AN OBJECT-ORIENTED APPROACH TO A SPEAKER-INDEPENDENT ISOLATED WORD RECOGNITION SYSTEM

by

ROSS PAUL GOERES

Bachelor of Science in Electrical Engineering

The University of Texas at Austin

1984

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1988

AN OBJECT-ORIENTED APPROACH TO A
SPEAKER-INDEPENDENT ISOLATED
WORD RECOGNITION SYSTEM

Thesis Approved:

C. M. Bacon
Thesis Adviser

Keith A. Teague

Richard L. Cummins

Norman N. Durham
Dean of the Graduate College

ACKNOWLEDGMENTS

When I started this, I had a lot of questions about speech recognition and cognition in general based on intuitive ideas rather than research or statistics. It seemed reasonable that if I could look at a plot and tell where numerical methods were lacking, then it should be possible to instruct a computer to use heuristics to mitigate this shortcoming. The idea seemed so straightforward that I was certain that it must have been tried before. I had searched the prevailing literature extensively to no avail. At this point I would like to thank my advisory committee for the encouragement, direction, and support that made it possible for me to pursue completion of a project that, at the outset, I had doubts would be feasible. Fortunately, to the degree that I could test it, it worked.

Those serving as my advisory committee are: Dr. Richard L. Cummins, my initial contact, who immediately demonstrated the haecceity of OSU; Dr. Keith A. Teague, the original and continuing inspiration for this particular project; and Dr. Charles M. Bacon who has always listened to my musings and offered constructive advice on how to focus the most diffuse

information into a coherent form.

I am also grateful for the understanding and patience shown by my wife, Mindy, during the trials of creating this document.

No listing of those deserving thanks would be complete without mentioning the Air Force Institute of Technology, without which none of this would have been possible.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Background

The purpose of this research is to investigate the theory and implementation of alternative approaches to a speaker-independent word recognition system. The original interest in this project stems from the success of a prototype system developed as an ECEN 5753 Digital Speech Signal Processing semester project. The purpose was to implement the system described in 'A Speaker-Independent Digit-Recognition System' [Sambur (1975)]. The method used four measures over 1Ø msec intervals (frames): average magnitude, zero-crossing rate (ZCR), linear predictive coding (LPC) coefficients, and the first difference of the LPC coefficients as an approximation of the first derivative. The average magnitude and ZCR were used to determine the endpoints of the utterance [Rabiner (1975)] and for the bulk of the classification. The LPC coefficients and their first differences were used as "auxiliary information" and were computed but not always used.

A striking feature of the Sambur algorithm is that it is strictly fine-tuned for the English digits only and defies modification or addition. For example, pronunciation variations of 'Ø' as 'zeero', 'ziro', and 'sero' are not taken into account, nor can they be easily accommodated.

Working on the assumption that the phonemes in the target words would be in the same order, and somewhat in proportion to the length of the utterance whether spoken quickly or slowly, the semester project design used the average magnitude and ZCR exclusively. The basic idea was to normalize time, magnitude, and ZCR to account for large and small amplitudes, along with short and long enunciations. This system, using just one voice (the author's) as a standard, was able to correctly identify over 7Ø percent of the test data on the initial run. The distance measure used to estimate how much an unknown word differed from a reference word was the square root of the sum of the squares of the differences. The major difficulty with this is that it suffers from phase problems in the matching routine, and as with the Rabiner endpoint detection algorithm, is very sensitive to background noise.

The methodology proposed here fundamentally differs from the above schemes in that the basis for recognition is the visual cues a human expert would see in the frame plots while attempting to match the

overall patterns of the unknown word to those of a specific word in a restricted vocabulary. A human glancing over the frame plots of the digits 'Ø' to '9' from one speaker will readily note how different the signatures of each digit can be (examples of this can be seen in Appendix A).

At the same time the observer will also note how remarkably similar the frame plots for a given number will be across different speakers. (For the sake of illustration, Appendix A presents a speaker ensemble uttering the digit '6'. Figure 44 is of particular interest. There is a considerable amount of background noise, the speaker slurred his speech, and the rate of speech was fast enough to violate the separation criteria for isolated utterances. Despite this, it does not take a human observer long to conclude that the word is unquestionably a '6'.) The codification of the process of how an observer can identify the unknown word from visual clues is the heart of this thesis.

The patterns of spoken digits are not stored as templates from which a numerical score may be calculated to quantify how well the unknown utterance matches the reference, but rather as a set of rules whereby the presence or absence of certain features will determine the classification. These rules comprise a shape description with each word treated as an object constructed of geometric primitives. This

allows virtually any word to be added to the vocabulary if its features can be described in the form of these shape primitives. The actual identification is automatically performed with the use of a backward-chaining inference engine (IE). The IE hides the details of implementation of how the inference is done so that the programmer may concentrate on treating the shapes on the frame plots as objects.

Objectives

The primary objective of this research is to perform isolated-word recognition of digits via visual information contained in signal patterns of the unknown word. Ideally, this rule-based system would be speaker-independent rather than having to be trained for each new person. This may be accomplished by the use of fuzzy logic [Schmucker (1984)] and a robust rule set to allow for variants in pronunciation (e.g. '8' as 'ay-yet', 'ate', and 'ay-tuh'). Quantifying uncertainty with fuzzy logic and heuristic distance measures provides the mechanisms with which to propagate certainty factors to assess the quality of a deduction. This gives the method the ability to make a guess in the face of uncertainty and issue a computerized version of the phrase: 'Pardon me?'.

A rule-based production system provides a means to add words arbitrarily to the vocabulary and to make use

of context information (e.g. in a telephone dialing system the second digit of the area code is always a '1' or a 'Ø'). These rules are in the form of if-then propositions. The rule antecedents (the left hand side or LHS) are the 'if' portion inquiring as to the presence or absence of features. The rule consequent (the right hand side or RHS) is the 'then' portion which is asserted as fact when all of the antecedents are evaluated to 'true', or discarded when an antecedent is proven false. Rules can be used to help cope with noise. Knowing whether the environment is noisy will indicate if weaker features may be obscured so that certainty factors can be set accordingly to compensate for this.

An important collateral goal is the development of an object-oriented endpoint placement algorithm. This seemingly innocuous task is crucial to any pattern matching operation.

An object-oriented paradigm is a natural application for a Lisp-based workstation such as the 'Explorer' from Texas Instruments. The Explorer's architecture embraces a form of parallel processing called pipelining. The numerically intensive portions of early processing may be performed on the next data set while the shape primitives of the current word are being scrutinized on the Lisp side of the Explorer.

Real-time applications are possible if these aspects of computational efficiency are properly exploited.

Summary

The overall patterns of speech parameters that are common from speaker to speaker in a given utterance are readily discernible to an observer. In the next chapter, issues involving automated recognition of these patterns are discussed.

A crucial issue in any recognition process is the proper placement of endpoints and noise mitigation. Chapter III presents an endpoint placement algorithm that is matched to the classification paradigm.

The details of data acquisition and low level processing are described in Chapter IV. This is to provide the reader some insight into the nature of the speech samples and how well they may be expected to represent the speech patterns of the general population.

A rule-based production system can be used to codify the process by which a human given the shape characteristics of each word in the vocabulary can distinguish the identity of the unknown word. Chapter V deals with how this process may then be implemented on a computer for automatic recognition.

The overall results and recommendations for future research are in Chapter VI.

CHAPTER II

RECOGNITION CONSIDERATIONS

Feature Selection by Region

The Sambur and Rabiner system [Sambur (1975)] divided the utterance into three broad regions. The processing of these regions is depicted by the flowchart in Figure 1. Although this could be easily implemented, given the endpoints, the utterance would more naturally be segmented into parts delimited by 'events' (e.g. sudden changes in magnitude levels) for the object-oriented approach as shown in Figure 2 (For this plot and the subsequent ASCII plots, 'A' is the symbol used on the ZCR plot, 'B' is for average magnitude. The abscissa composed of 1Ø msec frames, the ordinate is always scaled to integers from Ø to 7Ø.). Events are also natural boundaries for shape primitives. Another motivation to divide the word into discrete parts is that search techniques for finding primitives and events can be made more efficient if the search can be confined to specific regions. In this section two strategies for regionalization are presented.

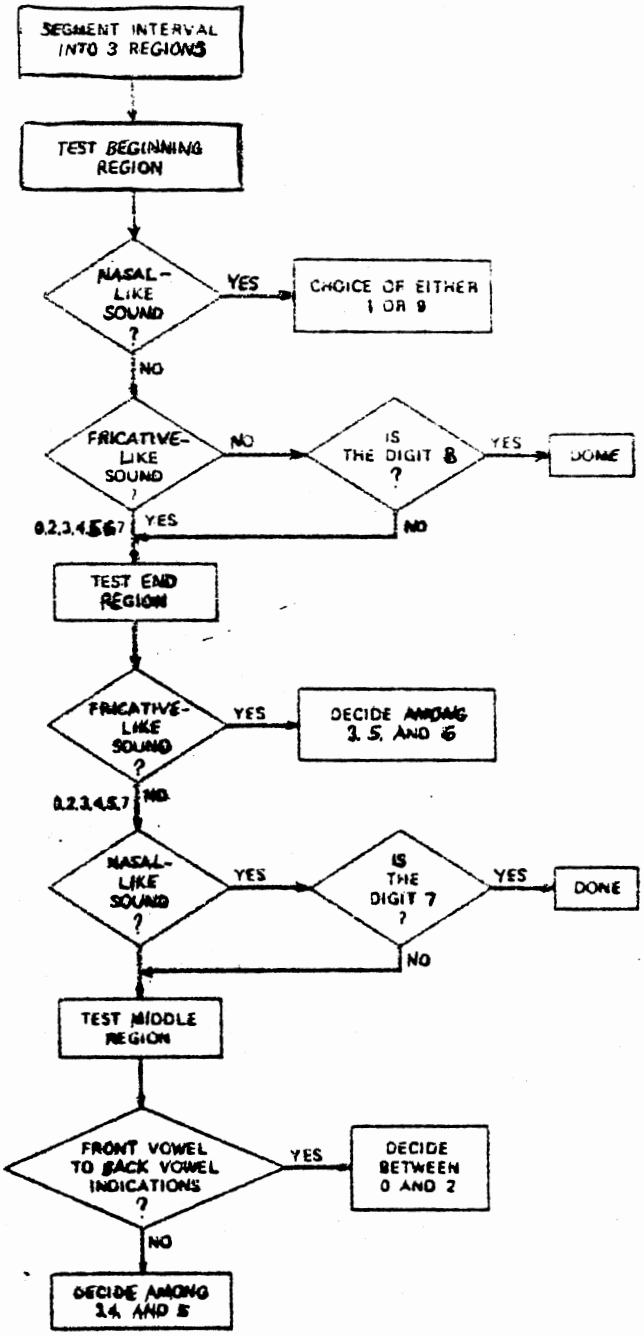


Figure 1. Recognition Flowchart (Sambur 1975)

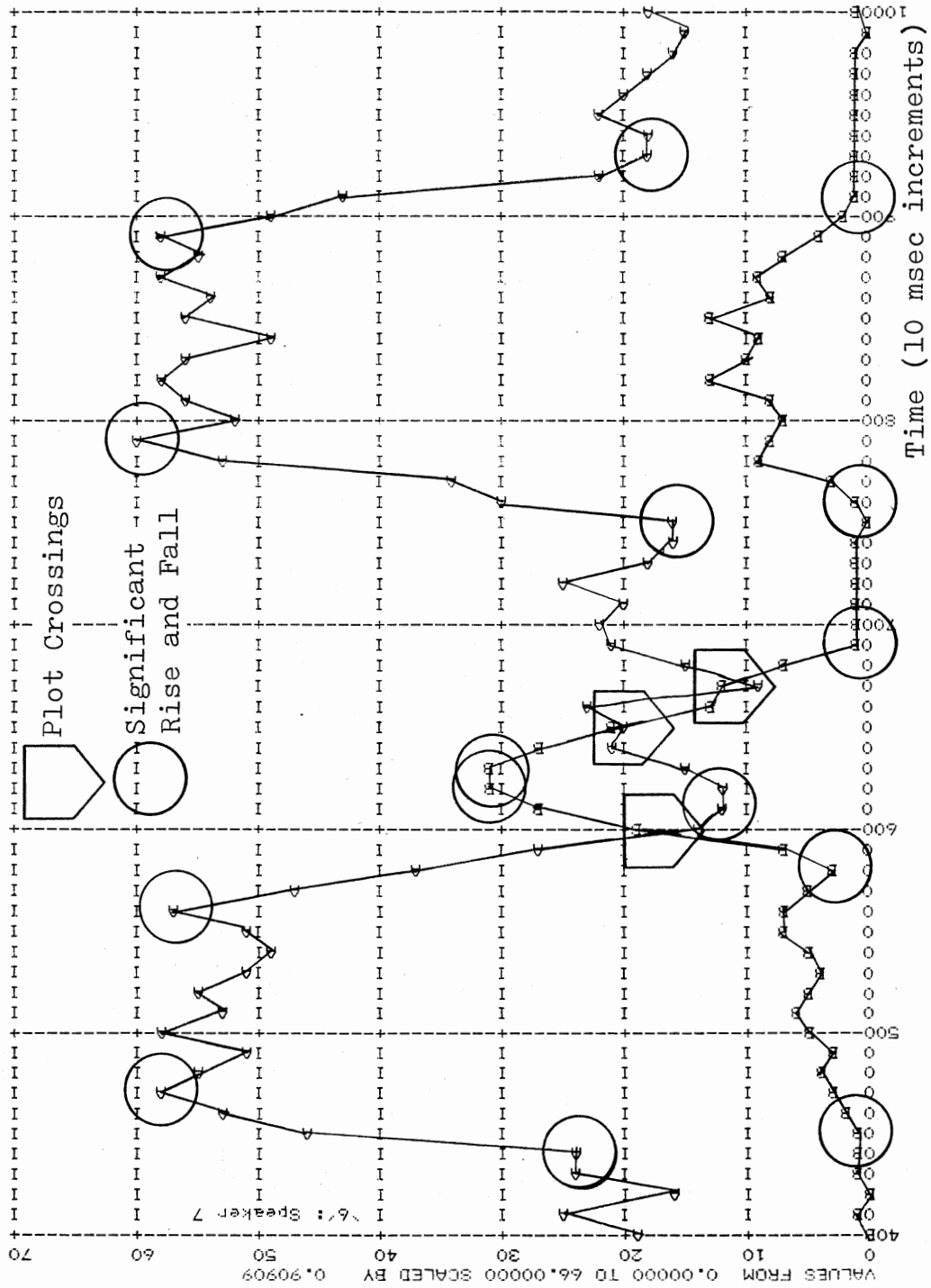


Figure 2. Frame Plot of '6' With Events Highlighted

Variable Region Size

Breaking the unknown word into variable size segments delimited by major events has the possible classification and computational advantages of using the minimum number of areas to be searched for shape primitives and a rough class indicator by virtue of the number of discrete regions detected.

The definition of fuzzy terms such as 'mesa', 'mound' (Figure 3), 'spike'(Figure 4), and 'speedbumps' (Figure 5) is made easier by the relative duration of the subinterval to the whole word and other regions. Locations of regions could be further defined by the location of the centroid of the shape in the region. The actual size of the regions can be described by visual rules-of-thumb to make sure that an individual shape primitive is encompassed by a single region.

Static Region Size

The major advantage of using static region sizes is that it facilitates the definition and construction of the original regions of 'front', 'back', and 'middle'. This allows a rough comparison of the heuristics used in each region for making an identification. The test frame plots of the vocabulary can then be examined to find where regional delimiters may be reasonably placed.

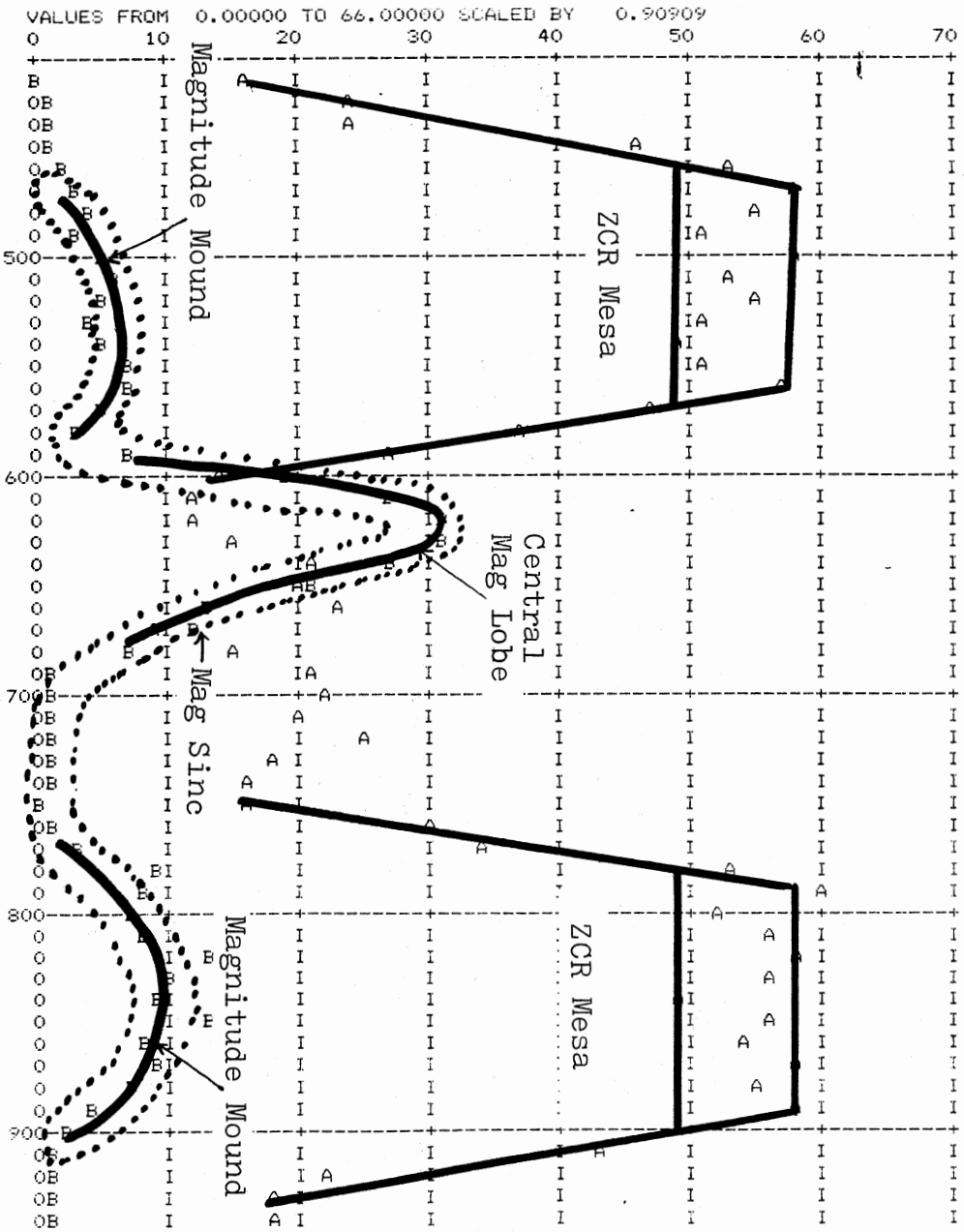


Figure 3. Utterance With Features Emphasized

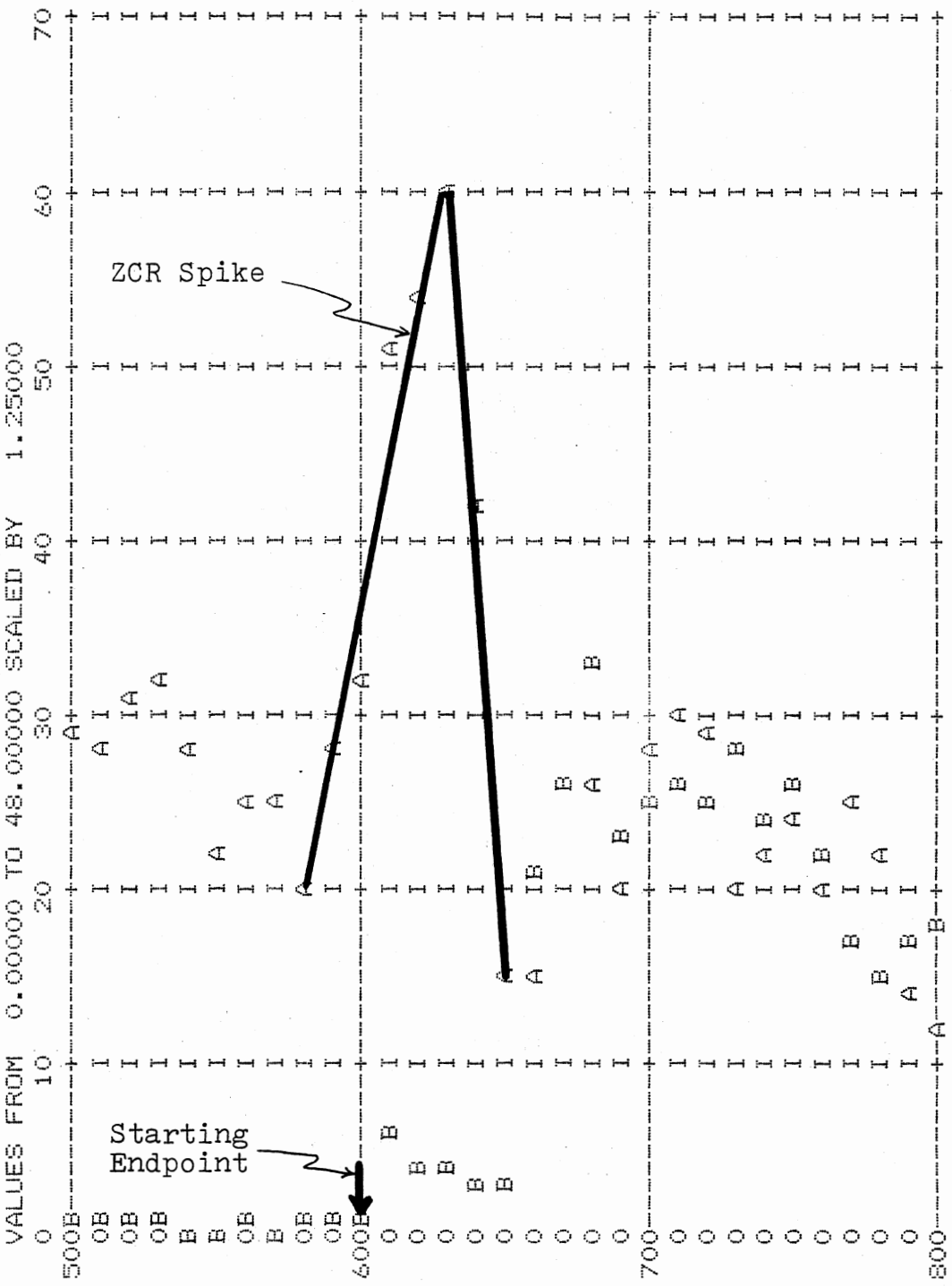


Figure 4. Plosive ZCR Spike From '2'

