maximum of 36 words) was allowed, but averaging was not allowed.

Setup 2 - Adding to the library was allowed and averaging was allowed.

Setup 3 - Adding to the library was allowed and averaging was allowed. The library was purged after 600 words (5 days of data) were compared.

Setup 4 - Adding to the library was allowed and averaging was allowed...
was allowed. The library was purged after every 240 words (2 days of data) were compared.

Table 6.1.1 gives the percentage of words that were matched correctly (right), incorrectly (wrong), ambiguously, or did not match with the library (no match).

Table 6.1.2 gives for each different word the number of times it matched correctly (out of 100) for the four different setups.

---

**TABLE 6.1.1**

RESULTS OF THE FOUR DIFFERENT SETUPS

<table>
<thead>
<tr>
<th>Setup #</th>
<th>Right</th>
<th>Wrong</th>
<th>Ambig.</th>
<th>No Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup 1</td>
<td>80%</td>
<td>4</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>Setup 2</td>
<td>82</td>
<td>4</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>Setup 3</td>
<td>85</td>
<td>3</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>Setup 4</td>
<td>84</td>
<td>4</td>
<td>1</td>
<td>11</td>
</tr>
</tbody>
</table>

**TABLE 6.1.2**

NUMBER OF CORRECT MATCHES FOR EACH WORD

<table>
<thead>
<tr>
<th>Word</th>
<th>Setup 1</th>
<th>Setup 2</th>
<th>Setup 3</th>
<th>Setup 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purple</td>
<td>53</td>
<td>60</td>
<td>58</td>
<td>49</td>
</tr>
<tr>
<td>Blue</td>
<td>75</td>
<td>73</td>
<td>72</td>
<td>78</td>
</tr>
<tr>
<td>Green</td>
<td>98</td>
<td>98</td>
<td>97</td>
<td>97</td>
</tr>
<tr>
<td>Yellow</td>
<td>98</td>
<td>78</td>
<td>91</td>
<td>95</td>
</tr>
<tr>
<td>Orange</td>
<td>83</td>
<td>93</td>
<td>94</td>
<td>94</td>
</tr>
<tr>
<td>Red</td>
<td>74</td>
<td>89</td>
<td>89</td>
<td>80</td>
</tr>
<tr>
<td>Brown</td>
<td>86</td>
<td>82</td>
<td>82</td>
<td>84</td>
</tr>
<tr>
<td>Black</td>
<td>70</td>
<td>80</td>
<td>82</td>
<td>82</td>
</tr>
<tr>
<td>White</td>
<td>90</td>
<td>85</td>
<td>84</td>
<td>93</td>
</tr>
<tr>
<td>Grey</td>
<td>67</td>
<td>96</td>
<td>96</td>
<td>87</td>
</tr>
<tr>
<td>Gold</td>
<td>86</td>
<td>81</td>
<td>95</td>
<td>93</td>
</tr>
<tr>
<td>Pink</td>
<td>80</td>
<td>70</td>
<td>73</td>
<td>75</td>
</tr>
</tbody>
</table>

Total | 962     | 985     | 1013    | 1007    |
Tables 6.1.3 gives for each day the percentage of words that matched correctly.

**TABLE 6.1.3**
PERCENTAGE OF CORRECT MATCHES PER DAY

<table>
<thead>
<tr>
<th>Date</th>
<th>Setup 1</th>
<th>Setup 2</th>
<th>Setup 3</th>
<th>Setup 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>07/11/85</td>
<td>83%</td>
<td>85</td>
<td>85</td>
<td>85</td>
</tr>
<tr>
<td>07/12</td>
<td>84</td>
<td>82</td>
<td>82</td>
<td>82</td>
</tr>
<tr>
<td>07/13</td>
<td>82</td>
<td>83</td>
<td>83</td>
<td>81</td>
</tr>
<tr>
<td>07/15</td>
<td>75</td>
<td>80</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>07/17</td>
<td>78</td>
<td>82</td>
<td>82</td>
<td>83</td>
</tr>
<tr>
<td>07/18</td>
<td>90</td>
<td>92</td>
<td>91</td>
<td>93</td>
</tr>
<tr>
<td>07/19</td>
<td>81</td>
<td>83</td>
<td>89</td>
<td>88</td>
</tr>
<tr>
<td>07/22</td>
<td>80</td>
<td>78</td>
<td>88</td>
<td>81</td>
</tr>
<tr>
<td>07/23</td>
<td>75</td>
<td>81</td>
<td>83</td>
<td>86</td>
</tr>
<tr>
<td>07/24</td>
<td>75</td>
<td>75</td>
<td>82</td>
<td>81</td>
</tr>
</tbody>
</table>

6.2 Experiment 2

In the second experiment the following twelve words were used as a vocabulary: zero, one, two, three, four, five, six, seven, eight, nine, ten, and dog. These words were said ten times, in the above order, on four different days. An original library was created the first day, and contained one copy of each word. The resulting 480 words were then compared to the library. The library was manipulated as in setup 2 of experiment 1. That is, adding to the library and averaging were allowed, but purging was not done.

Table 6.2.1 gives for each day the percentage of words matched correctly (right), incorrectly (wrong), ambiguously, or did not match with the library (no match). The totals are also given.
Table 6.2.1 gives for each word the number of times it matched correctly.

<table>
<thead>
<tr>
<th>Date</th>
<th>Right</th>
<th>Wrong</th>
<th>Ambig</th>
<th>No Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>07/30/85</td>
<td>89%</td>
<td>1</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>08/02</td>
<td>94</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>08/03</td>
<td>93</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>08/05</td>
<td>88</td>
<td>6</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>91%</strong></td>
<td>3</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 6.2.2 gives for each word the number of times it matched correctly.

Table 6.2.2
NUMBER OF CORRECT MATCHES FOR EACH WORD

<table>
<thead>
<tr>
<th>Word</th>
<th>Number of times correctly matched</th>
<th>Percentage of correct matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>37</td>
<td>93</td>
</tr>
<tr>
<td>One</td>
<td>35</td>
<td>85</td>
</tr>
<tr>
<td>Two</td>
<td>31</td>
<td>78</td>
</tr>
<tr>
<td>Three</td>
<td>37</td>
<td>93</td>
</tr>
<tr>
<td>Four</td>
<td>37</td>
<td>93</td>
</tr>
<tr>
<td>Five</td>
<td>39</td>
<td>98</td>
</tr>
<tr>
<td>Six</td>
<td>31</td>
<td>78</td>
</tr>
<tr>
<td>Seven</td>
<td>36</td>
<td>90</td>
</tr>
<tr>
<td>Eight</td>
<td>37</td>
<td>93</td>
</tr>
<tr>
<td>Nine</td>
<td>40</td>
<td>100</td>
</tr>
<tr>
<td>Ten</td>
<td>37</td>
<td>93</td>
</tr>
<tr>
<td>Dog</td>
<td>39</td>
<td>98</td>
</tr>
</tbody>
</table>

6.3 Experiment 3

In the third experiment the following ten words were used as a vocabulary: north, south, east, west, up, down, right, left, stop and go. These words were said ten times, in the above order, on four different days. An original library was created the first day and contained one copy of each word. The resulting 480 words
were then compared to the library. The library was manipulated as in setup 2 of experiment 1. That is, adding to the library and averaging were allowed, but purging was not done.

Table 6.3.1 gives for each day the percentage of words matched correctly (right), incorrectly (wrong), ambiguously, or did not match with the library (no match). The totals are also given.

**TABLE 6.3.1**

RESULTS OF THE THIRD EXPERIMENT

<table>
<thead>
<tr>
<th>Date</th>
<th>Right</th>
<th>Wrong</th>
<th>Ambig</th>
<th>No Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>08/06/85</td>
<td>78%</td>
<td>11</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>08/07</td>
<td>69</td>
<td>9</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>08/08</td>
<td>69</td>
<td>11</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>08/09</td>
<td>80</td>
<td>4</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>Total</td>
<td>74%</td>
<td>8</td>
<td>10</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 6.3.2 gives for each word the number of times, out of forty, that it matched correctly.

**TABLE 6.3.2**

NUMBER OF CORRECT MATCHES FOR EACH WORD

<table>
<thead>
<tr>
<th>Word</th>
<th>Number of times correctly matched</th>
<th>Percentage of correct matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>North</td>
<td>38</td>
<td>95</td>
</tr>
<tr>
<td>South</td>
<td>23</td>
<td>58</td>
</tr>
<tr>
<td>East</td>
<td>34</td>
<td>85</td>
</tr>
<tr>
<td>West</td>
<td>16</td>
<td>40</td>
</tr>
<tr>
<td>Up</td>
<td>33</td>
<td>83</td>
</tr>
<tr>
<td>Down</td>
<td>37</td>
<td>93</td>
</tr>
<tr>
<td>Right</td>
<td>27</td>
<td>68</td>
</tr>
<tr>
<td>Left</td>
<td>35</td>
<td>88</td>
</tr>
<tr>
<td>Stop</td>
<td>18</td>
<td>45</td>
</tr>
<tr>
<td>Go</td>
<td>35</td>
<td>88</td>
</tr>
</tbody>
</table>
6.4 Experiment 4

In the fourth experiment, a person not familiar with the system used his voice to generate the data. The vocabulary consisted of the words violet, blue, green, yellow, orange, red, brown, black, white, grey, and gold. These words were said 12 to 14 times each, in the above order, on two different days. An original library was created the first day and contained one copy of each word. The resulting 286 words were then compared to the library. The library was manipulated as in setup 2 of experiment 1. That is, adding to the library and averaging was allowed, but purging was not done.

These are the same words as used in experiment 1, with the exception that violet is used in this set and purple and pink are not. This experiment was done to show how well the system behaved with someone who has not repeatedly been saying words into the system, thereby possibly making his voice easier to recognize. Conclusive evidence about the performance of the system should not be drawn by comparing the results of experiment 1, with experiment 4, since a much more in depth study would be needed.

Table 6.4.1 gives for each day the percentage of words matched correctly (right), incorrectly (wrong), ambiguously, or did not match with the library (no match). The totals are also given.
### TABLE 6.4.1
RESULTS OF THE FOURTH EXPERIMENT

<table>
<thead>
<tr>
<th>Date</th>
<th>Right</th>
<th>Wrong</th>
<th>Ambig</th>
<th>No Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/04/85</td>
<td>81%</td>
<td>6</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>09/11</td>
<td>74%</td>
<td>8</td>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>77%</td>
<td>7</td>
<td>5</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 6.4.2 gives for each word the number of times, out of forty, that it matched correctly.

### TABLE 6.4.2
NUMBER OF CORRECT MATCHES FOR EACH WORD

<table>
<thead>
<tr>
<th>Word</th>
<th>Number of times correctly matched</th>
<th>Percentage of correct matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Violet</td>
<td>25</td>
<td>96</td>
</tr>
<tr>
<td>Blue</td>
<td>17</td>
<td>65</td>
</tr>
<tr>
<td>Green</td>
<td>16</td>
<td>62</td>
</tr>
<tr>
<td>Yellow</td>
<td>19</td>
<td>73</td>
</tr>
<tr>
<td>Orange</td>
<td>18</td>
<td>69</td>
</tr>
<tr>
<td>Red</td>
<td>22</td>
<td>85</td>
</tr>
<tr>
<td>Brown</td>
<td>21</td>
<td>81</td>
</tr>
<tr>
<td>Black</td>
<td>24</td>
<td>92</td>
</tr>
<tr>
<td>White</td>
<td>21</td>
<td>81</td>
</tr>
<tr>
<td>Grey</td>
<td>17</td>
<td>85</td>
</tr>
<tr>
<td>Gold</td>
<td>21</td>
<td>81</td>
</tr>
</tbody>
</table>
7.1 Discussion

In the experimentation (Chapter 6) it was shown that three sets of words, consisting of ten to twelve words per set, had a recognition rate between 70 and 90 percent. That is, the average recognition rate for words, within a given set, was between 70 and 90 percent.

For the set consisting of the words zero, one, ..., ten, and dog, the recognition rate was 91 percent. A word was incorrectly identified at a rate of only 3 percent. The remaining 6 percent of the time a word was not identified.

The system allows any set of words to be used as a vocabulary. In the experimentation, each of the three sets of words chosen belongs to a category of words. These three categories are colors, digits, and directions.

As can be seen in Chapter 6, the recognition rate does depend upon the group of words chosen. It is more difficult for
the system to differentiate between words that sound similar. But it is also true, that if one is careful to pick out words that have very distinctive sounds with respect to each other, then a high recognition rate can be achieved.

When the voice of someone who had not used the system before was used, the recognition rate went down slightly. This should be expected because that person would not pronounce the words as uniformly as a person who has been working with the system for some time. It is encouraging to see the recognition rate as high as 77 percent for this person.

7.2 Suggested Improvements

- The system is at the stage where the algorithms need to be refined and further experimentation conducted.
- Testing is needed in order to see the effects of pattern averaging, and library purging.
- Testing needs to be done to see how many different words can be used in one vocabulary (library) file.
- The speed of the system may now also be a factor to consider. Currently, with the system set up as described, it takes from 5 to 30 seconds to recognize a word. Speed has not been considered as a problem so far, and therefore something can probably be done to improve it.
- With a little more development, this system should be ready for implementation on another system which can be enhanced by receiving voiced words. One implementation is on a robot
whose movements may be controlled by a set of voiced commands.

7.3 Concluding Remarks

It is hard to show, in numbers, how much the system has been improved since its initial design. It has been in a constant state of development, with different people using their voice to test it, and different methods to measure performance. Also, more elaborate testing has been done on it recently. It is safe to say that the system has progressed very well. Using very common electronic components (resistors, capacitors, operational amplifiers, a microprocessor) for circuit design, and software run on a personal computer for pattern matching, a high recognition rate has been achieved for different groups of words.

Not only has the pattern matching software been improved, but much has been done to allow for further development of the system, and ease of experimentation.

Four matching algorithms, described in Chapter 3, are currently used. Many of the parameters that control these algorithms can easily be changed to allow the user to further refine the matching procedure.

A first version of the program to update the library (the patterns used for matching) was completed. Currently, the library may be updated by adding new word patterns and by averaging word patterns that match successfully (see Section 3.4). The averaging of word patterns slightly improved the system's recognition rate as can be seen in Chapter 6, experiment 1. Further testing needs to be done in order to show how much of an
Purging of the library was also introduced here (see Chapter 4). Again, a slight improvement was seen when the library was purged in experiment 1 (Chapter 6). More testing needs to be done to determine how much this improves the recognition rate.

Large word files can now be created and processed later by the user. The processing (attempting to recognize words) can be done without user interaction once the program is initiated. Statistics about the recognition rate are kept by the program, and statistics about each matching attempt can be printed out. All of this saves the user a considerable amount of time while doing experimentation.

