PROSTHESIS CONTROL USING A NEAREST NEIGHBOR ELECTROMYOGRAPHIC PATTERN CLASSIFIER

by

David Charles Dening

Dissertation submitted to the Graduate Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY
in
Electrical Engineering

APPROVED:

F. G. Gray, Chairman

A. A. Beex

R. M. Haralick

T. E. Leinhardt

R. Lumia

April, 1982

Blacksburg, Virginia

ACKNOWLEDGEMENTS

The work reported here was supported by the General Electric Electronics Laboratory. Special credit goes to Mr. W.H. Perkins, manager of the Solid State Circuits Subsection, for handling the overhead dislocations that this effort caused and to R. LaDon Brennan of employee relations for providing the Co-op and summer student support.

The hardware was constructed through the efforts of Sharon Oatman, Emily Martin, and Tim Waldron. Fabrication of the active EMG electrodes was done by Larry Westerman, who also wrote the pattern recognition software for the microcomputer based system.

Table of Contents

I. PROBLEM DESCRIPTION

Introduction

This dissertation addresses the problem of upper-extremity prosthesis control via myoelectrical signal pattern recognition. Since the author has a vested interest in a below elbow controller, the project was skewed toward a solution of that particular problem. However, the final pattern recognition system implemented for evaluation may be applied in a general fashion to various other prosthesis control functions.

All electromyographic (EMG) pattern-recognition-based prosthesis controllers rely on two basic premises. First, the amplitude and frequency content of an EMG signal is related to the loading on the muscle tissue in the proximity of the pickup electrodes. Second, the tension in various muscles results from the production of opposing forces or the transmission of a force to the ground. Thus, for example, when an object is to be lifted, the muscles in the shoulder region must transmit the required force to the trunk, legs, and eventually the earth. As a result, the patterns of muscle tension at various sites in the body contain information about movement and forces from

apparently unrelated regions. These muscle tensions may be detected electronically and used to control a prosthesis.

A. The Origin of EMG Signals and Their Relation to Muscle Contractions

Muscle cells are controlled by motor nerve cells from the central nervous system. The nerve cell, its axon, and the muscle fibres it drives, are referred to as a motor unit. The number of muscle fibers per motor unit in a human body can range from as low as 25 to as high as 2000 [1]. In general, muscles required for precise delicate work, will consist of motor units with a smaller number of muscle fibers, but if strength is an additional requirement, the muscle will contain a larger number of motor units.

The muscle cell wall maintains a resting potential of from -60mV to -100mV between the cell interior and the surrounding body fluids. When a section of the cell membrane is excited by the flow of an ionic current or by an externally applied source of energy, an avalanche effect occurs: the cell attempts to attain equilibrious ion concentrations on both sides. Since various ion types move at different transfer rates, a momentary action potential of approximately +20mv occurs before the new equilibrium is

reached. An inactive period then follows while the initial resting potential is reached. These various changes in a cell's potential occur as a consequence of the initiation of a motor unit contraction. When a motor unit is triggered, all the muscle fibers in that unit contract for a brief period of time and then relax. Thus, to produce a continous force from a muscle requires the retriggering of various motor units.

A muscle contraction's force is controlled by a combination of two mechanisms. As the force requirement increases, there is an orderly recruitment of motor units according to their size. In addition, the average output force may also be adjusted by altering the firing rate of single motor units [2]. Lest it be assumed that control of individual motor units is beyond conscious control, work has been conducted on voluntary control using feedback [3], which indicates that through training a great potential exists for improved EMG prosthesis control.

A combination of the various action potentials may be detected external to the body by their total induced potentials on a surface electrode pair. An electrode pair and ground and a differential amplifier are used to minimize the detection of external electrical noise. In addition, the amplifier must have high input impedence because of the

insulating properties of the skin between the electrodes and the motor units. As a consequence of this detection configuration, the amplifier output will be a weighted sum of all the motor unit activity in the vicinity of the electrodes. The weighting coefficients can be positive or negative, depending on their relative proximity to the two electrodes. In addition, the higher frequency components will tend to be further attenuated with increasing distance.

The amplitude and frequency content of EMG signals is related to the muscle contraction levels in the vicinity of the detection electrodes. The amplitude dependence derives from the increased number of firing motor units to produce a larger force. The various combinations of action potentials can then add up to form higher EMG signal peaks. The tendency for higher frequency contents in EMG signals due to increased output forces is again a function of the larger number of active motor units in addition to an increased firing rate.

Early quantitative experiments by Lippold [4] presented relations between isometric muscle contraction levels and electromyographic tracings. A linear relation was reported between the isometric tension and its integrated (absolute value) electromyogram. In later experiments, the same group [5] reported results recorded at a constant velocity of

lengthening or shortening where the integrated electrical activity was again linearly related to the tension. In addition, investigations conducted at constant tension showed that the electrical activity increased linearly with the velocity of shortening but was independent of the rate for lengthening. However, other researchers (Zuniga et al.) [6] have reported a non-linear (quadratic) relation between averaged EMG signals (RMS) and isometric tension. Differences in these results may be accounted for by the experimental procedures. While Zuniga used the right Biceps Brachii and a tension of 35 kg, Lippold performed his measurements on the calf muscles (Gastrochemius-Spleus muscle group) of the right leg and limited the maximum force to 45 kg. Lippold recognized that his experiment did not include maximum exertions and that a linear relation may not hold for this region.

Other researchers [7,8,9,10] have published data tending to support a linear relation between isometric tension and rectified, filtered, EMG signal amplitude.

From these published reports and initial experimentation, it may be concluded that at a minimum a monotonic relationship exists between RMS detected EMG signals and the isometric muscle contraction level. This will be sufficient for the intended use: pattern

recognition. However, for force estimations and prosthesis mechanical control, the assumption of a linear relationship will be useful although it may not be valid for the full range of muscular effort. There is evidence, however, that the linear assumption may be accurate over a substantial range.

B. Application of EMG Signals in Prosthesis Control

The serious development of electric arm prostheses began immediately following World War II. Early versions were limited to motions with few degrees of freedom, a constraint imposed by their control strategies. The control inputs were supplied by mechanically operated switches activated in various manners, i.e. harness and cable similar to previous mechanical prostheses, mechanical transducers measuring the volume change of a contracted muscle, toe operated switches, etc. These unnatural inputs required considerable practice and concentration for the amputee to control the prosthesis. A comprehensive review article by P. Herberts describes this early development effort [11].

Myoelectric signals had been suggested as a control source, but their successful utilization was hindered until

the development of the integrated circuit and its predecessor: the transistor. In general, the advances in myoelectric prosthesis control have reflected the improvements in microelectronics.

From 1955 to the present, various investigators have produced myoelectrically controlled prostheses. The first use of myoelectric potentials for powered prosthesis control was achieved by Battye, Nightingale, and Whillis [12]. The early devices used the input signals as an ON-OFF control for the prosthesis driver [13,14]. However, to date all myoelectrically controlled prostheses fall within two subcategories: those with pattern recognition controllers using signals obtained from single or multiple electrode sites, and those that employ the EMG signal at a single site to control a single degree of freedom.

The non-pattern-recognition approach may be successfully applied by amputees where few controlled motions (one or two) are desired. Prostheses from this subcategory that use a gross EMG signal amplitude to control a single degree of freedom have been reported [8,10,14,15,16,17]. In general, the applied control schemes are either ON-OFF or proportional to the signal amplitude.

The initial reports establishing the viability of an EMG pattern recognition approach to prosthesis control were

based on the work of two groups. Finley and Wirta at the Moss Rehabilition Hospital in Philadelphia were among the first to study myoelectric signal amplitudes from multiple electrode sites as people performed various motions [18,19,20,21,22]. In Sweden, a research group led by P. Herberts also studied the patterns in EMG amplitudes from multiple potential control sites [23] in addition to investigating the frequency spectrum of EMG signals under various muscle loadings [24].

Both of these groups produced an analog controller based on their results. In essence, the EMG outputs from multiple muscle sites were amplified, rectified, low pass filtered, and then used to drive a series of resistor summing networks. A separate summing network was needed for each control signal and a threshold level to inhibit unintended activation [25]. The pattern recognition algorithm, applied in these cases, assumes a multidimensional space spanned by the set of all input vectors whose components may or may not be orthogonal. The input vectors in this space consist of the set of EMG signal amplitudes measured at the multiple electrode sites while the resistor weighting network describes a set of hyperplanes in the data space. The outputs of the resistor-weighted summing networks satisfy a series of

conditions as to whether the input vector was on the positive or negative side of each hyperplane. The threshold circuit requires that the input vector be at least a minimum distance from the plane before recognition is acknowledged.

Heberts et al. [26] reported on the clinical evaluation of such a controller based on fitting four patients with it. The pickup electrodes were molded into the plastic socket over muscle sites having good signal correlation with the desired movements. Pattern programming was provided by computer analysis of EMG signals and intended motion. The purpose of the computer analysis was to compute the orientation of the various hyperplanes and the resistor summing networks that implement them.

The pattern classification test results were presented as a matrix for each of the four individuals studied. This hardware implementation could assign a pattern to zero, one, or more classes, depending upon the activation of the summing networks and threshold levels. As a result, a correct classification could be obtained in addition to an incorrect one. For example, after training, finger extension was correctly classified as finger extension 99% of the time for one subject. However, 26% of the time for the same subject it was also classified as forearm supination. In another case wrist flexion was correctly interpreted 88% of

the time but 95% of the time it was also interpreted as a command for finger flexion. In general, the intended motion was correctly recognized in almost all cases (90% - 100%), and the mistaken classifications were low. Except for the cases indicated above, and a very few others, the series of motions intended to be of a given class, (grasp, flex, for example) were not misclassified.

Various other approaches controlling artifical limbs. have been attempted. A hierarchy of primitive commands has been proposed by Saridis and Stephanou [27], which was combined with a dynamic Kalman filter model of the upper arm to provide useful motion for the wearer. The concept of treating a prosthesis or normal limb as a multistate system to which a controller may be applied has been further explored by Jacobsen and Mann [28] and Jerard and Jacobsen [29]. This work has led to the development of the "Utah" arm, which is now undergoing clinical testing.

One of the earliest applications of microcomputers to prosthesis control has been provided by the research group led by J. Lyman [30,31]. This work involving microcomputers has been expanded to provide sensory feedback [32,33].

From the surface it would appear that the problems relating the application of EMG signals to the control of an artifical limb have all been solved. However, work still

continues on the fundamental problem of recognizing the desired class or pattern in the detected signals so that this information may be utilized as an input command to the controller [34,35,36].