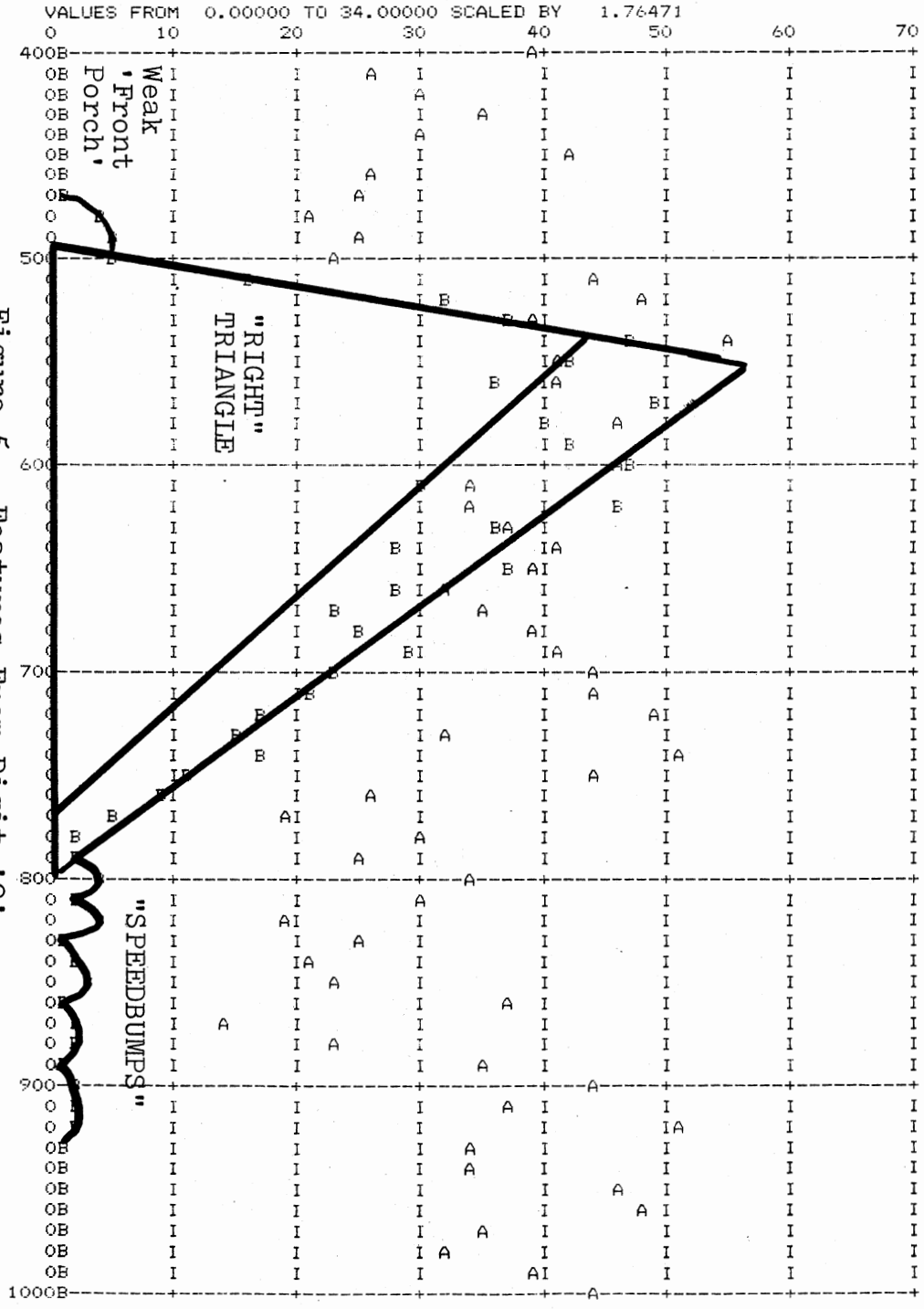


Figure 5. Features From Digit '9'

The scheme can be given an added measure of flexibility by concatenating subregions into larger search spaces, thus enabling it to capitalize on the advantages of a variable region size strategy. In a system using a larger vocabulary, the need for resolution means that more regions are required and will affect the search strategy accordingly.

In the current form, the strategy employed is that of the static region size. Other applications or programming considerations may dictate the use of variable sized regions.

Virtual Object Construction

Another avenue to gain visual insight into how the ZCR and average magnitude varied with respect to each other is to construct virtual objects which can be overlaid for direct comparison. The technique is to map the abscissa of normalized plot to the origin of an x-y plot and then rotate around this point with each ordinate of the normalized plot considered a radius. The steps in producing the virtual objects are:

1) Divide the ZCR and average magnitude frames of the sample into a predetermined number of intervals between the endpoints (ten 1Ø msec intervals in the case of the example in Figure 6).

2) Find the arithmetic mean over each interval.

3) Normalize each based on its largest component.

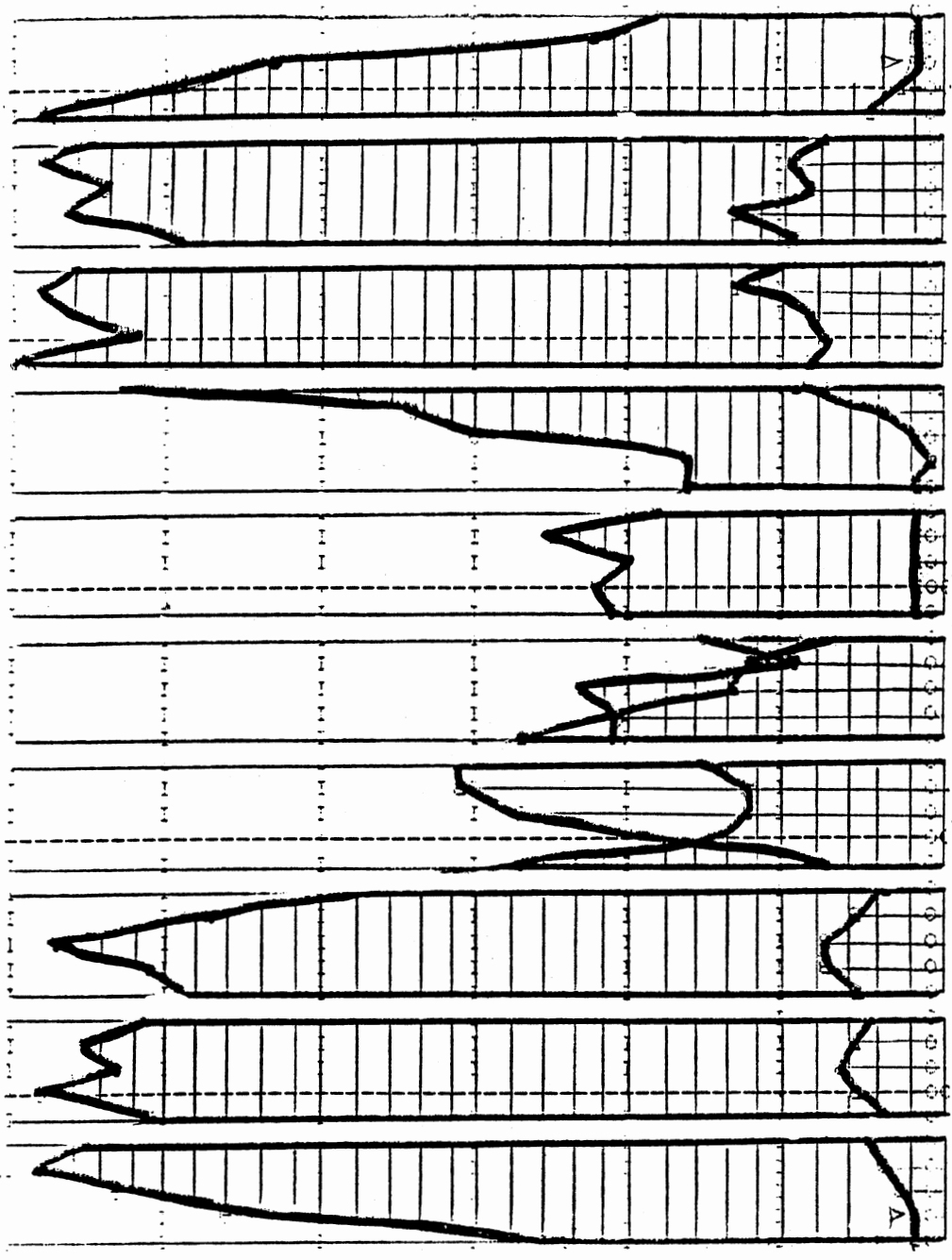


Figure 6. Digit '6' Divided Into Ten Intervals

4) Construct an x-y plot using each as a magnitude from the origin spaced 360/(number of intervals) degrees apart (See Figure 7).

5) Compute the centroid of each object.

The idea here is to produce two shapes that can be overlaid and compared directly for visual clues as to the identity of the word (Appendix B contains a utility for doing this: xyplotf). The centroids are computed as a discriminant (easily obtained but rough) to prune rules from consideration that characteristically have their centroids elsewhere. In addition to the mean, other attributes such as average deviation, standard deviation, variance, skewness, and kurtosis are easily calculated for use as rough shape discriminants in subregions delimited by events.

This method is extremely sensitive to proper endpoint alignment. Figure 8 illustrates how placing the endpoints too loosely may invalidate the normalization procedure. In this case, it may become necessary to invent some weighting scheme whereby the contributions of the frames near the endpoints are minimized accordingly.

Another source of objective trend comparison of these objects would be to use a power of two (16 or 32) as the number of intervals and then perform a Fast Fourier Transform (FFT) or discrete Fourier transform (DFT). As the idea is to check for general trends relative to each function, rather than true

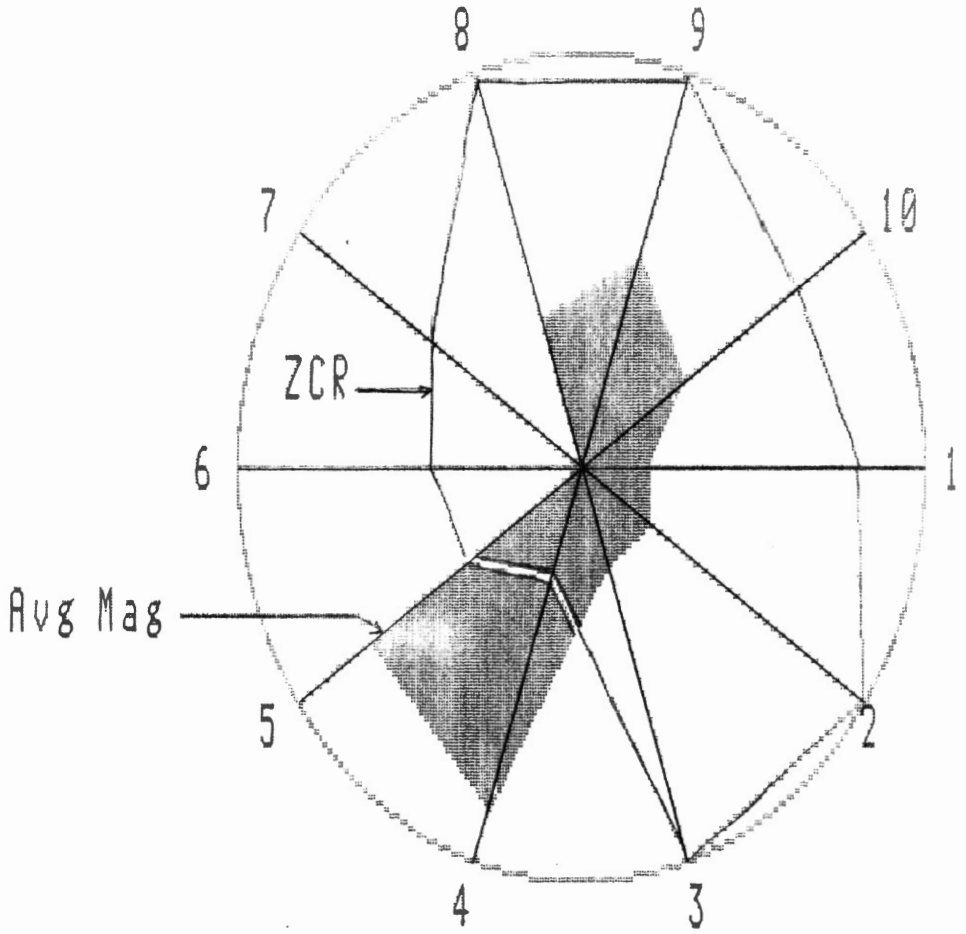


Figure 7. Normalized Virtual Object (ZCR & Magnitude)

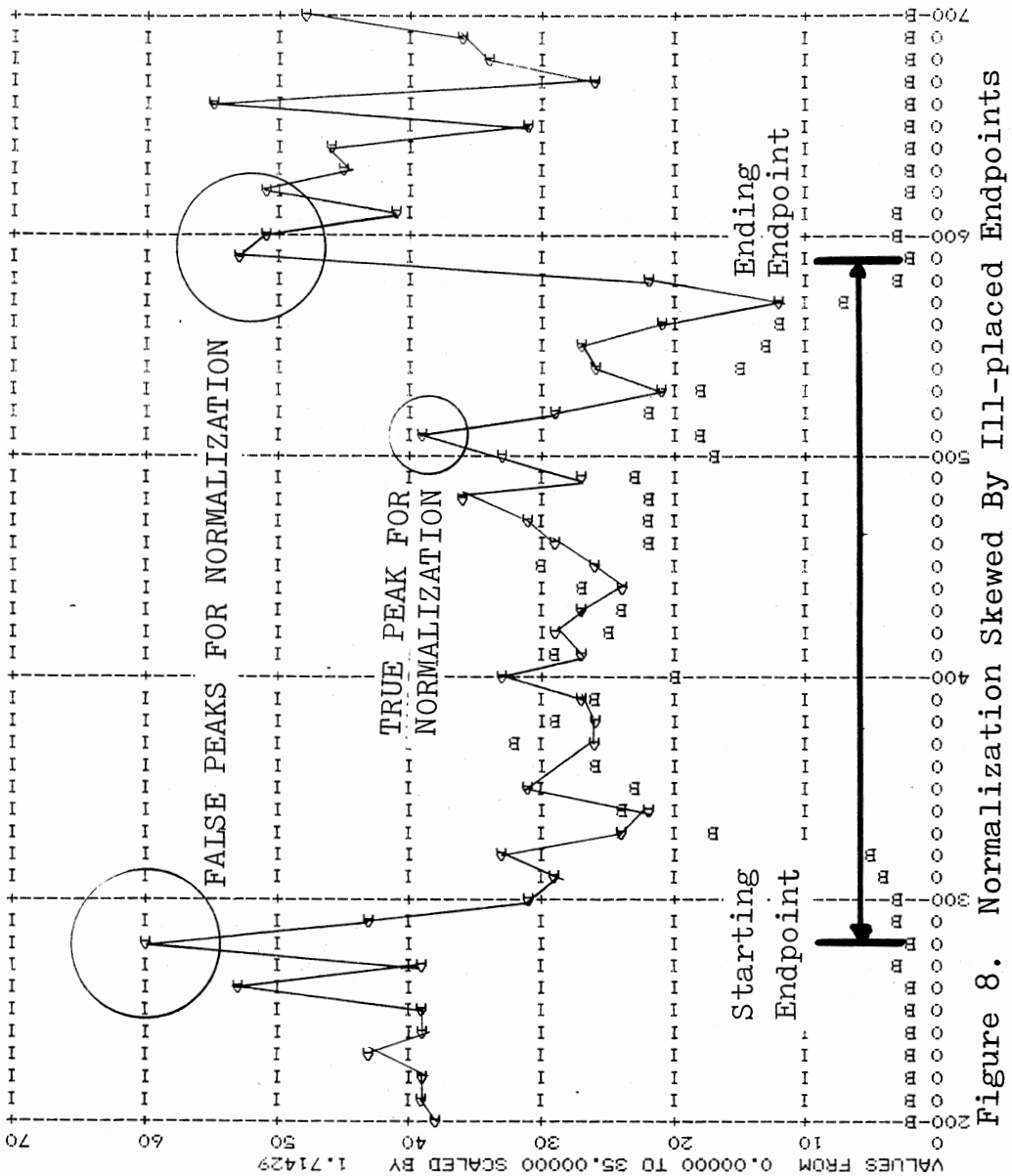


Figure 8. Normalization Skewed By Ill-placed Endpoints

frequencies, effects of aliasing might possibly be ignored.

Shape Primitives Extraction

Figure 9 shows the approximate features present in the English digits 'Ø' through '9' from which the prototype rulebase was developed. These simple shapes are sufficient to make an identification by an informed user. The digits are grouped into four classes based on some common feature. In the first column, the common feature is the basic shape of the average magnitude function in a shape reminiscent of a sinc function. A sharp rise and fall of the ZCR is the common thread of the next class. The sharp rise and fall of the ZCR due to a plosive sound at the end of the utterance places '8' in a class by itself. The shape of the magnitude curve at the top of the fourth column characterizes a trait found in '9', '5', '1', and '4'.

By finding a common feature, the search can be narrowed to just those words possessing this trait. Those words with differing pronunciations due to some form of accent may exhibit properties of more than one class. This difficulty is easily circumvented by coding a new set of rules for each variant which appears in another class. To make the system automatic, the user's judgement (or decision-making)

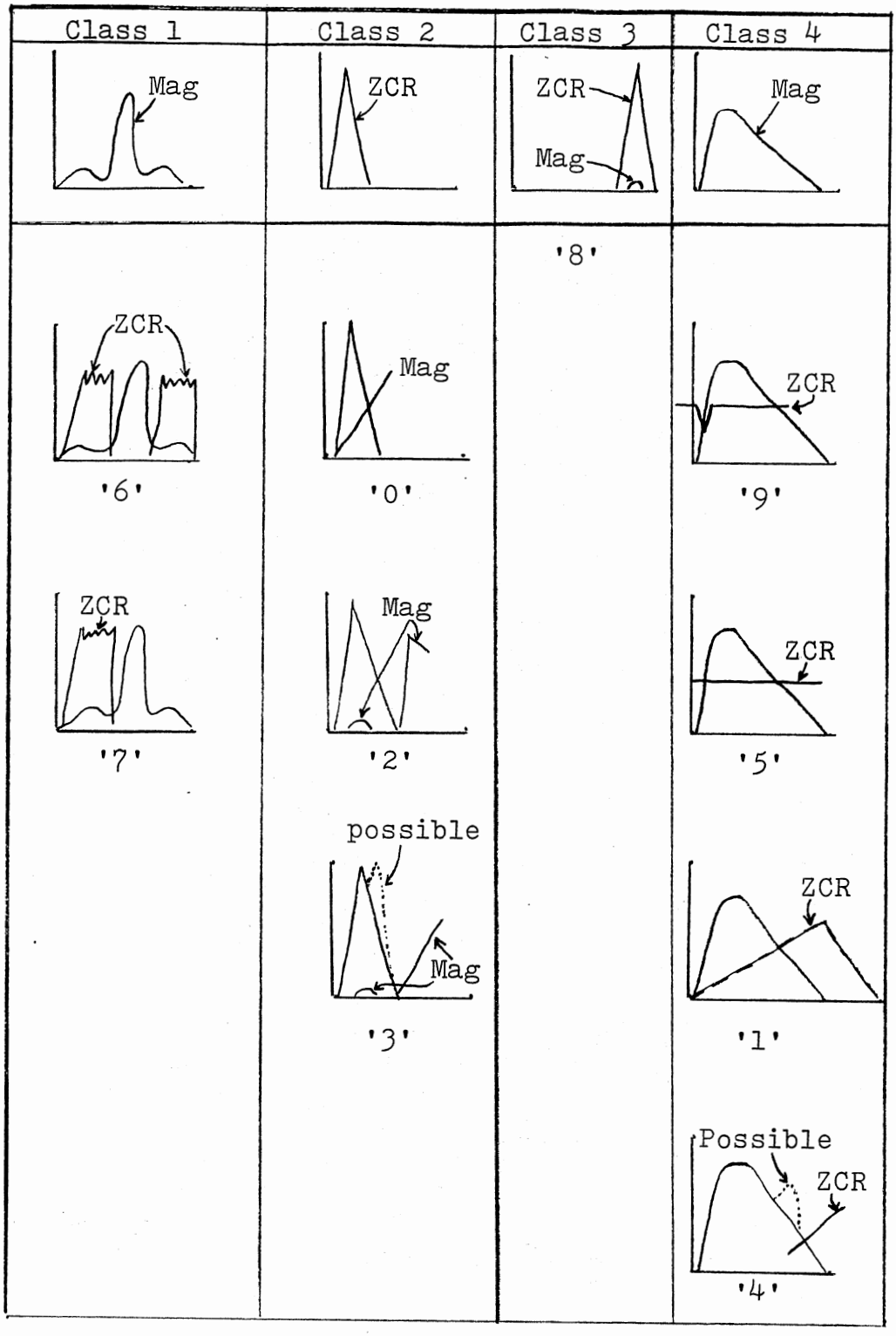


Figure 9. General Features of '0' to '9'

process must be codified to determine the presence or absence of features that the user can distinguish.

Shape primitives are directly analogous to the individual components of an erector set. Figure 3 depicts how, from these units, larger shapes may then be constructed (e.g. a 'mag sinc' from two 'mag mounds' separated by the main 'mag lobe'). The relative sizes and positions of these shapes are what constitute the basis for the visual clues perceived by a human observer that will eventually be codified as production rules. The synthesis of the larger shapes may be delayed until the actual classification process requires it, thus avoiding the wasting of time constructing shapes that will not be required to make an identification.

In addition to the shape primitives, the frame data between the endpoints and their associated integer plot levels are passed to the inferencing engine. This is typically only a couple of hundred of numbers so it will not slow the overall operation appreciably. Processing of these data is invoked by procedural 'demons'. Lisp allows function calls to be imbedded in the database so that if a piece of information is needed, the first time it is accessed, it runs the function. Subsequent queries access the result. This allows for 'hair-splitting'(when required) and

reduces the number of floating-point operations by comparing integer values whenever possible.

Spline Smoothing

Fitting smoother curves to data points rather than just using straight lines in a connect-the-dots fashion tends to enhance trends to an observer. Cubic and bicubic splines are popular methods for doing this kind of interpolation [Press (1986)]. The same techniques may also be used for extrapolation. The extent to which an extrapolated point matches the actual datum may be used as one way to gauge the degree of membership in a particular set assigned to a shape primitive.

The benefits of spline smoothing (a smoother line for the observer) did not seem to be worth the computational effort because the overall trends were not emphasized to any significant extent.