The use of storing word files also allows the user to process the same data but with different matching algorithms and/or different parameter settings. This makes for more accurate testing.
**PROGRAM RECOGN ( INPUT, OUTPUT );**

**CONST**

- `INF_DIST = 32767;`  
  (LARGEST INTEGER VALUE)
- `HSCORE = 255;`  
  (MAXIMUM SCORE DURING MATCHING)
- `MAX_SAMPLE = 60;`  
  (MAXIMUM SAMPLES PER WORD)
- `MIN_SAMPLE = 16;`  
  (MINIMUM SAMPLES PER WORD)
- `N_BPF = 7;`  
  (NUMBER OF ENERGY BANDS)
- `N_IC = 8;`  
  (NUMBER OF ZERO CROSSING BANDS (INCLUDES MIC))
- `N_ELH = 15;`  
  (N_BPF+N_IC, NUMBER OF ELEMENTS IN A FRAME)
- `MAX_VOCAB = 36;`  
  (MAXIMUM LIBRARY SIZE)
- `FE_START = 1;`  
  (STARTS F-E CONVERSION)
- `RESET = 0;`  
  (RESET THE F-E)
- `END_WORD = 6;`  
  (END OF WORD MARKER)
- `END_FILE = 2;`  
  (END OF FILE MARKER)

**TYPE**

- `TIME_SAMPLE = ARRAY [1..N_ELH] OF BYTE;`  
  (ONE FRAME OF DATA)
- `SAMPLE = ARRAY [1..MAX_SAMPLE] OF TIME_SAMPLE;`  
  (WORD PATTERN)
- `LINE_TYPE = ARRAY [1..40] OF CHAR;`  
  (INPUT LIME BUFFER)
- `BAND_SAMPLE = ARRAY [1..MAX_VOCAB,1..N_BPF] OF INTEGER;`
- `WS_SAMPLE = ARRAY [1..MAX_VOCAB] OF INTEGER;`

**VAR**

- `VOCAB : TEXT;`  
  (library_file, holds patterns for words)
- `VOCAB_FILE : STRING[19];`  
  (INPUT LIBRARY FILE NAME)
- `FILENAME2 : STRING[19];`  
  (COMPARISON FILE NAME)
- `VOCAB2 : TEXT;`  
  (COMPARISON FILE WORD PATTERNS)
- `C_FILES : ARRAY [1..10] OF STRING[12];`  
  (Desired comparison files)
- `CMD1,CMD2,CMD3,CMD4,CMD5 : CHAR;`  
  (User commands)
- `CMD6,CMD7,CMD8 : CHAR;`
- `NL : CHAR;`  
  (REPRESENTS ASCII VALUE FOR A NEWLINE)
- `OV_FLAG : INTEGER;`  
  (B751 BUFFER OVERFLOW FLAG)
- `N_WORDS : INTEGER;`  
  (NUMBER OF WORDS IN LIBRARY)
- `WORDS : ARRAY [1..MAX_VOCAB] OF RECORD;`  
  (DATA STRUCTURE FOR HOLDING EACH WORD)
- `CNT : INTEGER;`  
  (NUMBER OF CORRECT MATCHES FOR THIS WORD)
- `N : INTEGER;`  
  (NUMBER OF FRAMES)
- `TXT : STRING[20];`  
  (TEXT STRING OF THIS WORD)
- `DAT : SAMPLE;`  
  (WORD PATTERN)
- `END;`
- `M_ADJ : INTEGER;`  
  (MAXIMUM BOUNDARY ADJUSTMENT FOR TIME-WARPING)
- `S_P : INTEGER;`  
  (SILENCE PERIOD COUNT FROM FE BOX)
- `ZC_VAL : INTEGER;`  
  (ZERO CROSSING TIME-WARPING SCORE)
- `ZC_D : ARRAY [1..MAX_SAMPLE,1..MAX_SAMPLE] OF INTEGER;`  
  (ZERO CROSSING TIME-WARPING DISTANCE ARRAY)
- `D : ARRAY [1..MAX_SAMPLE,1..MAX_SAMPLE] OF INTEGER;`  
  (BANDPASS ENERGY TIME-WARPING DISTANCE ARRAY)
- `C_IND : INTEGER;`  
  (INDEX IN C_FILES TO DENOTE WHAT COMPARISON)
PROCEDURE READLINE ( VAR F : TEXT; VAR LINE: LINE_TYPE );

{ READS IN ONE LINE (FRAME) INTO AN ARRAY OF CHAR (LINE), TERMINATING
  WITH <NL> FROM THE TEXT FILE PASSED THROUGH F }

VAR I : INTEGER;

BEGIN
  I := 0;
  REPEAT
    I := I + 1;
    READ(F,LINE[I]);
    UNTIL LINE[I]=NL;
END;

PROCEDURE CONVERT ( LINE : LINE_TYPE; VAR ARY : SAMPLE; S : INTEGER;
                      VAR MNSFLAG : INTEGER);
ASCII to its numeric value and stores in row s of passed ary.
If it is found then a buffer overflow condition occurred in the 8751 buffer
and it flags this by setting the OV_FLAG=1. If at any time a negative
value is read for an energy or zero crossing value, or a value which is
greatly different than in the previous frame, then the MNSFAS is set to 1.
This problem sometimes occurs, cause unknown.

VAR  
  I     : INTEGER;
  ITNES : INTEGER;

FUNCTION HEX (.CH : CHAR) : INTEGER;

( CONVERT HEX DIGIT IN ASCII TO INTEGER )

BEGIN
  IF (CH)='O' AND (CH)='9')
    THEN HEX := ORD(CH) - ORD('0');  ( 0..9 )
  ELSE HEX := ORD(CH) - ORD('A') + 10;  ( A..F )
END;

FUNCTION MNSFAS:

VAR  
  INH5 : INTEGER;
  FOR I := 1 TO N, Elm DO BEGIN
    IF (LINE[I+I]=') OR (LINE[I]=') THEN OV_FLAG:=1;
    ITNES := HEX(LINE[I-I]) + 16 * HEX(LINE[I]);
    ARY5[I]=ITNES;
    IF S > 1 THEN IF ABS(ARY5[I]-ITNES) > 5 THEN MNSFAS:=1;
    IF ITNES < 0 THEN MNSFAS:=1;
END;

PROCEDURE INITIALIZE;

( READ IN LIBRARY FILE, SET PARAMETERS TO THEIR DEFAULT VALUES )

VAR I,J,K : INTEGER;  ( LOOP VARIABLES )
  NAME1 : STRING[20];  ( TEXT STRING OF A LIBRARY WORD )
  NAMED : STRING[20];  ( THE TEXT STRING IN UPPERCASE )
  CHR : CHAR;  ( STORAGE VARIABLE )
  LINE : LINE_TYPE;  ( AN ARRAY TO HOLD A FRAME OF DATA )

BEGIN
  WRITE('WHAT IS THE LIBRARY FILE NAME? ');  
  READLN(VOCAB_FILE);
  ASSIGN(VOCAB,VOCAB_FILE);
  RESET(VOCAB);
  FOR K:=1 TO MAX_SAMPLE DO ( CLEAR ARRAYS D AND ZC_D )
    FOR J:=1 TO MAX_SAMPLE DO ( TURBO DOES NOT CLEAR ARRAYS )
      BEGIN
        D[K,J]:=0;
        ZC_D[K,J]:=0;
      END;
  FOR K:=1 TO 40 DO ( CLEAR ARRAY LINE )

65
LINH$J:= ' ';
NL:= CHAR(10);
NUM:= 4;
THR1:= 40;
THR2:= 25;
THR3:= 25;
REC_DIF2:= 3;
REC_DIF3:= 5;
REC_DIF4:= 2;
FRK_THR:= 10;
L_DIST:= 3;
N_ADJ:= 1;
N_WORDS:= 0;
N_RIGHT:= 0;
N_WORNS:= 0;
N_AMBIG:= 0;
N_NONMATCH:= 0;
EN_COUNT:= 7;
THREND[1] := 250;
FOR J:= 2 TO 6 DO THREND[J] := 10;
THREND[7] := 3;
SPC:= 0;
{ WRITE (OUTPUT,"ENTER SILENT PERIOD COUNT: "); }
{ READLN (INPUT, SPC); }
WRITE (OUTPUT, "READING IN LIBRARY. PLEASE WAIT... ");
REPEAT
  N_WORDS:= N_WORDS + 1;
  READLN (VOCAB, NAME1);
  NAME1:= ' ';
  FOR K:= 1 TO LENGTH (NAME1) DO BEGIN
    CHR:= COPY (NAME1, K, 1);
    CHR:= UPCASE (CHR);
    NAME1:= CONCAT (NAME1, CHR);
  END;
  WORDSIN_WORDS1.TXT:= NAME1;
  READLN (VOCAB, WORDSIN_WORDS1.CNT);
  WRITE (WORDSIN_WORDS1.TXT: 12, ' - ', WORDSIN_WORDS1.CNT);
  I := 0;
  READLINE (VOCAB, LINE);
  WHILE LINE[I] <> END_FILE DO BEGIN
    IF LINE[I] <> END_WORD THEN BEGIN
      I := I + 1;
      CONVERT (LINE, WORDSIN_WORDS1.DAT, I, MNSFLAG);
      IF MNSFLAG=1 THEN I := I - 1;
    END;
    READLINE (VOCAB, LINE);
  END;
  WORDSIN_WORDS1.N := I
UNTIL EOF (VOCAB) OR (N_WORDS = MAX_VOCAB);
WRITE;
IF N_WORDS = MAX_VOCAB THEN BEGIN
  FULLFLAG2:= 1;
  (LIBRARY FULL FLAGS )
  FULLFLAG:= 1;
}
ELSE BESIH
FULLFLAS 1= O;
END}   
IIRITELN;
{ ELSE, CLEAR FLABS}
IIRlTE{SHOULD INPUT WAVES BE AVERAGED ? 'J;
READLN(CMD3);
WRITE;
WRITE('HOW IS THE LIBRARY TO BE CONFIGURED ? ');
WRITE('A - AVERAGING AND ADDING');
WRITE('B - NO AVERAGING, BUT ADDING ALLOWED');
WRITE('C - NO AVERAGING, AND NO ADDING');
WRITE;
WRITE('CHOOSE A, B, OR C ');READLN(CMD3);
WRITE;
WRITE('SHOULD RESULTS GO TO LST DEVICE ? ');
READLN(CMD15);
WRITE;
WRITE('WILL RECEIVED WORD PATTERNS BE FROM ');
WRITE('F - FILE ALREADY MADE');
WRITE('S - SPOKEN WORD VIA THE FEATURE EXTRACTOR ');
READLN(CMD3);
IF CMD3 = 'S' THEN BEGIN
FILENAME2:='LIVE DATA'; ( FOR DISPLAY (Routines WLOG )
WIDTH:=0; ( AND WLOGLST) PURPOSES )
END;
END;

(--------------------------------------------------------------}

PROCEDURE TOTAL_DIST ( ARY1 : SAMPLE; A : INTEGER;
ARY2 : SAMPLE; B : INTEGER;
VAR Val:INTEGER; VAR ZVal:INTEGER;
E_OR_ZC:CHAR);
( Subroutines within TOTAL_DIST ARE DIST, GET_D, DP_FCN, SET_BOUNDS
VARIABLE PASSED ARE THE TWO WORD PATTERNS TO BE COMPARED, ARY1 AND ARY2,
AND THE NUMBER OF FRAMES IN EACH PATTERN, A AND B RESPECTIVELY.
A FLAG DENOTING WHETHER THE ENERGY OR ZERO CROSSING VALUES ARE TO BE
COMPARED, E_OR_ZC, AND TWO VARIABLES IN WHICH TO RETURN THE
CALCULATED TIME-WARP SCORE, VAL (FOR ENERGY) AND ZIVAL (FOR ZERO-CROSSING).
NOTE THAT GLOBAL VARIABLES WARPFR AND WARPFR2 SHOULD BE SET TO DENOTE WHAT
BANDS SHOULD BE WARPED. TOTAL_DIST COMPUTES THE TIME-WARP SCORE BETWEEN ARY1
AND ARY2, FOR THE DESIRED BANDS, AND RETURNS THIS SCORE IN VAL OR ZIVAL
(DEPENDING UPON WHETHER ENERGIES OR ZERO CROSSINGS WERE BEING WARPED).
NOTE THAT A SHOULD BE GREATER THAN OR EQUAL TO B.)

VAR X, Y : INTEGER; ( ARRAY INDICES )
S : REAL; ( SLOPE OF BAND LIMITS )
Y_MIN, Y_MAX : ARRAY [1..MAX_SAMPLE] OF INTEGER; ( BOUNDS OF Y AS A FUNCTION OF X )

67
PROCEDURE DIST (I1,I2,WARPFR,WARPFR2 : INTEGER; E_OR_ZC:CHAR);

SET D[I1,I2]= ABSOLUTE DIFFERENCE BETWEEN THE BAND WITH THE GREATEST DIFFERENCE. THE BANDS RANGE FROM BAND NUMBER WARPFR TO WARPFR2, BETWEEN ROW I1 OF ARY1 AND ROW I2 OF ARY2. E OR ZC DETERMINE WHETHER THE BANDS'S ENERGY ('E') OR ZERO CROSSING ('Z') VALUE SHOULD BE COMPARED.

VAR DIF: INTEGER; ( TEMPORARY STORAGE VARIABLE )
I: INTEGER; ( LOOP VARIABLE, INDEX INTO FRAMES )
T1, T2: TIME_SAMPLE; ( HOLDS ONE FRAME OF DATA---THE FRAMES )

BEGIN IF E_OR_ZC = 'E' THEN BEGIN
D[I1,I2]:=0; ( ENERGIES TO BE WARPED )
FOR I:=1 TO N_BPF DO BEGIN ( READ IN FRAMES TO BE COMPARED )
T1[I,1]:=ARY1[I,1];
T2[I,1]:=ARY2[I,1];
END;
END;
ELSE BEGIN
ZC_D[I1,I2]:=0; ( ZERO-INGS TO BE WARPED )
FOR I:=9 TO 15 DO BEGIN
T1[I,1]:=ARY1[I,1];
T2[I,1]:=ARY2[I,1];
END;
END;
END;
END;

PROCEDURE GET_D (X, Y: INTEGER; VAR T3:INTEGER; VAR T4:INTEGER);

RETURNS T3=D(X,Y) AND T4=ZC_D(X,Y) IF X,Y ARE VALID VALUES IN THE TIME WARPING SCHEME. IF NOT, RETURN WITH T3, T4 = INF_DIST.

BEGIN IF (X=Y_MAX(X)) AND (Y=Y_MIN(Y)) THEN BEGIN
T3:=D(X,Y); ( IF THESE TWO FRAMES CAN BE COMPARED THEN )
T4:=ZC_D(X,Y); ( CALCULATE T3 AND T4 )
END;
ELSE BEGIN
T3:=INF_DIST;
T4:=INF_DIST;
END.

68
PROCEDURE Df_FCN ( X, Y: INTEGER; E_OR_ZC: CHAR );

( THIS IS THE DYNAMIC PROGRAMMING FUNCTION.
IF THE BAND ENERGIES ARE BEING WARPED THEN IT SETS
D(X,Y) = DIST(X,Y) + MINIMUM( D(X-1,Y1), D(X-1,Y-1), D(X-1,Y-2), ... ).
THE D ENTRIES MUST BE VALID ACCORDING TO Y_MAX[X-1] AND Y_MIN[X-1].
IF THE BAND ZERO CROSSINGS ARE BEING WARPED (IT SETS
ZC_D(X,Y) = DIST(X,Y) + MINIMUM( ZC_D(X-1,Y1), ZC_D(X-1,Y-1), ... ).
THE ZC_D ENTRIES MUST BE VALID ACCORDING TO Y_MAX[X-1] AND Y_MIN[X-1].

VAR 
VAL2 : INTEGER; ( IS SET TO THE MINIMUM TIME-WARP SCORE SO FAR )
T1 : INTEGER; ( PASSING PARAMETER TO GET_D )
T2 : INTEGER; ( PASSING PARAMETER TO GET_D )
T5 : INTEGER; ( PASSING PARAMETER TO GET_D )
ZC_MIN : INTEGER; ( IS SET TO THE MINIMUM TIME-WARP SCORE SO FAR )
( FOR THESE TWO WORDS IF THE ZERO CROSSINGS ARE BEING WARPED )
BEGIN
IF E_OR_ZC = 'E' THEN BEGIN
VAL2:=INF_DIST; ( ENERGIES ARE BEING WARPED )
FOR T5:=Y_MIN[X-1] TO Y_MAX[X-1] DO BEGIN
IF T5 <= Y THEN BEGIN
GET_D(T5,T5,T1,T2);
IF VAL2 = T1 THEN VAL2:=T1;
END;
END;
IF VAL2 < INF_DIST THEN BEGIN
DIST(X,Y,WARPFR,WARPFR2,E_OR_ZC);
D(X,Y):=D(X,Y)+VAL2;
END
ELSE D(X,Y):=INF_DIST;
END
ELSE BEGIN
ZC_MIN:=INF_DIST; ( ZERO-CROSSINGS BEING WARPED )
FOR T5:=Y_MIN[X-1] TO Y_MAX[X-1] DO BEGIN
IF T5 <= Y THEN BEGIN
GET_D(T5,T5,T1,T2);
IF ZC_MIN = T2 THEN ZC_MIN:=T2;
END;
END;
IF ZC_MIN < INF_DIST THEN BEGIN
DIST(X,Y,WARPFR,WARPFR2,E_OR_ZC);
ZC_D(X,Y):=ZC_D(X,Y)+ZC_MIN;
END
ELSE ZC_D(X,Y):=INF_DIST;
END
END;
PROCEDURE SET_BOUNDS(M_ADJ, A, B : INTEGER);

( ARRAYS Y_MIN AND Y_MAX ARE SET TO THE BOUNDS FOR ALLOWED TIME-WARPING.  
Y_MIN[I] AND Y_MAX[I] HOLD THE MINIMUM FRAME NUMBER (LEVEL IN ARRAY ARY2) AND THE 
MAXIMUM FRAME NUMBER WHICH FRAME I OF ARRAY ARY2 MAY BE 
COMPAIRED AGAINST IN ARRAY ARY1.)