An alternative approach to prosthesis control using myoelectric signals has been developed in a research group lead by D. Graupe. This approach is based on parameter identification in a myoelectric signal received from a single electrode site [37,38]. In essence, this approach utilizes the normal cross talk from various muscles that is detected at the electrode site, the spatially induced frequency attenuations in EMG signals that are produced in the more distant muscles, and the EMG frequency spectrum shifts resulting from muscle contractions. A microprocessor based system has been assembled [39] and various evaluations made to determine the classification success rates [40,41]. A success rate of 85% for discrimination between five limb functions was reported using a vector space discrimination method, and an 80% classification accuracy was reported with a parallel filter approach. No further elaboration was provided on the distribution of the incorrectly classified data inputs. The classification rate was raised to 99% with healthy volunteers after a training calibration procedure in which the users were encouraged to remember muscle

contraction combinations to produce non-overlapping pattern parameters for the various limb functions.

In summary, there are basic assumptions that this research is based upon. First, motions are caused by muscular exertions but not in the one-to-one-muscle-contraction to degree-of-movement pattern that might be expected or would be ultimately convenient. Instead, in many cases, a synergistic cooperation from many individual muscles is responsible for our various coordinated activities. Second, the amplitude of detected EMG signals is proportional to the muscle contraction levels in the vicinity of the electrodes. In addition, the EMG frequency spectrum shifts to higher frequency contents with greater exertion (but this information was not utilized in the work on this dissertation). As a result, it is clear that EMG signals contain information about the desired movement of a limb. If a portion of the limb is missing, as is the case with an amputee, the pattern of EMG signals in the remaining portion may be sufficient to reconstruct the intended motion.

II. APPLICATION OF MICROCOMPUTERS TO PROSTHESIS CONTROL

Introduction

A microcomputer based pattern classifier and artifical limb controller has a single advantage over approaches using hard wired analog devices. The presence of a microcomputer immediately adds flexibility to the system. Comparisons of size, weight, and power consumption may not be valid except in individual examples. However, a computer based system should in general be able to duplicate the performance of an analog system through the use of similar algorithms.

The algorithm flexibility contributed by the microcomputer allows various methods of pattern classification and prosthesis control to be used. This section will discuss the basis for the observed patterns in signal amplitude and clustering of input data. The algorithm discussion will concentrate on a nearest neighbor classifier of the signal amplitude vectors and the ramifications of such an approach with respect to multidimensional movements and to the training of the microcomputer program. The nearest neighbor classifier will be compared to a Bayes decision rule with an unlimited number of samples to provide a reference point in interpreting the experimental results.

A. EMG Pattern Clustering in Multidimensional Spaces

The electromyographic (EMG) patterns to be discussed in this section are the amplitudes of the signals recorded simultaneously from the electrode array. The hardware has the capability of utilizing eight individual electrode pairs, but these were limited to four to conserve memory space. It was felt that the four electrode pairs would be sufficient to detect unambiguous patterns for three degrees of freedom: i.e. wrist pronation and supination, wrist flexion and extension and a finger flexion (grasp) and extension (open).

For the initial data measurements, the electrode pairs were located around the upper forearm quadrants of a healthy limb (the author's left) in the following manner. The electrode pair for channel one was located over the pronator teres, which assists in forearm pronation, and over the flexor carpi radialis, which controls flexion and abduction of the wrist. The second pair of electrodes was located over the brachioradialis and was activated as a prime mover for elbow flexion. However, with the elbow flexed, this muscle tends to move the radio-ulnar joints to the midpoint from pronation or supination. As a result, the signals from this

channel may be smaller than other channels for the measured motions. The third channel electrode pair was positioned over the flexor carpi ulnaris, a prime mover for wrist flexion. In addition, these electrodes may also detect signals from the flexor digitorum superficialis, a primary flexor of the middle and adjacent fingers. The fourth electrode pair was positioned on the outside rear of the upper forearm and should receive signals from the extensor carpi ulnaris, a wrist extender, and the extensor digitorum, a finger extender [42].

Using a healthy limb (the author's left) six extreme configurations of limb displacement were assumed with exertions in the undesired degrees of freedom as small as possible (an orthogonal set). The limb positions assumed were:

a. Null - forearm as relaxed as possible for system calibration and noise reference measurements

b. Grasp - clenched fist with wrist neutral and neutral forearm supination/pronation

c. Open - fingers extended as far as possible with the other degrees of freedom neutral and relaxed

d. Flex - wrist flexed as far as possible with grasp relaxed and relaxed supinate/pronate

e. Extend - wrist extended as far as possible and the other degrees of freedom relaxed as before

f. Supinate - forearm rotated as far as possible with wrist and fingers relaxed

g. Pronate - forearm pronated as far as possible again with wrist and fingers relaxed

The data recorded during these limb configurations were treated as four dimensional vectors: the output of each of the four electrodes determined the four components. Since each vector component is proportional to an EMG signal amplitude, they are all positive with respect to the zero point. A small region around the origin was excluded for hardware reasons (which will be explained in a later section). Thus, all data vectors will fall into the positive sector (all components greater than zero) of the four dimensional data space, when plotted.

An an orthogonal coordinate system is assumed, and the average value of the vectors recorded for the various limb configurations are shown in Figures 1 and 2. Figure 1 is a three dimensional representation of the data projected so that the fourth component is zero, while Figure 2 is a projection with the second component zero. From these figures it can be seen how the average EMG amplitude patterns are positioned, at least for large exertions, in a four dimensional space. The null case "A" is offset from the origin by an amount also included in each data point.

For pattern recognition applications with simply

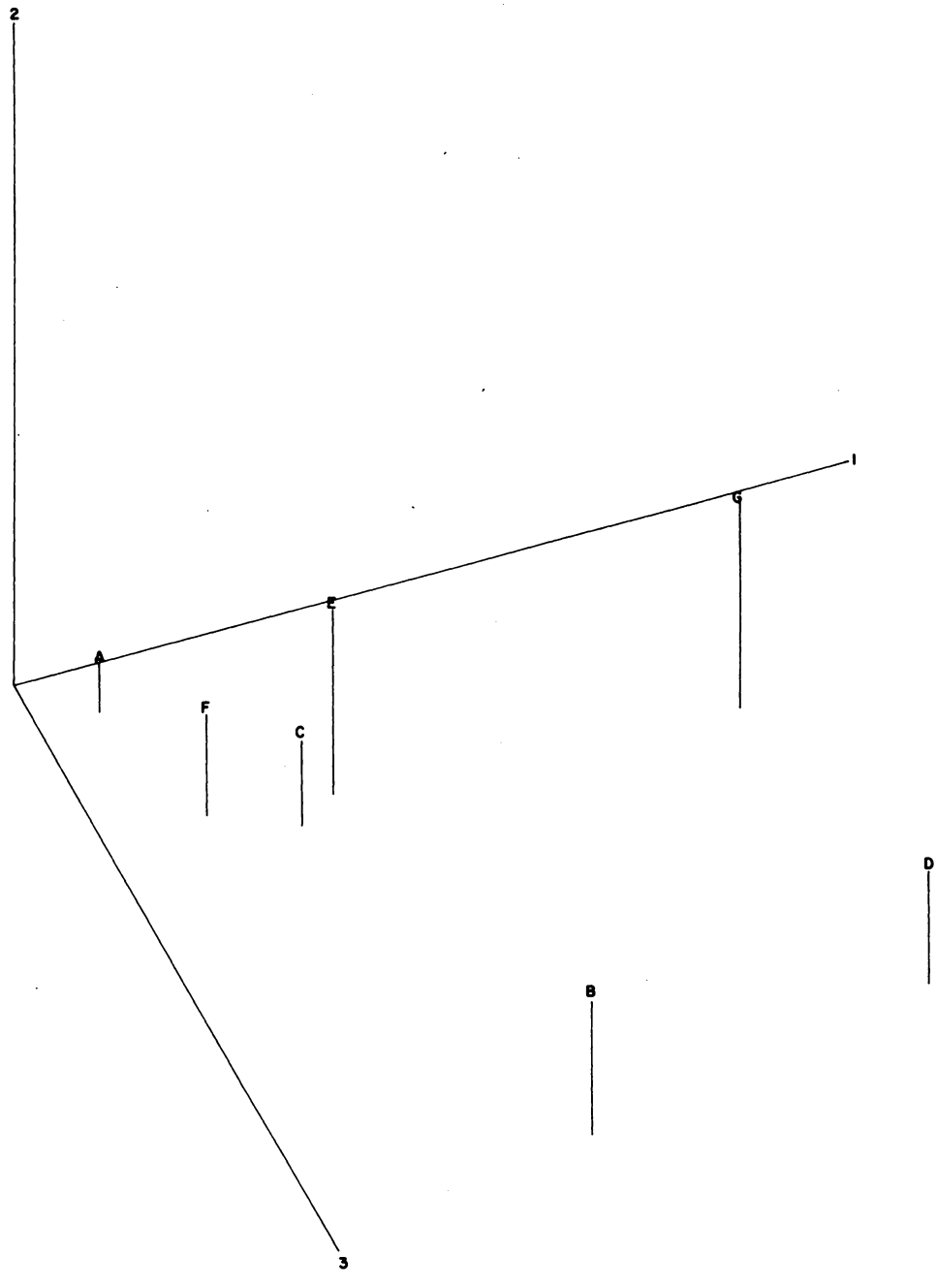


Figure 1 Average values of sample data clusters projected onto the fourth coordinate dimension