Median Filtering

A nonlinear tool used to smooth data while preserving sharp discontinuities (as long as the discontinuity exceeds some critical duration) is the median filter [Rabiner (1978)]. The top half of Figure 1Ø (from [Rabiner (1978)]) illustrates this. The bottom half show the effects of increasing filter length. There are algorithms for computing the median

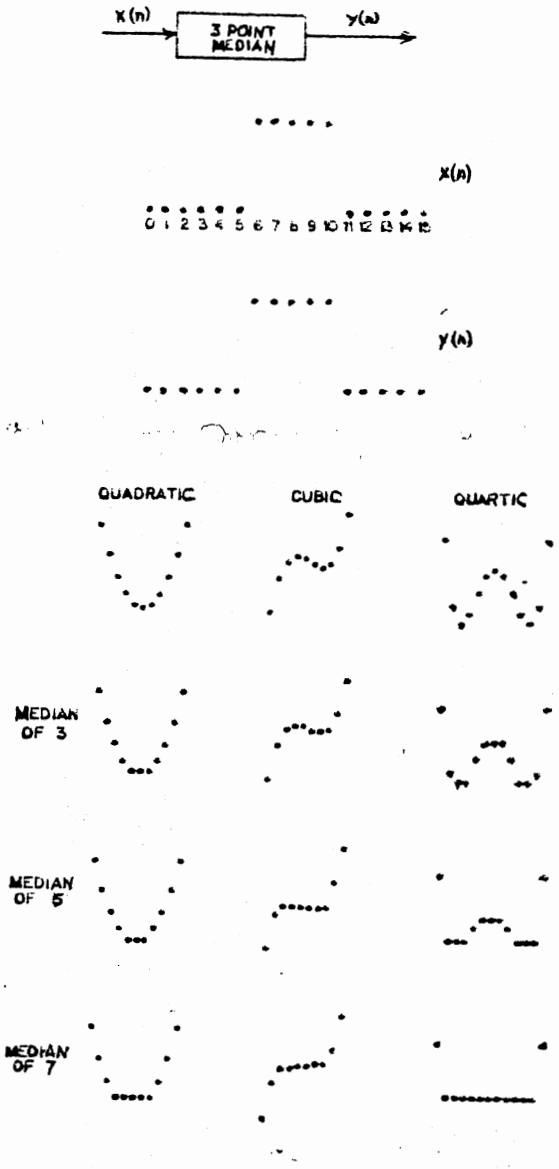
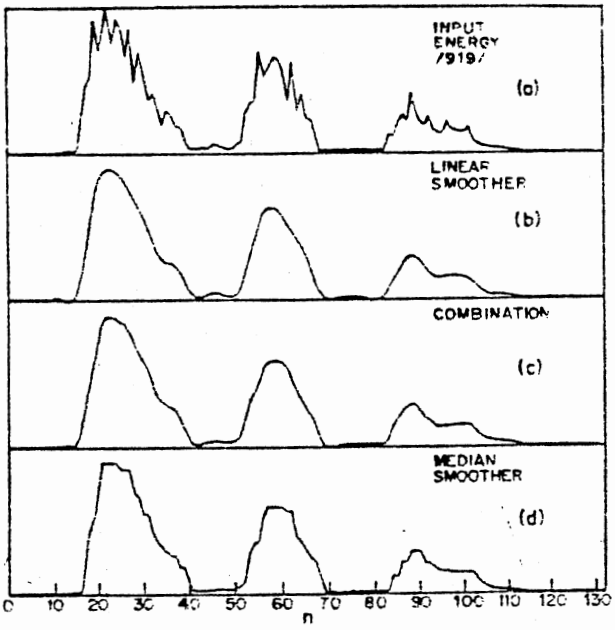


Figure 10. Median Filtering Operations



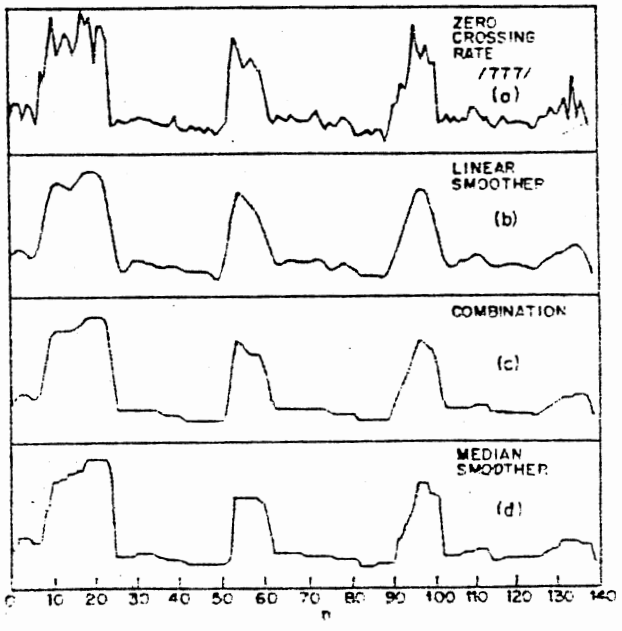


Figure 11. Effects of Median Filtering (from Rabiner 1978)

efficiently [Press (1986)], but a point of diminishing return is soon reached. The operation is simply to find the median of n data points, where n is the length of the filter. If an odd length filter is used, the middle of the filter is unambiguously defined. The result of this operation will be one of the original plot points rather than creating a new point as with an arithmetic mean. This has the advantage of being able to work strictly with integers (i.e. plot levels) and avoid floating-point operations.

Median filtering is a means by which the descriptors 'mesa' (a high, relatively flat shape) and 'spike' (steep rise and fall of ZCR or magnitude to a relatively high level in a short time) may be more strictly defined. The median will 'flatten' the mesa (while preserving the steep sides) while the spike will be clipped. These operations are carried out on copies of the data so that such techniques may be applied without corrupting the original data.

Figure 12 shows the effect of a median filter of length 3. Each new nth data point is the median of x(n-1), x(n), and x(n+1). Figure 11 illustrates the application of median filtering (length 5) and linear smoothing (3 point Hanning window).

The smoothed functions may then be used for comparison against more strictly defined shapes with the scored pattern matching algorithm described below.

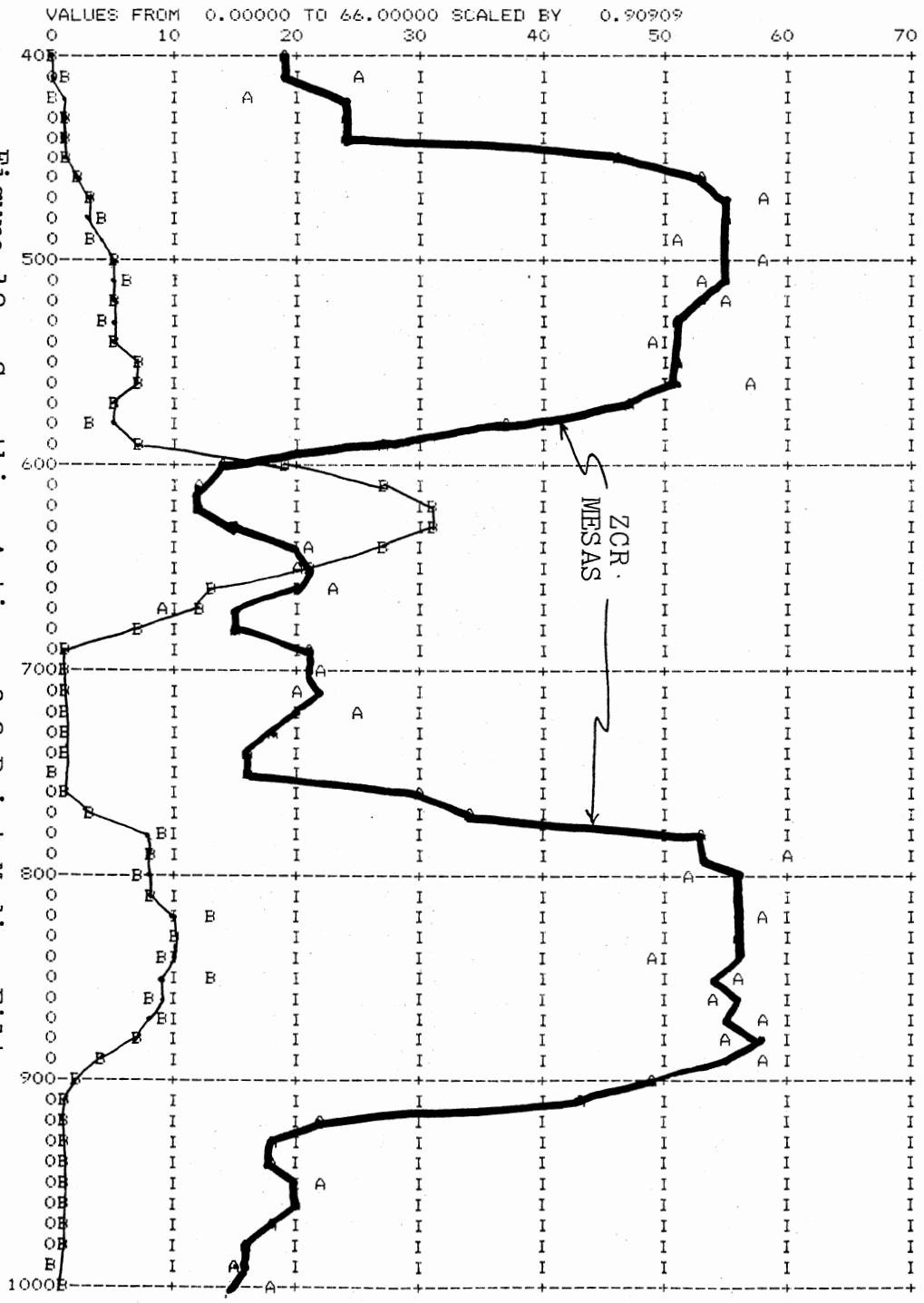


Figure 12. Smoothing Action of 3-Point Median Filter

Scored Pattern Matching

A quantitative measure of similarity may be obtained by using a hit-or-miss scoring system (Figure 13 shows this method in use with events as regional delimiters). For a scheme not keyed on events (e.g. static region size) each datum is compared with the reference primitives data set. If the datum falls within a predetermined distance, the occurrence is termed a 'hit'. The sequence is searched to find the longest string of consecutive hits. From this point the search is conducted to the right and to the left. The length of the greatest number of consecutive hits is the upper bound of how far to search for hits in the left and right segments of each. The similarity score is then two times the number of hits divided by the combined length of the two segments compared (this quantity may be multiplied by 100 to make it a percentage). The highest score wins. The entire reference set need not be searched if the similarity score exceeds some predetermined threshold (i.e. it's 'close enough').

This extraction procedure is attractive because it is insensitive to small phase distortions, provides a numeric quantity for fuzzy logic implementation, and is computationally efficient because of the self-pruning nature of the search.

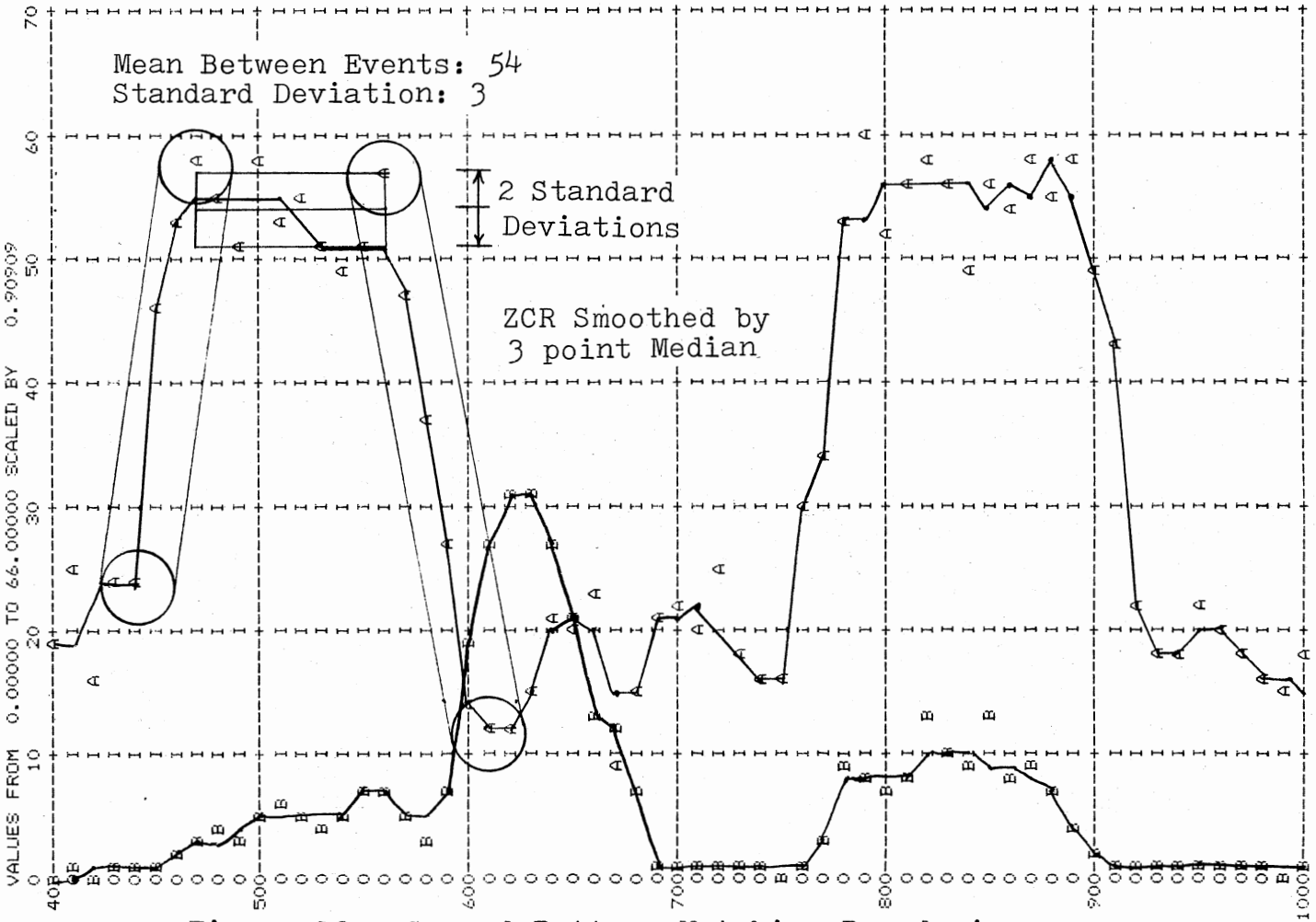


Figure 13. Scored Pattern Matching Boundaries

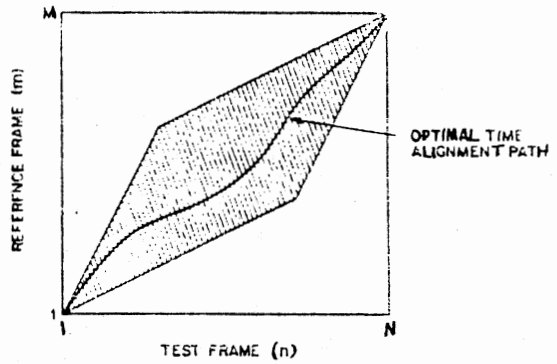
Dynamic Time Warping

Another method of matching the reference primitives set to regions of the unknown word is that of dynamic time warping (DTW) [Brown (1982), Rabiner (1978)]. Time warping is the process of warping the time scale of the reference so that the reference contour lines up with corresponding portions of the unknown word's contour (Figure 14). Boundary conditions force the endpoints to line up and a continuity constraint makes finding the optimal warping function relatively straightforward [Itakura (1975)].

The main advantage of DTW is that reliable time alignment between the reference and the unknown segment can be obtained to handle small phase distortions. The primary disadvantage is the computational expense of finding the optimal path. This expense necessitates limiting the alignment process to small, specific regions.

Summary

The advantages and disadvantages of static and variable region sizes were considered to allow a measure of choice of which to use in the final numeric implementation. Shapes in regions may be characterized by statistical and geometric properties to prune unnecessary rules from consideration and aid in



Region in the (n, m) plane for which a time alignment is calculated in dynamic time warping.

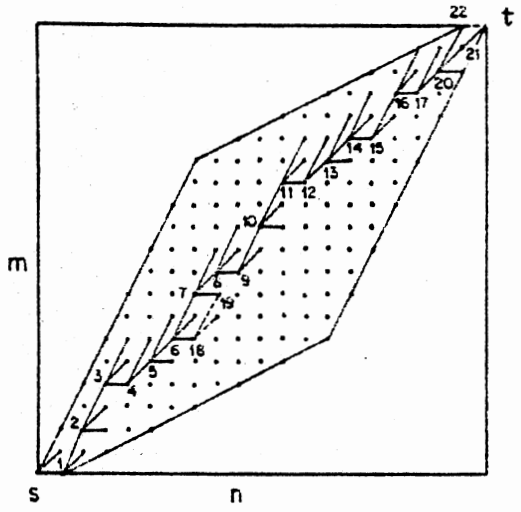


Illustration of the computation to determine a path from node s to node t, using the ordered graph searching concept.

Figure 14. Dynamic Time Warping

primitive feature extraction. Scored pattern matching and dynamic time warping were offered as procedures to quantify how much a given shape fits a pattern.

CHAPTER III

ENDPOINT DETECTION

An aspect of fundamental importance in any recognition scheme is that of endpoint placement. Background noise can mask out weak fricatives and low-level 'rumbling' at the ends (Figure 15). The original paradigm [Rabiner (1975)], shown in Figure 16, fails to place the endpoints appropriately (with respect to those methods proposed herein) in the presence of even small amounts of noise (Figure 17). The endpoint detector was improved later [Lamel (1981)], but had no separate stage for endpoint detection. The decision process was inexorably tied to the recognition process in an after-the-fact manner, thus making it useless for the pipelined, object-oriented approach. Because an observer will place the endpoints differently in a noisy context, one idea is to have two sets of heuristics to determine the bounds of the sample. If the first 100 msec. can be considered to be silence, then it is a simple matter to gather statistics during this interval to decide which method to use. For a specific system, the knowledge of the recording environment (e.g. tape versus microphone)

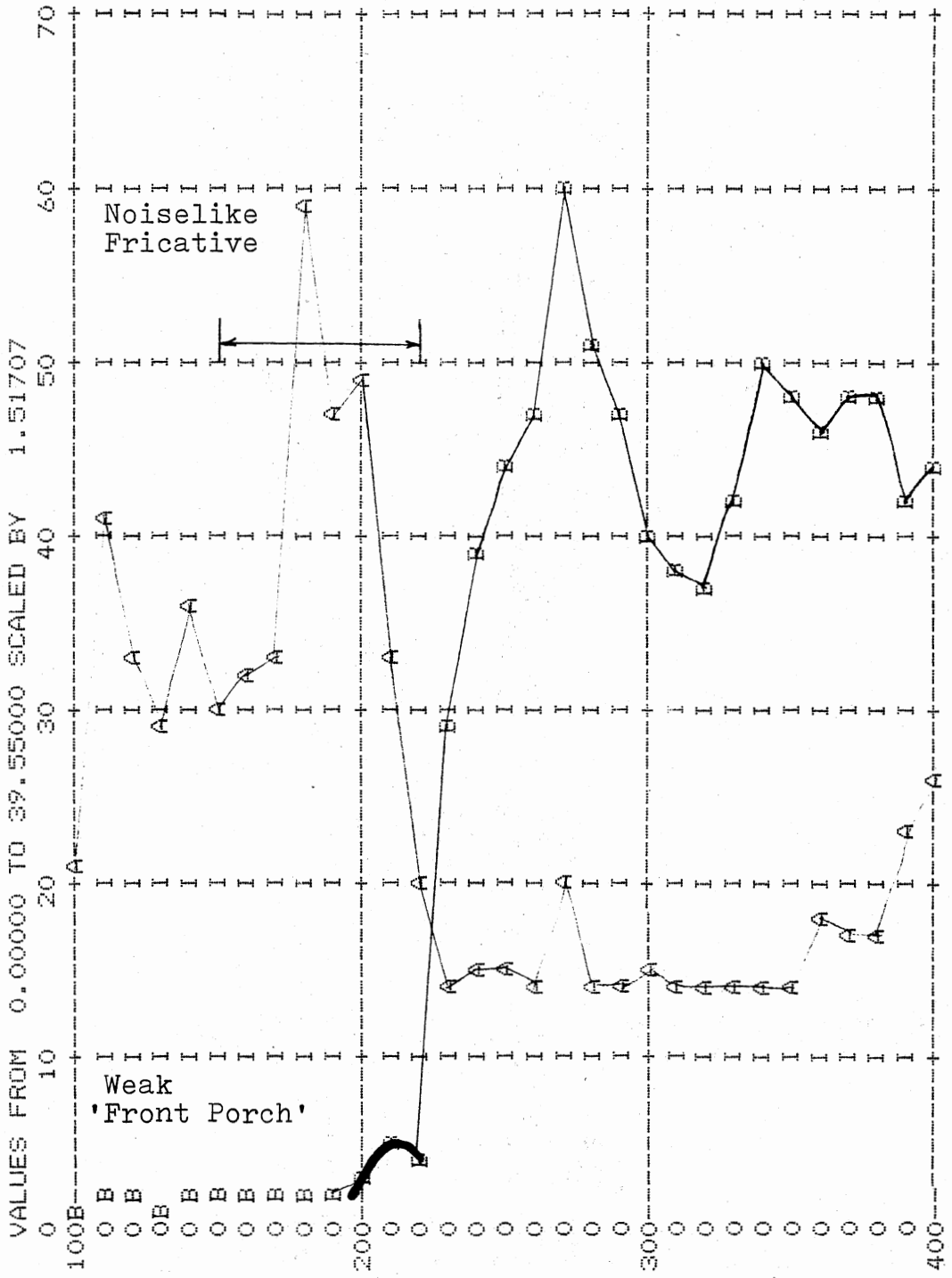


Figure 15. Fricative Beginning '4'