VAR  S : REAL;
    X : INTEGER;

BEGIN
  S := (A + M_ADJ) / (B + M_ADJ);
  FOR X := 1 TO B DO BEGIN
    Y_MIN[X] := ROUND(S*X) - M_ADJ;
    Y_MAX[X] := ROUND(S*X) + M_ADJ;
    IF Y_MAX[X] > A THEN Y_MAX[X] := A;
    IF Y_MIN[X] > A THEN Y_MIN[X] := A;
    IF Y_MIN[X] < 1 THEN Y_MIN[X] := 1;
  END;
END;

BEGIN (TOTAL_DIST)

SET_BOUNDS(M_ADJ, A, B);
FOR Y := Y_MIN[I] TO Y_MAX[I] DO DIST(I,Y,WARPFR,WARPFR2,E_OR_ZC);
  (WARP THE FIRST FRAME)
FOR X := 2 TO B DO { WARP THE REMAINING FRAMES }
  FOR Y := Y_MIN[X] TO Y_MAX[X] DO
  DP_FCN(X,Y,E_OR_ZC);
  IF E_OR_ZC = 'E' THEN BEGIN { IF ENERGY BEING WARPED THEN FIND }
    VAL := INF_DIST; { THE LOWEST VALUE OF THE TIME WARP IN THE }
    FOR Y := Y_MIN[0] TO Y_MAX[0] DO ( ROW OF ARRAY D (LAST FRAME WARPED) )
      IF D[I,B,Y] < VAL THEN VAL := D[I,B,Y]; { DIVIDE THIS NUMBER BY THE NUMBER }
    IF VAL < INF_DIST THEN VAL := (VAL + B DIV 2) DIV B; { OF FRAMES WARPED TO }
      (AVERAGE SCORE PER FRAME WARPED)}
  END
ELSE BEGIN { IF ZERO-XINGS BEING WARPED, DO SAME AS }
  ZVAL := INF_DIST; { DESCRIBED ABOVE }
  FOR Y := Y_MIN[I] TO Y_MAX[I] DO
    IF IC_D[I,G,Y] < ZVAL THEN ZVAL := IC_D[I,G,Y];
    IF ZVAL < INF_DIST THEN ZVAL := (ZVAL + B DIV 2) DIV B;
  END;

  RETURN WITH VAL AND ZVAL SET
END;

(---------------------------------------------------------------------)
PROCEDURE 60_REC06N;

( 60_REC06N IS ONE OF THE MAIN DRIVING ROUTINES. IT READS IN THE COMPARISON
WORD, AND CALLS PROCEDURE SCORE. SCORE RETURNS WITH TWO PARAMETERS,
BEST AND WFLAG, SET AND 60_REC06N REACTS AS FOLLOWS:

IF WFLAG = 'A' THEN THE WORD WAS AMBIGUOUS AND BEST IS SET TO THE INDEX
OF THE BEST MATCHING WORD. IF THE BEST MATCHING WORD WAS
THIS WORD AND THE LIBRARIAN IS IN NODE A, THEN THE BEST
MATCHING WORD GETS AVERAGED WITH THE COMPARISON WORD. IF
THE BEST MATCHINGWORD IS NOT THE SAME AS THE COMPARISON
WORD AND THE LIBRARIAN IS IN NODE A OR B, THEN THE
COMPARISON WORD IS ADDED TO THE LIBRARY IF THE LIBRARY IS
NOT FULL.

IF WFLAG = 'N' THEN THE COMPARISON COULD NOT BE MATCHED AND IS ADDED TO
THE LIBRARY IF THE LIBRARIAN IS IN NODE A OR B AND THE LIBRARY
IS NOT FULL

IF WFLAG = 'Y' THEN THE COMPARISON WORD WAS MATCHED WITH THE LIBRARY WORD
POINTED TO BY BEST (BEST BEING AN INDEX INTO THE LIBRARY). IF
THE BEST MATCHING WORD WAS THE SAME AS THIS WORD, AND THE
LIBRARIAN IS IN NODE A, THEN THE BEST MATCHING WORD AND THE
COMPARISON WORD ARE AVERAGED. IF THE BEST MATCHING WORD
WAS NOT THIS WORD, AND THE LIBRARIAN IS IN NODE A OR B,
THEN THIS WORD IS ADDED TO THE LIBRARY IF THE LIBRARY IS
NOT FULL. )

TYPE DDIIST=ARRAY(1..N_VOCABJ) OF INTEGER;

VAR TEST : SAMPLE;  (COMPARISON WORD PATTERN )
TEST2: SAMPLE;  (NORMALIZED COMPARISON WORD PATTERN )
TSTSUM = ARRAY(1..N_BPF) OF INTEGER;  (SUM OF ALL THE ENERGIES
OF EACH BAND )
TOTALTS : INTEGER;  (TOTAL ENERGY OF THE COMPARISON WORD )
W : INTEGER;  (NUMBER OF FRAMES IN THE COMPARISON PATTERN )
I,J,W,K,J : INTEGER;  (LOOP VARIABLES )
LINE : LINE_TYPE;  (INPUT LINE BUFFER--HOLDS ONE FRAME )
BEST : INTEGER;  (INDEX OF BEST SCORE FOUND )
SECOND : INTEGER;  (INDEX OF SECOND BEST SCORE )
N3 : INTEGER;  (TEMPORARY STORAGE VARIABLE )
NAME2 : STRING(20);  (TEXT STRING OF COMPARISON WORD )
NAMED : STRING(20);  (TEXT STRING CONVERTED TO UPPERCASE )
CHR : CHAR;  (TEMPORARY STORAGE VARIABLE )
MFLAGS : ARRAY(1..4) OF BYTE;  (FLAGS TO DENOTE OF MATCHING )
( ALGORITHMS HAVE BEEN ENTERED )
VER_FLAG : BYTE;  (FLAG TO DENOTE IF THE VERIFY )
(VS00M : BYTE;  (PROCEDURE HAS BEEN ENTERED )
WFLAG6 : BYTE;  (MATCH3 VERIFY SCORE )
WFLAG : CHAR;  (FLAG SET AS DESCRIBED EARLIER )
VFLAG : CHAR;  (FLAG TO DENOTE IF VERIFY WAS )
(BNKS : INTEGER;  (DIFFERENCE BETWEEN BEST AND SECOND )
(BEST1,BEST2,BEST3,BEST4 : INTEGER;  (INDEXES OF BEST WORDS AT THE )
(CORRESPONDING STAGES )

71
PROCEDURE A872(VAR LINE:LINE_TYPE);EXTERNAL 'A872.COM';

{ ASSEMBLY ROUTINE TO INTERFACE WITH THE F-E BOX. THE ROUTINE MUST BE CALLED WITH AN ARRAY AS THE ONLY VARIABLE BEING PASSED. THE ROUTINE WILL SEND OUT A NL TO THE F-E BOX AND THEN READ INTO THE PASSED ARRAY THE DATA THAT THE F-E BOX SENDS BACK UNTIL EITHER A NL OR CR IS SENT BY THE F-E BOX. THE NL OR CR IS THE LAST CHARACTER PLACED IN THE ARRAY. }

( --------------------------------------------- )

PROCEDURE SCORE (VAR BEST : INTEGER; VAR WFLAG : CHAR);

{ BY USING THE DIFFERENT MATCHING ALGORITHMS AND CRITERIA, PROCEDURE SCORE FINDS THE BEST MATCHING LIBRARY WORD, OR A NO MATCH CONDITION. THE VARIABLES THAT SCORE RETURNS ARE DESCRIBED IN PROCEDURE 60_REC06H }

VAR VAL : INTEGER;  { HOLDS TIME WARP SCORES FOR ENERGIES }
ZVAL : INTEGER;  { HOLDS TIME WARP SCORES FOR ZERO-XIN6S }
K,J,J2 : INTEGER;  { INDEX VARIABLES }
WFLAG : CHAR;

(--------------------------------------------------------------- )

PROCEDURE SUNTEST;

{ SUMS UP THE ENERGIES (PEAK DETECTED VALUES) IN EACH BAND OF THE OF THE COMPARISON AND STORES THESE SEVEN VALUES IN ARRAY TSTSUM. SETS TOTALTS TO THE SUM OF ALL THE ENERGIES IN THE COMPARISON WORD }

VAR J,K,TEMP : INTEGER;

BEGIN
TOTALTS:=0;
FOR J:= 1 TO N_BPF DO BEGIN
TSTSUM(J):=0;
FOR K:=1 TO N DO TSTSUM(J):=TSTSUM(J) + TEST(K,J);
TOTALTS:= TOTALTS + TSTSUM(J);
END;

END;

{ ------------------------------------------------------------------------- }  

PROCEDURE MATCHI (VAR DIST1:DDIST; FRM_THR: INTEGER);

{ RETURNS WITH DIST1(I) EQUAL TO THE SUM OF THE DIFFERENCES OF THE AVERAGE ENERGIES IN EACH OF THE FIRST SIX BANDS BETWEEN LIBRARY WORD I AND THE... }
COMPARISON WORD.

BEFORE CALCULATING THIS SUM, THE TWO PATTERNS ARE NORMALIZED (IN PROCEDURE
ENS_DIF). DIST1 = 255 (HIGHEST POSSIBLE SCORE) IF THE LIBRARY WORD 1 AND
THE COMPARISON WORD DIFFER BY MORE THAN FRR_THR NUMBER OF FRAMES IN LENGTH.
FOR AN EXPLANATION OF THE NORMALIZATION PLEASE SEE MR. WIGBER’S THESIS.)

VAR:  ENSDIF: INTEGER;

PROCEDURE ENS_DIF(S: INTEGER; VAR ENSDIF: INTEGER);

VAR  TEMP: REAL;
    J, TEMP2: INTEGER;
    TSM: ARRAY[1..N_BPF] OF INTEGER;
    TSM2: ARRAY[1..N_BPF] OF INTEGER;

BEGIN

{ NORMALIZATION }

TEMP2 := ROUND((JUSTSUM(S)-JUSTSUM(J))/JUSTSUM(J));
TEMP := ROUND((TOTALTS-TSTSUM(J))/N) - TEMP2;
FOR J := 1 TO N_BPF DO BEGIN
    RR2 := ROUND(JUSTSUM(S,J)/JUSTSUM(S,N)/TEMP2);
    TSM(J) := ROUND(JUSTSUM(S,J)/JUSTSUM(S,N) + ROUND(TEMP+RR2));
END;
FOR J := 1 TO 6 DO TSM2(J) := ROUND(TSTSUM(J)/N);

{ SUM UP THE DIFFERENCES }

ENS_DIF := 0;
FOR J := 1 TO 6 DO ENSDIF := ENSDIF + ABS(TSM(J) - TSM2(J));

END;

{ --------------------------------------- }

BEGIN { PROCEDURE MATCH1 }

MFLA(S)[1] := 1;
FOR I := 1 TO N_WOROS DO BEGIN
    IF ABS(WOROS[I] - W) > FRR_THR THEN DIST1[I] = HSCORE
    ELSE BEGIN
        ENS_DIF(I, ENSDIF);
        DIST1[I] := ENSDIF;
        WRITE('WORDS[I].TXT ', DIST1[I]);
    END;
END;

END;

{ --------------------------------------- }

PROCEDURE MATCH2 (VAR DIST2:DDIST; DIST1:DDIST; THR:INTEGER; E_DG:ZC:CHAR);

{ SETS DIST2[I] EQUAL TO THE TIME-WARP SCORE FOR ALL ENERGY BANDS COMBINED
FOR LIBRARY WORD I AND THE COMPARISON WORD. IF THE LIBRARY WORD I HAD A
STAGE 1 (DIST1) SCORE ABOVE THRESHOLD 1 (THR) THEN DIST2[I] = 255 (HIGH

73
SCORE: WITHOUT PERFORMING THE TIME-WARPING, I.E. LIBRARY WORD I IS NOT
CONSIDERED TO BE A POSSIBLE MATCH. BEFORE PERFORMING THE TIME WARPING
THE COMPARISON WORD IS NORMALIZED TO THE LIBRARY WORD. FOR AN EXPLANATION
OF THE NORMALIZATION PLEASE SEE MR. WIBER'S MASTER THESIS.

VAR  I : INTEGER;

PROCEDURE NORMALIZE(I: INTEGER);

VAR  RR2,RR,DF,TD : INTEGER;
    N_TEST : ARRAY[1..N_BPF] OF REAL;
    J,K   : INTEGER;
    RR3   : REAL;

BEGIN
    RR2:=ROUND(TOTALTS/N);
    RR:=ROUND(WSUM[I]/WORDS[I].N);
    DF:=RR-RR2;
    FOR J:=1 TO N_BPF DO BEGIN
        RR3:=TSTSUM(JJ/TOTALTS;
        N_TEST[IJ]:=DF*RR3*N;
    END;
    FOR K:=1 TO N_BPF DO BEGIN
        IF N_TEST[IK] <> 0 THEN BEGIN
            FOR J:=1 TO N DO BEGIN
                RR3:=TEST[J,K]/TSTSUM(K);
                TD:=ROUND(N_TEST[IK]*RR3);
                TEST2[IJ,K]:=TEST[J,K]+TD;
                IF TEST2[IJ,K] > 255 THEN TEST2[IJ,K]:=255;
                IF TEST2[IJ,K] < 0 THEN TEST2[IJ,K]:=0;
            END;
        END;
        ELSE FOR J:=1 TO N DO TEST2[IJ,K]:=TEST[IJ,K];
    END;
END;

BEGIN  ( PROCEDURE MATCH2 )
    IFLAGS[I]:=1;
    WARPF:=1;
    WARPF2:=7;
    E_OR_IC:= 'E';
    FOR I:= 1 TO N_WORDS DO BEGIN  ( SEE IF WORD HAS BEEN )
        IF DIST[I] > THRI THEN DIST2[I]:=HSCORE  ( RULED OUT ).
        ELSE BEGIN
            NORMALIZE(I);
            IF WORDS[I].N < N  ( PASS PATTERNS ACCORDING TO NUMBER OF FRAMES )
                THEN TOTAL_DIST(TEST2,W,WORDS[I].DAT,WORDS[I].N,VAL,ZVAL,E_OR_IC)
                ELSE TOTAL_DIST(WORDS[I].DAT,WORDS[I].N,TEST2,W,VAL,ZVAL,E_OR_IC);
            DIST2[I]:=VAL;
        END;
    END;
PROCEDURE MATCH3(VAR DIST3:DDIST; DIST1_2:DDIST);

{ SETS DIST3[i] EQUAL TO THE SUM OF THE TIME-WARP SCORES FOR ZERO CROSSING BANDS 5, 6, AND 7 (WARPED INDIVIDUALLY) BETWEEN LIBRARY WORD I AND THE COMPARISON WORD. IF THE LIBRARY WORD WAS DETERMINED BY MATCH1 AND/OR MATCH2 TO LONGER BE CONSIDERED AS A MATCHING WORD, THEN DIST1_2[i] WILL EQUAL THE HIGH SCORE (255), AND MATCH3 WILL SET DIST3[i] EQUAL TO THE HIGH SCORE. }

VAR I,J : INTEGER;
BEGIN

PROCEDURE MATCH4 (BEST3 : INTEGER; SECOND3 : INTEGER; DIST2_3 : DDIST; VAR MFLAG : CHAR);

{ SETS MFLAG EQUAL TO 'Y' IF THE LOWEST SCORING WORD IS NOT EQUAL, SO FAR, TO THE SECOND LOWEST SCORING WORD, AND THE FRAME NUMBER DIFFERENCE BETWEEN THE LOWEST SCORING WORD AND THE COMPARISON WORD AND THE SECOND LOWEST SCORING WORD AND THE COMPARISON WORD IS GREATER THAN OR EQUAL TO REC_DIF4. ELSE MFLAG IS SET EQUAL TO 'N'. }

VAR TEMP : INTEGER;
BEGIN

MFLAGS[4]:=1;
BEST4:=BEST3;

75
SECOND4 := SECOND3;
IF DIST2_3 < BEST3 THEN BEGIN
  TEMP = ABS(WORDS(SECOND3.N - M) - ABS(WORDS(BEST3.N - M));
  IF TEMP >= DELIF THEN M4FLAG := 'Y'
  ELSE M4FLAG := 'N';
END;
END;

PROCEDURE SIFT(VAR ARY : DDIST; L_CNT : INTEGER);
{ SETS ALL THE ENTRIES IN THE PASSED ARRAY ARY TO HSCORE, EXCEPT THE
  L_CNT NUMBER OF LOWEST SCORES IN THE ARRAY, THESE ENTRIES RETAIN THEIR
  SAME VALUES. }
VAR  TMPARY : ARRAY [1..MAX.VOCAB,1..2] OF INTEGER;
1,J,TEMP1,TEMP2 : INTEGER;
DONE : INTEGER;
BEGIN
  FOR I:=1 TO N_WORDS DO BEGIN
    TMPARY[I,1] := ARY[I];
    TMPARY[I,1] := I;
  END;
  DONE := 0;
  I:=1;
  WHILE I < N_WORDS AND (DONE=0) DO BEGIN
    DONE := 1;
    FOR J:=1 TO N_WORDS DO BEGIN
      IF TMPARY[I,J] > TMPARY[I,J+1] THEN BEGIN
        DONE := 0;
        TEMP1 := TMPARY[I,J+1];
        TEMP2 := TMPARY[I,J];
        TMPARY[I,J] := TEMP1;
        TMPARY[I,J+1] := TEMP2;
      END;
    END;
    I:=I+1;
  END;
  FOR I:=1 TO N_WORDS DO BEGIN
    IF I <= L_CNT THEN ARY[TMPARY[I,1]] := TMPARY[I,2]
    ELSE ARY[TMPARY[I,1]] := HSCORE;
  END;

PROCEDURE COMBINE(VECA:DDIST; VECB:DDIST; VAR VECAB:DDIST; THR:INTEGER);
{ THIS PROCEDURE SUMS THE TWO VECTORS VECA AND VECB TO FORM VECAB. IF
  VECB[I] IS GREATER THAN THR THEN VECAB IS SET EQUAL TO THE HIGH SCORE }

76
VAR.   I : INTEGER;

BEGIN
FOR j:=1 TO N_WORDS DO
  IF VECB[j] > THR THEN VECAB[j] := HSCORE
  ELSE VECAB[j] := VECAB[j] + VECB[j];
END;

PROCEDURE FMDBST(VEC:DDIST; VAR LW : INTEGER; VAR SLW :INTEGER; VAR DIFF : INTEGER);
(FINDS THE LOWEST SCORE IN ARRAY DDIST. IT SETS LW TO THE INDEX OF
THE LOWEST SCORE, AND SLW TO THE INDEX OF THE SECOND LOWEST SCORE.
VARIABLE DIFF IS SET TO THE DIFFERENCE BETWEEN THE TWO LOWEST SCORES.)