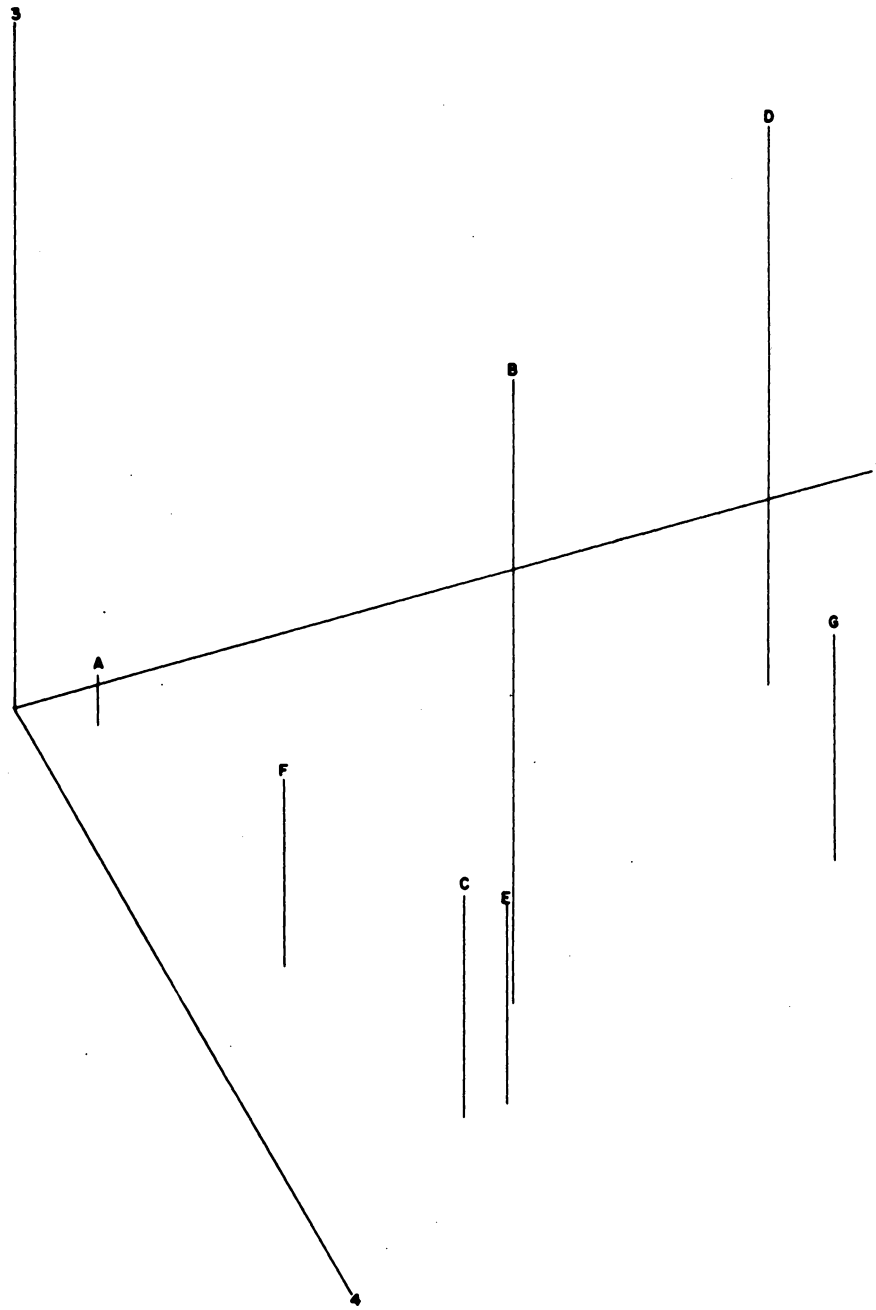


Figure 2 Average values of sample data clusters (Same as Figure 1) except projected onto the second coordinate dimension

defined pattern boundaries, the spreading (covariance matrix) of the data clusters in addition to the mean values are the critical parameters. Figures 3 and 4 illustrate the data clustering for the various limb configurations; the individual vectors shown in the first two figures are used.

The figures show a definite clustering of the data for the various limb configurations. However, this was a result of a single experiment in which muscle contraction was held at a uniform effort. In an actual working environment, a complete spectrum of limb positions and muscle exertions would be experienced. Figures 5 and 6 present data for such muscle exertions along a continuum of movement. Vectors labeled "A" are the calibration (completely relaxed) data while "B" and "C" represent the grasp and wrist extend vectors previously presented in Figures 3 and 4. The vector points labeled "D" and "E" were recorded for various levels of muscle contraction effort for the grasp and wrist extend limb positions. In these two figures it can be seen that the data clusters tend to elongate toward the relaxed state (cluster "A"). As a result, the pattern clusters in the four dimensional data space tend to extend radially outward from the zero state.

An additional hardware calibration problem may be observed in data clusters "B" and "D" in Figure 5. This limb