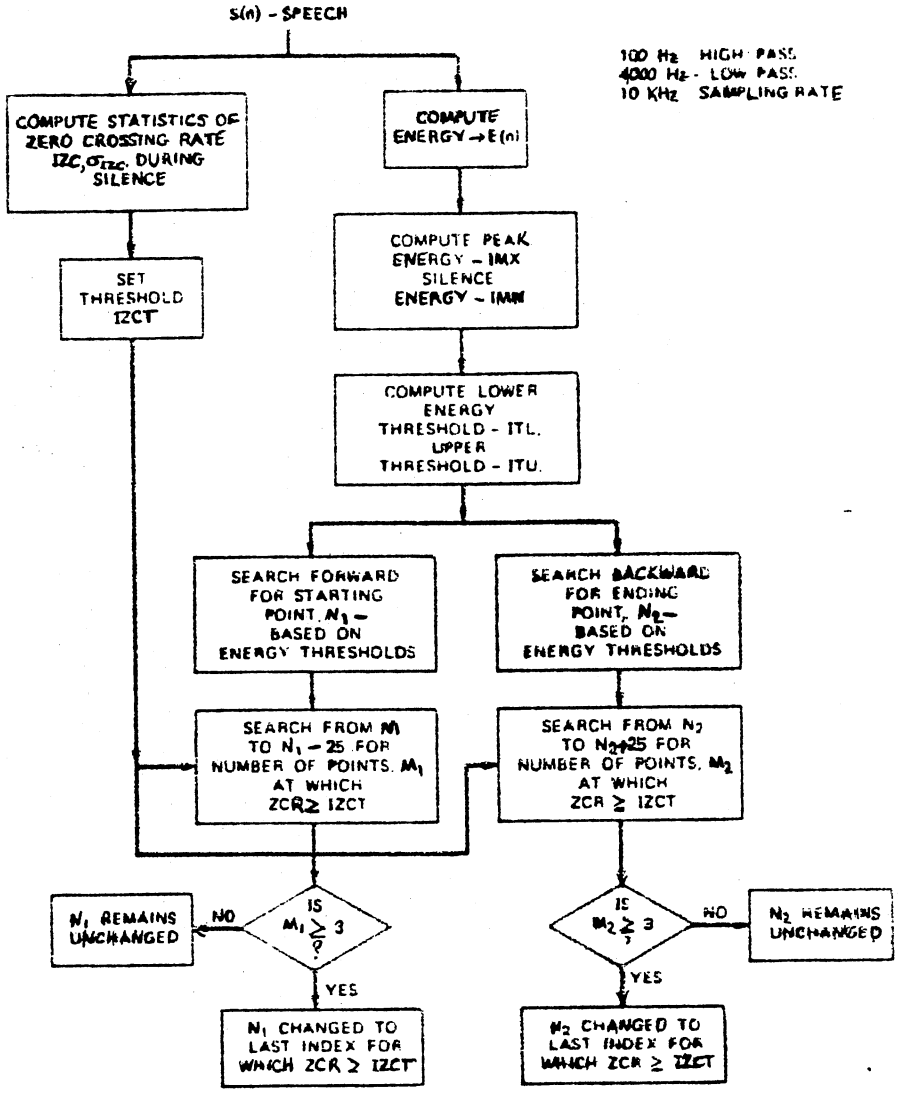


Figure 16. Endpoint Placement Flowchart

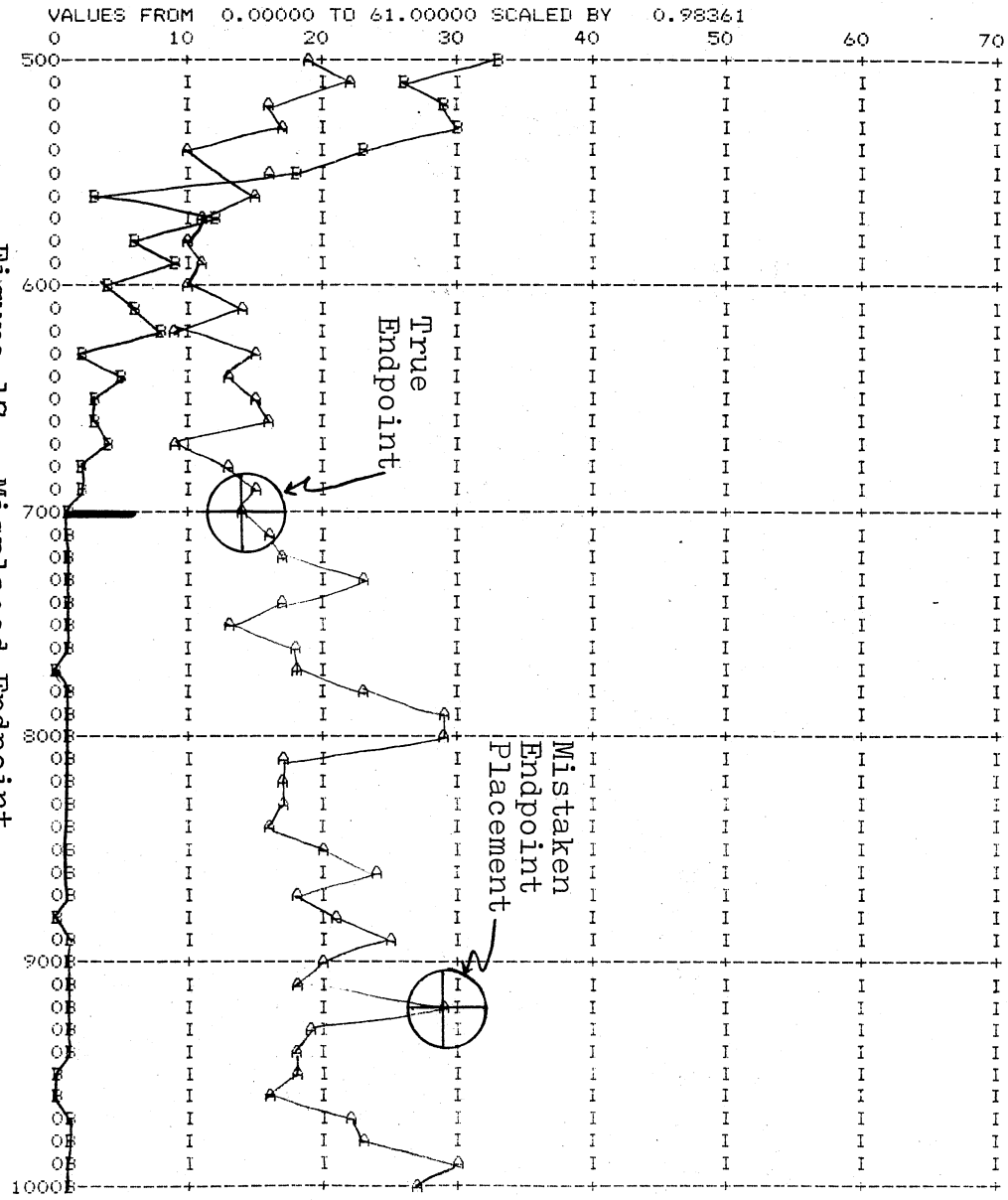


Figure 17. Misplaced Endpoint

and the digitizer (e.g. DC offset and quantization error) would provide a basis to 'fine-tune' the algorithms accordingly.

The endpoint placement scheme in this system is matched to the methodology of the classification process, i.e. visual cues. The frame plots were studied to determine where the endpoints should be placed, why they should be there, and especially how these locations were decided. This is not really an extra task as this sort of scrutiny must be performed in the course of developing rules for identification.

A three-stage strategy emerged with rules that could easily be procedurally coded. These stages were dubbed 'initial magnitude zero' (points 1 and 2 of Figure 18), 'bump extension' (point 3 of Figure 18), and 'ZCR extension'. These names were derived from visual descriptions of phenomena peculiar to the plotting procedure (source code listing in Appendix B).

The sheer number of plots to be produced dictated that the plotting routine be fast and easy to implement. The plot is constructed from the standard ASCII character set so no special graphics mode is required to send the output to any printer. This produces a coarse graph quickly at the expense of connecting lines between the data points. The plotting routine also has a self-scaling option so that all of the plots appear at the same relative positions despite

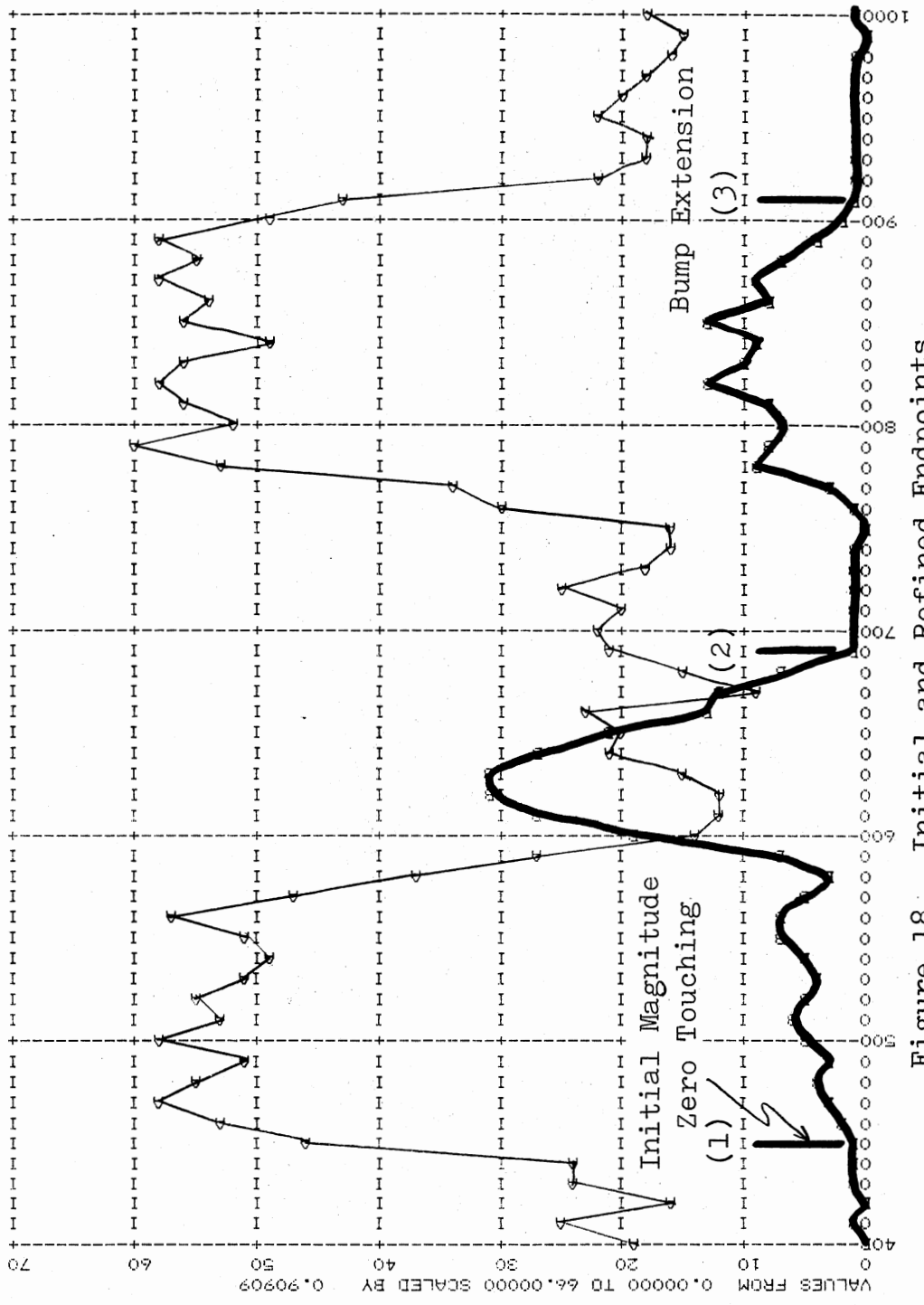


Figure 18. Initial and Refined Endpoints

moderate changes in background noise and loudness of the speech. The frame array starts at index 1 so that the lower limit of the graph may be consistently set to zero by setting the array element at index Ø to Ø.Ø. The upper limit will then be the largest value of the ZCR and average magnitude frames. Knowing that the lower limit is always Ø.Ø means that the scale is based directly on the largest value to be plotted. The granularity of the graph and the associated scaling factor are the key to endpoint placement in this case.

A function called Plotlevel (See Appendix B) uses the scale factor to return an integer corresponding to where a given argument would appear on the ordinate. In this way, rules based on where symbols appear visually on the plot may be coded directly.

The reader should note that this endpoint detection method has not been tested for generality nor robustness at the time of this writing. The following procedure uses limits obtained empirically from the available frame plots and may have to be adjusted to suit another implementation.

The first stage starts the search from the index of the largest average magnitude frame in the interval (frame 62 of Figure 18). Extraneous clicks and noise are eliminated from consideration. The search proceeds forward and backward to the first points where the average magnitude plot level is equal to or less than

the average level of the magnitude in the segment considered to be silence (points 1 and 2 of Figure 18).

The second stage starts by looking to the right of the right index set in the first stage. If the zero level is exceeded more than once for durations greater than one frame (a 'bump') within 11 frames of the initial estimate, the index of where the bump (or bumps) ends is noted. This process is called 'bump extension'. It is used to mark the end of any trailing 'rumbles' in words that 'dip' early (point 3 of Figure 18) or do not terminate crisply (Figure 19).

The last stage (ZCR extension) is used primarily to include noise-like fricatives (unvoiced sounds generated by forcing air through a constriction at the mouth end of the vocal tract fast enough to produce a broad-spectrum noise source) in the region for consideration. The ZCR of the bump extension and the next two ZCR levels are averaged and compared to the mean of the ZCR during the silent segment. The index is incremented until the average of these three points falls within two standard deviations (may be adjusted dynamically in the event of extreme cases) of the ZCR silence mean or the index is incremented six times, whichever occurs first.

There appears to be only a slight improvement in performance if a weighted average is used, with the points lying outward receiving the greater emphasis.

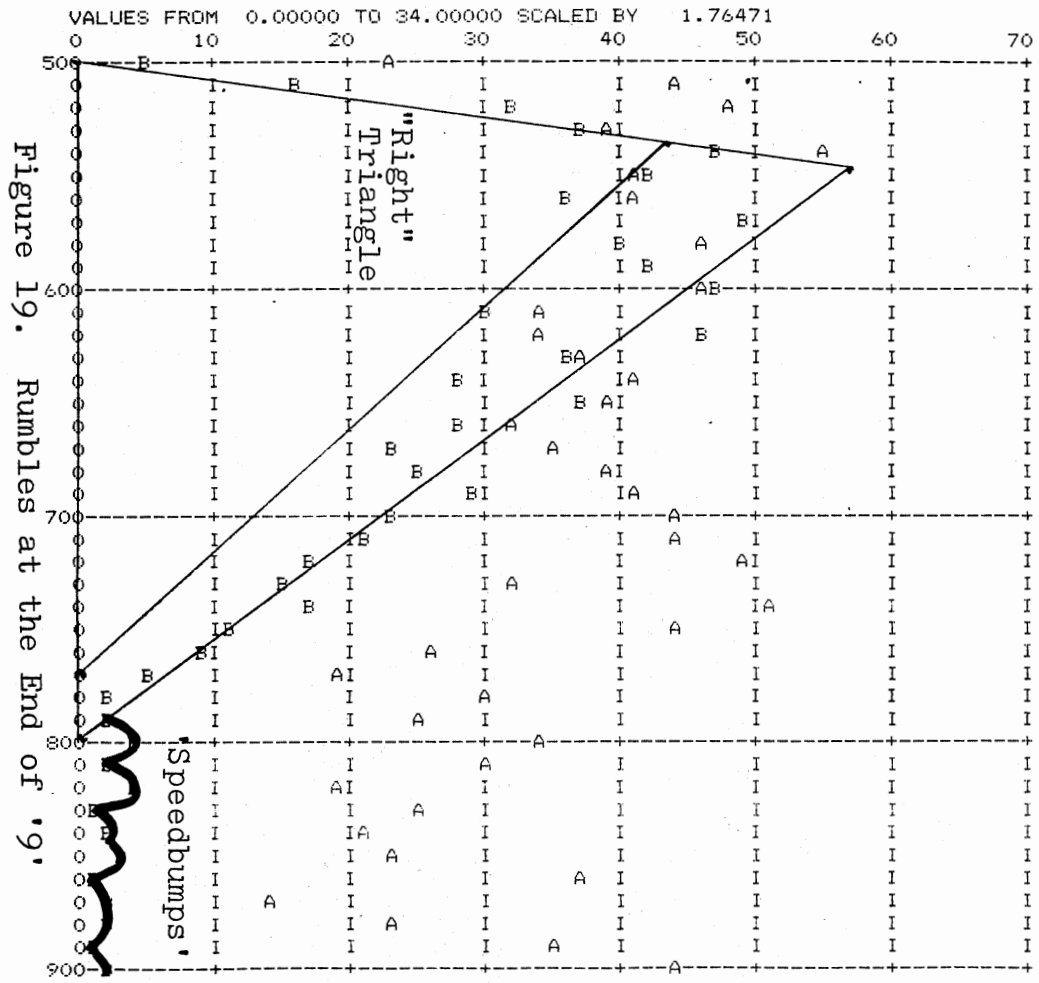


Figure 19. Rumbles at the End of '9'

Testing an operational system is the only reasonable way to determine if this is necessary.

The process is then repeated for the left side with the relative directions reversed. The left side has an additional heuristic in the bump extension part that the zero magnitude level must be exceeded by two levels or have an anomalous ZCR to be extended. This is to account for smaller bumps that may be obscured by noise, but will not let the endpoint be placed too far afield should it be a 'false alarm'.

Figure 2Ø shows the endpoint placed at frame 1ØØ. While the ZCR of the next three frames is a little high, the magnitude there is insufficient to be part of any feature for which the interval will be searched. The bumps on frames 1Ø6 through 113 are to be ignored completely.

Summary

An object oriented approach to the problem of endpoint placement has a major advantage over the Rabiner procedure in this case. This approach corresponds to the inferencing method, allowing the endpoints to be set correctly in terms of what features are considered important to the observer, instead of a cryptic numeric parameter. The potential for noise mitigation makes this methodology particularly attractive.

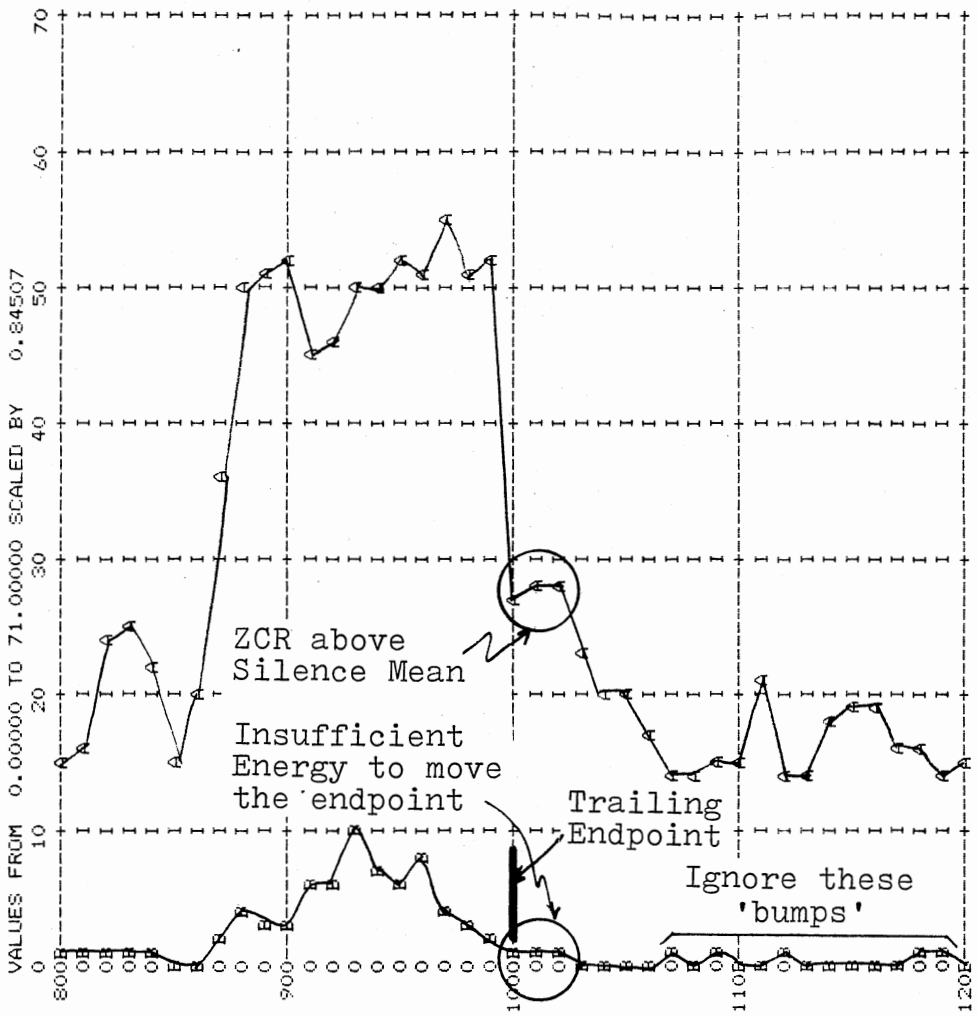


Figure 20. Endpoint Refinement Notes

The source code is provided in Appendix B for easy modification and implementation. These heuristics of endpoint placement may be easily adjusted to be system specific.

CHAPTER IV

DATA ACQUISITION

The test vocabulary chosen was that of the digits 'Ø' through '9' because this was the set used by Rabiner in his original work. This set provides a performance standard with which to gauge how well a particular method works and lends itself more readily to practical application.

The underlying assumptions in each case are that the first 1ØØ milliseconds of the raw data is to be considered silence (from which statistics characterizing the quiet portion are extracted) and that the utterances be separated by at least 25Ø milliseconds of silence. All words are presumed to be contained in a two second segment. These restrictions may be loosened somewhat if it is further assumed that lip-smacking, inhalations, and other extraneous noise may be handled by noting anomalous duration or intensity during the classification process.

The speech samples were collected under less than ideal circumstances to simulate 'real world' conditions. The utterances were recorded on a hand-held cassette recorder using the built-in

microphone at a social gathering. These recordings were then digitized at 8 KHz to simulate the quality one would expect from a voice grade telephone line.

The eight-bit digitizer and sound editor employed was 'Perfect Sound' from Sunrize Industries attached to a Commodore Amiga. The digitizer had a slight DC offset (obtained by noting output levels with no input applied) which was subtracted from the magnitude when the raw data file was read into an array. Another recognition scheme [Lau (1985)] using only ZCR and energy (but speaker trained) ignored the DC offset by using a biased zero-crossing rate (BZCR). The bias point was selected to be just above the system noise level.

The raw data were then transferred to MS-DOS formatted diskettes for processing on an IBM PC/XT compatible. During the transfer, several of the files were inadvertently saved as Interchange File Format (IFF) files. The difference between a raw data dump and a IFF file is that the IFF file is prefaced with a header containing information on sampling rate, size of file, and so forth. This information is placed in the position of the file from which silence statistics are gathered and may be erroneously interpreted as speech data. To prevent this from skewing the silence characterization, the first 10 milliseconds was ignored leaving the subsequent 100 milliseconds to be silence.

The choice of the width of the frame (1Ø msec.) is a common choice in speech processing stemming from a compromise between the pitch period of a child or high-pitched female (about 2 msec.) and the pitch period of a deep-voiced male (about 25 msec.). There were no female utterances in the test sample. This means that one can expect the speech characteristics not to vary dramatically within the frame period and that the short-time speech measures will track events faithfully.

The ten-millisecond frames were rectangularly windowed and not overlapped to save computation time. It has been reported [Lau (1985)] that no significant increase in successful recognition was gained by overlapping. This is reasonable when considered in the light of the intended identification process. The classification is based on general trends perceived visually, so more accurate 'averages' are not critical.

The subjects doing the talking were Eta Kappa Nu pledges on the occasion of their initiation. At the outset, it seemed to be a natural source of unpaid volunteers with various accents willing to do precisely what was asked of them. The problem with this was that although the situation was convenient, it was also very tense. One apprehensive pledge actually miscounted. The constant suspicion of a trap gave rise to a rate of delivery that was usually too deliberate or too fast.

The cold weather was also a factor in producing mumbling and 'clicks'. A heater fan in the background afforded the opportunity to record samples heavily corrupted by noise.

Summary

The nature of the source of speech data was examined. To provide a more robust solution, it will be necessary to gather new speech samples in the environment of the intended application.

CHAPTER V

EXPLORER IMPLEMENTATION

The following is a program specification for implementing a prototype system on the Explorer II Lisp workstation manufactured by Texas Instruments. The Explorer's architecture (Figure 21) is such that the entire machine operates within a Lisp environment. Within this environment is a separate processor board with a Motorola 68Ø2Ø microprocessor running under a dialect of the UNIX operating system. This setup is well suited to the tasks at hand. The initial number crunching is handled on the 68Ø2Ø side using the C programming language.