VAR   I : INTEGER;

BEGIN
LW:=1;
SLW:=2;
LW:=2;
SLW:=1;
END;
END;
FOR i:=3 TO N_WORDS DO BEGIN
  IF VEC[i] < VEC[SLW] THEN
    BEGIN
      IF VEC[i] < VEC[LW] THEN BEGIN
        IF IIORDS[i].TXT = IIORDS[SLW].TXT THEN LW:=i
        ELSE BEGIN
          SLW:=LW;
          LW:=i;
        END;
      END;
    END;
  END;
DIFF:=VEC[SLW]-VEC[LW];
END;

PROCEDURE VERIFY(J:INTEGER; VAR VFLAG : CHAR; THR : INTEGER);
(THIS PROCEDURE IS CALLED IF THE BEST MATCHING WORD AFTER STAGE 2 HAS
A COMBINED SCORE WHICH IS LOWER BY A CERTAIN THRESHOLD (REC_DIF2) THAN
THE SECOND LOWEST SCORE. THIS PROCEDURE THEN CHECKS ONLY THIS WORD
WITH THE MATCH TEST—SUM OF THE TIME WARP VALUES OF THE ZERO CROSSINGS
OF BANDS 5,6, AND 7. IF THIS SUM IS LESS THAN OR EQUAL TO THR THEN VFLAG
IS SET TO "Y", ELSE VFLAG IS SET TO "N". VARIABLE X HOLDS THE INDEX INTO THE
LIBRARY WHERE THIS BEST MATCHING WORD LOCATED.)

VAR   J : INTEGER;
BEGIN
E_OR_IC:='Z';
VER_FLAG:=0;
VSUM:=0;
FOR J:=1 TO 3 DO BEGIN
WARPFR:=J;
WARPFR2:=J;
IF WORDS(J).M < N
THEN TOTAL_DIST(WORDS(J).DAT,TEST,N,VAL,ZVAL,E_OR_IC);
ELSE TOTAL_DIST(WORDS(J).DAT,TEST,N,VAL,ZVAL,E_OR_IC);
VSUM:=VSUM + ZVAL;
END;
IF VSUM <= THR3 THEN VFLAG:='Y'
ELSE VFLAG:='N';
END;

{ --------------------------------------------------

BEGIN ( PROCEDURE SCORE )
SUMTEST;
FOR I:=1 TO 4 DO MFLAGS[I]:=0;
VER_FLAG:=0;
WRITELN('--------------------------------------------------');
WRITELN('CALLING MATCH1');
MATCH1(DIST1,THR1); (PERFORM FIRST MATCHING ALGORITHM )
SIFT(DIST1,L_DIST1); (KEEP L_DIST1 LOWEST SCORES )
WRITELN('CALLING MATCH2');
MATCH2(DIST2,DIST1,THR1,E_OR_IC); (CALL SECOND MATCHING ALGORITHM )
COMBINE(DIST1,DIST2,DIST1_2,THR2); (COMBINE SCORES FROM FIRST TWO )
WRITELN('CALLING MATCH3');
MATCH3(DIST3,DIST1_2); (CALL THIRD MATCHING ALG. )
COMBINE(DIST1_2,DIST3,DIST1_2_3,THR3); (COMBINE ALL SCORES )
FNOBST(DIST1_2_3,BEST1,BEST2,SECOND1,BNS1); (FIND BEST SCORES )
BEST:=BEST1;
IF DIST1_2_3[BEST1] <> HSCORE THEN BEGIN (SEE IF WORD IS A VALID MATCH )
VFLAG:='Y'; (IF YES, SET VFLAG )
END
ELSE BEGIN
IF DIST1_2_3[BEST1] <> HSCORE THEN BEGIN (ANY WORDS LEFT?)
WRITELN('CALLING MATCH4');
MATCH4(DIST4,DIST1_2_3); (CALL FOURTH MATCHING ALG. )
COMBINE(DIST1_2_3,DIST4,DIST1_2_3_4,THR4); (COMBINE ALL SCORES )
FNOBST(DIST1_2_3_4,BEST4,BEST5,SECOND4,BNS4); (FIND BEST SCORES )
BEST:=BEST4;
IF DIST1_2_3_4[BEST4] <> HSCORE THEN BEGIN (SEE IF WORD IS A VALID MATCH )
VFLAG:='Y'; (IF YES, SET VFLAG )
END
ELSE BEGIN
IF DIST1_2_3[BEST1] <> HSCORE THEN BEGIN (ANY WORDS LEFT?)
WRITELN('CALLING MATCH5');

78
MATCH4(BEST3,SECOND3,DIST2_3,WFLAGS); (CALL FOURTH ALG.)

IF WFLAGS = 'Y' THEN BEGIN
  BEST3=BEST3; (IF FOURTH ALG. DETERMINE THE WORD)
  WFLAGS='Y'; (THEN SET BEST AND WFLAGS)
END
ELSE WFLAGS='A'; (AMBIGUOUS)
END
ELSE WFLAGS='N'; (NO MATCH)
END;

ELSE IFAB='N'; (NO MATCH)
END;

(----------------------------------------------)

PROCEDURE WLOG;

(WRITES ON THE SCREEN THE TWO LOWEST SCORING WORDS AFTER EACH STAGE
AND WHETHER THE COMPARISON WAS FOUND TO MATCH CORRECTLY OR INCORRECTLY,
OR IT WAS FOUND TO BE AMBIGUOUS, OR IT WAS FOUND TO NOT MATCH AT ALL)

VAR DIFF : INTEGER;

PROCEDURE HEADER2;

BEGIN
  WRITELN;
  WRITELN('PROGRAM MATCH9');
  WRITELN('LIBRARY FILE - ',VOCAB_FILE,' COMP. FILE - ',FILENAME2);
  WRITELN('COMPARISON WORD - ',NAME2,' INDEX - ',INDEX);
  WRITELN;
END;

(----------------------------------------------)

BEGIN
  HEADER2;
  IF IFAB=1 THEN BEGIN
    BEST1:=1;
    SECOND1:=2;
  END
  IF DIST111 > DIST112 THEN BEGIN
    BEST1:=2;
    SECOND1:=1;
  END;
  FOR I:=3 TO N_WORDS DO BEGIN
    IF DIST111 < DIST11(SECOND1) THEN
      IF DIST111 < DIST11(BEST1) THEN BEGIN
        IF WORDS11.TXT = WORDS(BEST1).TXT THEN BEST1:=I
        ELSE BEGIN
          SECOND1:=BEST1;
          BEST1:=I;
        END
      END
    END
  END;
END
ELSE IF WORDS[1].TXT <> WORDS[BEST1].TXT THEN SECOND1:=I;
END;
WRITE('STAGE 1; ',WORDS[BEST1].TXT, ' (',BEST1,')', ' - ',DIST[1][BEST1]);
WRITE(' ',WORDS[SECOND1].TXT, ' (',SECOND1,')', ' - ',DIST[1][SECOND1]);
END;
IF MFLAGS[2]=I THEN BEGIN
WRITE('STAGE 2; ',WORDS[BEST2].TXT, ' (',BEST2,')', ' - ',DIST[2][BEST2]);
WRITE(' ',WORDS[SECOND2].TXT, ' (',SECOND2,')', ' - ',DIST[2][SECOND2]);
END;
IF VER_FLAG=1 THEN BEGIN
IF WFLAG = 'Y' THEN WRITE('VERIFY GOOD; STAGE 3 SCORE - ',VSUM);
ELSE WRITE('VERIFY NOT GOOD; STAGE 3 SCORE - ',VSUM);
END;
IF MFLAGS[3]=I THEN BEGIN
WRITE('STAGE 3; ',WORDS[BEST3].TXT, ' (',BEST3,')', ' - ',DIST[3][BEST3]);
WRITE(' ',WORDS[SECOND3].TXT, ' (',SECOND3,')', ' - ',DIST[3][SECOND3]);
END;
IF MFLAGS[4]=I THEN BEGIN
DIF1:=ABS(WORDS[BEST1].N - N);
DIF2:=ABS(WORDS[SECOND1].N - N);
WRITE('STAGE 4; ',WORDS[BEST4].TXT, ' (',BEST4,')', ' - ',DIF1);
WRITE(' ',WORDS[SECOND4].TXT, ' (',SECOND4,')', ' - ',DIF2);
END;
IF WFLAG = 'N' THEN WRITE('NO MATCH');
IF WFLAG = 'A' THEN WRITE('AMBIGUOUS');
IF WFLAG = 'Y' THEN WRITE('MATCHED WITH WORD ',WORDS[BEST1].TXT);
WRITE;
END;

PROCEDURE WLOGLST;

< WRITES ONTO THE LST DEVICE THE TWO LOWEST SCORING WORDS AFTER EACH STAGE
AND WHETHER THE COMPARISON WAS FOUND TO MATCH CORRECTLY OR INCORRECTLY,
OR IT WAS FOUND TO BE AMBIGUOUS, OR IT WAS FOUND TO NOT MATCH AT ALL >

PROCEDURE HEADER3;

BEGIN
(WRITE(LST,'PROGRAM-RMATCH9');)
WRITE(LST,'LIBRARY FILE - ',VOCAB_FILE,' COMP. FILE - ',FILENAME2);
WRITE(LST,'COMPARISON WORD - ',NAME2,' INDEX - ',WIDX);
(WRITE(LST);
END;

PROCEDURE HEADER3;

IF MFLAGS[1]=I THEN BEGIN
WRITE(LST,'STAGE 1; ',WORDS[BEST1].TXT, ' (',BEST1,')', ' - ',DIST[1][BEST1]);

END;
IIRITELN(LST,' ',IIORDS[SECOND1l, TXT
IIRITELN(LST[' ',IIORDS[SECOND2l, TXT
IIRITELN(LST[' ',IIORDS[SECOND31, TXT
IIRITELN(LST[' ',IIORDS[SECOND4l, TXT

WRITE(LST,' "WORDS[SECOND1].TXT", (" Second1", )", ", DIST1[SECOND1l); END;
IF NFLA6ST2l=1 THEN BEGIN
WRITE(LST, 'STAGE 2; ",WORDS[BEST2].TXT", ("BEST2", )", ",DIST1[BEST2l); WRITE(LST, "WORDS[SECOND2].TXT", ("SECOND2", )", ",DIST1[SECOND2l); END;
IF VER_FLAG=1 THEN BEGIN
IF VFLA6 = 'Y'
    THEN WRITELN(LST, 'VERIFY GOOD; STAGE 3 SCORE = ', VSUM);
    ELSE WRITELN(LST, 'VERIFY NOT GOOD; STAGE 3 SCORE = ', VSUM);
END;
IF NFLA6ST3l=1 THEN BEGIN
WRITELN(LST, 'STAGE 3; ",WORDS[BEST3].TXT", ("BEST3", )", ",DIST[BEST3l); WRITE(LST, "WORDS[SECOND3].TXT", ("SECOND3", )", ",DIST1[SECOND3l); END;
IF NFLA6ST4l=1 THEN BEGIN
WRITE(LST, 'STAGE 4; ",WORDS[BEST4].TXT", ("BEST4", )", ",DIST[BEST4l); WRITE(LST, "WORDS[SECOND4].TXT", ("SECOND4", )", ",DIST1[SECOND4l); END;

BEGIN;
(WRITELN(LST));
IF WFLAG = 'N' THEN BEGIN
WRITELN(LST, 'NO MATCH; ', N_WORDS);
ELSE WRITELN(LST, 'LIBRARY IS FULL - WORD COULD NOT BE ADDED.'); END;
IF WFLAG = 'A' THEN BEGIN
WRITELN(LST,'AMBIGUOUS; ', A_WORDS);
IF CMDI = 'A' AND NAME2 = WORDS[BEST1].TXT THEN BEGIN
  WRITE(LST,'WORD WAS AVERAGED WITH ',WORDS[BEST1].TXT);
  WRITELN(LST, 'INDEX INTO LIB. - ',BEST1);
END;
IF CMDI = 'A' OR CMDI = 'B' AND NAME2 <> WORDS[BEST1].TXT THEN BEGIN
  IF FULLFLAG2 <> 1
    THEN WRITELN(LST, 'WORD WAS ADDED TO THE LIBRARY, INDEX - ', N_WORDS);
  ELSE WRITELN(LST, 'LIBRARY IS FULL - WORD COULD NOT BE ADDED.'); END;
END;
IF WFLAG = 'Y' THEN BEGIN
IF NAME2 = WORDS[BEST1].TXT THEN BEGIN
  WRITELN(LST, 'THE WORD WAS MATCHED CORRECTLY; ', R_WORDS);
  IF CMDI = 'A' THEN
    WRITE(LST, 'THE WORD WAS AVERAGED WITH ',WORDS[BEST1].TXT);
    WRITELN(LST, 'INDEX INTO LIB. - ',BEST1);
END;
ELSE BEGIN
  WRITELN(LST, 'THE WORD WAS MATCHED INCORRECTLY; ', W_WORDS);
  IF CMDI = 'A' OR CMDI = 'D' THEN BEGIN
    IF FULLFLAG2 <> 1
      THEN WRITELN(LST, 'IT WAS ADDED TO THE LIBRARY, INDEX - ', N_WORDS);
    ELSE WRITELN(LST, 'LIBRARY IS FULL - WORD COULD NOT BE ADDED.'); END;
END;
END;
END;
IF FULLFLAG = 1 THEN FULLFLAG2 := 1;
WRITE(LST,'--------------------------------------------------------');

END;

(---------------------------------------------------------------)
PROCEDURE WAVEAVE(BEST:INTEGER);

{ THIS PROCEDURE AVERAGES THE WORD PATTERN IN ARRAY TEST WITH THE WORD
IN THE LIBRARY GIVEN BY THE INDEX PASSED IN BEST. THE AVERAGING
IS A WEIGHTED AVERAGE, WITH THE LIBRARY WORD GIVEN A WEIGHT OF NUM-1 AND
THE COMPARISON WORD A WEIGHT OF 1. THE SMALLER, FRAMWISE, OF THE TWO
WORDS IS 'CENTERED' UPON THE LARGER WORD BEFORE AVERAGING. THE FINAL
WORD THUS IS THE SIZE OF THE LARGER OF THE TWO WORDS. THE FRAMES IN THE
LARGER WORD WHICH DO NOT HAVE CORRESPONDING FRAMES IN THE SMALLER WORD
(THE FRAMES AT EACH END) ARE LEFT AS IS. NORMALIZING IS DONE BEFORE
AVERAGING.)