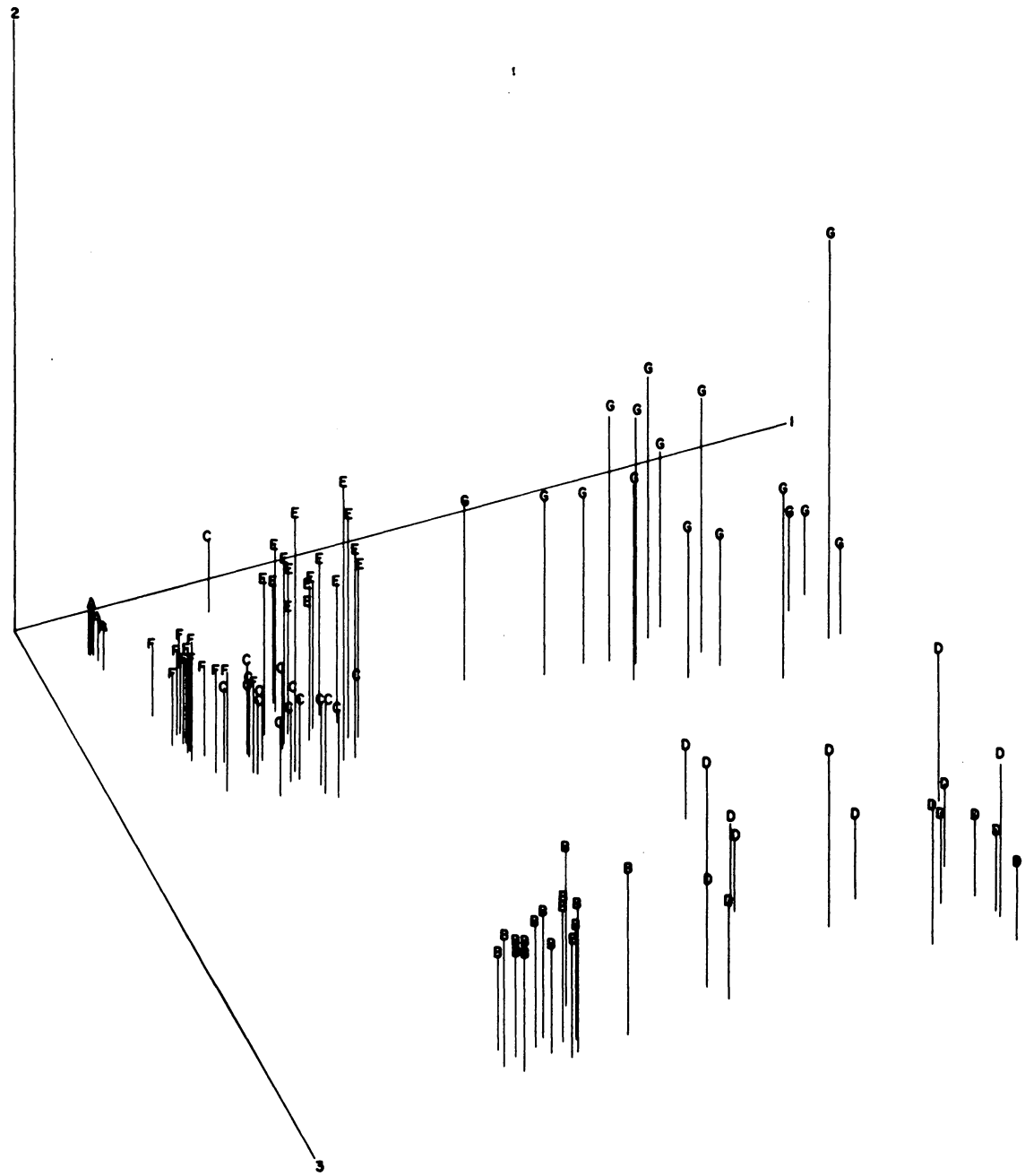


Figure 3 Complete sample data clusters as shown in Figure 1

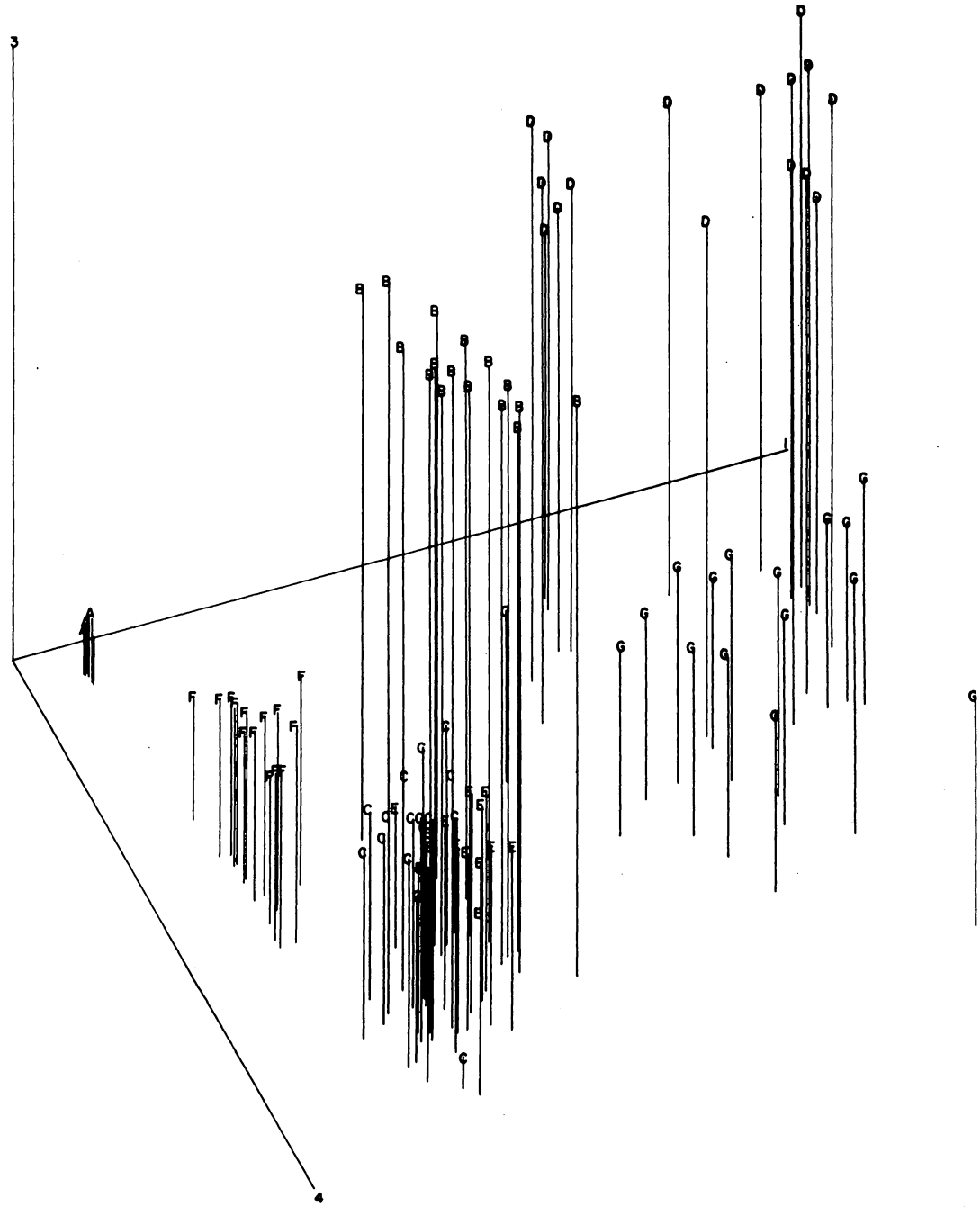


Figure 4 Complete sample data clusters as shown in Figure 2

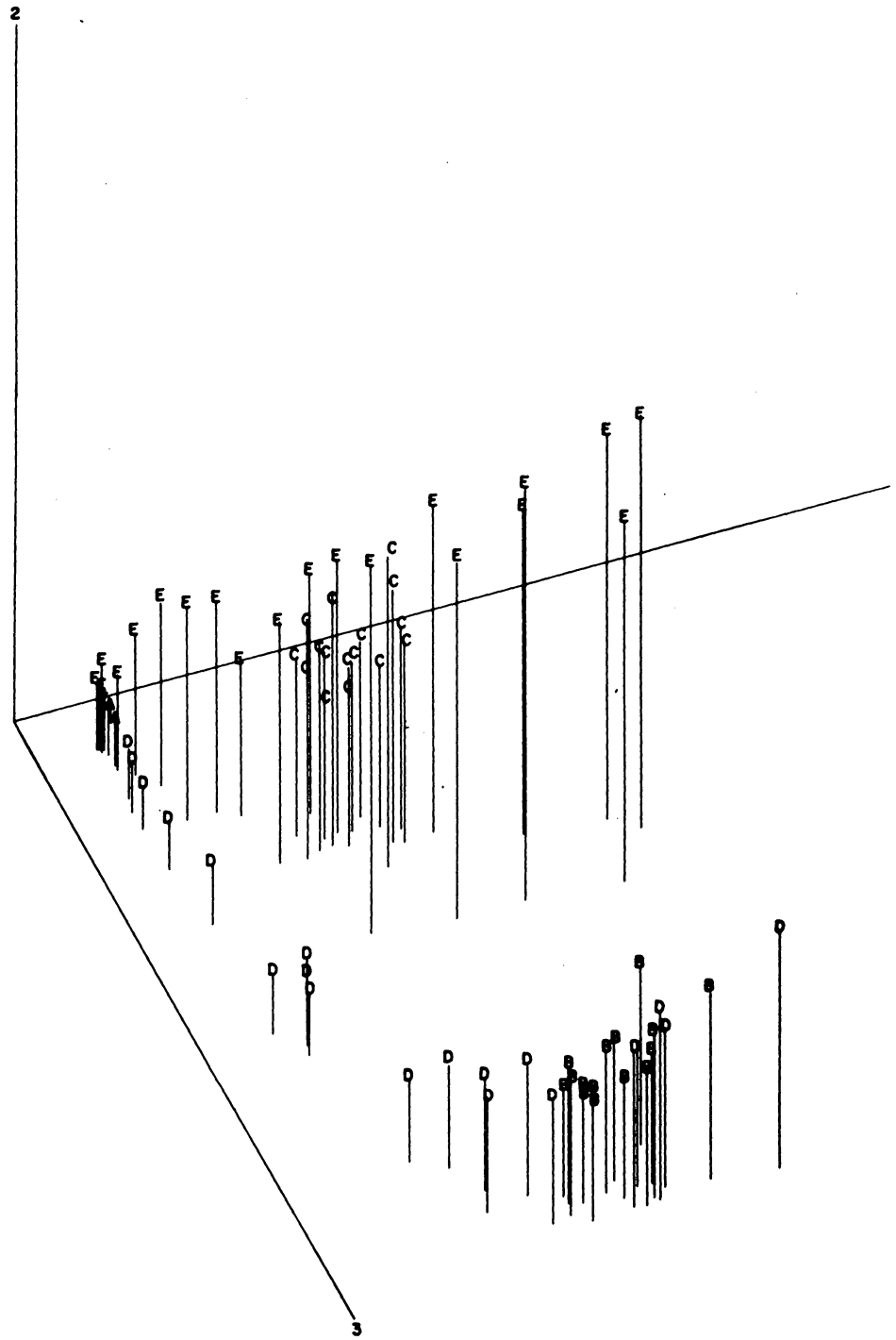


Figure 5 Grasp and wrist extend data from Figure 3 and 4 with additional data from a continuum of exertions

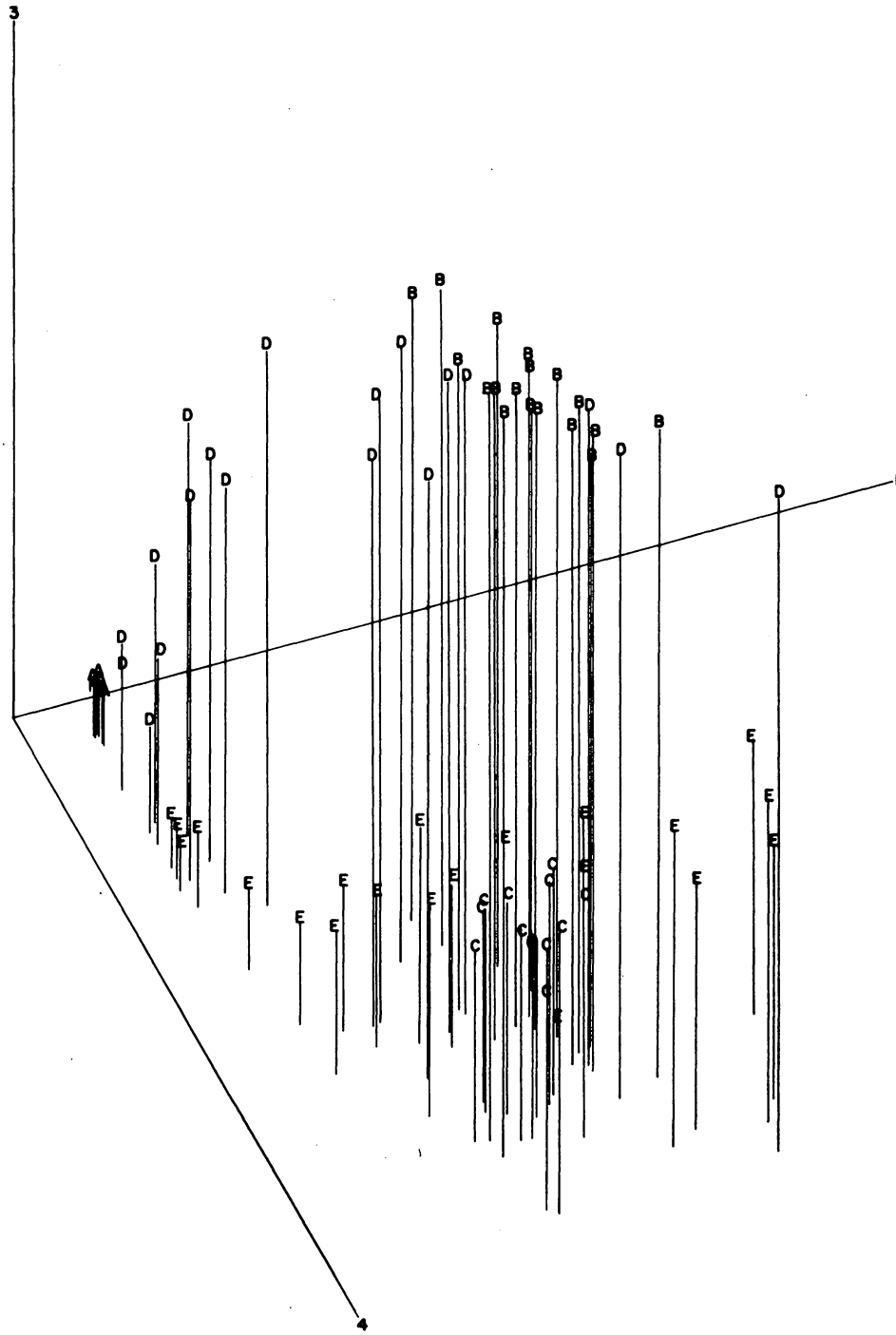


Figure 6 Grasp and wrist extend data for fixed and continuum levels of exertion projected into the second coordinate

position is strongly correlated with the EMG signal received from channel three. When the amplifiers in channel three saturate, as seen in the figure, increasing exertions tend to produce crosstalk and more weakly correlated signals from other channels such as channel one. As a result, the cluster is not radial; it exhibits a sharp kink. This problem may be alleviated by reducing the gain in the offending channel to scale it back to a linear region of the amplifiers.

B. Pattern Recognition Algorithm

The original approach to the pattern recognition algorithm was to be a digital implementation of the analog pattern classifier employed by the Herberts group [25]. A discriminant function for each movement can be expressed as:

$$F_j(X) = a_{j1} x_1 + a_{j2} x_2 + \ldots + a_{ji} x_i + c_j \quad . \quad (1)$$

This is a weighting of the input data vector performed by an operational amplifier and resistor summing network.

Equation (1) may also be interpreted as a scalar product between the normal vector to a plane and the input data vector. The discriminant function can then be

visualized physically as the distance to the plane. For multiple movements, the sets of weighting coefficients can be assembled into a matrix for efficient numerical operations. The pattern recognition problem reduces to the determination of planes in the multidimensional data space that would separate the various clusters. Although preliminary experiments have shown that the matrix calculations could be completed within a reasonable cycle time with a microprocessor and hardware multiplier, this approach was not taken for several reasons.