C is a very portable procedural language that produces executable code which can be optimized for speed of execution. The output from the 68Ø2Ø side (frames between endpoints, their plotting data, and possibly shape primitives) is then sent to the Lisp side where it is treated as a set of objects and symbols during the inferencing. There is some flexibility concerning how much primitive feature extraction and shape analysis is done on the 68Ø2Ø side. Whichever side happens to be the computational

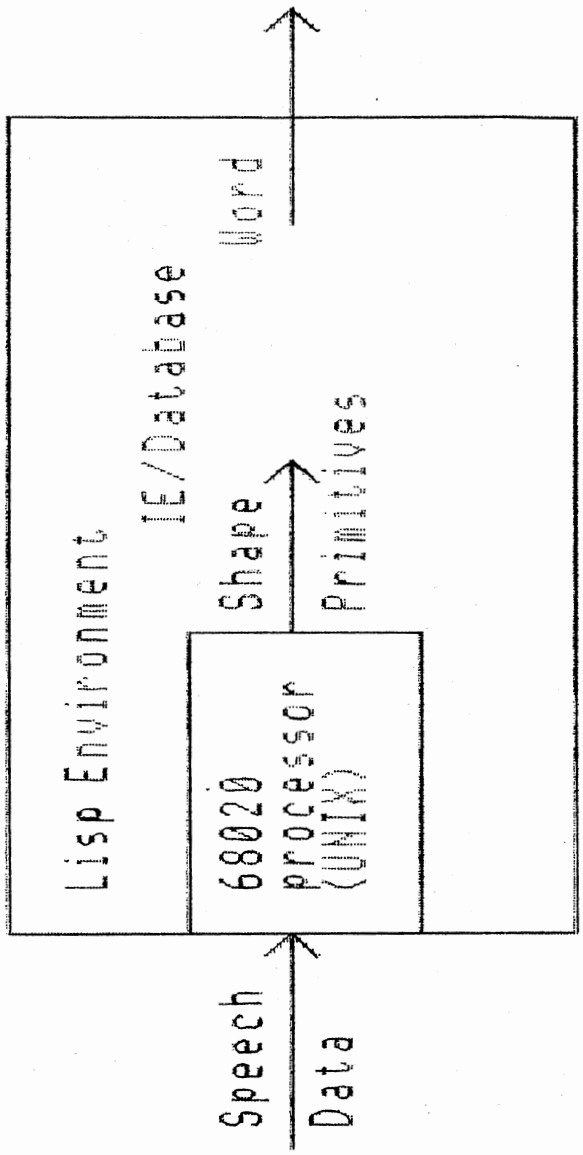


Figure 21. Explorer Environment

bottleneck, the opposite side will be given a greater share of the load until the two are more or less balanced. The ability to shift portions of the burden back and forth will help the performance of the multiprocessor strategy. Ideally, the Lisp side should just be finishing the identification of one word while the 68Ø2Ø is just finishing the initial processing of the next word.

The Explorer comes with built-in stream handling functions to pass data between the two sides. However, if the 68Ø2Ø side produces symbols at a much greater rate than the symbols can be processed, files can be used to queue up symbols awaiting processing.

Coding Primitive Feature Detectors

The key to good resolution in the extraction of primitive features is the proper maintenance of a catalog of reference primitives. The feature detector should return a degree of membership or similarity factor. The similarity factor can be used to control the search when a primitive is ruled 'close enough'. There are only a few basic shapes that have to be fit to the contours of the word to be classified. In the case of two similar shapes that correspond to different words, more primitives will have to be stored in the catalog to provide a greater degree of precision for the similarity factor.

The attributes of the lines and curves comprising the shape primitives are dependent upon such items as slope, relative duration, radius of curvature, and the like. The basic guideline to keep in mind at all times when deciding which attributes to use and with what weighting is how different two items must be before an observer will perceive them as belonging to separate classes.

Expert System Approach

The advantages of using a rule-based production system, or 'expert system', to do the classification are many. It is very natural to express relationships in the form: 'If these preconditions are satisfied, then this is the result'. This method also provides an easy way to handle variations in pronunciation. For example, enunciating 'Ø' as 'zeeero', 'ziro', or 'sero' requires at most the addition of another set of rules for each case. This is not always the case as some identification heuristics do not use that portion which varies from word to word because most of the rest of the word fits the overall pattern.

Rather than perform all of the shape analysis then make a decision, a backward-chaining inferencing paradigm would automatically construct a series of smaller goals it needs to satisfy in order to meet the larger goal of identification. The concept behind this

strategy is to do only the minimum processing necessary to make a decision. There are procedures to enhance the performance in this regard. One way is to realize that the probability of occurrence is basically the same for all of the digits so that the rules may be ordered to perform expensive 'hair-splitting' procedures as seldom as possible [Thompson (1986)]. Another is to put the rule attributes in matrix form [Tschudi (1988)] where the rows are attributes and the columns are the vocabulary for identification (the classic 'plant classification' knowledge base is used for an example in Figure 22). The ternary elements are: '1' if the row attribute must be true for the column candidate, '-1' if it must be false, and 'Ø' if it does not matter to the outcome.

The entropy of the non-zero entries in each column is computed to determine the attributes which will provide the most information to split the candidates into classes. The entropy represents the amount of uncertainty of an outcome, so the split is made on the smallest entropy of classification. The top part of the example in Figure 23 is an illustrative set. The middle part is the result of a split on 'age'. The bottom part is the result of another split on 'competition'.

Membership in a class is found by a similarity measure found by matching the non-zero attributes

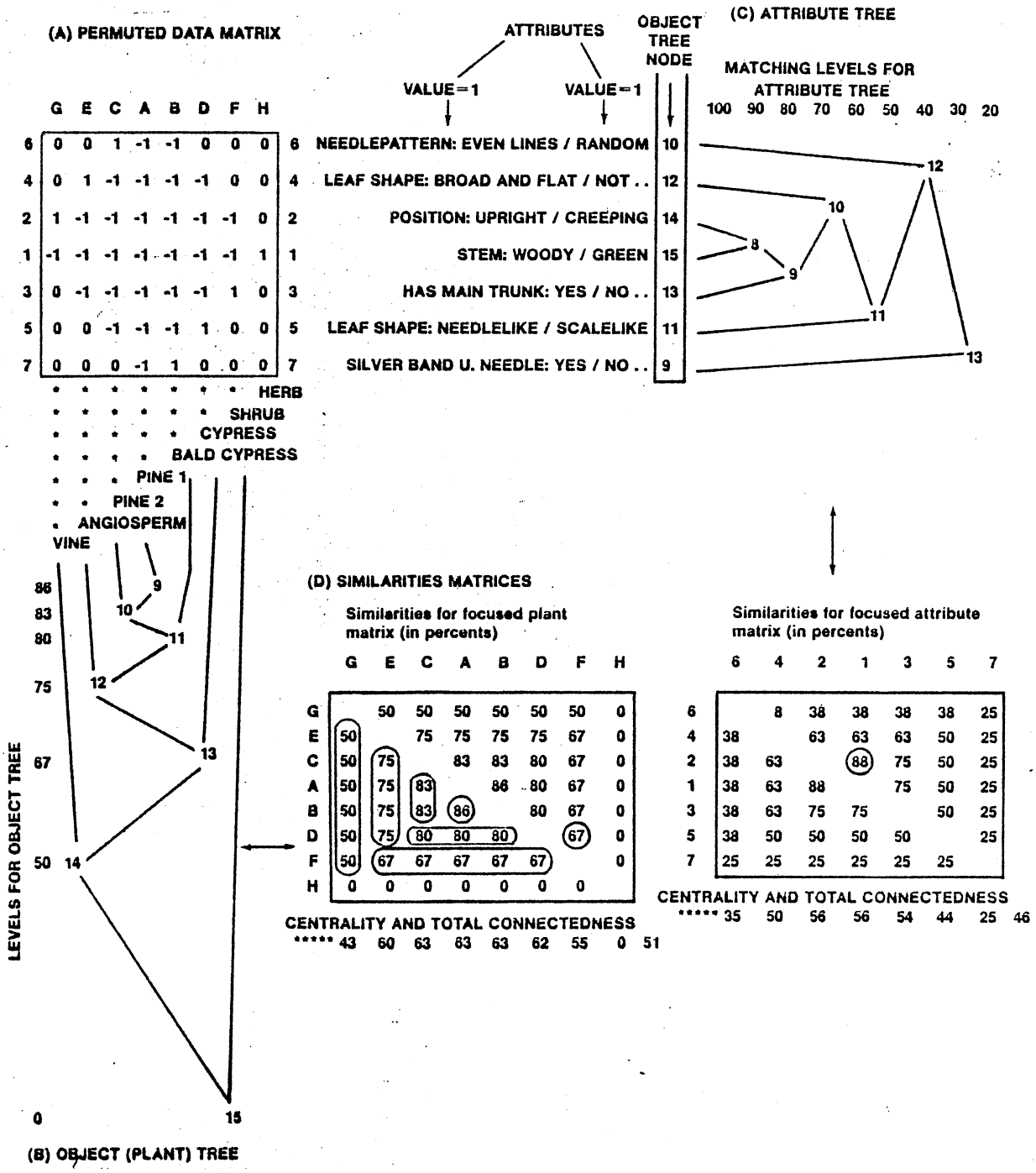


Figure 22. Plant Classification Example (from Tchudi 1988)

Profit	Age	Competition	Type
Down	Old	No	Software
Down	Midlife	Yes	Software
Up	Midlife	No	Hardware
Down	Old	No	Hardware
Up	New	No	Hardware
Up	New	No	Software
Up	Midlife	No	Software
Up	New	Yes	Software
Down	Midlife	Yes	Hardware
Down	Old	Yes	Software

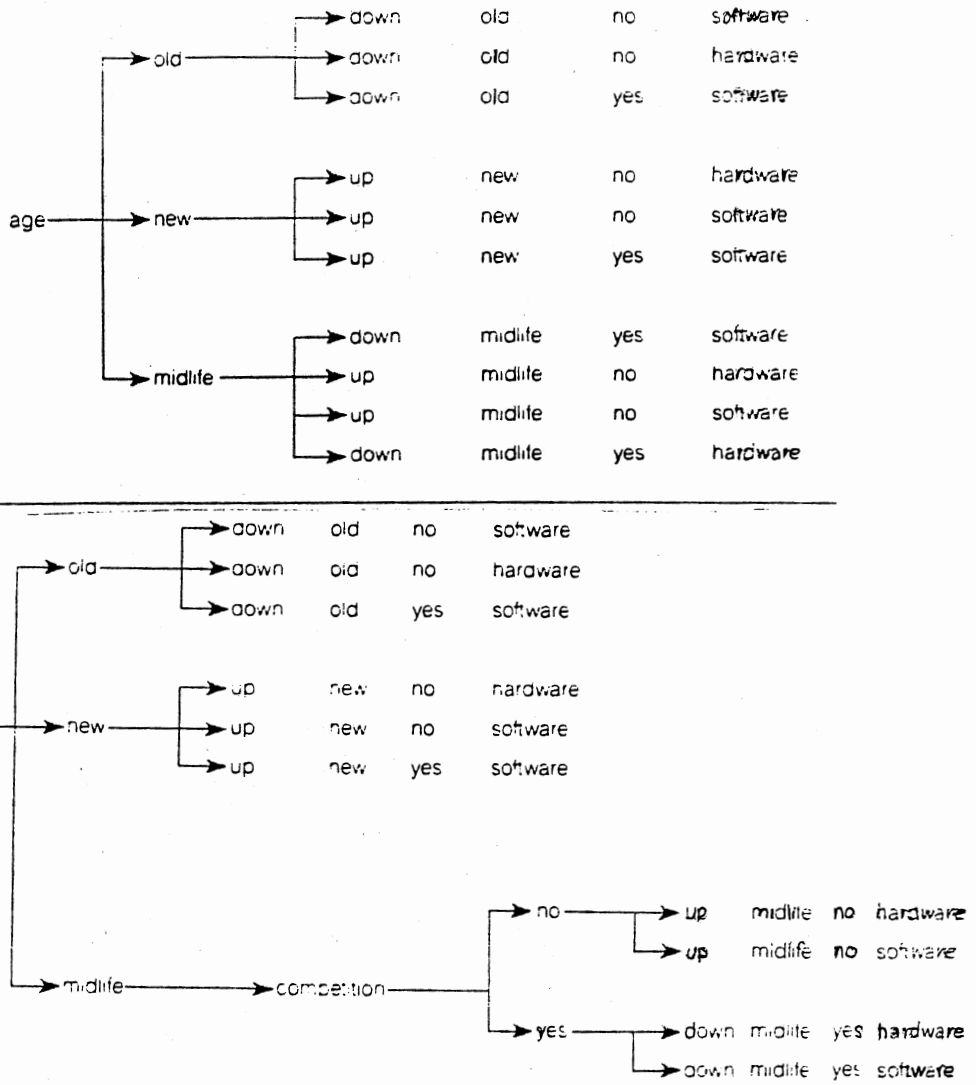


Figure 23. Rule Split by Entropy

between columns and computing a score proportional to the number of exact non-zero matches. The effect is to produce an optimum binary search tree and eliminate unnecessary or redundant computations to evaluate an attribute.

The precedence of rules can be easily enforced by adopting the biological hierarchy of kingdom to species, where kingdom is the first split in the search tree and the species is a word in the vocabulary.

The method of delaying evaluation of an attribute until it is absolutely necessary can be implemented by taking advantage of the ability of Lisp to pass procedures and functions as data. These procedures are known as 'demons' and are invoked by the inference engine during the process of attempting to satisfy a goal.

Inference Engine Operation

The classification is performed by the manipulation of four list structures (Figure 24). The 'rule list' is a database of if-then production rules (prototype source code in Appendix C) . The 'goal list' is a stack of current subgoals to be satisfied. As they are discovered, the facts are stored in the 'context list'. Those rules proven false are removed from consideration by being placed on the 'discard list'.

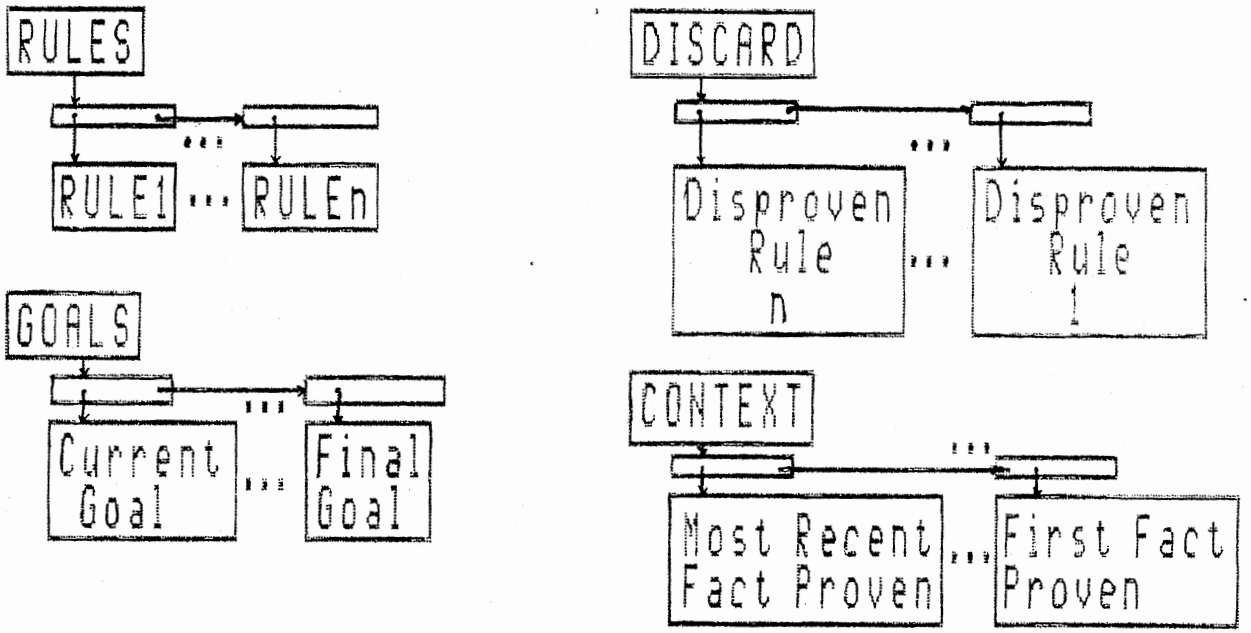


Figure 24. Inference Engine List Structure

The basic operation (source code in Appendix C) is to place the final classification as the current goal on the goal list. Until there are no more goals to satisfy, choose a rule to evaluate. This is done by first checking the consequent (RHS) of each rule to find a rule that can satisfy the current goal. If no appropriate RHS can be found, a flag is set to note this event. If this flag is set, the user is prompted for more information (or a function call is made to acquire this information in the case of an automatic system). If this flag is not set, then the rule chosen is 'evaluated'.

Evaluation starts by checking each of the antecedents (LHS) of the chosen rule against the context list to see if the value of this attribute is already known. Three cases are possible at this point:

Case 1: One or more attributes of the LHS is unknown. In this case, the first unknown attribute of the antecedents is placed on the goal stack as the current goal, and a new rule must be chosen.

Case 2: A contradiction is found. The rule is considered to be proven false. The false rule is then removed from the rule list and placed on the discard list. A new rule must now be chosen.

Case 3: All antecedents are found to have matches in the context list. In this case the rule is considered to be proven true. The current goal is

removed from the goal stack. The consequent of the true rule is added to the context list. Evaluation of this rule is complete.

This process continues until the last goal is satisfied or the rules are exhausted. In the event of rule exhaustion, the certainty scores of the candidate words are compared to hazard a guess as to the identity of the utterance.

Summary

The Explorer offers an environment conducive to the implementation of a real-time system by providing a means by which one processor will be making a classification based upon features extracted from one word while the other processor is extracting features from the next word. The individual processors are assigned tasks which are more naturally handled by them: number-crunching for the UNIX side, object manipulation on the Lisp side.

The quality of the classification may be quantified by the propagation of certainty factors. These factors are not implemented in the current prototype. The actual details of the symbol handling and the source code for a prototype system are provided in Appendix C.

CHAPTER VI

SUMMARY AND CONCLUSIONS

Summary

The basic question is 'Do patterns exist in speech such that a human, given a grahical representation of an unknown word, can determine to which of a reference vocabulary the word most closely corresponds?'. The implicit assumption is that the classification will be accomplished based on visual aspects of the plot.

In order to find out, speech samples of about 3Ø speakers were recorded and digitized. From these raw data, two short-time measures (ZCR and average magnitude over 1Ø msec intervals called frames) were plotted with the intent of finding common visual cues that would permit identification based on these two measures alone.

Conclusions

The most important answer to the basic question is that it can be done, with some caveats and limitations. The first problem that must be dealt with is that of determining where the word begins and ends (especially in the presence of noise). Endpoint placement for this

particular system can be best handled by object oriented heuristics. With proper endpoints, the virtual object and FFT parameters are much more likely to be reliable recognition characteristics.

The system works well (near 100 per cent accuracy) interactively with an informed user on the limited speaker samples available. This is to be expected as the test cases were, to a large extent, the source of the rules by which the classification is made. If it is known a priori that the system will only be used by a few people, the object oriented approach can also be used for a trained system. This would simplify matters a great deal and allow for thick accents and custom vocabularies.

How large a vocabulary may be is limited by how similar two different words may sound. In this case, contextual information may be the best discriminant. For the case of a word with more than one pronunciation (e.g. '0'), it is a simple matter to let the word appear in more than one place in the search tree.

Suggested Future Research

It seems appropriate to test the performance of an object oriented method aided by numerical techniques against a purely numeric strategy. For example, the scored pattern matching algorithm may be applied over the entire utterance to match an overall shape

template. This works best on plots that have been smoothed. The primary disadvantages are the additional computational burden of a search method that is no longer constrained to small regions and this method is less object oriented and more numeric in nature. Similar observations may be made about dynamic time warping.

For a global match, application of image processing techniques to the boundaries of the virtual object may provide a set of Fourier coefficients (phase information is important) that can be compared to a reference set of coefficients for each word in the vocabulary.

The system now exists in discrete pieces (pattern matching provided by the user). The next step is the synthesis of a single system from the components. After the inference engine is fully tested and debugged, the next step will be the codification of the numeric matching functions (complete with certainty factors). There are algorithms designed to match text words that are misspelled. If the source code for this can be found, it may be possible to adapt it into a version of the scored pattern matching described in Chapter II (depending on how the mismatch is determined and corrected).

Additional applications and speaking environments should be tried and tested for performance. Collecting

and testing more speech samples will provide a basis by which it may be possible to gauge how robust the object oriented approach is in general.

REFERENCES

Brown, M. K. & Rabiner, L. R. (1982). An Adaptive, Ordered, Graph Search Technique for Dynamic Time Warping for Isolated Word Recognition, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-30, No. 4, pp. 535-544.

Itakura, F. (1975). Minimum Prediction Residual Principle Applied to Speech Recognition, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-23, No. 1, pp. 67-72.

Lamel, L., Rabiner, L. R., Rosenberg, A. E., & Wilpon, J. (1981). An Improved Endpoint Detector for Isolated Word Recognition, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-29, No. 4, pp. 777-785.

Lau, Y., & Chan, C. (1985). Speech Recognition Based on Zero Crossing Rate and Energy, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-33, No. 1, pp. 320-323.

Press, W. H. et al (1986). Numerical Recipes: The Art of Scientific Computing, Cambridge University Press, Cambridge, 1986.

Rabiner, L. R., & Sambur, M. R. (1975). An Algorithm for Determining the Endpoints of Isolated Utterances, The Bell System Technical Journal, Vol. 54, No. 2, pp. 297-315.

Rabiner, L. R., & Schafer, R. W. (1978). Digital Processing of Speech Signals, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1978.

Sambur, M. R., & Rabiner, L. R. (1975). A Speaker-Independent Digit-Recognition System, The Bell System Technical Journal, Vol. 54, No. 1, pp. 81-1Ø2.

Schmucker, K. J. (1984). Fuzzy Sets, Natural Language Computations, and Risk Analysis, Computer Science Press, 1984.

Thompson, B., & Thompson, W. (1986). Finding Rules in Data, Byte, November 1986, pp. 149-158.

Tschudi, F. (1988). Matrix Representation of Expert Systems, AI Expert, Vol. 5, No. 1Ø, pp. 44-53, October 1988.

APPENDIXES

APPENDIX A

TIME PLOTS OF ZCR AND MAGNITUDE FOR SELECTED SPOKEN DIGITS

Figures 25-34: Plots of ten digits by same speaker.

Figures 35-44: Plots of '6' by different speakers.