VAR DIFFQ:INTEGER;
SHIFT : INTEGER;
I,J : INTEGER;

PROCEDURE RESUM(I : INTEGER);

{ CHANGE ARRAY, WHICH HOLD INFORMATION ABOUT THE CURRENT LIBRARY,
ENTRIES FOR THE AVERAGED LIBRARY WORD }

VAR J,K,TENP : INTEGER;
BEGIN
FOR J:=1 TO N_BPF DO BEGIN
BDSUM[I,J]:=0;
FOR K:=1 TO WORDS[I].N DO
BDSUM[I,J]:=BDSUM[I,J] + WORDS[I].DATA[K,J];
END;
WDSUM[I]:=0;
FOR J:=1 TO N_BPF DO WDSUM[I]:=WDSUM[I] + BDSUM[I,J];
END;

{ ---------------------------------------------------------------

PROCEDURE NORMALIZE3(IND:INTEGER);

VAR RR,RR2,DF : INTEGER;
J,K,TC : INTEGER;
RR3 : REAL;
N_TEST : ARRAY[1..N_BPF] OF REAL;

BEGIN
RR2:=ROUND(TOTALTS/N);
RR:=ROUND(WDSUM(IND)/WORDS(IND).N);
DF:=RR-RR2;
FOR J:=1 TO N_BPF DO BEGIN

82
RR3:=TSTSUM[J]/TOTALTS;
N_TEST(J):=DF*RR3*N;
END;
FOR K:=1 TO N_BPFF DO BEGIN
IF M_TEST(K) <> 0 THEN BEGIN
FOR J:=1 TO N DO BEGIN
RR3:=TEST(J,K)/TSTSUM(K);
TD:=ROUND(N_TEST(K)+RR3);
TEST2(J,K)=TEST(J,K)+TD;
IF TEST2(J,K) > 255 THEN TEST2(J,K):=255;
IF TEST2(J,K) < 0 THEN TEST2(J,K):=0;
END;
END;
ELSE FOR J:=1 TO N DO TEST2(J,K):=TEST(J,K);
END;
FOR K:=1 TO N DO
FOR J:=6 TO 15 DO
TEST2(K,J):=TEST(K,J);
END;
{ ----------------------------------------------- }
BEGIN ( WAYEAYE }
WRITELM('AVERAGE WITH WORD ’,WORDS[BEST].TXT,’ INDEX: ’,BEST);
NORMALIZE[BEST];
SHIFT:=WORDS[BEST].N-N; ( CALCULATE FRAME DIFFERENCE )
IF SHIFT=0 THEN BEGIN ( IF PATTERN LENGTHS ARE EQUAL )
DIFFQ:=1;
FOR I:=DIFFQ TO WORDS[BEST].N DO
FOR J:=1 TO N_ELM DO
WORDS[BEST].DAT[I,J]:=ROUND(((NUM-1)*WORDS[BEST].DAT[I,J]+TEST2[I,J])/NUM);
END;
IF SHIFT > 0 THEN BEGIN ( IF LIBRARY WORD IS LONGER )
DIFFQ:=ROUND((WORDS[BEST].N-N)/2); ( DIVIDE DIFFERENCE BY 2 )
IF ODD(DIFFQ) THEN BEGIN ( IF DIFFERENCE IS ODD )
FOR I:=DIFFQ+1 TO WORDS[BEST].N-DIFFQ DO
FOR J:=1 TO N_ELM DO
WORDS[BEST].DAT[I,J]:=ROUND(((NUM-1)*WORDS[BEST].DAT[I,J]+TEST2[I-DIFFQ,J])/NUM);
END ELSE BEGIN ( IF DIFFERENCE IS EVEN )
FOR I:=DIFFQ+1 TO WORDS[BEST].N-DIFFQ DO
FOR J:=1 TO N_ELM DO
WORDS[BEST].DAT[I,J]:=ROUND(((NUM-1)*WORDS[BEST].DAT[I,J]+TEST2[I-DIFFQ,J])/NUM);
END;
END;
IF SHIFT < 0 THEN BEGIN ( TEST WORD IS LONGER )
DIFFQ:=ROUND((N-WORDS[BEST].N)/2); ( DIVIDE DIFFERENCE BY 2 )
IF ODD(DIFFQ) THEN BEGIN ( IF DIFFERENCE IS ODD )
FOR I:=DIFFQ+1 TO N-DIFFQ DO
FOR J:=1 TO N_ELM DO
TEST2[I,J]:=ROUND(((NUM-1)*WORDS[BEST].DAT[I-DIFFQ,J]+TEST2[I,J])/NUM);
END ELSE BEGIN ( IF EVEN )

FOR I:=DIFF+1 TO N-DIFF DO
  FOR J:=1 TO N_ELM DO
    TEST2[I,J]:=ROUND(((NUM-1)\*WORDS[BEST1,DAT[I-DIFF,Q,J]+TEST2[I,J]])/NUM);
END;
FOR J:=1 TO N DO
  FOR J:=1 TO N_ELM DO
    WORDS[BEST1,DAT[I,TEST2[I,J]]:=TEST2[I,J];
  WORDS[BEST1,N:=N;
END;
RESUM(BEST); { RECALCULATE ARRAY ENTRIES FOR THIS WORD }
END;

/*---------------------------------------------*/
PROCEDURE EXPANDOR;
( THIS PROCEDURE ADDS THE WORD PATTERN IN ARRAY TEST AS A NEW WORD AT THE
  END OF THE LIBRARY. ARRAYS WHICH HOLD INFORMATION ABOUT THE LIBRARY
  WORDS ARE UPDATED (ARRAYS BOSUN AND WDSUM). )
VAR I : INTEGER;
  J : INTEGER;
BEGIN
  IF (CMD1 = 'U') AND (N_WORDS >= MAX_VOCAB) THEN BEGIN
    WRITELN('LIBRARY IS FULL - WORD CANNOT BE ADDED');
  END ELSE BEGIN
    N_WORDS:=N_WORDS + 1;
  FOR I:=1 TO N DO
    FOR J:=1 TO N_ELM DO
      WORDS[N_WORDS,DAT[I,J]]:=TEST[I,J];
  WORDS[N_WORDS,N]:=N;
  WORDS[N_WORDS,T:=NAMEZ;
  WORDS[N_WORDS,CNT:=O;
  FOR J:=1 TO N_BPF DO BOSUN[N_WORDS,J]:=TSTSUM[J];
  WDSUM[N_WORDS]:=TOTALTS;
  IF N_WORDS = MAX_VOCAB THEN FULLFLG:=I;
END;

/*---------------------------------------------*/
PROCEDURE BANDAVEVAR ARY:SAMPLE; NF:INTEGER);
( THIS ROUTINE AVERAGES THE INCOMING COMPARISON WAVE BEFORE MATCHING IS
  DONE. IT DOES THIS AVERAGING BY SETTING THE ENERGY IN BAND X FRAME Y
  EQUAL TO THE AVERAGE ENERGY IN BAND X FRAME Y AND BAND X FRAME Y-1.
  THE FIRST FRAME IS UNTOUCHED. )
VAR I,J : INTEGER;
BEGIN
FOR I:=1 TO N_BPF DO
 FOR J:=2 TO NF DO
  ARY[J,J]:=ROUND((ARY[J-1,J]+ARY[J,J])/2);
END;

{ -------------------------------------- }
PROCEDURE READLINE2 (VAR LINE : LINE_TYPE);

( READ IN ONE FRAME OF DATA FROM THE F-E BY THE ROUTINE AB72. THE OVERFLOW
MARKER IS CHECKED FOR. IF FOUND IT IS DELETED FROM THE FRAME (ALL ENTRIES
MOVE UP ONE) AND THE OVERFLOW FLAG OVFF IS SET. THE CALLING ROUTINE DOES
NOT CHECK THIS FLAG. )

VAR  I,J : INTEGER;

BEGIN

I:=1;
OVFF:=FALSE;
WRITE('.');
AB72(LINE);
.
WHILE LINE(I) <> CHAR(0) DO BEGIN
   IF LINE(I) = OVFF THEN BEGIN
      OVFF:=TRUE;
      J:=I;
      WHILE LINE(J) <> CHAR(0) DO BEGIN
         LINE(J):=LINE(J+1);
         J:=J+1;
      END;
      I:=I+1;
      END;
   END;
   I:=I+1;
END;
END;

{ ------------------------------- }

BEGIN ( SET_WORD )

NAME2:='UNKNOWN';
WIDIX:=WIDX+1;
REPEAT
   ST_CHK:=0;
   END_FLAG:=0;
   N:=0;
   TOO_LARGE:=FALSE;
   FOR I:=1 TO 40 DO LINE[I]:=' ';
   WRITELN;
   WRITELN('SAY THE \DRD');
   WRITE(LST,FE_START);
   READLINE2(LINE);
   
   WHILE (LINE[I] <> END_FILE) AND (END_FLAG <> I) DO BEGIN 
     CHECK FOR 
     IF LINE[I] <> END_WORD THEN BEGIN 
       IF N < MAX_SAMPLE THEN BEGIN 
         N:=N+1;
         [ INCREMENT FRAME COUNT ]
         CONVERT(LINE,TEST,N,WNSFLAG); [ CONVERT FROM ASCII TO DECIMAL ]
         IF WNSFLAG = 1 THEN N:=N-1; [ IF ERROR, THROW AWAY FRAME ]
         ST_CHK:=ST_CHK+1; [ INCREMENT START COUNTER ]
         IF ST_CHK >15 THEN CHECKEND(END_FLAG,W); [ LOOK FOR END OF WORD ]
       END;
       [ AFTER FIRST 15 FRAMES ]
     ELSE BEGIN
       TOO_LARGE:=TRUE; [ WORDToo_LONG ]
       WRITELN('!'); [ SHOW USER ]
     END;
   END;
END.
BEGIN (GO RECOGN)
RESET(VOCAB); IF CMDS <> 'S' THEN BEGIN
  MAXX:=MAXX+1; (IF COMPARISON WORDS ARE IN A FILE)
  READLN(VOCAB2,NAME2); (READ IN NAME AND CONVERT TO UPPER CASE)
  NAMED:='';
  FOR K:=1 TO LENGTH(NAME2) DO BEGIN
    CHR:=COPY(NAME2,K,1); CHR:=UPCASE(CHR);
    NAMED:=CONCAT(NAMED,CHR);
  END;
  NAME2:=NAMED;
  READLINE(VOCAB2,LINE); (READ FIRST FRAME)
  N3:=0; (FRAME COUNT IS CLEARED)
  WHILE LINE[i][i] <> '#' DO BEGIN (CHECK FOR FIRST END OF WORD MARKER)
    N3:=N3+1; (INCREMENT FRAME COUNT)
    CONVERT(LINE,TEST,N3,MNSFLAG); (CONVERT FROM ASCII TO DECIMAL)
    IF MNSFLAG = 1 THEN N3:=N3-1; (ERROR, THEN THROW FRAME AWAY)
    READLINE(VOCAB2,LINE); (READ IN NEXT FRAME)
  END;
  READLINE(VOCAB2,LINE); (READ IN NEXT FRAME)
  WHILE LINE[i][i] <> '#' DO BEGIN (CHECK SECOND END OF WORD MARKER)
    N3:=N3+1;
    CONVERT(LINE,TEST,N3,MNSFLAG);
    IF MNSFLAG = 1 THEN N3:=N3-1;
    READLINE(VOCAB2,LINE);
  END;
  N:=N3; (SET FRAME COUNT)
END ELSE GET WORD; (ELSE GET WORD PATTERN FROM THE F-E)
IF CMDS = 'Y' THEN BANDAVE(EST,N); (BAND AVERAGING DESIRED?)
SCORE:=BEST,WFLG); (GET SCORE)
IF WFLAS='Y' THEN BEGIN ( SEE DESCRIPTION OF GD_RECOGN FOR WHAT FOLLOWS )
  IF CMD5 = 'S' THEN BEGIN
    WRITE('THE WORD SAID WAS ', WORDS[BEST].TXT);
    WRITE('R - RIGHT OR W - WRONG ? ');
    READLN(CMD6);
    IF CMD6 <> 'R' THEN BEGIN
      WRITE('WHAT WAS THE WORD ? ');
      READLN(NAME2);
    END
    ELSE NAME2=WORDS[BEST].TXT;
  END
  IF NAME2=WORDS[BEST].TXT THEN BEGIN
    N_RIGHT:=N_RIGHT+1;
    WORDS[BEST].CNT:=WORDS[BEST].CNT+1;
    IF CMDI='A' THEN BEGIN
      WRITE('WORD WAS MATCHED CORRECTLY AND WILL BE AVERAGED.');
      MAVEAVE(BEST);
    END
    ELSE WRITELN('WORD WAS MATCHED CORRECTLY');
  END
  ELSE BEGIN
    N_WRONG:=N_WRONG+1;
    WRITE('WORD WAS MATCHED INCORRECTLY');
    IF (CMDI='A') OR (CMDI='B') THEN BEGIN
      IF FULLFLAS <> 1 THEN BEGIN
        WRITE('AND WILL BE ADDED TO THE LIBRARY.');
        EXPANDOR;
      END
      ELSE WRITELN('. LIBRARY IS FULL - WORD NOT ADDED.');
    END
    ELSE WRITELN;
  END;
END;
IF WFLAS='N' THEN BEGIN
  IF CMD5 = 'S' THEN BEGIN
    WRITE('WHAT WAS THE WORD ? ');
    READLN(NAME2);
  END
  N_NOMATCH:=N_NOMATCH+1;
  WRITE('WORD COULD NOT BE MATCHED');
  IF (CMDI='A') OR (CMDI='B') THEN BEGIN
    IF FULLFLAS <> 1 THEN BEGIN
      WRITE('AND WILL BE ADDED TO THE LIBRARY.');
      EXPANDOR;
    END
    ELSE WRITELN('. LIBRARY IS FULL - WORD NOT ADDED.');
  END
  ELSE WRITELN;
END;
IF WFLAS='A' THEN BEGIN
  IF CMD5 = 'S' THEN BEGIN
    WRITE('WHAT WAS THE WORD ? ');
    READLN(NAME2);
  END
N_AMBIG:=N_AMBIG+1;
IF NAME2 = WORDSIBEST.TXT THEN BEGIN
  WRITELN('THE WORD WAS AMBIGUOUS, BUT THE BEST MATCHING WORD WAS THIS WORD.');
  IF CMD1='A' THEN BEGIN
    WRITELN('THEREFORE IT WILL BE AVERAGED WITH THE LIBRARY WORD.');
    WAVERAVE(BEST);
  END;
END
ELSE BEGIN
  WRITELN('THE WORD WAS AMBIGUOUS AND THE BEST MATCHING WORD WAS NOT THIS WORD');
  IF CMD1='A' OR CMD1='B' THEN BEGIN
    IF FULLFLAG < I THEN BEGIN
      WRITELN('THEREFORE IT WILL BE ADDED TO THE LIBRARY');
      EXPANDOR;
    END;
    ELSE WRITELN('LIBRARY IS FULL - WORD NOT ADDED. ');
  END;
END;
END;
IF CND1 = T THEN LIST;
WRITELN('R-' ,R_RIGHT
          'W-' ,N_IIRON6
          'A-' ,N_AMBIG
          'M-' ,N_NONMATCH);
END;
{ -------------------------- -------------------------------------------- }
PROCEDURE DISPLAY_PARAMS;
( DISPLAYS THE ADJUSTABLE PARAMETERS )
BEGIN
  WRITELN('1) FRAME THRESHOLD: ',FRM_THR);
  WRITELN('2) STAGE 1 THRESHOLD: ',THR1);
  WRITELN('3) STAGE 2 THRESHOLD: ',THR2);
  WRITELN('4) STAGE 2 DIFFERENCE: ',REC_DIF2);
  WRITELN('5) STAGE 3 THRESHOLD: ',THR3);
  WRITELN('6) STAGE 3 DIFFERENCE: ',REC_DIF3);
  WRITELN('7) STAGE 4 DIFFERENCE: ',REC_DIF4);
  WRITELN('8) AVERAGING WEIGHT FACTOR: ',NUM);
  WRITELN('9) STAGE 1 KEEP ',L_DIST1);
  WRITELN;
END;
{ -------------------------- -------------------------------------------- }
PROCEDURE ADJUST;
( ALLOWS ANY OF THE ABOVE PARAMETERS TO BE ADJUSTED BY THE USER AT
  RUN TIME. )
VAR SELECT: INTEGER;
  CH : CHAR;
  NEWTHR: INTEGER;
BEGIN
  REPEAT
    WRITELN;
    WRITELN('0) No change');
    DISPLAY_PARAMS;
    WRITE('Select parameter to adjust:');
    READLN(INPUT, SELECT);
    CASE SELECT OF
      0 : (Nothing done here)
    ELSE BEGIN
      WRITE('New threshold:');
      READLN(NEWTHR);
      CASE SELECT OF
        1 : FRM_THR:=NEWTHR;
        2 : THR1:=NEWTHR;
        3 : THR2:=NEWTHR;
        4 : REC_DIFF:=NEWTHR;
        5 : THR3:=NEWTHR;
        6 : REC_DIF4:=NEWTHR;
        7 : REC_DIF4:=NEWTHR;
        8 : NUM:=NEWTHR;
        9 : L_DIST:=NEWTHR;
      END;
    END;
    END;
    UNTIL SELECT = 0;
    WRITELN;
  END;

{ -----------------------------------------------}
PROCEDURE BANDSUM(VAR BDSUM:BAND_SAMPLE;
                   VAR WDOSUM:WS_SAMPLE);
( Puts into array BDSUM[I,J] the total energy of band J, word I, for
  all seven energy bands for all the Library words.
  Puts into array WDOSUM[I,J] the total energy of each word I, for all
  Library words.)
VAR I,J,K : INTEGER;
BEGIN
  I:=1 TO N_WORDS DO
    FOR J:=1 TO N_BPF DO BEGIN
      BDSUM[I,J]:=0;
      K:=1 TO WORDS[I].N DO
        BDSUM[I,J]:=BDSUM[I,J]+WORDS[I].DAT[K,J];
    END;
  WOSUM[I,J]:=0;
  FOR J:=1 TO N_BPF DO WOSUM[I,J]:=WOSUM[I,J]+BDSUM[I,J];
END;
{ ----------------------------------------------- }
PROCEDURE WAVEWRITER;
(Writes all the word patterns in the library (all the library words)
back into the library file on disk. The library may be changed
during the program since wave averaging and adding words to the library
may have been done. All data is store in ASCII form.)