The perceived problems were the determination of the a priori probabilities for the various motions and the stability of the clusters with time. To implement a Bayes decision rule in the orientation of the planes, the sample data set must also contain information on the probability of occurrence of a certain motion. The weighting coefficients can then be skewed to minimize the total error rate of the pattern classifier. However, the determination of the a priori probabilities for the occurrence of a movement can not be determined since it will be closely related to an individual's daily activities. The stability of the pattern clusters with time was viewed to be a problem that would continually require recalibration of the decision rule. For example, in cold environments, the lack of perspiration on

the skin tends to increase the impedance seen by the metal electrodes and degrades signal quality. This could change cluster locations and require a new sample data set to be recorded in order to recompute the weighting coefficients.

To eliminate these problems and provide a more versatile prosthesis controller a nearest neighbor pattern classifier was used. A nearest neighbor classifier compares an input vector with a previously established list and uses a predefined metric to determine which list entry is closest to the input value. The pattern or class associated with the chosen list entry is assigned to the input. The advantages of such a classifier are ease of implementation and adaptability to changes. A potential disadvantage that may occur for long reference lists is a reduced cycle time; the entire list must be searched before a nearest neighbor is identified.

The data stored in the reference list for each of the pattern classes forms clusters in the data space. Physically, the unclassified input vector is assigned to the cluster to which the nearest neighbor element belongs. The reference list contains information on the probability density of each class in the data space. However, the a priori probability of the occurrence of each class may or may not be contained in the incidence of that class in the

reference list.

The error rate for a nearest neighbor classifier will always be higher than the Bayes error rate. However, for a large number of samples, the nearest neighbor error rate is never larger than twice the Bayes rate [43]. As a result, if the actual probability of an input vector belonging to a specific class is close to unity, as determined by a Bayes decision rule, a nearest neighbor classifier would have a similar probability of also choosing that class.

The information to be classified, as is shown in Figures 5 and 6, is contained in the direction of the elongated clusters from the null point. Thus, the input data vectors, if greater than a preset threshold, were normalized to a standard length. The preset threshold was employed to determine that a valid EMG vector was available and not the residual signal from completely relaxed muscles. The various components (X1 TO X4) were then proportional to the direction cosines. The magnitude of the input vector containing the overall muscular exertion information was saved for input into the prosthesis controller after a movement goal was identified by the classifier.

C. Simultaneous Multidimensional Control

Multidimensional prosthesis control has been implemented in various ways. Graupe's novel approach [40] uses multiple digital filters to estimate the value of the next EMG data point from a single electrode pair:

$$Y_k(\text{est}) = a_1 Y_{k-1} + a_2 Y_{k-2} + \ldots + a_n Y_{k-n} . \qquad (2)$$

The weighting coefficients provide a best estimate of the next EMG data point for a given limb configuration. Thus, to implement a multidimensional control, similar parallel filters are employed for each limb motion desired. The filter producing the smallest running error (difference squared) (if the error is below a preset threshold) determines the class of the intended motion.

The analog signal based controller developed by Herberts et al. [25], which uses a resistor summing network, has the capability of simultaneously controlling multiple movements. Since the pattern discriminator is a set of hyperplanes in the data space, any input data vector may be positioned far enough from the positive side of more than one plane to activate those servomotors. This provides the apparent simultaneous control of multiple complex motions as

the input vectors move around the data space with time.

Before proceeding with the control capabilities provided by a nearest neighbor pattern classifier, we will digress to examine the relation between received EMG signals (muscle contractions) and the resulting motion.

A muscle may only provide a tension force between its origin and its insertion. The relationship between the applied muscle tension and the resulting movement may be divided into four areas that are of interest to this investigation [1]. First, a sustained force movement is one in which a continuing resistance is experienced. For example, in lifting a weight the object may be raised, lowered, or supported in a stationary fashion with roughly the same muscular exertion. In general, the mover muscle groups (agonists) are contracted while the antagonists are relaxed. Second, a passive movement is one which takes place without a continuing muscle contraction. This may result from inertial effects, gravitational forces, or other external influences but in all cases is characterized by movement without muscular exertion. Third, a ballistic movement is a compound type of movement. Initially it starts as a sustained force movement with body parts accelerated by agonists followed by an inertial movement and finally a sustained force movement with deceleration provided by

antagonists. Fourth, a guided movement is one in which steadiness and accuracy are of importance. In this case both the prime movers and the antagonistic muscle groups are contracted in opposition to provide fine motion control.

The force requirements from an individual muscle and consequently the information pattern available to the computer will depend on the type of movement. The most common is the sustained force movement since there are many resistive forces created by the stretching ligaments, the antagonistic muscle groups, and the flexing joints. However, the information available to the computer may be misleading since a measure of the resistive forces encountered is absent. The muscle contraction pattern as seen by the computer would, for example, present little change in slowly lifting, supporting, or lowering an object. Likewise, no pattern change would be seen for guided movements, while passive motions would provide no indication of muscle contractions.

Muscle contraction level data seen by the control computer must be interpreted with care to provide a reasonable approximation to the desired movement. The control strategy which was also employed by other research groups (Finley et al., Herberts et al., and Graupe et al.), uses a velocity control in which a detected pattern class

energizes the servo motors to produce a constant velocity motion. When the input data drops below a preset threshold or when the pattern class is no longer seen, the servomotors are turned off. Preliminary qualitative experiments indicated that the velocity control was not difficult for the user to operate and that it was easy to implement.

For this study, the output servo drivers drive a "set point" style servo in which a control loop within the servo causes it to track the input data. Velocity control was implemented with set point servomotors by storing a data word for each servo. This setpoint is periodically output to the servo controller after being updated by the pattern classifier routine. To synthesize a velocity control the data words are incremented or decremented by various amounts (with underflow and overflow constraints), depending on the pattern class.

Multidimensional simultaneous motions may be controlled easily by using the coding of the pattern classification. The pattern class code was designed to handle the operation of four servomotors with the following structure:

Bit	7	6	5	4	3	2	1	0
Function	INC	DEC	INC	DEC	INC	DEC	INC	DEC
Servo	0	0	1	1	2	2	3	3

Thus a class code of 81 (hex) would translate into

increasing the set point value for servomotor zero while decreasing the setpoint for servo number three. The utility of this implementation is immediately obvious since most movements are not composed of single-degree-of-freedom increments but are fluid patterns with multiple directions.

The flexibility resulting from this interpretation of the pattern classes allows the user to associate a specific motion with an arbitrary EMG pattern. This was thought to be potentially useful in cases in which different but desireable movements resulted from EMG patterns that could not be resolved with sufficient accuracy by the classifier. Under these circumstances, an alternate pattern could be associated with the desired class for activating the desired movement. The complexity of the movement is completely arbitrary; it depends only on the pattern classification coding. This allows for all possible clusters in the data space to be associated with a movement that is completely at the discretion of the user.

D. Inherent Training Capability of the Controller

A nearest neighbor classifier has an inherent supervised training capability. As was previously described, the

microcomputer compares the input data with a stored list and selects the class of the nearest neighbor using a Euclidean distance. The adaptability of this style of pattern classifier stems from its ability to allow the user to add to and alter the reference list.

The training capabilities are best presented through the following description of the initial operation of the prototype. The prosthesis user would train his appliance in a similar fashion. When the reference list is empty and the system up and running with four electrode pairs in place, no movement occurs since a default class of zero is chosen. After the program was switched into the training mode, four vectors each were added to the lookup list under the grasp and open classification. Upon returning to the run mode, the system correctly classified grasp and open motions (without error for this short run). However, as would be expected, all other intended motions providing EMG signals larger than the treshold were also classified either as grasp or open. This apparent error actually demonstrates an advantage of tnis system. As soon as he discovers the lack of a desired motion, the user can immediately switch to the training (table building) mode, and with the proper classification code, he can add entries to the nearest neighbor list to implement the missing function. In this case, entries were

added for wrist flex and extend and later for forearm supination and pronation. In a final version, static CMOS memory could provide low power nonvolatile storage for the pattern reference list, even when the system is turned off.

The training mode contains provisions for pruning the nearest neighbor list to keep it within reasonable bounds. This can be accomplished in various ways. The vector table entry farthest from the class mean may be removed from the list arbritrarily or upon adding a new entry. Note: new entries may also be made without deletions. In addition, the last list entry for a class may be removed if this is desired.

The entries in the nearest neighbor reference list have no restrictions on the pattern classification or on the number of entries associated with a given pattern (except memory limits). This provides another variable in the prototype system. Instead of just using the nearest neighbor to determine the pattern class, the system may also use n-nearest neighbors with a voting procedure for pattern classification. The additional variable allows the density of a pattern class to also skew the final pattern determinations.

III. DESCRIPTION OF HARDWARE

Introduction

The microcomputer will be briefly discussed to show how its features influenced the design of the various peripherals. A complete design and interface description will be provided for the multiplier, A/D converter, and servo driver, followed by the analog design for the active EMG electrodes.

A. Micromputer and Interface

The microcomputer employed in this project was an Intel single board computer (80/10) using an 8080A central processor. The microcomputer contained 1024 8-bit words of read/write memory starting at location 3C00 (hex) and sockets for four (2708) chips of 1k by 8 bits of read-only memory starting at address location zero. A complete memory map is shown in Figure 7. The first two ROM chips contain the system monitor, which also controls the microcomputer's front panel. The third ROM chip, which originally contained programs for reading and writing paper tape, programming

ADDRESS (hex)	FUNCTION
00	
.	ROM socket #1
.	(monitor)
3FF	
400	
.	ROM socket #2
.	(monitor)
7FF	
800	
.	ROM socket #3
.	(pattern recognition program)
BFF	
C00	
.	ROM socket #4
.	(pattern recognition program)
FFF	
1000	
.	
.	UNUSED
3BFF	
3C00	RAM reserved for monitor
3C01	
3C02	
.	
.	RAM available to user
3F90	
3F91	
.	
.	RAM reserved for monitor
3FFF	
4000	
.	
.	UNUSED
FFFF	

Figure 7 SBC 80/10 microcomputer memory map

2708 EPROMS, moving blocks of memory, and transferring PROM contents into memory from the front panel socket, was not needed for the monitor operation. This third ROM socket and the fourth (unused) socket were employed for the EMG pattern recognition program.

The 80/10 microcomputer also contains two 8255 programmable interface chips, each with 24 programmable input/output signals. The first 8255, located at output ports E4 to E7 (hex), is dedicated to the front panel control. The second, located at ports E8 to EB (hex),controls the 24-pin front panel socket and in addition has the 24 I/O lines available at an output connector in parallel with the socket. The second programmable interface chip forms the basis for all peripheral control signals and data flow. Port E8 (hex) is specified as the data output channel from all peripherals. Port E9 (hex) serves as the data input channel from all peripherals. Port EA (hex) is programmed as an output port and passes the control signals to all peripherals for operational control and data flow. A system block diagram, Figure 8, shows how the various peripherals interface with the microcomputer.