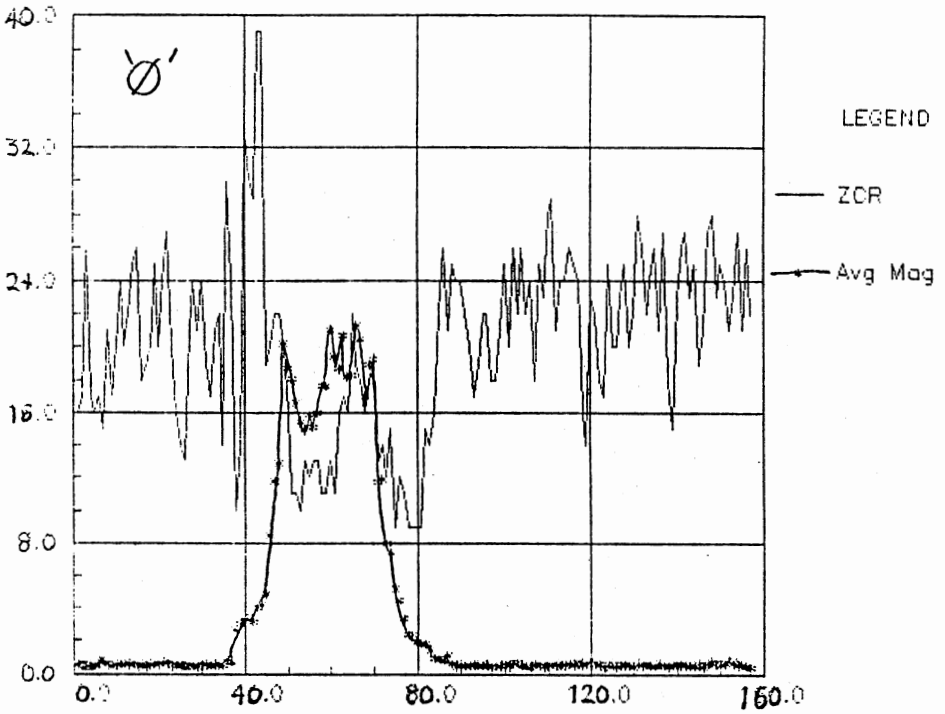


Figure 25. Time Plot of '0' Speaker 1

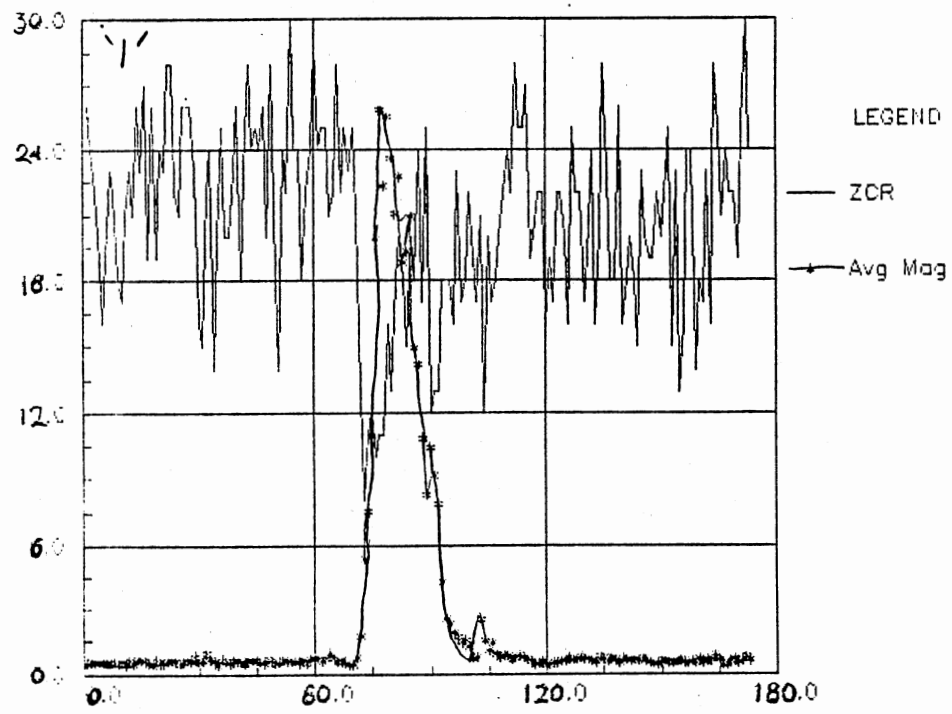


Figure 26. Time Plot of '1' Speaker 1

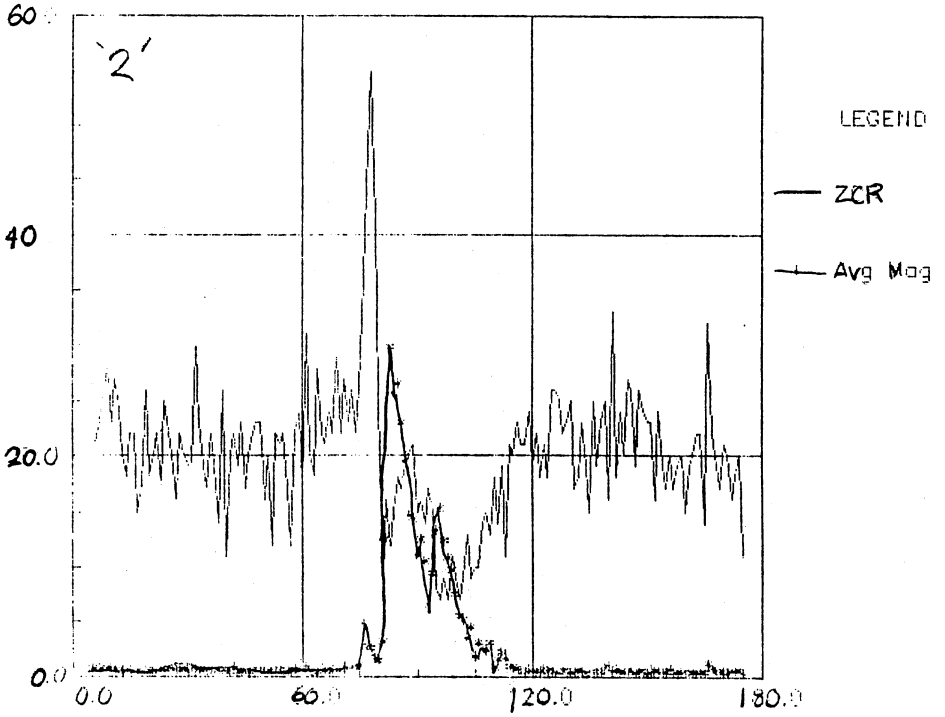


Figure 27. Time Plot of '2' Speaker 1

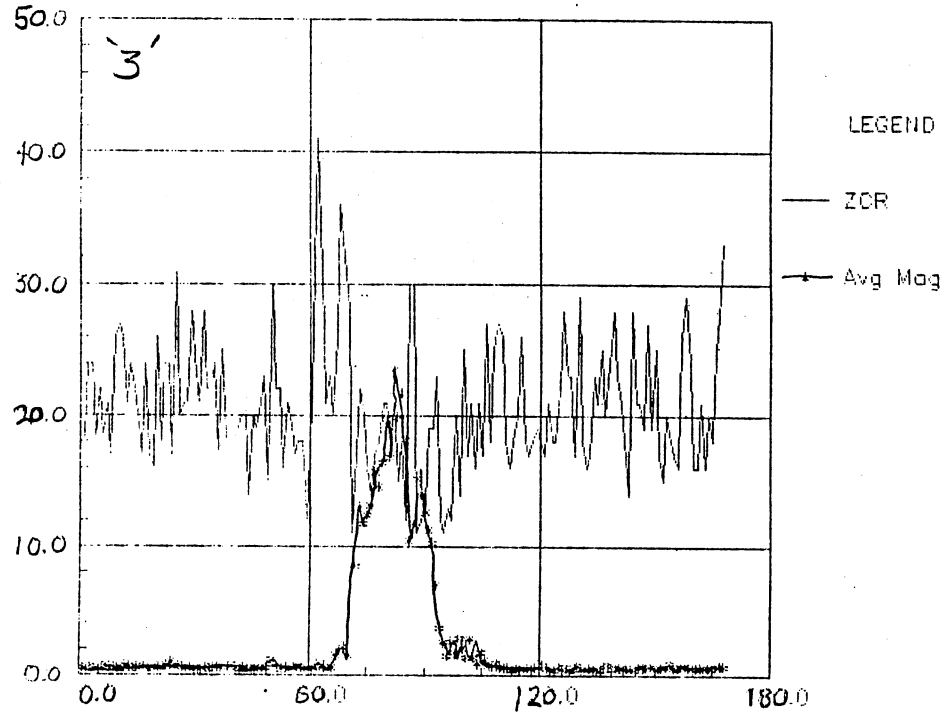


Figure 28. Time Plot of '3' Speaker 1

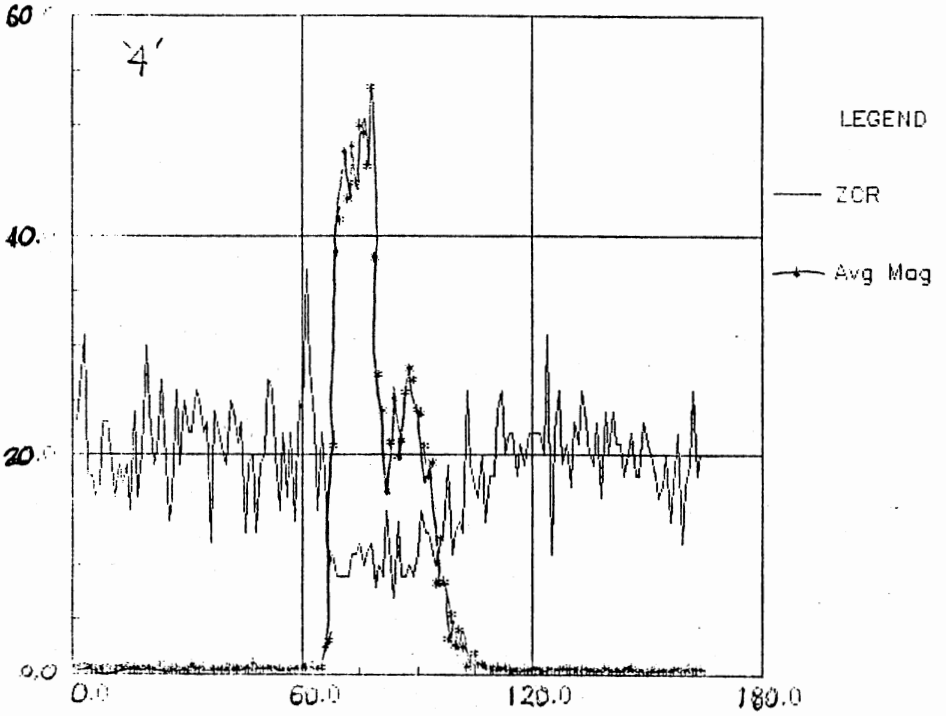


Figure 29. Time Plot of '4' Speaker 1

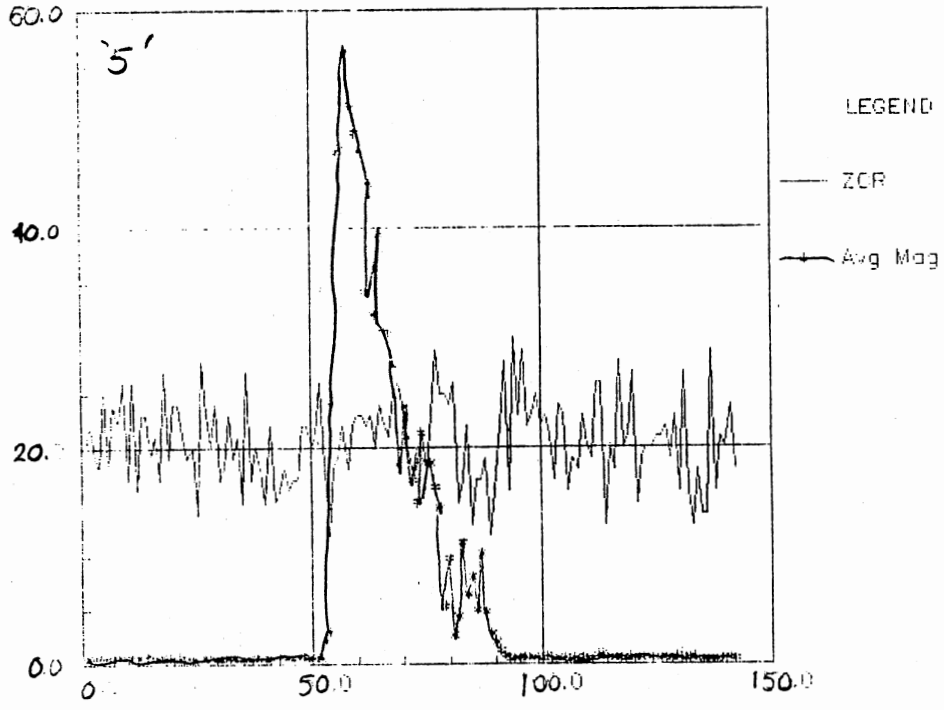


Figure 30. Time Plot of '5' Speaker 1

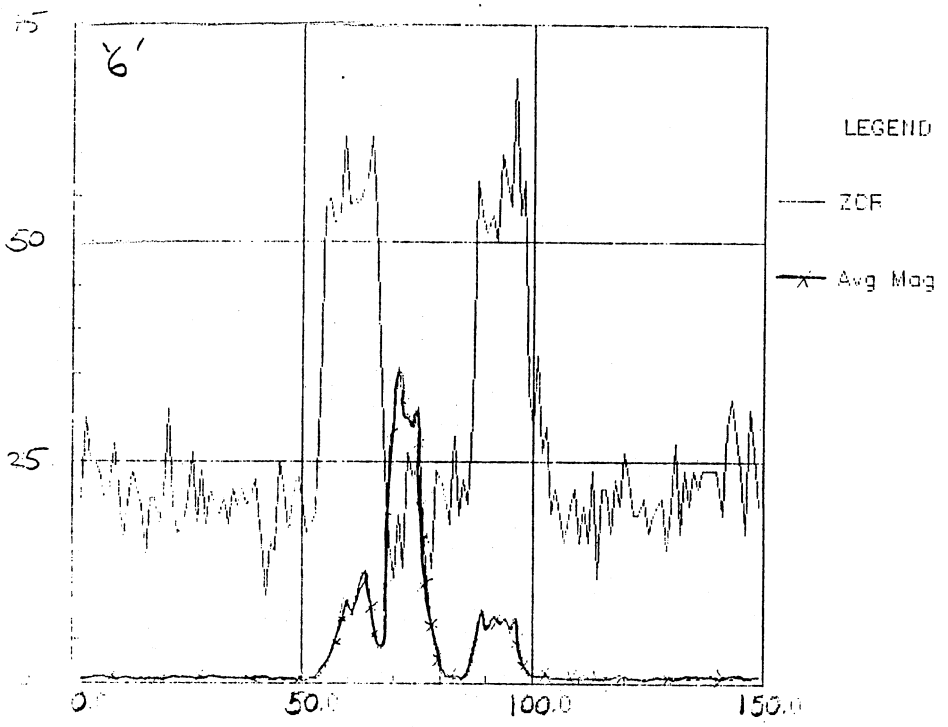


Figure 31. Time Plot of '6' Speaker 1

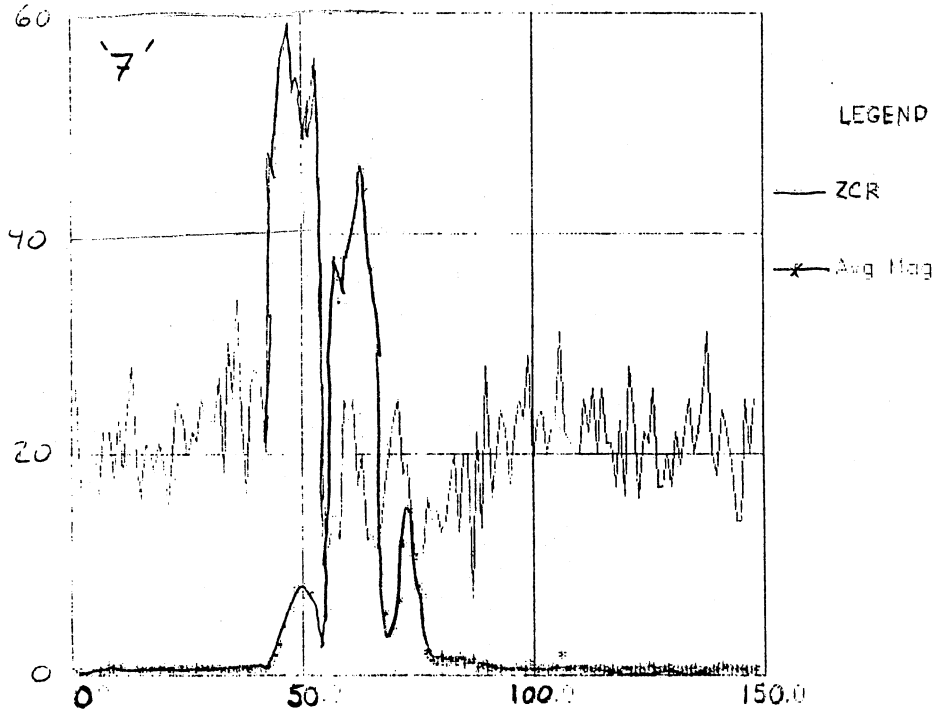


Figure 32. Time Plot of '7' Speaker 1

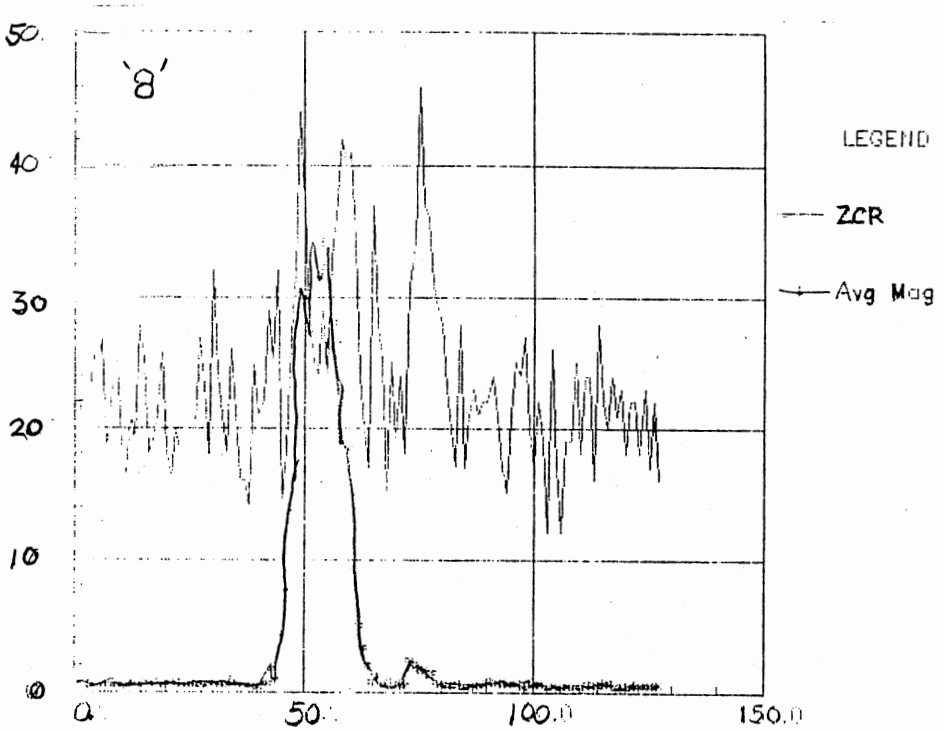


Figure 33. Time Plot of '8' Speaker 1

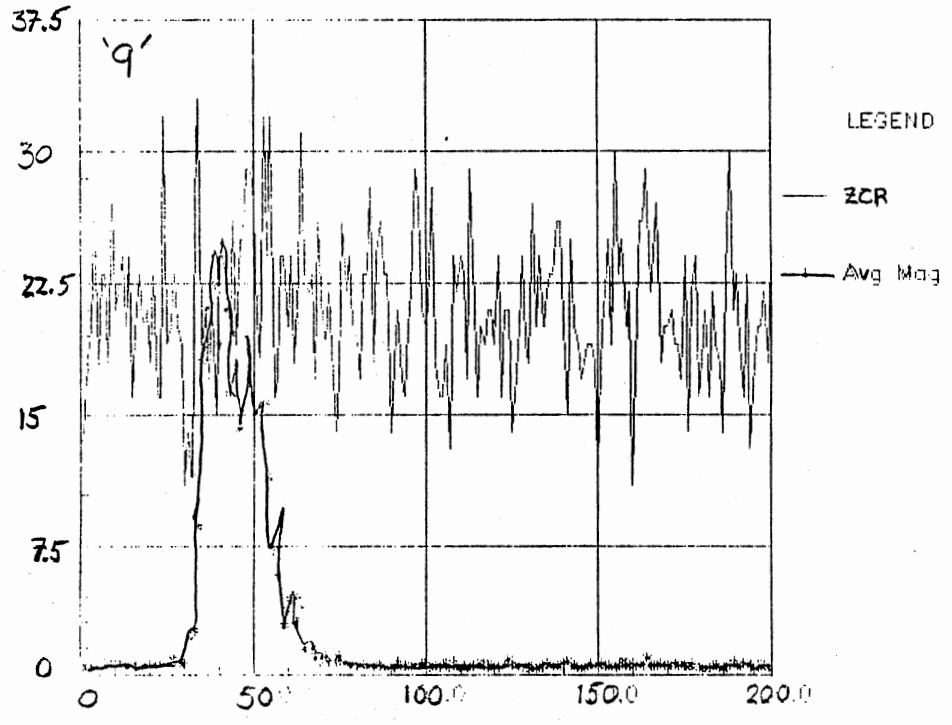


Figure 34. Time Plot of '9' Speaker 1

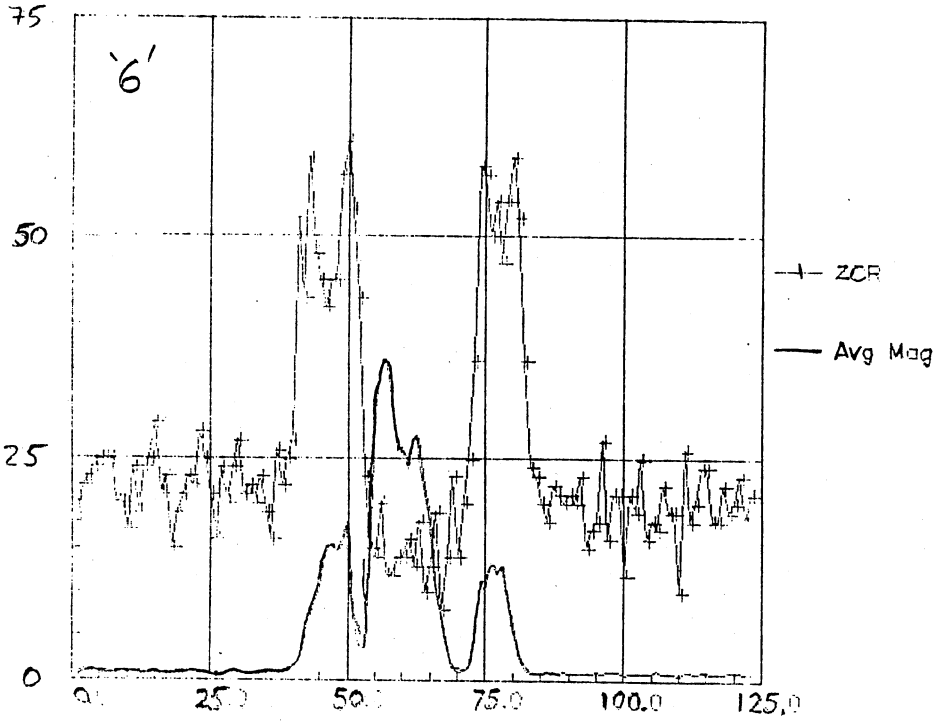


Figure 35. Time Plot of '6' Speaker 1

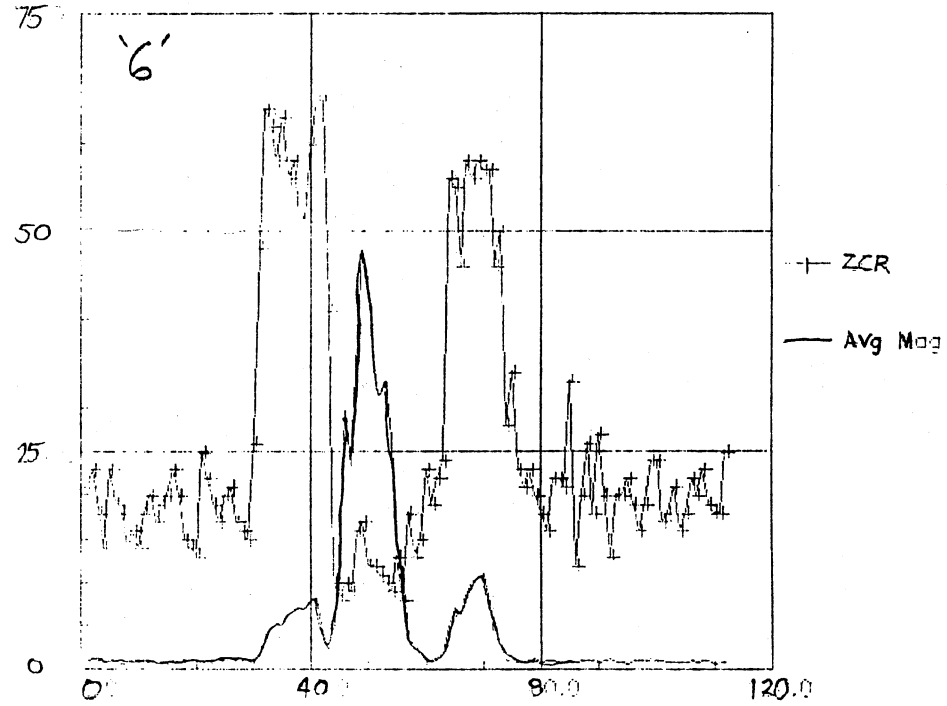


Figure 36. Time Plot of '6' Speaker 2

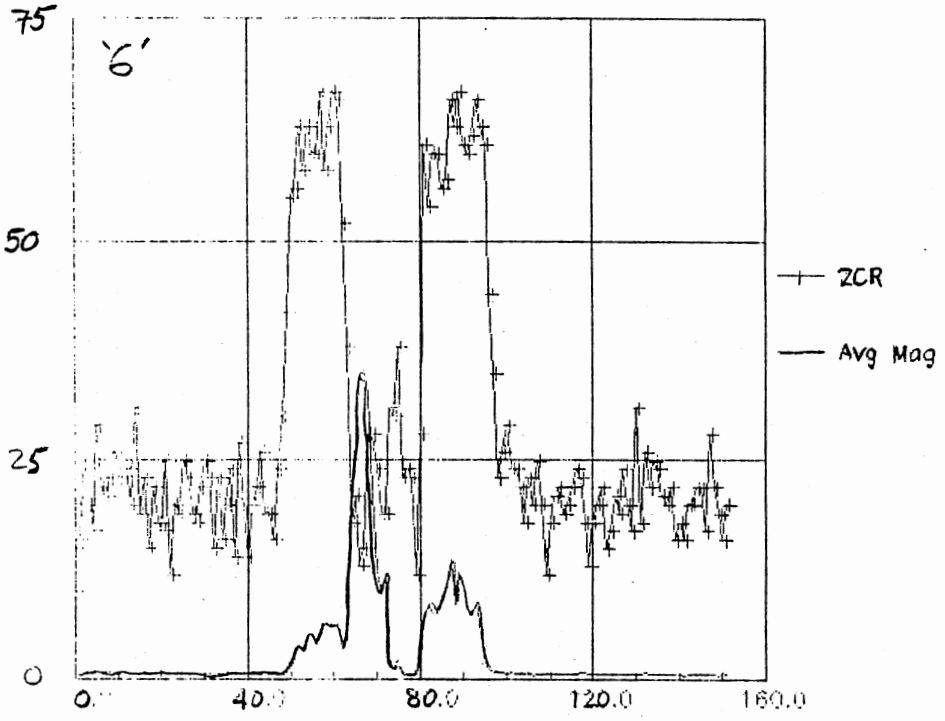


Figure 37. Time Plot of '6' Speaker 3

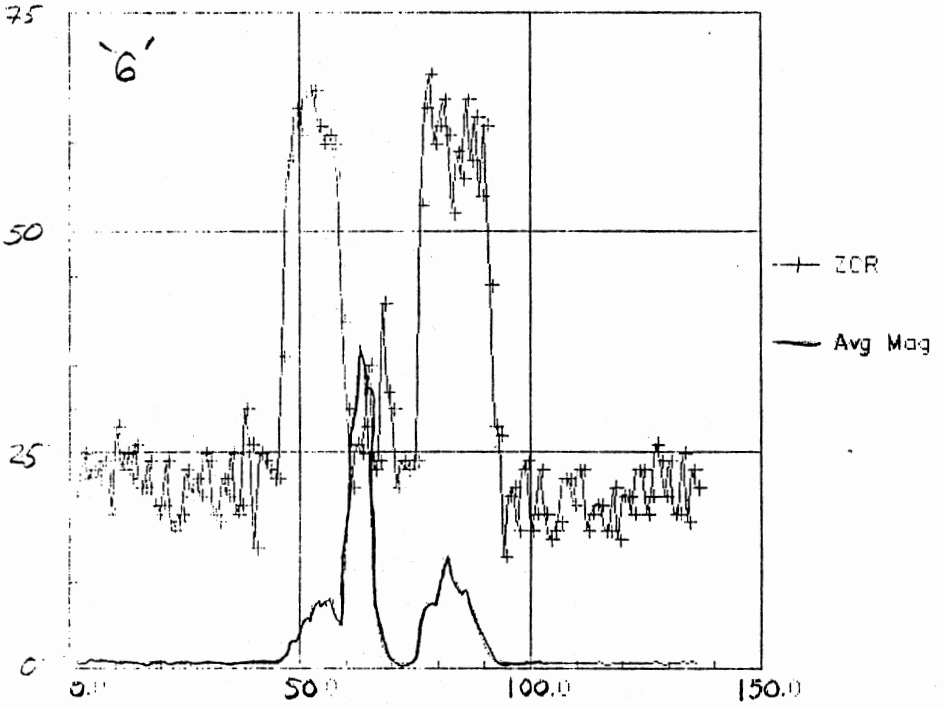


Figure 38. Time Plot of '6' Speaker 4

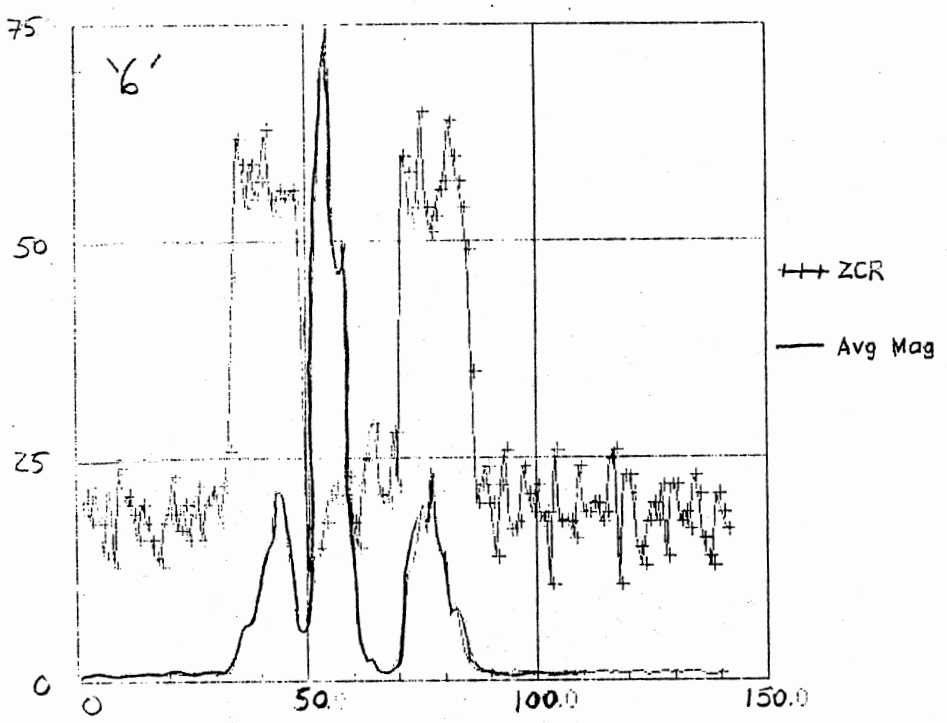


Figure 39. Time Plot of '6' Speaker 5

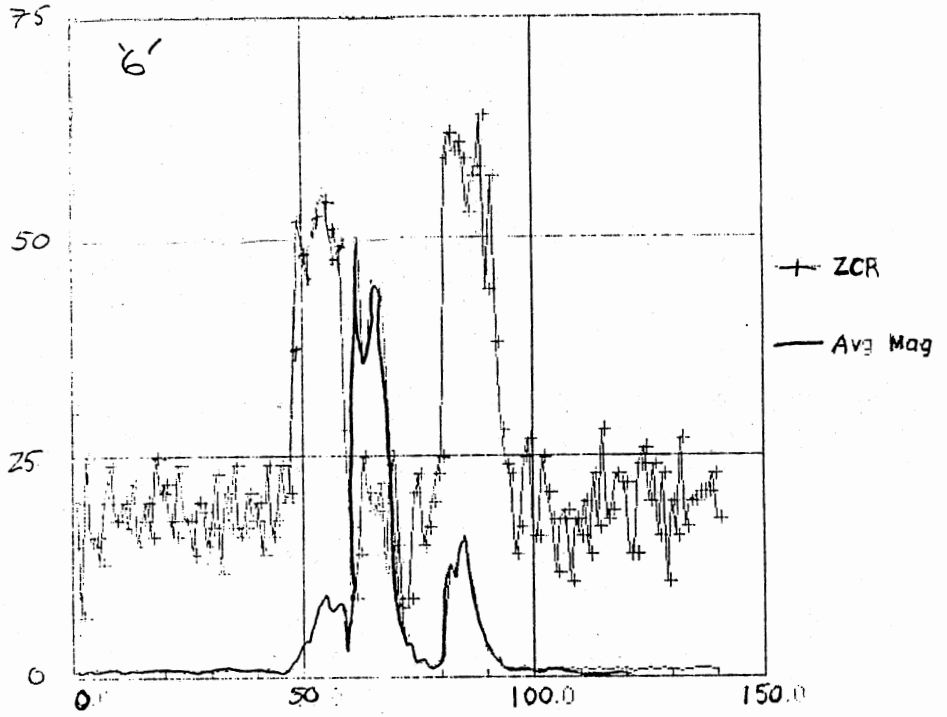


Figure 40. Time Plot of '6' Speaker 6

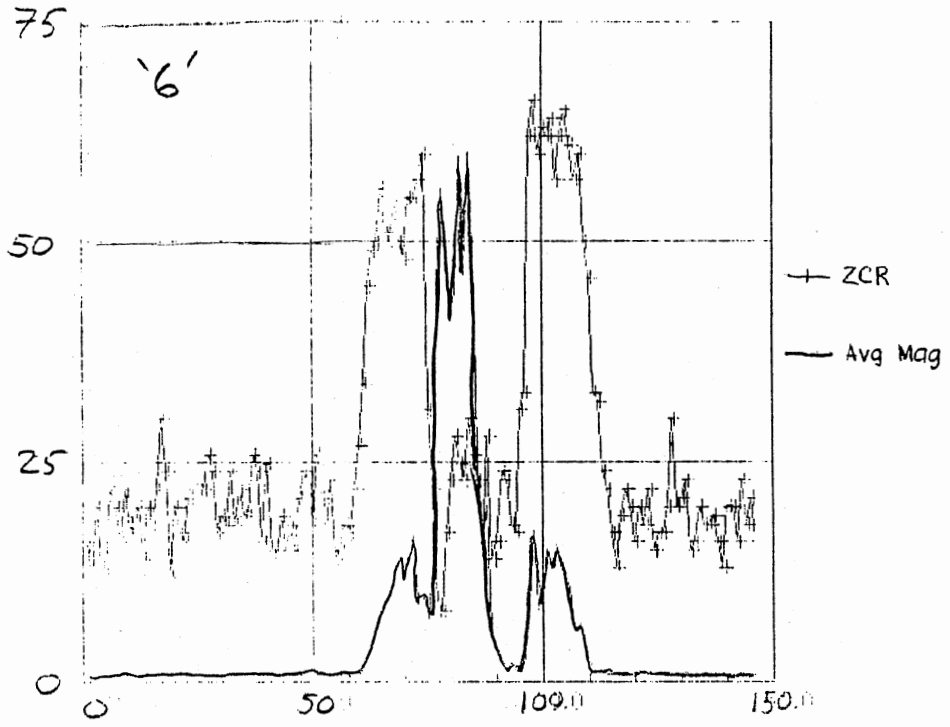


Figure 41. Time Plot of '6' Speaker 7

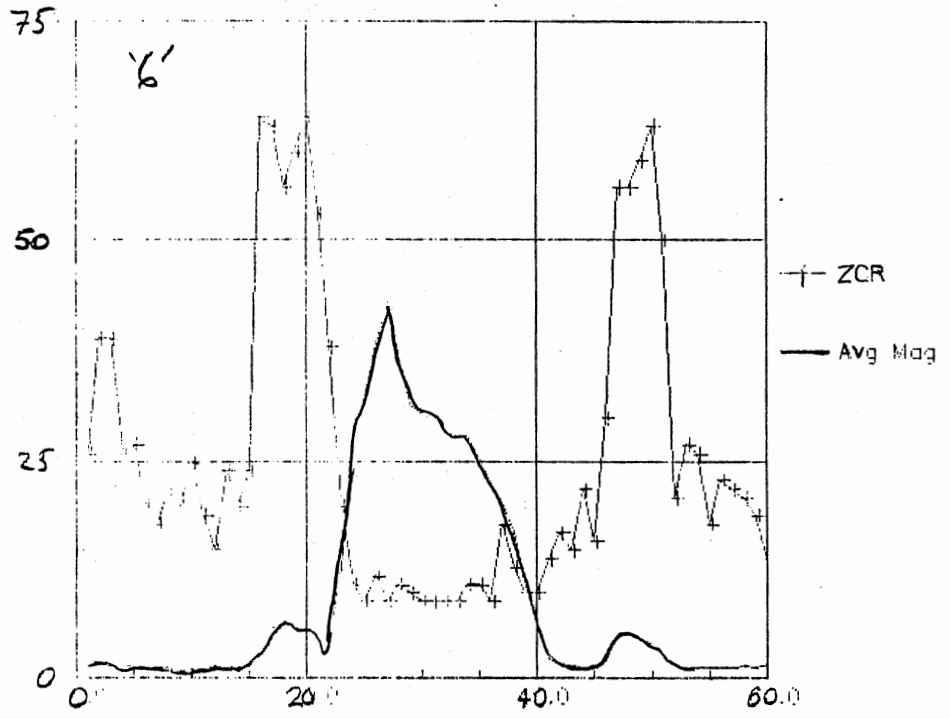


Figure 42. Time Plot of '6' Speaker 8

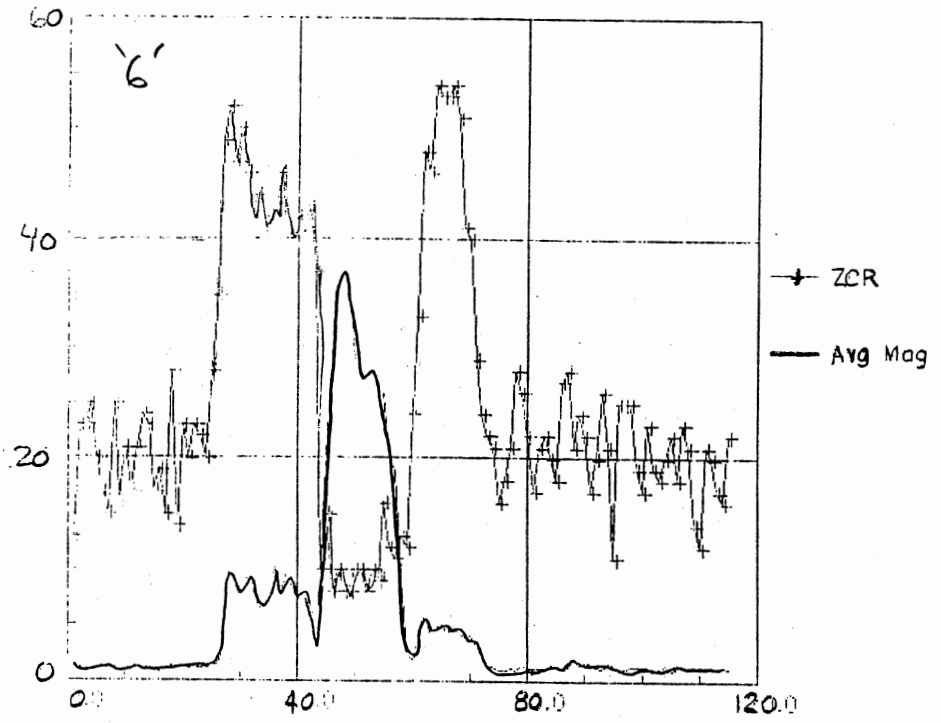


Figure 43. Time Plot of '6' Speaker 9

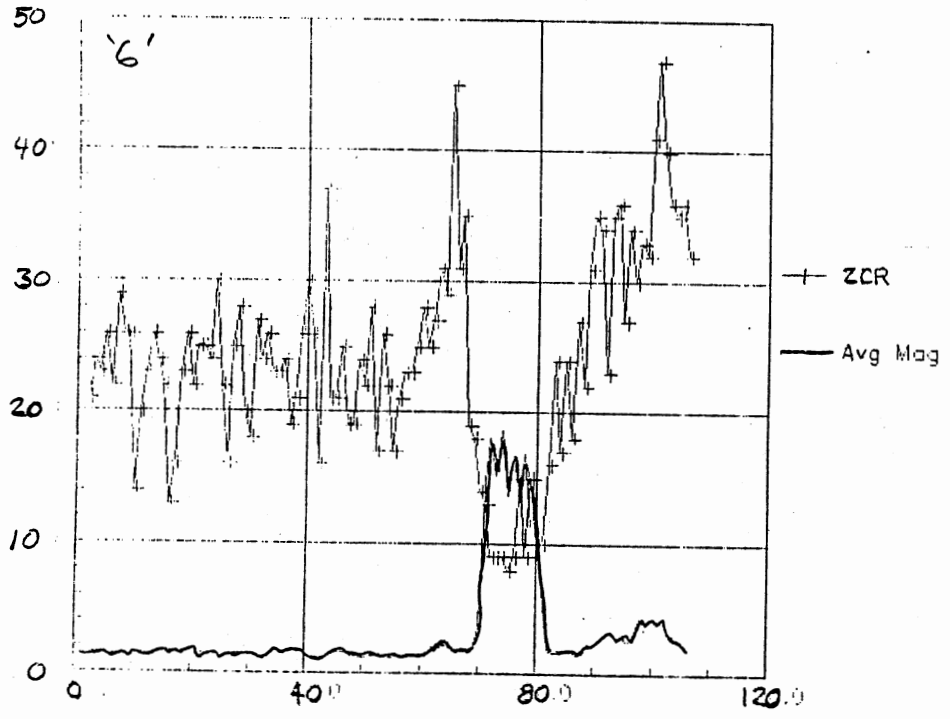


Figure 44. Time Plot of '6' Speaker 10

APPENDIX B

NUMERIC PROCESSING SOURCE CODE

```
PROCEDURE scrplotf (var y:plotarray; nf, nti, ordmx : integer);
{THIS VERSION FOR DUMPING PLOT INFO TO A FILE }
{  plotarray = array[1..2,0..200] of real
          y = 2-D array with the values to be plotted
         nf = number of functions to be plotted
        nti = number of time increments to be plotted
      ordmx = max ordinal to be plotted (999 to self scale) }
type carray = array[0..70] of char;
var i, j, ordlbl, row, ydata, yzi : integer;
    ymax, ymin, yscl, yzr         : real;
    grid1, grid2, grid3           : char;
    line                          : carray;
    filespec : string[20];
    textfile : text;
begin {scrplotf}
  write(' File to hold the screen plot : ');
  readln(filespec);
  assign(textfile,filespec);
  rewrite(textfile);
  writeln(textfile,filespec);
  yzr := 0;
  ymin := 0;
  yscl := 1;
  if ordmx = 999 then
    begin
      ymax := -1.0E+30;
      ymin := 1.0E+30;
      for j := 1 to nf do
        for i := 0 to nti do
          begin
            if y[j,i] > ymax then
              ymax := y[j,i];
            if y[j,i] < ymin then
              ymin := y[j,i];
          end;
      yscl := 60/(ymax - ymin);
      writeln(textfile,'      VALUES FROM ',ymin:8:5,
              ' TO ',ymax:8:5,' SCALED BY ',yscl:9:5);
      yzr := -ymin * yscl;
      yzi := round(yzr + (10 - round(yzr) mod 10 ));
      if (yzi - yzr) >= 10 then
        yzi := yzi - 10;
      ordmx := 70 - yzi;
    end;
  write(textfile,'    ');
  for i := 0 to 7 do
    begin
      ordlbl := 10 * i - 70 + ordmx;
      write(textfile, ordlbl:4,'      ');
    end;
  writeln(textfile);
  for row := 0 to nti do
    begin
      if ( row mod 10 ) = 0 then
```

```
        begin
          grid1 := '-';
          grid2 := '+';
          write(textfile,' ','  ',row:4);
        end
      else
        begin
          grid1 := ' ';
          grid2 := 'I';
          write(textfile,' ','      ');
        end;
      for i := 0 to 70 do
        line[i] := grid1;
      for i := 0 to 7 do
        if (10 * i) = (70 - ordmx) then
          line[10 * i] := '0'
        else
          line[10 * i] := grid2;
      grid3 := 'A';
      if row <= nti then
        for j := 1 to nf do
          begin
            ydata := round((y[j,row] - ymin) * yscl - yzr - ordmx +70);
            if (ydata >= 0) and (ydata <= 70) then line[ydata] := grid3
            else
              if ydata < 0 then line[0] := '$'
              else line[70] := '$';
            grid3 := succ(grid3);
          end;
      for i := 0 to 70 do write(textfile,line[i]);
      writeln(textfile);
    end;
  close(textfile);
end; {scrplotf}
```

```
{ ASCII plot of y vs x. >>>>>>>>>> TO A NAMED FILESPEC <<<<<<<<<<< }
procedure xyplotf(npts : integer; var x,y : array200;
                  abrange, abmax, ordmax : integer );
(* npts     -- number of points to plot           *)
(* x,y      -- array[1..200] of real to be plotted *)
(* abrange  -- abscissa range (999 to self scale)  *)
(* abmax    -- maximum abscissa value (uses default w/ selfscaling)*)
(* ordmax   -- maximum ordinal value (uses default w/ selfscale)   *)

const abscl = 0.600000;   (* scaling factor for printing *)
type carray = array[0..70] of char; (* character array for each line *)
var abslbl, i, j, k, ordlbl, row, xdata, xzi, ydata, yzi : integer;
    temp, xscl, ymax, ymin, yscl, yzr, xzr               : real;
    grid1, grid2                                         : char;
    line                                                 : carray;
    filespec : string[20];
    texttile : text;
begin
 yzi := 0;
 xzi := 0;
 for i := 1 to (npts - 1) do
   for j := (i + 1) to npts do
     if x[i] > x[j] then
      begin
       temp := x[i];
       x[i] := x[j];
       x[j] := temp;
       temp := y[i];
       y[i] := y[j];
       y[j] := temp;
      end;
 write(' Filename for xyplot?: ');
 readln(filespec);
 assign(textfile,filespec);
 rewrite(textfile);
 writeln(textfile,filespec);

 if abrange = 999 then
  begin
   ymax := -1.0e+30;
   ymin :=  1.0e+30;
   for i := 1 to npts do
    begin
     if y[i] > ymax then ymax := y[i];
     if y[i] < ymin then ymin := y[i];
    end;
   yscl := 60/(ymax - ymin);
   xscl := 80/(x[npts] - x[1]);
   writeln( textfile,'      VALUES OF X FROM ',x[1]:9:5,' to ',x[npts]:9:5,' scaled by ',
          xscl:9:5 );
   writeln( textfile,'      VALUES OF Y FROM ',ymin:9:5,' to ',ymax:9:5,' scaled by ',
          yscl:9:5 );
   yzr := -ymin * yscl;
   yzi := round(yzr + (10 - round(yzr) mod 10));
```

```
  if (yzi - yzr) >= 10 then yzi := yzi - 10;
  xzr := -x[1] * xscl * abscl;
  xzi := round(xzr + (6 - round(xzr) mod 6));
  if (xzi - xzr) >= 6 then xzi := xzi - 6;
  for i := npts downto 1 do
   begin
    y[i] := (y[i] - ymin) * yscl + yzi - yzr;
    x[i] := (x[i] - x[1]) * xscl + (xzi - xzr)/abscl;
   end;
  abmax := 90; (* new defaults set *)
  ordmax := 70;
  abrange := 90;
 end;
write( textfile,'0   ');
for i := 0 to 7 do
 begin
  ordlbl := 10 * i - 70 + ordmax - yzi;
  write( textfile,ordlbl:4,'      ');
 end;
writeln( textfile );
k := 1;
for row := 0 to round(abrange*abscl) do
 begin
  if (row mod 6) = 0 then
   begin
    grid1 := '-';
    grid2 := '+';
    abslbl := 10 * ((row - xzi) div 6) + abmax - abrange;
    write( textfile,'   ',abslbl:4);
   end
  else
   begin
    grid1 := ' ';
    grid2 := 'I';
    write( textfile,'       ');
   end;
  for i := 0 to 70 do line[i] := grid1;
  for i := 0 to 7 do line[10*i] := grid2;
  repeat
   if k <= npts then
    begin
     xdata := round(abscl * (x[k] - abmax + abrange));
     ydata := round(y[k] - ordmax + 70);
    end;
   if xdata = row then
     if (ydata >= 0) and (ydata <= 70) then line[ydata] := 'X'
     else
       if ydata < 0 then line[0] := '$'
       else line[70] := '$';
   if xdata <= row then k := k + 1;
  until (k > npts) or (xdata > row);
  for i := 0 to 70 do write( textfile,line[i]);
  writeln( textfile );
 end;
```

```
 close(textfile);
end;
```

```
PROGRAM dio (input, output);
(* Ross Goeres Fall 88 *)
(* new endpoint placement algorithm *)
(* Reads input from ASCII frame file of pt:integer ZCR:real Mag:real *)
CONST
    maxframes = 200;  {2 seconds of 10 msec frames}
    110ms = 80;       {length of 10 msec at 8 KHz}
    quietframes = 10; {100 msec silence }
    bumplook = 11;    {max number of frames to look in bump extension}
    zcrlook = 6;      {"    "     "    "     "   "     ZCR    "      }
    lenavgz = 3;      {number of frames to average ZCR over}
    looklimit = 25;
    dcoffset = 2;     {system specific constant}
    numdevs = 3.0; {avg silence ZCR +/- numdevs * std dev = normal range}
TYPE
    framearray   = array[1..maxframes] of real; {2 sec. of 10 msec frames}
    intplotarray = array[1..2,1..maxframes] of integer;
    dataarray    = array[1..2,1..maxframes] of real;
    plotarray    = array[1..5,0..maxframes] of real;
VAR
    data         : dataarray;
    datalevels   : intplotarray;
    filespec     : string[20];
    textfile     : text;
    itu,itl,
    izcr,izct,
    imax,maxedat,
    n1,n2,
    mzero1,mzero2,          {initial endpoint markers}
    bext1,bext2,
    zext1,zext2,
    numread      : integer;
    zavg, sdzcr,
    mag0, scale  : real;
    a            : plotarray;
    gostr        : string[3];
    goagain      : boolean;