const stcon = '0123456789abcdef';

var n_file : integer;
l,m,n : integer;
askey : string[11];
dumb2,ff : char;
hx16 : integer;
rm16 : integer;

begin
ff:=char(12); 
rewite(vocab); 
if cmdis = 'y' then begin
   writeln;
   writeln;
   writeln(lst,'UPDATED LIBRARY');
end;
writeln('WRITING UPDATED LIBRARY INTO FILE ',vocab_file);
for i:=1 to n_words do begin
   write(words[i].tit:12,'-',words[i].cnt:3);
   if cmdis = 'y' then writeln(lst,words[i].tit,'-',words[i].cnt);
   writeln(vocab,words[i].tit);  (write spelling of word)
   writeln(vocab,words[i].cnt);  (write word match count)
end;
for l:=1 to words[i].m do begin  (write word pattern)
   for m:=1 to n_elm do begin
      hx16:=trunc(words[i].datl(m)/16);  (convert from ascii)
      rm16:=words[i].datl(m)-16*hx16;  (to decimal each value)
      askey:=copy(stcon,hx16+1,1);  (before writing)
      write(vocab,askey:1); 
      askey:=copy(stcon,rm16+1,1);
      write(vocab,askey:1);
   end;
   writeln(vocab,nl);  (end frame with newline)
   end;
   writeln(vocab,'*':nl);  (end pattern with two end of word markers)
   write(vocab,'*':nl);
end;
close(vocab);  (close file)
if cmdis = 'y' then writeln(lst,ff);
end;

{-----------------------------------------------)

procedure set_files;

{allows the user to input into array c_files ten comparison file
names which are to be processed in the order entered.)
VAR  I : INTEGER;
   C_NAME : STRING[12];

BEGIN
  FOR I:=1 TO 10 DO C_FILES[I]:='';
  WRITELN;
  WRITELN('LIST THE DESIRED COMPARISON FILES, # WHEN COMPLETE');
  C_NAME:='START';
  I:=1;
  READLN(C_NAME);
  WHILE (C_NAME <> '#' AND (I <> 11)) DO BEGIN
    C_FILES[I]:=C_NAME;
    READLN(C_NAME);
    I:=I+1;
  END;
  WRITELN(I-1,' FILES ENTERED');
END;

{ --------------------------------------------------------------- }

BEGIN 'Main program'
  INITIALIZE;
  BANDSUM(BSUM,WBSUM);
  WRITELN;
  WRITELN ('DEFAULT PARAMETERS ARE: ');;
  WRITELN;
  ADJUST;
  IF CMD5 <> 'S' THEN BEGIN
    SET_FILES;
    FILENAME2:=C_FILES[I];
    C_INDEX:=I;
    WHILE (FILENAME2 <> '') AND (C_INDEX < 11) DO BEGIN
      WRITELN;
      WRITELN('NEXT COMPARISON FILE IS ',FILENAME2,', NUMBER ',C_INDEX);
      ASSIGN(VOCAB2,FILENAME2);
      RESET(VOCAB2);
      WIDX:=0;
      WHILE NOT EOF(VOCAB2) DO SO_REC06N;
      C_INDEX:=C_INDEX+1;
      IF C_INDEX <> 11 THEN FILENAME2:=C_FILES[C_INDEX];
      WRITELN('-----------------------------------------------------');
      WRITELN('RIGHT':10,'WRONG':10,'AMBIG':10,'NO MATCH':10);
      WRITELN(N_RIGHT:10,N_WRONG:10,N_AMBIG:10,N_NONMATCH:10);
      WRITELN('-----------------------------------------------------');
    END;
  END
  ELSE BEGIN
    WRITELN(LIST,RINTEL);'; { reset f-e }
    CMD5:='B';'; { g for bg }
    WHILE CMD5 <> 'E' DO BEGIN
      GO_RECOGN;
      WRITE('PRESS RETURN TO CONTINUE; OR 'E' TO EXIT ');
      READLN(CMD5);
    END;
END;
IF (CMD1='A') OR (CMD1='B') THEN WAVewriter;
WRITEln;
WRITEln('MEMORY LEFT [16 BYTE PARAGRAPHS]: ',MemAvail));

END.
APPENDIX B

TRAIN5 PROGRAM LISTING
PROGRAM TRAIN (INPUT, OUTPUT);

LABEL OUT;  
( Jump to OUT if 8751 ext buffer overflows )

CONST  
END_CMD = '"';  
( Command to signify user wants to end input )
END_WORD = '"';  
( End of word marker from F-E box )
FE_START = '"';  
( Start looking for word command to F-E box )
MAX_SAMPLE = 65;  
( Maximum number of frames per word )
OVERFLOW = '"';  
( Overflow marker from F-E box )
END_FILE = '"';  
( End of file marker from F-E box )
CP = '"';  
( Command to signify user wants to change )  
( the 8751 parameters )
RESTART = '"';  
( Command to warm boot or reset the 8751 )
CINTER = '"';  
( Command to cold boot the 8751 )
WORLD = '"';  
( Command to signify user wants to boot 8751 )
N_BPF = 7;
ZERO_CNT = 0;

TYPE  
LINE_TYPE = ARRAY [1 .. 401] OF CHAR;  
( Input line buffer )
PARA_TYPE = ARRAY [1 .. 401] OF CHAR;  
( Input line buffer, for parameter procedure )

VAR  
VOCAB : TEXT;  
( Vocabulary file variable )
VOCAB_FILE: STRING[19];  
( Name of the vocab file on disk )
WORD : STRING[803];  
( Input word or end command )
SAMPLE : ARRAY [1 .. MAX_SAMPLE] OF LINE_TYPE;  
( Storage for the frames of a word )
DUMMY : LINE_TYPE;  
( Used when word has too many frames )
TOO_LARGE : BOOLEAN;  
( Indicates word has too many frames )
N : INTEGER;  
( Number of frames in this word )
EN : INTEGER;  
( Number of frames up to end of word marker )
l, j : INTEGER;  
( Loop count )
ACTIVE : BOOLEAN;  
( Indicates activity from F-E box )
CH : CHAR;  
( Answer to yes/no question )
MIN_SAMPLE : INTEGER;  
( Minimum number of frames per word )
SPC : INTEGER;  
( Number of frames to chop off end of word )
OVERFLOW : BOOLEAN;  
( 8751 ext buffer overflow flag )
GOFF : BOOLEAN;  
( Set when want to restart main loop )
ANS : CHAR;  
( Answer for choice of either to )  
( warm boot or cold boot the 8751 )
NL : CHAR;  
( Declare line feed character )
THEND : ARRAY [1 .. N_BPF] OF INTEGER;
COUNT10W : INTEGER;
PROCEDURE INITIALIZE;

( Initialization procedure )
( Set up vocabulary file on disk to store the frames of data )

VAR  I : INTEGER;

BEGIN

NL := CHAR(10);   { Line feed character }
WRITE('What is the name of the new vocabulary file ? : ');
READLN(VOCAB_FILE);   { Name of the vocabulary file }
ASSIGN(VOCAB, VOCAB_FILE);  { Associate VOCAB with vocabulary file }
REWITE(VOCAB);   { Initialize the vocabulary file }

{ Enter minimum frame count: ' };
READLN(INPUT, MIN_SAMPLE);    { Read min frame count }
MIN_SAMPLE := 16;

WRITE('Number of frames to chop off the end of word: ');
READLN(INPUT, SPC);   { Read num of frames to chop off }
SPC := 7;
EN_COUNT := 7;
THREN[:7] := 255;
FOR I := 2 TO 6 DO THREN[I] := 10;
THREN[7] := 3;

END;

PROCEDURE A872(VAR LINE: LINE_TYPE); EXTERNAL 'A872.COM';

( This is an assembly language routine that will send out a request )
( command (NL) to the F-E box and then it will collect a frame of )
( data, including the NL at the end of the frame. When reading a )
( frame, this routine reads in bytes from the serial port, until a )
( NL character is received, in which case the procedure ends. )

PROCEDURE READLNE (VAR LINE : LINE_TYPE );

( Reads one frame into the variable LINE, this includes the NL )

VAR  I,J : INTEGER;    { Index into LINE }

BEGIN

I := 1;  OVFF := FALSE;    { Reset overflow flag }
WRITE('.');
A872(LINE);   { Read in a frame of data }

96
WHILE LINE[I] <> CHAR(10) DO BEGIN (While not NL)
    IF LINE[I] = OVFM THEN BEGIN
        OVFF:=TRUE;
        j:=1;
        WHILE LINE[I] <> CHAR(10) DO BEGIN
            LINE[I]:=LINE[I+1];
            j:=j+1;
        END;
    END;
    I := I+1;
END;
END;

PROCEDURE AB7P(VAR TNPAR:PARA_TYPE); EXTERNAL 'AB7P.COM';

{ This assembly language routine will read in the parameters of the }
{ 8751. This is used in procedure READPARA. }

PROCEDURE READPARA ( VAR LINEOUT:PARA_TYPE);

{ This procedure will read the string of parameters from the 8751 }
{ into TMPAR. The string is terminated with a CR. Then this procedure }
{ deletes all the blanks from the string and puts the result in }
{ LINEOUT. Blanks occur after every two characters. }

CONST BLANK = ' '; { Blank character }

VAR C,D: INTEGER;
    TMPAR : PARA_TYPE; { A temporary array that holds the }
    { parameter data before any blanks }
    { are deleted }

BEGIN
    WRITELN; { Blank character }
    WRITELN('Now we are reading in the parameters');
    C := 1; D := 1;
    AB7P(TMPAR); { Read in whole line of parameters }
    WHILE TMPAR[C] <> CHAR(13) DO (If not a CR)
    BEGIN { Delete all inserted blanks }
        IF TMPAR[C] <> BLANK THEN
            BEGIN
                LINEOUT[D]:=TMPAR[C];
                D := D + 1;
            END;
        C := C + 1;
    END;
    LINEOUT[D]:=CHAR(13); { Insert a CR at the end }
END;
PROCEDURE AB70; VAR TEMINPUT : PARA_TYPE; EXTERNAL 'AB70.COM';

{ This assembly language routine sends the first character in TEMINPUT )
( put the serial port to the B751. Then it waits for two bytes to be    )
( sent, the first byte sent is put into TEMINPUT[2], the second byte    )
( sent is ignored. }

PROCEDURE PARAMETERS;

{ This procedure displays the current B751 parameters, and then allows the )
( user to input new values for these parameters. Note the parameters    )
( should be entered in hexadecimal format. This procedure also checks )
( for serial communication errors. }

CONST AECMD = 'A'; { Command for averaging adjacent frames }
DACMD = 'D'; { Command for not averaging adjacent frames }
TSCMD = 'T'; { Command for collecting 5 extra frames }
WTSCMD = 'W'; { Command for not collecting 5 extra frames }

VAR TEM : ARRAY [1..7] OF STRING [80]; { Array to hold output strings }
STOREP : PARA TYPE; { B751 parameters stored here }
DUMPARA : PARA TYPE;
TEM INPUT : PARA TYPE;
PARVALUE : STRING [80]; { New parameter from user }
VALID : BOOLEAN; { Flags if input parameter is valid }
I, J : INTEGER;
P, KT : INTEGER;
NF : INTEGER;
ERRF : BOOLEAN; { Serial communications error flag }
COM : CHAR;
CMDI : CHAR;

PROCEDURE ADJ_THRESH;

VAR I, SELECT, NEWTH : INTEGER;

BEGIN
REPEAT
  WRITELN;
  WRITELN('0) No Change');
  FOR I:=1 TO N_BPF DO
    WRITELN('1) Ending threshold value for filter', I, ': ', THRESH(I));
    WRITELN('2) Ending silent period count: ', EN_COUNT(I));
    WRITELN('3) # of frames to chop off end of a word: ', SPC(I));
  WRITELN;
  WRITE('Select parameter to adjust: ');
  READLN(SELECT);
  IF (SELECT < N_BPF+2) AND (SELECT > 0) THEN BEGIN
    WRITE('New value: ');
  END
  ELSE WRITELN('Invalid selection');
  WRITELN;
END

98
BEGIN
  { First initialize output string variables. }
  TEMP[7]='4000 Hz';
  WRITE;
  WRITELN('Would you like to change: '); 
  WRITELN('F - Feature Extractor parameter values');
  WRITELN('S - Software parameter values');
  WRITELN('A - All parameter values');
  READLN(CMD);
  IF (CMD='A') OR (CMD='F') THEN ADJ_THREND;
  IF (CMD='A') OR (CMD='S') THEN ADJ_THREND;
  READPARA(STOREP);  { Read in the line of parameters from 8751. }
  WRITE;
  WRITELN('Current 8751 parameters '); 
  WRITELN;
  WRITELN('Sampling period value (SPER) = '); 
  FOR I := 1 TO 4 { Display the 4 bytes of SPER },
     DO WRITE(STOREP[I]);
  WRITELN;
  WRITELN('Settling tile constant for the log amp (STLTN) = '); 
  FOR I := 5 TO 6 { Display the 2 bytes of STLTN },
     DO WRITE(STOREP[I]);
  WRITELN;
  NF := 7; { Pointer to thresholds within parameter line. }
  FOR I := 1 TO 7 DO 
     BEGIN { This loop will display the 7 PD thresholds }
       WRITE('THR',I,' = ', Threshold value for band ',I,' ',TEMP[I],') = '); 
       FOR J := NF TO NF+1
         DO WRITE(STOREP[J]);
       WRITELN;
       NF := NF + 2; { Point to beginning of next threshold }
     END;
  WRITE('SPCC = '); 
  FOR I := 21 TO 22 { This will display the 2 bytes of SPCC },
     DO WRITE(STOREP[I]);
  WRITELN;
  WRITE('TSAM = '); 
  FOR I := 23 TO 24 { This will display the 2 bytes of TSAM },
     DO WRITE(STOREP[I]);
  WRITELN;
  WRITELN;
  WRITELN('Input new parameters in hexadecimal format ');
WRITELN;
WRITELN;
VALID := FALSE;
WHILE (VALID = FALSE) DO
BEGIN  // Here we input a new value for SPER
  WRITE('Sampling period value (SPER) = ');
  READLN(INPUT,PARVALUE);  // Input new value
  IF LENGTH(PARVALUE) > 5 THEN  // If too long
    BEGIN
      VALID := FALSE;
      WRITELN('Exceeded the maximum length for SPER, Please re-type!');
    END
  ELSE VALID := TRUE;
  END;
IF COPY(PARVALUE,1,1) <> NL THEN  // If first char not NL
  FOR I := 1 TO 4  // Store new SPER in parameter line
    DO STOREP[II] := COPY(PARVALUE,1,1);
  VALID := FALSE;
WHILE (VALID = FALSE) DO
BEGIN  // Here we input a new value for STLTH
  WRITE('Settling time constant for log amplifier (STLTH) = ');
  READLN(INPUT,PARVALUE);  // Input new value
  IF LENGTH(PARVALUE) > 3 THEN  // If too long
    BEGIN
      VALID := FALSE;
      WRITELN('Exceeded the maximum length of STLTH, Please re-type!');
    END
  ELSE VALID := TRUE;
  END;
IF COPY(PARVALUE,1,1) <> NL THEN  // If first char not NL
  BEGIN
    P := 1;
    FOR I := 5 TO 6  // Store new STLTH in parameter line
      DO BEGIN
        STOREP[II] := COPY(PARVALUE,P,1);
        P := P + 1;
        END;
  END;
KT := 7;  // Index to first threshold in parameter line
FOR I := 1 TO 7 DO
BEGIN  // This loop will read in the 7 new thresholds
  VALID := FALSE;
  (and make sure they are correct length)
  WHILE (VALID = FALSE) DO  // Here, read in each threshold
    BEGIN
      WRITE('TH',I,' = ','Threshold value for band ',
              I,'; (' ',TEMP[II],') = ');  // Input new value
      READLN(INPUT,PARVALUE);
      IF LENGTH(PARVALUE) > 3 THEN  // If too long
        BEGIN
          VALID := FALSE;
          WRITE('Exceeded the maximum length (2 bytes) for ');
          WRITELN('the Threshold value, Please re-type!');
        END
      ELSE VALID := TRUE;
    END;
END;
IF COPY(PARVALUE,i,1) <> NL THEN (If input isn't NL)
BEGIN
  P := 1;
  FOR J := KT TO KT + 1 DO
  BEGIN (Store new threshold in parameter line)
    STOREP[J] := COPY(PARVALUE,P,1);
    P := P + 1;
  END;
END;
KT := KT + 2 (Index to next threshold)
END;
VALID := FALSE;
WHILE (VALID = FALSE) DO
BEGIN (Input new SPCC value)
  WRITE('SPCC:::');
  READLN(INPUT,PARVALUE); (Read new value)
  IF LENGTH(PARVALUE) > 3 THEN (If too long)
  BEGIN
    VALID := FALSE;
    WRITE('Exceeded the maximum length of (2 bytes)');
    WRITELN('for SPCC, Please re-type!');
  END
  ELSE VALID := TRUE;
END;
IF COPY(PARVALUE,i,1,1) <> NL THEN (If first char not NL)
BEGIN
  P := 1;
  FOR I := 21 TO 22
  DO BEGIN (Store new SPCC value in parameter line)
    STOREP[I] := COPY(PARVALUE,P,1);
    P := P + 1
  END;
END;
VALID := FALSE;
WHILE (VALID = FALSE) DO
BEGIN (Here we input new TSAM value)
  WRITE('TSAM = ');
  READLN(INPUT,PARVALUE); (Input new value)
  IF LENGTH(PARVALUE) > 3 THEN (If too long)
  BEGIN
    VALID := FALSE;
    WRITELN('Exceeded the maximum length of (2 bytes) for',
             'the TSAM value, Please re-type!');
  END
  ELSE VALID := TRUE;
END;
IF COPY(PARVALUE,i,1) <> NL THEN (If first char not NL)
BEGIN
  P := 1;
  FOR I := 23 TO 24
  DO BEGIN (Store new TSAM value in parameter line)
    STOREP[I] := COPY(PARVALUE,P,1);
    P := P + 1;
  END;
END;
ERRF := FALSE;
FOR I := 1 TO 24
DO BEGIN
  TEMPINPUT[1] := STOREP[I];
  TEMPINPUT[2] := '0';
  AB70(TEMPINPUT);  ( Send out first byte and receive echo )
    BEGIN
      WRITELN;
      WRITELN('**** serial communication error! ****');
      WRITELN;
      ERRF := TRUE;
    END;
  IF ERRF = TRUE THEN I := 24;
END;