The low order four bits of port EA (hex) are used to generate strobe signals through a 74LS138 three to eight line decoder for the following operations:

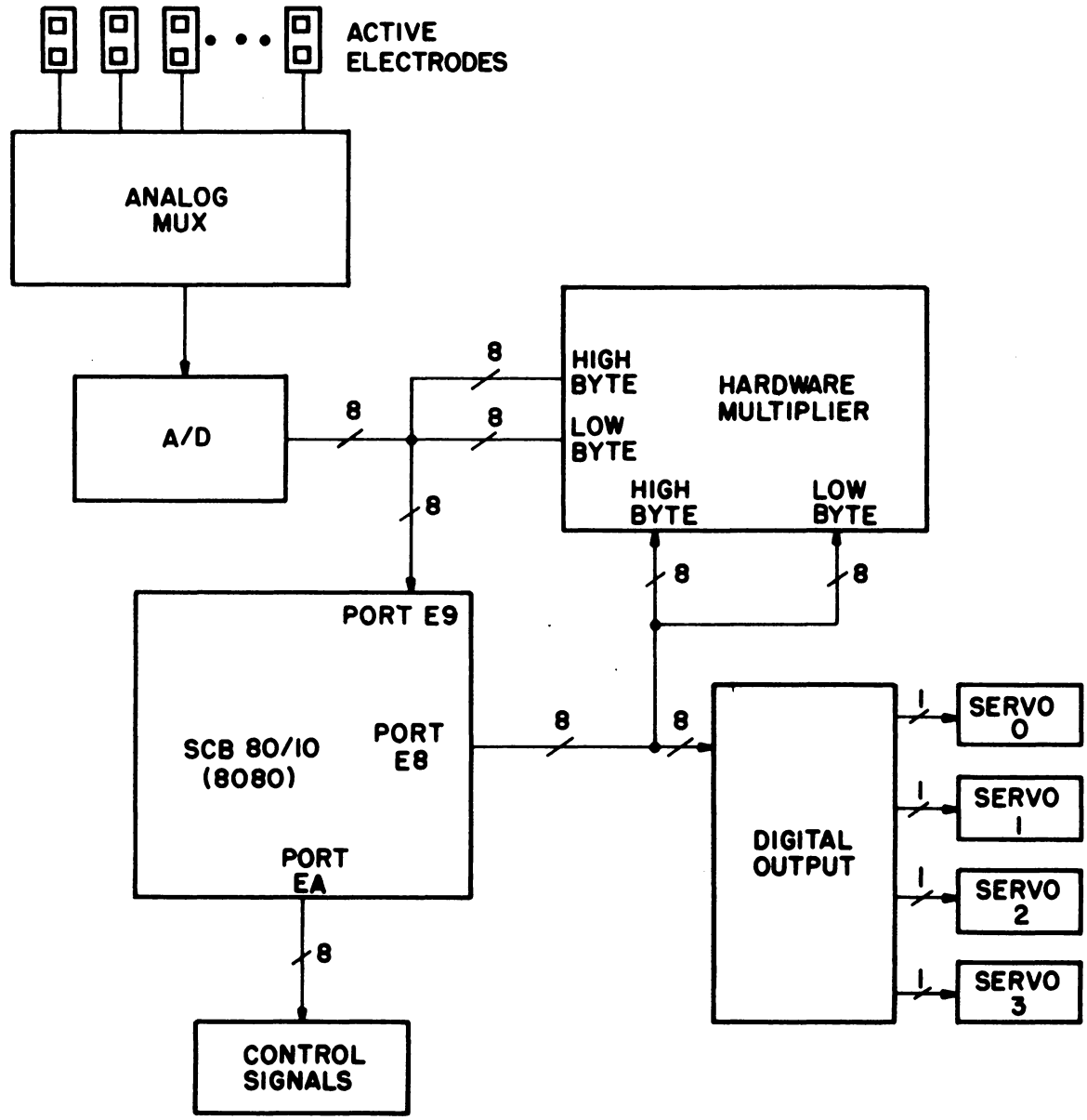


Figure 8 Hardware block diagram for prosthesis controller

Bit 2	Bit 1	Bit 0	Output	Control Function
0	0	0	Y0	- no connection
0	0	1	Y1	- load multiplier (8 bits)
0	1	0	Y2	- load multiplicand (8 bits)
0	1	1	Y3	- output enable product register (low byte)
1	0	0	Y4	- output enable product register (high byte)
1	0	1	Y5	- output enable A/D data register
1	1	0	Y6	- two's complement multiplication
1	1	1	Y7	- straight binary (magnitude) multiplication

Bit 3	0 - enable control strobes
	1 - disable control strobes

The upper four bits, i.e. 4,5,6, and 7 of port EA (hex), are used to output data to the servo motor controller. Bits 4 and 5 specify the servo being controlled. Bit 6 must be low to enable the servo controller, and bit 7 is strobed from a zero to a one and back to generate the write enable signal necessary to transfer data into the servo controller memory.

B. Hardware Multiplier

The hardware multiplier (see Figure 9) was designed at the initiation of this project in anticipation of the need for rapid calculations. The design is a result of tradeoffs between ease of interfacing to the microcomputer output

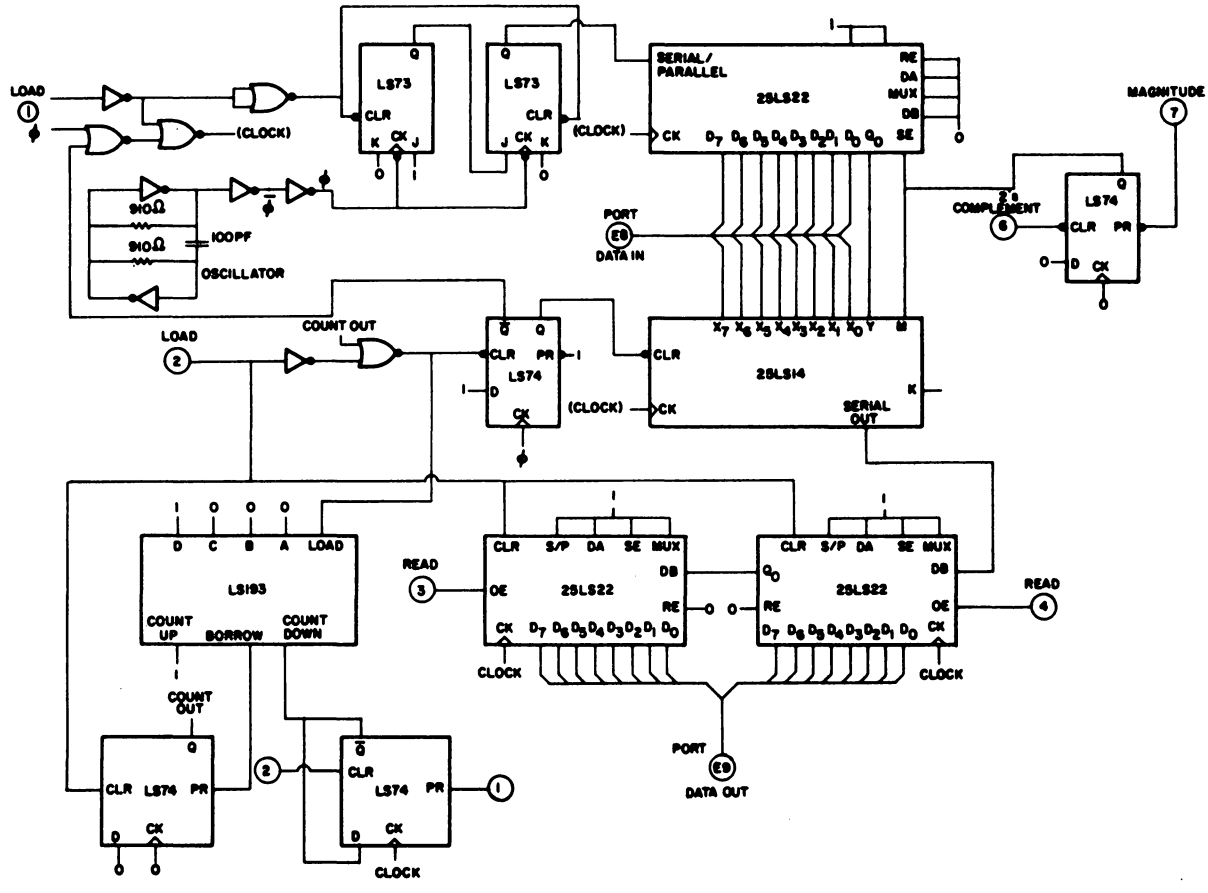


Figure 9 Schematic diagram of the 8 bit by 8 bit hardware multiplier

lines and the highest possible throughput.

The multiplier is based on the Advanced Micro Devices serial/parallel multiplier (25LS14), which computes a two's complement product. In addition, straight binary multiplication may be performed by proper selection of the mode input and the sign extended input bits. As constructed, the mode is selected by strobing the set or reset of a flip-flop using output 6 (hex) or 7 (hex) from port EA (hex), which was discussed in the previous section.

The multiplication rate is determined by an internal clock running at 3.3 MHz. A full 16 bit product is available about 6 microseconds after a multiplication is initiated. During operation, the multiplier byte is latched out by the computer onto port E8 (hex), and a strobe signal to load this value is generated on port EA (hex). The strobe signal (load 1 in Figure 9) is synchronized to the multiplier clock and generates a single clock cycle load pulse into the input shift register (25LS22). The multiplicand is then latched into the output port (E8 (hex)) and a strobe signal(2) generated on port EA (hex). This second strobe (load 2 in Figure 9) is also synchronized with the multiplier clock to generate the clear for the multiplier. The multiplier clear loads the multiplicand and on the trailing edge starts the multiplication cycle. The full 16 bit product (of two 8 bit

bytes) is available after 18 multiplier clock cycles. This product is read into the computer through port E9 (hex) by strobing the output enables (3) and (4) of the multiplier output registers.