{$I SCRPLOTF.INC }

  FUNCTION plotlevel( x, scale : real):integer;
  { returns the integer plot placement }
   begin
    plotlevel := round(x * scale);
   end;

  FUNCTION signum(x : shortint):integer;
  { returns the sign of an argument }
  begin
   if ((abs(x) + x) >= abs(x)) then signum := 1
   else signum := -1;
  end;
```

```
PROCEDURE getspeechdata( var data : dataarray;
                         var datalevels : intplotarray;
                         var numread, maxedat, imax : integer;
                         var scale : real);
var lcv, temp : integer;
    ymax : real;
begin
 write(' ASCII Frame Data Filename?: ');
 readln(filespec);
 assign(textfile,filespec);
 reset(textfile);
 lcv := 1;
 numread := 0;
 imax := 0;
 ymax := 0.0;
 while ( (lcv < maxframes) and (not(eof(textfile))) ) do
  begin
   readln(textfile,temp,data[1,lcv],data[2,lcv]);
   numread := numread + 1;
   if ((data[2,lcv] * 110ms) > imax) then {imax based on energy}
    begin
     imax := round(data[2,lcv] * 110ms);
     maxedat := lcv;
    end;
   if (data[1,lcv] > ymax) then ymax := data[1,lcv];
   if (data[2,lcv] > ymax) then ymax := data[2,lcv];
   if (data[1,lcv] < 1) then {edited to zero}
    begin
     numread := numread - 1; {don't trust the last set}
     lcv := lcv + maxframes; {exit the loop}
    end
   else lcv := lcv + 1;
  end; (* while *)
 close(textfile);
 if (numread < maxframes) then {zero pad to end}
  begin
   for lcv := (numread + 1) to maxframes do
    begin
     data[1,lcv] := 0.0;
     data[2,lcv] := 0.0;
    end; (* for *)
  end; (* if *)
 scale := 60.0 / ymax;  {use plotting scale}
 writeln(' numread:',numread:4,' scale: ',scale:9:5);
 for lcv := 1 to numread do
  begin
   datalevels[1,lcv] := plotlevel(data[1,lcv],scale);
   datalevels[2,lcv] := plotlevel(data[2,lcv],scale);
  end;
 if (numread < maxframes) then
  begin
   for lcv := (numread + 1) to maxframes do
    begin
     datalevels[1,lcv] := 0;
```

```
    datalevels[2,lcv] := 0;
   end; (* for *)
  end; (* if *)
end; (* getspeechdata *)


PROCEDURE classifysilence(var data : dataarray;
                          var numread, itu, itl, izct, imax : integer;
                          var zavg,sdzcr,avgmag0,scale : real);
var sumsq,sumzcr,summag,temp1,zupper,zlower : real;
    imn,i1,i2, lcv1, zalev, zllev, zulev,m0lev : integer;
begin
 sumsq := 0.0;
 sumzcr := 0.0;
 summag := 0.0;
 for lcv1 := 2 to (quietframes + 1) do  (skip first frame in case of IFF )
  begin
   sumzcr := sumzcr + data[1,lcv1];
   summag := summag + data[2,lcv1]*110ms;
   sumsq := sumsq + data[1,lcv1]*data[1,lcv1];
  end;
 temp1 := sumzcr/quietframes;
 zavg := temp1;
 izcr := round(temp1);
 temp1 := sumsq - sumzcr*sumzcr/quietframes;
 sdzcr := sqrt(temp1/quietframes);
 imn := round(summag / quietframes);
 i1 := round(0.03 * (imax - imn)) + imn;
 i2 := 4 * imn;
 if (i1 < i2) then itl := i1
 else itl := i2;
 itu := 5 * itl;
 if ((izcr + 2.0*sdzcr) > 25.0) then izct := izcr + round(2.0*sdzcr)
 else izct := 25;

 zupper := zavg + numdevs * sdzcr;
 zlower := zavg - numdevs * sdzcr;
 avgmag0 := summag / (quietframes * 110ms);
 writeln('zavg:',zavg:8:4,' zu:',zupper:8:4,' zl:',zlower:8:4,' mag0:',
         avgmag0:8:4);

 zalev := plotlevel(zavg,scale);    (set range for anomaly decision)
 zllev := plotlevel(zlower,scale);
 zulev := plotlevel(zupper,scale);
 m0lev := plotlevel(avgmag0,scale);
 writeln(' levels: zavg: ',zalev:3,' zu: ',zulev:3,' zl: ',zllev:3,
        ' mag0: ',m0lev:3);

 write(' Save the FRAME PLOT to disk (y)? ');
 readln(gostr);
 if ((gostr[1] <> 'n') and (gostr[1] <> 'N')) then
  begin
   a[4,0] := 0.0; (puts autoscale lower bound at 0)
   a[5,0] := 0.0;
```

```
   a[1,0] := zavg;
   a[2,0] := zupper;
   a[3,0] := zlower;
   for lcvi := 1 to numread do
    begin
     a[1,lcvi] := zavg;
     a[2,lcvi] := zupper;
     a[3,lcvi] := zlower;
     a[4,lcvi] := data[1,lcvi];
     a[5,lcvi] := data[2,lcvi];
    end;
   scrplotf(a,5,numread,999);
  end;
end; (* classify silence *)


PROCEDURE estendpts (itu,itl : integer;
                     data  : dataarray;
                     datalevels : intplotarray;
                     var numread,n1,n2,maxedat,mzero1,mzero2 : integer;
                     mag0, scale : real);
var i : integer;
    mag0level : real;
begin {Sambur and Rabiner algorithm}
 i := 0;
 repeat
  i := i + 1;
 until (110ms*data[2,i] >= itu);
 while (110ms*data[2,i] > itl) do i := i - 1;
 n1 := i;
 i := maxframes;
 repeat
  i := i-1;
 until (110ms*data[2,i] >= itu);
 while (110ms*data[2,i] > itl) do i := i + 1;
 n2 := i;

 {new algorithm}
 i := maxedat;
 mag0level := plotlevel(mag0,scale);
 repeat
  i := i + 1;
 until ((datalevels[2,i] <= mag0level) or (i = maxframes));
 if i = maxframes then
   writeln(' ERROR: failed to find mag0level before maxframes');
 mzero2 := i;
 i := maxedat;
 repeat
  i := i - 1;
 until ((datalevels[2,i] <= mag0level) or (i = 0));
 if i = 0 then
   writeln(' ERROR: failed to find mag0level before 0 index');
 mzero1 := i;
end;
```

```
PROCEDURE refineendpts(data : dataarray;
                       datalevels : intplotarray;
                       numread : integer;
                       zavg, sdzcr, mag0, scale : real;
                       var n1,n2,izct,
                       mzero1,mzero2,bext1,bext2,zext1,zext2 : integer);
var jlst,jlast,lcount,rcount,lcv,bumplevel,mag0level,i, index : integer;
    zupper, zlower, ztemp : real;
    inrange : boolean;

begin {Sambur and Rabiner algorithm}
 jlst := n1;
 jlast := n2;
 lcount := 0;
 rcount := 0;
 for lcv := 1 to looklimit do
  begin
   if (data[1,n1-lcv] > izct) then
    begin
     jlst := n1 - lcv;
     lcount := lcount + 1;
    end;
   if (data[1,n2+lcv] > izct) then
    begin
     jlast := n2 + lcv;
     rcount := rcount + 1;
    end;
  end; (* for *)
 if (lcount > 2) then n1 := jlst;
 if (rcount > 2) then n2 := jlast;
 write('N1: ',n1:4,' N2: ',n2:4);

 {new algorithm ,,,,recycle lcount,rcount,jlst,jlast}
 bext1 := mzero1;
 bext2 := mzero2;
 rcount := 0;
 lcount := 0;
 lcv := 0;
 jlast := 0;
 jlst := 0;
 mag0level := plotlevel(mag0,scale);
 writeln(' mag0level: ',mag0level:3);
 bumplevel := mag0level + 1;
 while (lcv < bumplook) do
  begin
   lcv := lcv + 1;
   if (datalevels[2,mzero2 + lcv] >= bumplevel) then  {bump found}
    begin
     rcount := rcount + 1; {increment bump counter}
     while (datalevels[2,mzero2 + lcv] >= bumplevel) do
       lcv := lcv + 1; {advance to end of bump }
     jlast := lcv; {save index of end of last bump}
```

```
   end; (* if *)
 end; (* while *)

if (rcount = 1) then (* just one bump-- check if significant *)
 begin
  if (datalevels[2,mzero2 + jlast - 2] >= bumplevel) then
    bext2 := mzero2 + jlast;
 end
else                          {for rcount = 0 => no extension}
  if (rcount > 1) then bext2 := mzero2 + jlast;

zupper := zavg + numdevs * sdzcr; {widen or narrow to taste}
zlower := zavg - numdevs * sdzcr;
lcv := 0; { extend based on zcr in silence }
inrange := false;
while ((lcv < zcrlook) and (not inrange))do
 begin
  ztemp := 0.0;
  for i := 0 to (lenavgz - 1) do
   begin
    ztemp := ztemp + data[1,bext2 + lcv + i];
   end;
  ztemp := ztemp / lenavgz; {average over next lenavgz frames}
  if ((ztemp >= zlower) and (ztemp <= zupper)) then inrange := true
  else  lcv := lcv + 1;
 end; (* while *)
zext2 := bext2 + lcv;

{now set left endpoint extensions}
bext1 := mzero1;
i := 1;
while ( i < bumplook) do
 begin
  index := bext1 - i;
  if (datalevels[2,index] > mag0level) then
   begin
    if (( data[1,index] > zupper) or (data[1,index] < zlower)) then
     begin {advance to end of bump iff zcr is anomalous}
      jlst := index;
      while (datalevels[2,jlst] > mag0level ) do
        jlst := jlst -1;
      i := bumplook; {exit while loop}
      bext1 := jlst;
     end;
   end;
  i := i + 1;
 end; (* while *)

lcv := 0; { extend based on zcr in silence }
inrange := false;
while ((lcv < zcrlook) and (not inrange))do
 begin
  ztemp := 0.0;
  for i := 0 to (lenavgz - 1) do
```

```
      begin
       ztemp := ztemp + data[1,bext1 - lcv - i];
      end;
     ztemp := ztemp / lenavgz; {average over next lenavgz frames}
     if ((ztemp >= zlower) or (ztemp <= zupper)) then inrange := true
     else  lcv := lcv + 1;
    end; (* while *)
   zext1 := bext1 - lcv;

   writeln(' zext1: ',zext1:3,' bext1:',bext1:3,' mzero1:',mzero1:3,
           ' mzero2:',mzero2:3,' bext2:',bext2:3,' zext2:',zext2:3);

  end; (* refine end points *)

begin
 goagain := false;
 repeat
  getspeechdata(data,datalevels,numread,maxedat,imax,scale);
  classifysilence(data,numread,itu,itl,izct,imax,zavg,sdzcr,mag0,scale);
  estendpts(itu,itl,data,datalevels,numread,
            n1,n2,maxedat,mzero1,mzero2,mag0,scale);
  refineendpts(data,datalevels,numread,zavg,sdzcr,mag0,scale,n1,n2,izct,
               mzero1,mzero2,bext1,bext2,zext1,zext2);
  write(' Would you like to process another set (y/n) <y> ? ');
  readln(gostr);