{ This part of the parameter procedure will give the user some options, }
{ and then will issue the commands to the 8751 based on user response. }

IF ERRF = FALSE THEN  { If no communication error }
BEGIN
  WRITELN('Average adjacent frames (Y/N) ? ');
  READLN(INPUT, COM);
  COM := 'N';
  IF (COM = 'Y') OR (COM = 'y') THEN
    WRITE(LST, ACMD);
  IF (COM = 'N') OR (COM = 'n') THEN
    WRITE(LST, DACMD);
  WRITELN('Collect 5 frames after the end of word (Y/N) ? ');
  READLN(INPUT, COM);
  COM := 'N';
  IF (COM = 'Y') OR (COM = 'y') THEN
    WRITE(LST, TSCMD);
  IF (COM = 'N') OR (COM = 'n') THEN
    WRITE(LST, MTSCMD);
END;  { END OF A LARGE IF-THEN BLOCK }

PROCEDURE CHECKEND (VAR END_FLAG : INTEGER; N : INTEGER);

VAR
  SMNLINE : ARRAY[1..N_EOL] OF INTEGER;
  I : INTEGER;
  TFLAG : INTEGER;

FUNCTION HEX(CH:CHAR) : INTEGER;
BEGIN
  IF (CH >= '0') AND (CH <= '9');
THEN HEX:=ORD(CH) - ORD('0')
ELSE HEX:=ORD(CH) - ORD('A') + 10;
END;

BEGIN ( CHECKEND );
FOR I:=1 TO N_BPF DO
   SMLINE[I]:=HEX(SAMPLE[I*I-I]) * 16 + HEX(SAMPLE[I*I+I]);
TFLAG:=0;
FOR I:=1 TO N_BPF DO
   IF SMLINE[I] > THRESH[I] THEN TFLAG:=1;
IF TFLAG = 1 THEN COUNT:=0
ELSE BEGIN
   COUNT:=COUNT + 1;
   IF COUNT = EN_COUNT THEN BEGIN
      END_FLAG:=1;
      WRITE(LST,INTERLE; \( RESET 8751 \) :);
      END;
      END;
END;

BEGIN ( Main program )
WRITELN; \( This is the main program, responsible for running the training phase \)
WRITELN("********** Training phase has started **********");
WRITELN;
WRITELN("********** Training phase has started **********");
WRITELN;
INITIALIZE; \( Initialize this program \)
REPEAT \( Command loop \)
   GOLF := FALSE;
   READLN(INPUT,WORD); \( Get typed input \);
   IF WORD = CP THEN \( If we want to change the parameters \)
   PARAMETERS; \( Change 8751 parameters \)
ELSE IF WORD = WORM THEN BEGIN  (Here the user wants to boot the 8751.)

WRITE('Warm boot the 8751? (Y/N) ?');
READLN(INPUT,ANS);
IF (ANS='Y') OR (ANS='y') THEN WRITE(LST,RINTEL);
ELSE IF (ANS = 'N') OR (ANS = 'n') THEN BEGIN
WRITE('Cold boot the 8751? (Y/N) ?');
READLN(INPUT,ANS);
IF (ANS='Y') OR (ANS='y') THEN WRITE(LST,RINTEL);
END;
END;
ELSE IF WORD <> END_CMD THEN BEGIN  (If we didn't type the end command)

N := 0;
END_FLAG:=0;
CNTLOW:=0;
ST_CHK:=0;
WRITE('Say the word typed above into the microphone.');
WRITE(LST,F STARTH);
TOO_LARGE := FALSE;
ACTIVE := FALSE;
DVFF := FALSE;
REPEAT
{Here we read in all the frames up to the word marker.}

IF N < MAX_SAMPLE THEN BEGIN  (This executes when not too many frames)
N := N + 1;
READLINE(SAMPLEIN);  (Read one frame from the F-E box)
ST_CHK:=ST_CHK+1;
IF ST_CHK = 15 THEN CHECKEMD(END_FLAG,N);
IF NOT ACTIVE THEN BEGIN  (Executes only when we get the 1st frame)
ACTIVE := TRUE
END;
END;
ELSE BEGIN  (Executes when we get too many frames)
READLINE(DUMMY);  (Throw away frame since too many frames)
END_FLAG:=1;
WRITE('!');
WRITE(LST,RINTEL);  (Reset 8751)
TOO_LARGE := TRUE  (Indicate too many frames)
END;
UNTIL (END_FLAG=1);
WRITEH;
IF TOO_LARGE <> TRUE THEN

WRITEH('The word was ',N,' frames in length');
IF N < MIN_SAMPLE THEN BEGIN  (Detect if too few frames)
WRITEH('Word too short, Input ignored.');
WRITEH(LST,RINTEL);
END  (Send warm boot command to 8751)
ELSE IF TOO_LARGE THEN BEGIN  (Detect if too many frames)
WRITEH('Word too long, Input ignored.');
WRITEH(LST,RINTEL);  (Reset the 8751)
END
ELSE BEGIN
WRITEH('Would you like to store this prototype? ');
READLN(INPUT,CH);  (Get answer)
IF (CH='Y') OR (CH='y') THEN BEGIN

104
(STORE THE FRAMES INTO THE VOCABULARY FILE.)

WRITELN('Storing ',WORD);
WRITELN(VOCAB,WORD);  (* Store text word in file *)
WRITELN(VOCAB,ZERO_CNT); (* Store zero counter for each word *)
FOR I := 1 TO N-SPC DO BEGIN
  J := 0;
  REPEAT
    J := J + 1;
    WRITE(VOCAB,SAMPLE[I,J]) (* Each byte *)
    UNTIL SAMPLE[I,J]=NL; (* Until end of frame *)
  END;
  WRITE(VOCAB,END_WRD);
  (* Insert end of word marker in vocab file *)
  WRITE(VOCAB,END.FILE);
  (* Insert end of file marker in vocab file *)
  WRITE(VOCAB,END);
END
ELSE WRITELN('Input ignored!');
END;
END;
UNTIL WORD = END_CMD; (* Until end command detected *)
WRITELN('The training phase has now ended.');
CLOSE(VOCAB);
END.
APPENDIX C
PROFILE PROGRAM LISTING
CONST
N_ELM = 15;  \{ NUMBER OF READINGS IN A FRAME \}
MAX_VOCAB = 36; \{ MAXIMUM NUMBER OF WORDS IN A FILE \}
END_WORD = "F"; \{ FIRST END OF WORD PATTERN MARKER \}
END_FILE = "F"; \{ SECOND END OF WORD PATTERN MARKER \}
MAX_SAMPLE = 60; \{ MAXIMUM NUMBER OF FRAMES IN A WORD PATTERN \}

TYPE
LINE_TYPE = ARRAY[1.40] OF CHAR; \{ INPUT LINE BUFFER \}
TIME_SAMPLE = ARRAY[1..N_ELM] OF BYTE; \{ ONE FRAME OF A PATTERN \}
SAMPLE = ARRAY[1..MAX_SAMPLE] OF TIME_SAMPLE; \{ WORD PATTERN \}
NAME_TYPE = STRING[12];
CHART_TYPE = ARRAY[1..MAX_VOCAB] OF CHAR;

VAR
LBMFILE : TEXT; \{ FILE TO BE PURGED \}
FILENAME : NAME_TYPE; \{ FILE NAME OF FILE TO BE CREAT ED \}
FILENAME2 : NAME_TYPE; \{ FILE NAME OF FILE TO BE CREAT ED \}
LINE : LINE_TYPE; \{ BUFFER TO READ IN ONE FRAME \}
N_WORDS : INTEGER; \{ NUMBER OF WORDS IN THE FILE \}
TOTAL_CNT : INTEGER; \{ TOTAL NUMBER OF COUNTS \} MATCHES \}
THR1 : INTEGER; \{ THRESHOLD \} TO ALLOW PURGING \}
THR2 : INTEGER; \{ THRESHOLD \} FOR KEEPING A WORD \}
CHART : CHART_TYPE; \{ KEEPS TRACK \} OF WHICH WORDS WILL BE KEPT \}
WORDS : ARRAY[1..MAX_VOCAB] OF RECORD \{ DATA STRUCTURE \} FOR HOLDING EACH WORD \}
N : INTEGER; \{ NUMBER \} OF FRAMES IN THE WORD \}
CNT : INTEGER; \{ NUMBER \} OF CORRECT MATCHES FOR THIS \WORD \}
TT : STRING[20]; \{ SPELLING OF THIS \WORD \}
DAT : SAMPLE \{ WORD \} PATTERN \}
END;
PRMT : CHAR; \{ DUMMY \} VARIABLE \}

{ -----------------------------------------------

PROCEDURE READLINE (VAR F : TEXT; VAR LINE : LINE_TYPE);
{ READS IN ONE LINE (FRAME) INTO AN ARRAY OF CHARACTERS (LINE),
TERMINATING WITH (NL) FROM THE TEXT FILE PASSED THROUGH F ;

VAR
I : INTEGER;
NL : CHAR;

BEGIN
I:=0;
NL:=CHAR(10);
REPEAT
I:=I+1;
READ(F,LINE(I));
UNTIL LINE(I) = NL;
END;
PROCEDURE CONVERT (LINE : LINE_TYPE; VAR ARY : SAMPLE; S : INTEGER; VAR NNSFLAG : INTEGER);  

{ CONVERT THE INPUT ASCII CHARS IN LINE (REPRESENTS ONE FRAME OF DATA) FROM ASCII TO ITS NUMERIC VALUE AND STORES IT IN ROW S OF PASSED ARY. IF AT ANY TIME A NEGATIVE VALUE IS READ FOR AN ENERGY OF ZERO CROSSING VALUE, OR A VALUE WHICH IS GREATLY DIFFERENT THAN IN THE PREVIOUS FRAME, THEN THE NNSFLAG IS SET. THIS HAPPENS WHEN A BYTE HAS BEEN LOST FROM A FRAME DURING THE TRAINING PHASE. }

VAR I,ITNEG : INTEGER;

FUNCTION HEX (CH : CHAR) : INTEGER;
BEGIN
IF (CH)='0' AND (CH<='9')
  THEN HEX:=ORD(CH) - ORD('0')
  ELSE HEX:=ORD(CH) - ORD('A') + 10;
END;

BEGIN ( CONVERT )
NNSFLAG:=0;
FOR I=1 TO N_ELM DO BEGIN
  ITNEG:=HEX(LINE[I+1]) * 16 + HEX(LINE[I+2]);
  ARY[S,I]:=ITNEG;
  IF S > 1 THEN IF ABS(ARY[S-1,I]-ITNEG) > 50 THEN NNSFLAG:=1;
  IF ITNEG < 0 THEN NNSFLAG:=1;
END;
END;

PROCEDURE IN_FILE;

VAR NAME1 : STRING[20];
NAME2 : STRING[20];
CHR : CHAR;
I,X : INTEGER;
NNSFLAG : INTEGER;

BEGIN
WRITE('WHAT IS THE LIBRARY FILE NAME ? ');
READLN(FILENAME);  { OBTAIN NAME OF FILE TO BE PURSED }
ASSIGN(LBFILE,FILENAME);  { FROM THE USER. RESET THE FILE }
RESET(LBFILE);  { MARKER }
WRITELN('READING IN LIBRARY. PLEASE WAIT...');
N_WORDS:=0;  { CLEAR WORD COUNT }

108
REPEAT
  N_WORDS:=N_WORDS+1;  { INCREMENT WORD COUNT }
  READLN(LBRFLE,NAMEx);  { READ IN SPELLING OF THE NEXT WORD }
  NAME:='';  { IN THE FILE, CONVERT IT TO UPPER }
  FOR K:=1 TO LENGTH(NAME) DO BEGIN { CASE BEFORE STORING IT IN THE }
    CHR:=COPY(NAME,K,1);  { DATA STRUCTURE }
    CHR:=UPCASE(CHR);  { STORE IT IN THE DATA STRUCTURE }
    NAME:=CONCAT(NAME,CHR);
  END;
  WORDS[N_WORDS].tit:=NAME;  { STORE IT IN THE DATA STRUCTURE }
  READLN(LBRFLE,WORDS[N_WORDS].cnt);  { READ IN WORD COUNT AND STORE }
  WRITE(WORDS[N_WORDS].tit,'-','WORDS[N_WORDS].cnt);
  I:=0;
  READLINE(LBRFLE,LINe);  { READ IN FIRST FRAME OF WORD PATTERN }
  WHILE LINE<>END_FILE DO BEGIN { WHILE NOT SECOND END OF FILE MARKER }
    IF LINE<>END_WORD THEN BEGIN { WHILE NOT FIRST END OF FILE MARKER }
      I:=I+1;  { INCREMENT FRAME COUNT }
      CONVERT(LINE,WORDS[N_WORDS].dat,(I,NSFLAG));  { ASCII TO DECIMAL AND STORE }
    END;
    IF NSFLAG=1 THEN I:=I-1;  { IF MNS FLAG IS SET THEN DISCARD }
    END;  { FRAME BY DECREMENTING THE FRAME COUNT }
  READLINE(LBRFLE,LINe);
  WORDS[N_WORDS].n:=I;  { STORE THE FRAME COUNT }
  UNTIL EOF(LBRFLE) OR (N_WORDS = MAX_VOCAB);
  WRITELN;
  CLOSE(LBRFLE);
END;

{ ------------------------------------------------------------------------ }

PROCEDURE SET_THRI (VAR THRI : INTEGER);
{ THE FIRST THRESHOLD REPRESENT THE MINIMUM NUMBER OF MATCHES MADE TO }
{ THE FILE BEING PURGED, IN ORDER TO ALLOW THE FILE TO BE PURGED. }
{ THREE TIMES THE NUMBER OF WORDS IN THE FILE IS THE CURRENT THRESHOLD. }
BEGIN
  THRI:=3*N_WORDS;
END;

{ ------------------------------------------------------------------------ }

PROCEDURE SUM_CNT (VAR TOTAL_CNT : INTEGER);
{ SUM UP THE TOTAL COUNT (MATCHES) OF THIS FILE. }
VAR  I : INTEGER;
BEGIN
  TOTAL_CNT:=0;
  FOR I:=1 TO N_WORDS DO
    TOTAL_CNT:=TOTAL_CNT + WORDS[I].cnt;
  WRITELN('THE TOTAL COUNT IS: ',TOTAL_CNT);
END.
PROCEDURE SET_THR2 (VAR THR2 : INTEGER);

{ THE SECOND THRESHOLD IS THE MINIMUM NUMBER OF MATCHES WHICH MUST
  HAVE BEEN MADE TO A WORD, IN ORDER TO KEEP THIS WORD IN THE NEW
  FILE. }

BEGIN
  THR2:=ROUND((TOTAL_CNT/N_WORDS)/3);
  WRITELN('THE MINIMUM COUNT VALUE FOR ACCEPTING A WORD IS: ', THR2);
  WRITE('PRESS RETURN TO CONTINUE, '));
  READLN(PRNT);
END;

PROCEDURE THROWOUT (THR2 : INTEGER; VAR CHART : CHART_TYPE);

{ IF WORD I OF THE FILE IS TO BE INCLUDED IN THE NEW FILE, THEN
  CHART[I] SHOULD BE SET TO 'Y', ELSE IT SHOULD BE SET TO 'N'.
  A WORD WILL BE INCLUDED IN THE NEW FILE IF EITHER 1) IT'S COUNT
  VALUE IS GREATER THAN THR2, OR 2) IT HAS THE HIGHEST COUNT VALUE
  OF ALL THE WORDS IN THIS FILE WHICH HAVE THE SAME SPELLING (I.E.
  THE SAME WORDS) }