The microcomputer control of the hardware multiplier is illustrated in the following example. The computer (8080) calls this subroutine with the multiplier in the accumulator and the multiplicand in the B register. The output ports have previously been set up for I/O and the multiplication mode specified.

```
MULT;  OUT 0E8H     ;LATCH OUT MULTIPLIER
       MVI A,01H
       OUT 0EAH     ;START STROBE (1) OUTPUT
       INR A
       OUT 0EAH     ;STOP STROBE (1), START STROBE (2)
       MOV A,B
       OUT 0E8H     ;LATCH OUT MULTIPLICAND
       MVI A,03H
       OUT 0EAH     ;STOP STROBE (2), START STROBE (3)
       NOP          ;TIME FOR MULTIPLICATION
       IN  0E9H     ;INPUT LOW BYTE OF PRODUCT
       MOV B,A      ;SAVE LOW BYTE
       MVI A,04H
       OUT OEAH     ;ENABLE HIGH BYTE, STROBE (4)
       IN  0E9H     ;INPUT HIGH BYTE OF PRODUCT
       RET
```

The multiplication is performed during the "NOP" cycle, and the instruction is fetched for the following data input. From the example it may be seen that most of the multiplication subroutine is dedicated to generating strobe

pulses, a constraint imposed by the microcomputer interface design.

A benchmark test for the multiplier was conducted by multipling two 8x8 matrices in a continuous loop. Observation of a specially generated strobe indicated that the total cycle required approximately 60 ms.

C. Analog-to-Digital Converter

An analog-to-digital converter is required for converting the EMG signal amplitudes into a form usable in the microcomputer. The circuit employed (see Figure 10) is a free running successive approximation converter.

The free running feature was desired to minimize the computer overhead in communicating with the A/D. The end of conversion signal from the successive approximation register is used to trigger the start convert input signal and to simultaneously clock the data byte into a tri-state output latch. The output of the tri-state latch is connected to port E9 (hex) and is enabled by a 05 (hex) output on port EA (hex) (see section III(A) on the computer and interface). The computer may obtain the most recent conversion data by executing the following instructions:

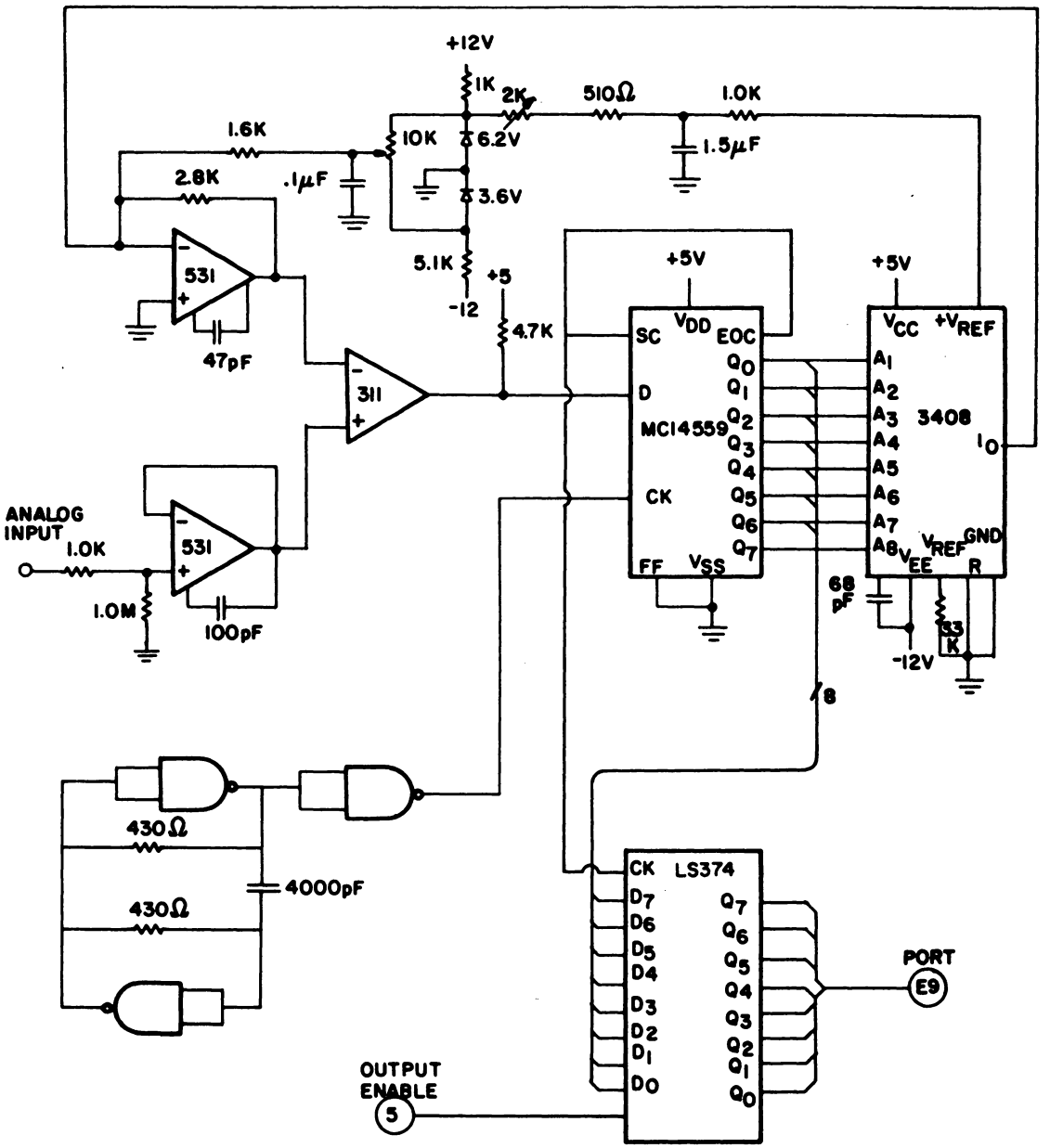


Figure 10 Schematic diagram of free running analog to digital converter with latched tri-state outputs

```
MVI   A,05H
OUT   0EAH     ;ENABLE A/D OUTPUT BUFFER
IN    0E9H     ;GET DATA
```

The A/D was designed to convert a 0 to 10 volt input signal into a straight binary value of 0 to FF (hex). A calibration adjustment was included for both range and offset. The local oscillator runs at 192 KHz so that complete conversions are obtained at about a 20 KHz rate. A sample and hold was not included since the data would be slowly varying over the conversion period (see the section on the analog multiplexor).

D. Servomotor Drivers

As an aid in training the microcomputer based pattern classifier and to illustrate the potential for application as a prosthesis controller, a digital interface was constructed to drive four "Heathkit" model aircraft servomotors. The servomotors were high torque digital proportional servos, model GDA-1205-8. The servomotor's input signal is a TTL logic level positive pulse, which varies in duration from 1 ms to 2 ms and repeats at a 60 Hz rate. The servomotor output setting (0 to 180 degrees or 0 to 90 degrees, jumper selectable) is directly proportional

to the pulse width. Thus, the most desirable computer interface would take an eight bit data byte and convert the numerical value into a pulse width in the specified range without further computer supervision.

The servomotor driver interface shown in Figure 11 operates as follows. The data word on port E8 (hex) is buffered and supplied to the data inputs of two 74LS670 four bit by four bit memory chips. This implements four eight bit data registers addressed by the high four bits from port EA (hex). The data registers have the added feature that data can be read from one location while it is simultaneously being written into another location. As a result, the computer involvement in controlling the servomotors is limited to outputting the setpoint data byte to the correct register.

During operation, the servomotor driver is governed by two clocks. A 256 kHz clock is used to drive a counter that converts the data byte, 00 to FF (hex), into a zero-to-one millisecond pulse spacing. A 60 Hz clock is used to control the pulse repetition rate.

As is shown in the Figure 11, the 60 Hz clock drives the second section of a 74LS123 monostable multivibrator, which generates a short isolation pulse, loading the hardwired preset data (1011 binary) into the pulse counter

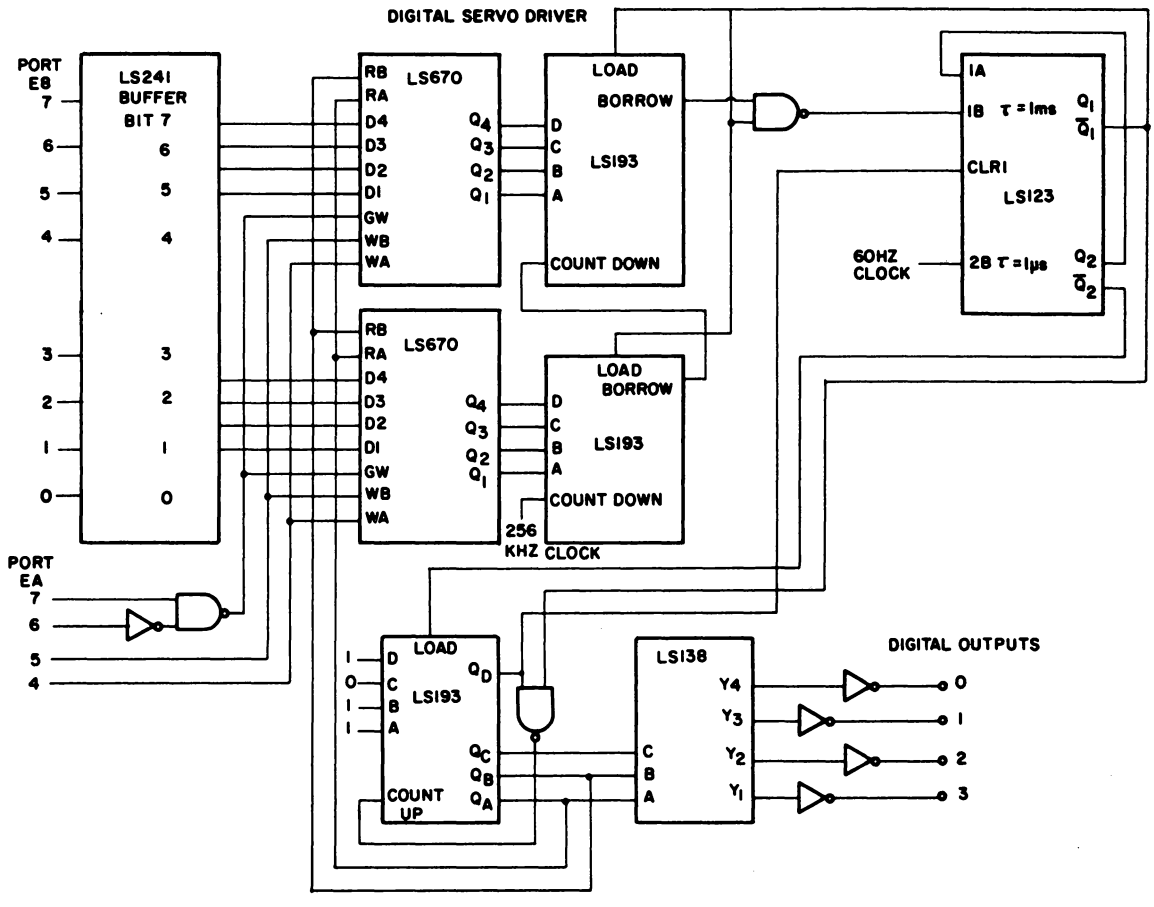


Figure 11 Schematic diagram of the digital servomotor-driver

and triggering the first section of the multivibrator. The first section produces a 1 ms wide pulse that clocks the pulse counter (bottom 74LS193) and loads an eight bit synchronous down counter (2 - 74LS193's) with the addressed data byte. The down counter starts counting in the trailing edge of the 1 ms load pulse and converts the loaded data byte into a zero to 1 ms delay (depending on the value) before the borrow pulse is generated. The borrow pulse retriggers the first section of the multivibrator causing the sequence to repeat using the second data byte. The output signals that drive the servomotors are generated from the outputs of the pulse counter. The time interval between clock inputs to this counter is the sum of the 1 ms data load pulse to the down counter and the time this down counter takes to reach zero and generate a borrow pulse (i.e. 1 ms to 2 ms total). Decoding the pulse counter states produces the four digital output signals to control the servomotors.