  if ((gostr[1] <> 'n') and (gostr[1] <> 'N')) then
    goagain := true
  else goagain := false;
 until (goagain = false);
end.
```

APPENDIX C

SYMBOLIC PROCESSING SOURCE CODE

```
; Prototype rulebase for Scheme Inference Engine
( define database '(
   ( rule1              ; rule name -- not used in
inferencing
     ((sinclike yes)) ; antecedent list (LHS)
     (classis magsinclike) ; consequent (RHS)
   )
   ( rule2
     ((classis plosive.at.end))
     (species eight)
   )
   ( rule3
     ((endplosive yes))
     (classis plosive.at.end)
   )
   ( rule4
     ((frontzhi yes))
     (classis zcr.hi.at.front)
   )
   ( rule5
     ((rttri yes))
     (classis mag.right.triangle)
   )
   ( rule6
     ((classis magsinclike)
      (onemesa no))
     (species six)
   )
   ( rule7
     ((classis magsinclike)
      (onemesa yes))
     (species seven)
   )
   ( rule8
     ((classis zcr.hi.at.front)
      (slorise yes))
     (species zero)
   )
   ( rule9
     ((classis zcr.hi.at.front)
      (jumpsup no))
     (species three)
   )
   ( rule10
     ((classis zcr.hi.at.front)
      (jumpsup yes))
     (species two)
   )
   ( rule11
     ((classis mag.right.triangle)
      (order rollercoaster))
     (species four)
   )
```

```
    ( rule12
      ((roller yes))
      (order rollercoaster)
    )
    ( rule13
      ((classis mag.right.triangle)
       (zupdown yes))
      (order zupmagdown)
    )
    ( rule14
      ((classis mag.right.triangle)
       (order zupmagdown))
      (species four)
    )
    ( rule15
      ((classis mag.right.triangle)
       (zconst yes))
      (order constzcr)
    )
    ( rule16
      ((classis mag.right.triangle)
       (order constzcr))
      (species five)
    )
    ( rule17
      ((classis mag.right.triangle)
       (zconst no))
      (order nine.or.one)
    )
    ( rule18
      ((order nine.or.one)
       (longtriang yes))
      (species one)
    )
    ( rule19
      ((order nine.or.one)
       (longtriang no))
      (species nine)
    )
    ( rule20
      ((endplosive no)
       (sinclike   no)
       (frontzhi   no)
       (rttri      no)
       (roller     no))
      (classis unknown.class)
    )
    ( rule21
      ((classis unknown.class))
      (species of.bad.class)
    )
  )
)
```

```
; asks about unknown attributes
; to be replaced by calls to numeric pattern matching functions
(define askabout
 (lambda (currgoal)
  (let ( (answer1 nil) (answer2 nil) )
   (case currgoal
    ((sinclike)   (writeln
                    "Does the main mag lobe have a mound on both sides?"))
    ((endplosive) (writeln
                    "Is there a ZCR spike at the end with a mag bump?"))
    ((frontzhi)   (writeln
                    "Is there a sharp ZCR lobe at the front?"))
    ((rttri)      (writeln
                    "Mag form a (very) rough triangle (right?),peak to left?"))
    ((onemesa)    (writeln
                    "Is the ZCR mesa just on the front?"))
    ((slorise)    (writeln
                    "Does the mag rise slowly during the high ZCR?"))
    ((jumpsup)    (writeln
                    "Does the mag jump up fast right after the high ZCR?"))
    ((zconst)     (writeln
                    "Is the ZCR constant over the entire utterance?"))
    ((roller)     (writeln
                    "Does mag resemble a 2-nump roller coaster(larger left)?"))
    ((longtriang) (writeln
                    "Does ZCR form a triangle peaking after mag?"))
    ((zupdown)    (writeln
                    "End: ZCR on ramp rise  mag mostly ramp going down?"))
    (else (error "Unknown attribute to find:" currgoal))
   ) ;case
(set! answer1 (read))
(cond ((member answer1 '(yes y no n)) ;check for legal response
       (set! answer2 (append (list currgoal answer1) context))
       (set! context answer2) ; update context information
       (set! goals (cdr goals)) ;remove subgoal
       (set! endtester t) ) ; non-nil to allow rule 1 to process
      (else (error "Illegal answer: " answer1))
 ) ;cond

 ) ;let
) ;lambda
) ;define
```

```
;; file: ie.s -- rules in: database.s -- prompts in ask.s
(define rulebase)       (define numcontra) ;move local to evalarule
(define goals)          (define numtrue)
(define context)        (define antcount)
(define discard)        (define templist)
(define endtester)      (define attrfound)
(define goalsleft)      (define rptr)
(define rulepicked)     (define targetr)
(define testdummy)      (define maxlen)
(define stopper)

(define (choosearule)
  (do ( (ruleptr rulebase (cdr ruleptr)) ; search RHS for current goal
        (count 1 (+ 1 count)) )
      ( (or (null? ruleptr) (equal? (car goals) (car (caddar ruleptr))))
        (set! rulepicked count) )
      (set! endtester (cdr ruleptr)) ; nil if gets to end w/o finding it
  )
)

(define (evalarule rulenum) ; global lists: rule,goals,context, and discard
   (set! numcontra 0) (set! numtrue 0) (set! antcount 0) (set! templist nil)
   (set! attrfound nil) (set! rptr nil) ; initializations
   (assert ( rulenum 0) "In evalarule: rulenum is " rulenum) ; error check
   (do ((rulelist rulebase (cdr rulelist)) ;find rule # rulenum
        (count 1 (+ 1 count)) )
       ((= count rulenum) (set! rptr rulelist)) )

   (set! stopper nil)
   (do ( (antlist (cadar rptr) (cdr antlist)) )      ;check each antecedent
       ( (or (null? antlist) (equal? stopper t)) )  ;against the context list
        (set! antcount (+ 1 antcount))
        (set! attrfound nil)
        (do ( (conlist context (cddr conlist)) )
            ( (or (null? conlist) ( numcontra 0)) )
             (cond ((equal? (caar antlist) (car conlist)) ;attribute found
                    (set! attrfound t)
                    (cond ((equal? (cadar antlist) (cadr conlist)) ;matches
                           (set! numtrue (+ 1 numtrue)) )
                          (else (set! numcontra (+ 1 numcontra))
                                (set! stopper t)  )
                    ); inner cond
                   ) ;outer guard
                   (else nil)
             ) ;outer cond
        ) ;inner do

        (cond ( (null? attrfound)
                (set! goals (append (list (caar antlist)) goals))
                (set! stopper t)  ;exit outer loop
                (set! attrfound t)
              )
              (  else nil )
         ); cond
```

```
   ) ; outer loop limit of do to check each antecedent

    (cond (( numcontra 0) ;move rule to discard list
           (set! targetr (- rulenum 1)) ; set index of target rule
           (set! maxlen (length rulebase))
           (set! templist nil)
           (do ( (ctr 0 (+ 1 ctr)) )
               ( (= ctr maxlen) (set! rulebase (reverse templist)) )
               (if (= ctr targetr) ; on rule to be moved
                  (set! discard (cons (car (list-tail rulebase ctr))
                                      discard)) ;put on discard list
                  (set! templist (cons (car (list-tail rulebase ctr))
                                       templist)) ;keep this rule
               )
           );do
          ); 1st guard
           (else nil) )

    (cond ((= antcount numtrue) ;then rule has been proven
            (set! context (append (caddar  rptr) context)) ;update context list
            (set! goals (cdr goals)) ) ;remove goal from consideration
           (else nil) )
)    ;define

(define (ie)
   (set! rulebase database)
   (set! goals '(species) )
   (set! goalsleft t)
   (set! context nil)
   (set! discard nil)
   (set! rulepicked 0)
   (set! endtester nil) ; initializations
   (do ( (dummyvar 0 (1+ dummyvar)) )
       ( (null? goalsleft) (writeln (car context) " is " (cadr context)) )
         (choosearule)
         (if (null? endtester) ;goal not found in RHS
             (askabout (car goals))
             (evalarule rulepicked) )
         (set! goalsleft goals)
    );do
) ;define
```

VITA

Ross P. Goeres

Candidate for the Degree of

Master of Science

Thesis: AN OBJECT-ORIENTED APPROACH TO A SPEAKER INDEPENDENT ISOLATED-WORD RECOGNITION SYSTEM

Major Field: Electrical and Computer Engineering

Biographical:

Personal Data: Born in Lemmon, South Dakota, the son of John William and Laurentina Margaret Goeres, December 22, 1955.

Education: Graduated from Lemmon High School, Lemmon, South Dakota, in May 1974; attended University of Minnesota and Brown Institute in Minneapolis, Minnesota; attended Mississippi Gulf Coast Junior College, Biloxi, Mississippi; received Associate of Applied Science Degree from Community College of the Air Force, December, 1981; received Bachelor of Science Degree in Electrical Engineering from The University of Texas at Austin, Austin, Texas, December, 1984; attended Oklahoma City University, Oklahoma City, Oklahoma; completed requirements for the Master of Science degree at Oklahoma State University in December, 1988.

Professional Experience: Telecommunication Design Engineer at the Engineering Installation Division, Tinker Air Force Base, Oklahoma, June, 1985, to June, 1987.