VAR I, J : INTEGER;
  TEMP : STRING(10);

BEGIN
  FOR J:=1 TO N_WORDS DO CHART[I]:='Y';  { INITIALIZE ARRAY CHART }
  FOR I:=1 TO N_WORDS DO BEGIN
    IF WORDS[I].CNT < THR2 THEN BEGIN { DETERMINE IF A WORD SHOULD }
      ( NOT BE INCLUDED )
      ( ACCORDING TO THE ABOVE CRITERIA )
    END;
    FOR J:=1 TO N_WORDS DO BEGIN
      IF (WORDS[I].TXT = WORDS[J].TXT) AND (I<>J)
      AND (CHART[I] = 'N') AND (WORDS[I].CNT <= WORDS[J].CNT) THEN
      CHART[I]:='N';
    END;
  END;
  WRITELN; { WRITE WHAT WORDS WILL BE KEPT AND }
  ( WORDS WILL BE DELETED WHEN MAKING }
  ( THE NEW FILE )

  WRITELN('LIBRARY LINE: ', FILENAME);
  WRITELN('INDEX WORD COUNT STATUS');
  FOR I:=1 TO N_WORDS DO BEGIN
    TEMP:='KEEP';
    IF CHART[I] = 'N' THEN TEMP:='DELETE';
    WRITELN(I,WORDS[I].TXT,WORDS[I].CNT,TEMP);
  END;

  WRITELN;
PROCEDURE OUT_FILE (FILENAME : NAME_TYPE; CHART : CHART_TYPE);

CONST  STCON = '0123456789ABCDDE';

VAR   NL : CHAR;
      TEMP : NAME_TYPE;
      I,L,M : INTEGER;
      HI16,RM16 : INTEGER;
      CHR : CHAR;
      ASKEY : STRING[1];

BEGIN
  NL:=CHAR(IO);
  FILENAME2:="; ( GIVE THE NEW FILE THE SAME )
  Writeln('FORMING NEW FILE NAME'); ( NAME AS THE OLD FILE BUT )
  IF POS('.',FILENAME) <> 0 THEN BEGIN
    TEMP:=FILENAME;
    I:=1;
    WHILE COPY(TEMP,1,1) <> '.' DO BEGIN
      CHR:=COPY(TEMP,1,1);
      FILENAME2:=CONCAT(FILENAME2,CHR);
      I:=I+1;
    END;
    FILENAME2:=CONCAT(FILENAME2,'.NEW');
  END
  ELSE BEGIN
    FILENAME2:=FILENAME; ( IF THE FILE DOES NOT HAVE AN )
    FILENAME2:=CONCAT(FILENAME2,'.NEW'); ( EXTENSION THEN ADD .NEW )
  END;
  Writeln('NEW FILE NAME IS: ',FILENAME2); ( IF THE FILE DOES NOT HAVE AN )
  Writeln('PRESS RETURN TO CONTINUE');
  Readln(PMNAT);
  Writeln;
  Assign(LBRFILE,FILENAME2); ( WRITE THE NEW FILE )
  Rewrite(LBRFILE);
  Writeln('WRITING NEW LIBRARY INTO FILE ',FILENAME2);
  FOR I:=1 TO N_WORDS DO BEGIN
    IF CHART[I] = 'Y' THEN BEGIN
      WRITE(WORDS[I].TXT:12,'-','WORDS[I].CNT:3);
      Writeln(LBRFILE,WORDS[I].TXT); ( THE WORDS MUST BE WRITTEN OUT )
      WORDS[I].CNT:=0; ( IN THE SAME WAY THAT THE TRAINS )
      Writeln(LBRFILE,WORDS[I].CNT); ( PROGRAM STORES THEM )
    END;
    FOR L:= 1 TO WORDS[I].N DO BEGIN
      FOR M:= 1 TO N_ELM DO BEGIN
        HX16:=TRUNC(WORDS[I].DATIL,M/16);
        RM16:=WORDSI].DATIL,M-16*HX16; ( THEREFORE THESE VALUES ARE )
        ASKEY:=COPY(STCON,HX16+1,1); ( CONVERTED FROM DECIMAL TO ASCII )
        WRITE(LBRFILE,ASKEY); ASKEY:=COPY(STCON,RM16+1,1);
      END;
    END;
  END;
END;
END;
WRITE(LBRFILE,'$',NL);
END;
WRITE(LBRFILE,'$',NL);
END;
CLOSE(LBRFILE);

BEGIN ( MAIN PROGRAM )
IN_FILE;
SET_THR1(THR1);
SUM_CNT(TOTAL_CNT);
IF TOTAL_CNT > THR1 THEN BEGIN
  SET_THR2(THR2);
  THROWOUT(THR2,CHART);
  WRITE('PRESS RETURN TO CONTINUE, OR 'E' TO ESCAPE ');
  READLN(PRMt);
  IF PRNT <> 'E' THEN OUT_FLE(IFILENAME,CHART); ( NEW FILE )
END
ELSE WRITE('LIBRARY FILE HAS NOT BEEN USED ENOUGH TO WARRANT PURGING ');

END. ( MAIN PROGRAM )
PROGRAM GRAPH (INPUT, OUTPUT);

TYPE
  TIME_SAMPLE=ARRAY [1..7] OF INTEGER; { EACH TIME SAMPLE }
  LINE_TYPE=ARRAY [1..32] OF CHAR; { # OF BYTES PER FRAME }
  ZERO_SAMPLE=ARRAY [1..7] OF INTEGER; { HOLDS ZERO CROSSINGS }
    ( AT EACH FREQ, DOESN'T CONTAIN MIC ZERO X'S)
  ZCROSS=ARRAY [1..100] OF ZERO_SAMPLE; { HOLDS ZERO CROSSINGS FOR WORD }
  XOMIC_TYPE=ARRAY [1..100] OF INTEGER; { MIC ZERO CROSSINGS FOR WORD }
  DATA=ARRAY [1..100] OF TIME_SAMPLE; { HOLDS BAND PASS VALUES FOR WORD }

VAR
  VOCAB : TEXT; { VOCABULARY FILE }
  CMD : CHAR; { USED AS USER INPUT VARIABLE }
  SAMPLE : DATA; { BAND PASS FILTER VALUE ARRAY FOR A WORD }
  Z_SAMPLE : ZCROSS; { ZEROS X'S ARRAY FOR FILTERS FOR A WORD }
  LINE : LINE_TYPE; { USED TO HOLD ASCII VALUES OF A FRAME }
  N,M : INTEGER;
  N_WORDS : INTEGER; { NUMBER OF WORDS IN VOCABULARY FILE }
  NAME : STRING [80]; { NAME OF THE WORD BEING GRAPHED }
  NL : CHAR;
  ICNIC : XOMIC_TYPE; { MIC ZERO CROSSINGS FOR A WORD }
  FILENAME : STRING [80]; { NAME OF INPUT FILE }
  E_FLAG : BYTE; { FLAG TO DENOTE IF GRAPHING IS DESIRED }
    ( FOR CURRENT WORD }
  FF : CHAR;
  CMD3 : STRING [3];
  MNG5FLAG : INTEGER; { FLAG FOR A BAD FRAME READING }

PROCEDURE READLINE (VAR F : TEXT; VAR LINE : LINE_TYPE);
( READS IN ONE LINE (FRAME) INTO AN ARRAY OF CHARACTERS (LINE),
  TERMINATING WITH <NL>, FROM THE TEII FILE PASSED THROUGH F )

VAR
  I : INTEGER; { INDEX INTO ARRAY }
  E_O_L : BOOLEAN; { INDICATES NL RECEIVED }

BEGIN
  I:=O;
  REPEAT
    I:=I+1;
    READ (F,LINE[I]);
    E_O_L:= (LINE[I]=NL);
  UNTIL E_O_L
END;

PROCEDURE CONVEI: LINE ; LINE_TYPE; VAR ARY : DATA; VAR AZCY : ZCROSS; S : INTEGER);
FUNCTION HEX(CH:CHAR): INTEGER;
    \( \text{CONVERT HEX DIGIT IN ASCII TO CORRESPONDING INTEGER} \)
BEGIN
IF \( \text{CH} = '0' \) AND \( \text{CH} = '9' \) THEN \( \text{HEX} := \text{ORD} (\text{CH}) - \text{ORD} ('0') \) \( \text{\{ 0..9 \}} \)
ELSE \( \text{HEX} := \text{ORD} (\text{CH}) - \text{ORD} ('A') + 10 \) \( \text{\{ A..F \}} \)
END;

VAR I: INTEGER;

BEGIN
MNSFLAG:=0;
FOR I:=1 TO 7 DO \( \{ \text{FOR EACH BAND} \} \)
BEGIN
ARY[S,I]:=HEX(LINE[I*1-1]) \*16 + HEX(LINE[I*1]);
\( \{ \text{CONVERT FILTER VALUE} \} \)
IF S > 1 THEN IF \( \text{ABS} (\text{ARY}[S-1,I]-\text{ARY}[S,I]) > 50 \) THEN MNSFLAG:=1;
IF \( \text{ARY}[S,I] < 0 \) THEN MNSFLAG:=1;
END;
AZCY[S,1]:=HEX(LINE[31-I*1]) \*16 + HEX(LINE[32-I*1]);
\( \{ \text{CONVERT SAME FILTER'S ZERO CROSSINGS} \} \)
IF S > 1 THEN IF \( \text{ABS} (\text{AZCY}[S-1,I]-\text{AZCY}[S,I]) > 50 \) THEN MNSFLAG:=1;
IF \( \text{AZCY}[S,I] < 0 \) THEN MNSFLAG:=1;
END;
ZCMIC[S]:=HEX(LINE[15]) \*16 + HEX(LINE[16]);
\( \{ \text{CONVERT MIC ZERO CROSSINGS} \} \)
IF S > 1 THEN IF \( \text{ABS} (\text{ZCMIC}[S-1,I]-\text{ZCMIC}[S,I]) > 50 \) THEN MNSFLAG:=1;
IF \( \text{ZCMIC}[S] < 0 \) THEN MNSFLAG:=1;
END;

PROCEDURE PRINTOUT \{ N:INTEGER \};

VAR K : 1..35;
J : 1..35;
I : INTEGER;
FREQ : ARRAY[1..7] OF INTEGER;

BEGIN
FREQ[1]:=62;
FREQ[2]:=125;
FREQ[3]:=250;
FREQ[4]:=500;
FREQ[5]:=1000;
FREQ[6]:=2000;

115
FREQ(K) = 4000;
K:=1;
WHILE K < 8 DO  ( PRINT GRAPHS FOR ALL 7 BAND-PASS FILTERS )
BEGIN
WRITE(LST,' ');
( WRITE TEN BLANKS )
WRITE(LST,'BANDPASS FILTER ','K',' CENTER FREQUENCY ','FREQ(K)," Hz"');
WRITE(LST,' ');
( WRITE 25 BLANKS )
WRITE(LST,'BANDPASS FILTER ','K',' # OF ZERO CROSSINGS');
WRITE(LST);
FOR J:=1 TO N DO  ( N IS THE NUMBER OF FRAMES )
BEGIN
WRITE(LST,' 
( 1'S ARE SUPPOSE TO CONSTRUCT A SORT OF AXIS )
FOR I:=1 TO ROUND(SAMPLE(I,K)/4) DO WRITE(LST,"*");
( PRINT OUT BPF VALUE (DIVIDED BY FOUR) FOR THIS FRAME )
FOR I:=1 TO (65-ROUND(SAMPLE(I,K)/4)) DO WRITE(LST,' '); ( MOVE TO COLUMN 66 TO PRINT Z-XING VALUE OF SAME FILTER )
WRITE(LST,' 
IF Z_SAMPLE(I,K) > 50 THEN Z_SAMPLE(I,K):=50;
FOR I:=1 TO Z_SAMPLE(I,K) DO WRITE(LST,'*');
( PRINT BPF ZERO CROSSING VALUE FOR THIS FRAME )
END;
WRITE(LST);
FOR I:=1 TO 130 DO WRITE(LST,'-');  ( SEPARATE EACH FILTER'S GRAPH )
WRITE(LST);
WRITE(LST);
WRITE(LST);
K:=K+1;
END;
WRITE(LST)
WRITE(LST,' 
WRITE(LST,' 
WRITE(LST,'MICROPHONE OUTPUT , # OF ZERO CROSSINGS (DIV BY 2)');
WRITE(LST);
( NEXT PRINT OUT GRAPH OF MIC'S ZERO CROSSINGS VALUES DIVIDED BY 2 )
FOR J:=1 TO N DO
BEGIN
WRITE(LST);
WRITE(LST,' 
WRITE(LST,' 
WRITE(LST)
WRITE(LST)
WRITE(LST)
WRITE(LST, '------------------------------------------------------------- 
WRITE(LST,
END
( **********************************************************************

BEGIN ( THIS IS THE MAIN PROGRAM )
WRITE("WHAT IS THE NAME OF THE INPUT FILE ? ");
READLN(FILENAME);
ASSIGN(VOCAB,FILENAME);
RESET(VOCAB);
N_WORDS:=0;
FF:=CHAR(12);
NL:=CHAR(10);
WRITE ('WOULD YOU LIKE THE WHOLE FILE GRAPHED? ') ;
READLN(CMD3);
IF (CMD3='y') OR (CMD3='yes') OR (CMD3='YES')
  THEN CMD3:='Y';
IF(CMD3='Y') THEN BEGIN
  6_FLAG:=1;
  CMD:= 'Y';
END;
REPEAT
READLN(VOCAB,NAME);  ( GET NAME OF WORD )
IF CMD3='Y' THEN BEGIN  ( IF WHOLE FILE IS NOT TO BE GRAPHED )
  WRITE;
  ( SEE IF GRAPH OF THIS WORD IS DESIRED )
  WRITE('WOULD YOU LIKE THE GRAPH FOR THE WORD ',' NAME:', '?') ;
  WRITE('Y - YES  N - NO  * - QUIT GRAPhING PROGRAM') ;
  READLN(CMD);
  WHILE(CMD> 'Y') AND (CMD> 'y') AND (CMD> 'N') AND (CMD> 'n')
  AND (CMD<> '*' ) DO BEGIN
    WRITE('Y, N, or *');
    READLN(CMD);
  END;
  IF CMD <> '*' THEN ( IF CMD='*', THEN EXIT PROGRAM )
  BEGIN
    ( SET 6_FLAG TO 1 IF THIS WORD IS TO BE GRAPHED )
    IF (CMD='y') OR (CMD='y') THEN 6_FLAG:=1;
    ELSE 6_FLAG:=0;
    N_WORDS:=N_WORDS+1; ( INCREMENT WORD COUNT )
    IF 6_FLAG=1 THEN BEGIN ( IF GRAPH OF THIS WORD DESIRED, THEN PRINT )
      WRITE(LST); ( OUT AN IDENTIFYING HEADER )
      WRITE(LST);
      WRITE(LST);
      WRITE(LST, 'CURRENTLY ON WORD NUMBER ',N_WORDS);
      WRITE(LST, 'FILE: ',FILENAME,' ');
      WRITE(LST, 'WORD CURRENTLY BEING ANALYZED: ',NAME);
      WRITE(LST);
    END;
    N:=0;
    READLINE(VOCAB,LINE);  ( READ FIRST FRAME OF WORD )
    WHILE LINE[1] <> '*' DO ( SEE IF FIRST END OF WORD MARKER REACHED )
    BEGIN
      ( IF GRAPHING DESIRED THEN CONVERT FRAME FROM ASCII DATA )
      TO INTEGER, AND STORE IN APPROPRIATE ARRAYS FOR GRAPHING )
      ( THEN READ IN NEXT FRAME )
      N:=N+1;
      IF 6_FLAG=1 THEN BEGIN
        CONVERT(LINE,SAMPLE,Z_SAMPLE,N);
        IF MNSFLAS=1 THEN N:=N-1;
      END;
    READLINE(VOCAB,LINE);  
  END;
  READLINE(VOCAB,LINE);
  WHILE LINE[1] <> '*' DO ( SEE IF SECOND END OF WORD MARKER REACHED )
BEGIN

( IF GRAPHING DESIRED THEN CONVERT FRAME FROM ASCII DATA )
( TO INTEGER, AND STORE IN APPROPRIATE ARRAYS FOR GRAPHING )
( THEN READ IN NEXT FRAME )
N:=N+1;
IF G_FLAG=1 THEN CONVERT(LINE,SAMPLE,1_SAMPLE,N);
READLINE(VOCAB,LINE);
END;
IF G_FLAG=1 THEN PRINTOUT(N); ( GRAPH WORD IF DESIRED )
IF G_FLAG=1 THEN WRITELN('THE WORD ',NAME,' WAS JUST GRAPHED');
END;
UNTIL EOF(VOCAB) OR (CMD='*');
CLOSE(VOCAB);
WRITELN;
WRITELN('END OF FILE ',FILENAME,' HAS BEEN REACHED -- EXITING PROGRAM');
END.
REFERENCES