In practice, it was found that a low impedance power supply was necessary to power the four servomotors. Although the quiescent current was about 100 mA, the impulse current drawn when a servomotor started or stopped was almost 1 amp. Unless the power supply could respond to this widely ranging demand the movement in one servomotor would cause errors in

the others, which resulted in all servomotors continually "hunting."

E. EMG Electrodes

The most important pieces of experimental equipment for EMG activity measurements are the electrodes. The characteristics and limitations of the electrodes impose constraints on the remainder of the system. Two types of electrodes were considered: remote electrodes from which the detected EMG signal is transmitted along three wires (two for the differential signals and one for ground) to a remote amplifier, and active electrodes in which a local differential amplifier is included in the electrode design to increase the signal level and lower the output impedance. Initial experimentation indicated that the undesirable detection of external noise (especially 60 Hz) could be minimized by placing the differential amplifier as close to the detection electrodes as possible. As a result, it was decided to stack the differential amplifier printed circuit board and the EMG amplitude detector board on top of the electrode board to form as compact a package as possible. Only three wires (for power, ground, and the EMG signal

amplitude) were necessary to connect the electrode assembly to the computer system.

The differental amplifier, implemented with the first two operational amplifiers, shown in Figure 12, provides a nigh input impedance for the detected signal and a design gain of 201. This is followed by a low pass 2-pole Butterworth filter constructed around the third operational amplifier. The low pass filter was designed for a frequency rolloff starting at 1500 Hz and a low frequency gain of five. The total gain for this amplifier/filter stage is about 1000. The fourth operational amplifier in the chip (MLM324) supplies a pseudo-ground; therefore, the circuit does not require a positive and negative power supply.

This amplifier and electrode configuration provided excellent immunity to external noise (see Figure 13a) while also supplying a good quality signal when the muscles in proximity to the electrodes were under contraction (see Figure 13b). Note that the signal obtained in Figure 13a was with the muscles as relaxed as possible.

The data desired for the pattern recognition algorithm was the EMG signal amplitude instead of the EMG signal itself. Therefore, an additional signal processing capability was designed to process the EMG signal into its

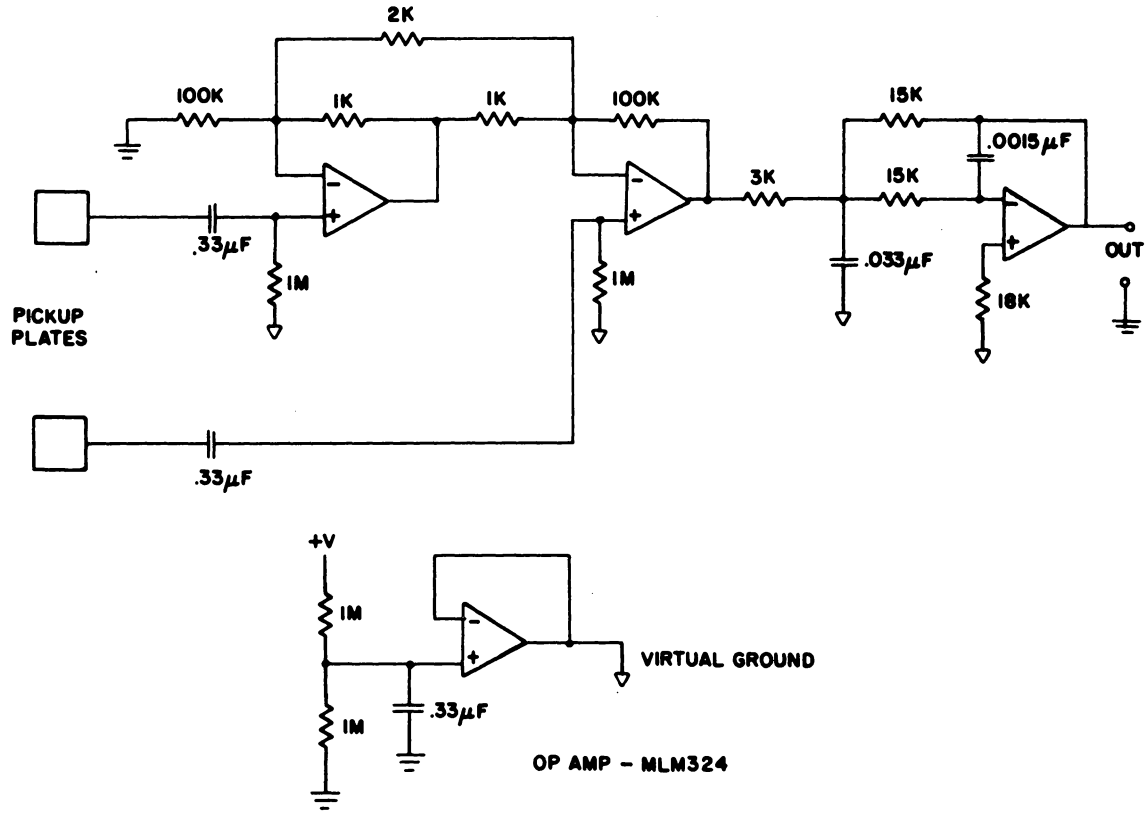


Figure 12 Schematic of the high impedance differential amplifier for EMG signal detection and low pass filtering

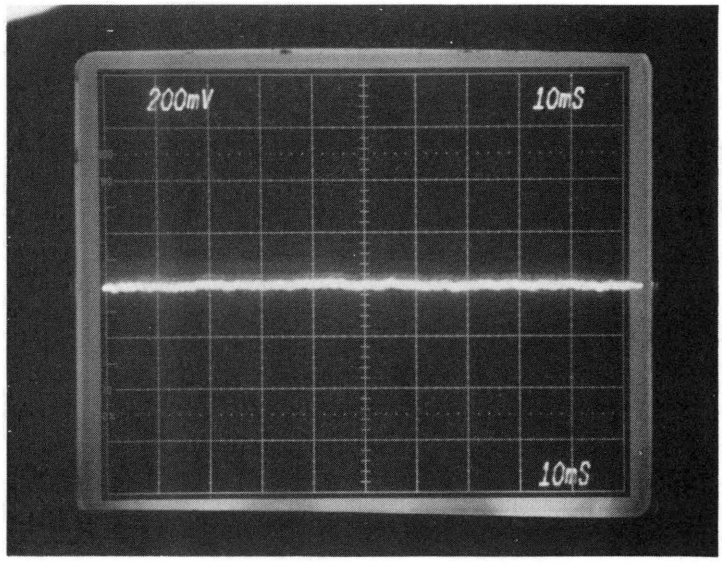
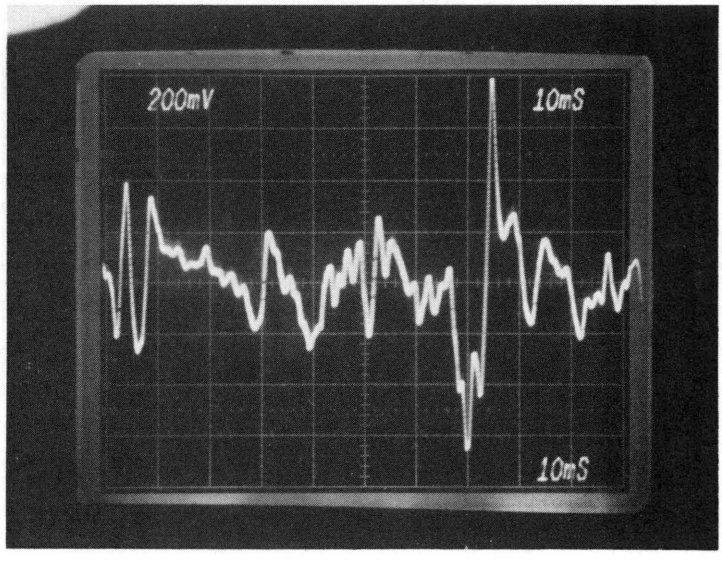


Figure 13a. Output of the differential amplifier and low pass filter with pickup electrodes over relaxed muscles



Figrue 13b. Signal output from amplifier and low pass filter with electrodes over a contracted muscle (forearm with clenched fist)

modulation envelope. This was performed with the circuit shown in Figure 14, which was constructed on a second printed circuit board (amplitude detector) stacked on top of the electrodes. The first operational amplifier (MLM324) on this board forms a two pole high pass Butterworth filter that rolls off below 100 Hz with a pass band gain of 10. This filtering is desirable because it attenuates any residual 60 Hz pickup and eliminates the very low frequency signals associated with the movement of the electrodes (motion artifact). The high pass filter is followed by a half wave rectifier to demodulate the EMG signal and a non-inverting low pass filter to smooth the envelope. The time constants of the smoothing low pass filter were chosen so that the final output signal's rise and fall time would closely track the most rapid muscle contraction/relaxation response expected. The fourth operational amplifier in the chip was utilized to produce a pseudo-ground.

An example of the total electrode package response is shown in Figure 15. The top signal is the amplified EMG signal from the differential amplifier board. The bottom signal trace is the corresponding amplitude envelope taken from the output of the electrode package. These signals were produced by a quick muscle contraction followed by an interval in which the muscle was relaxed as completely as

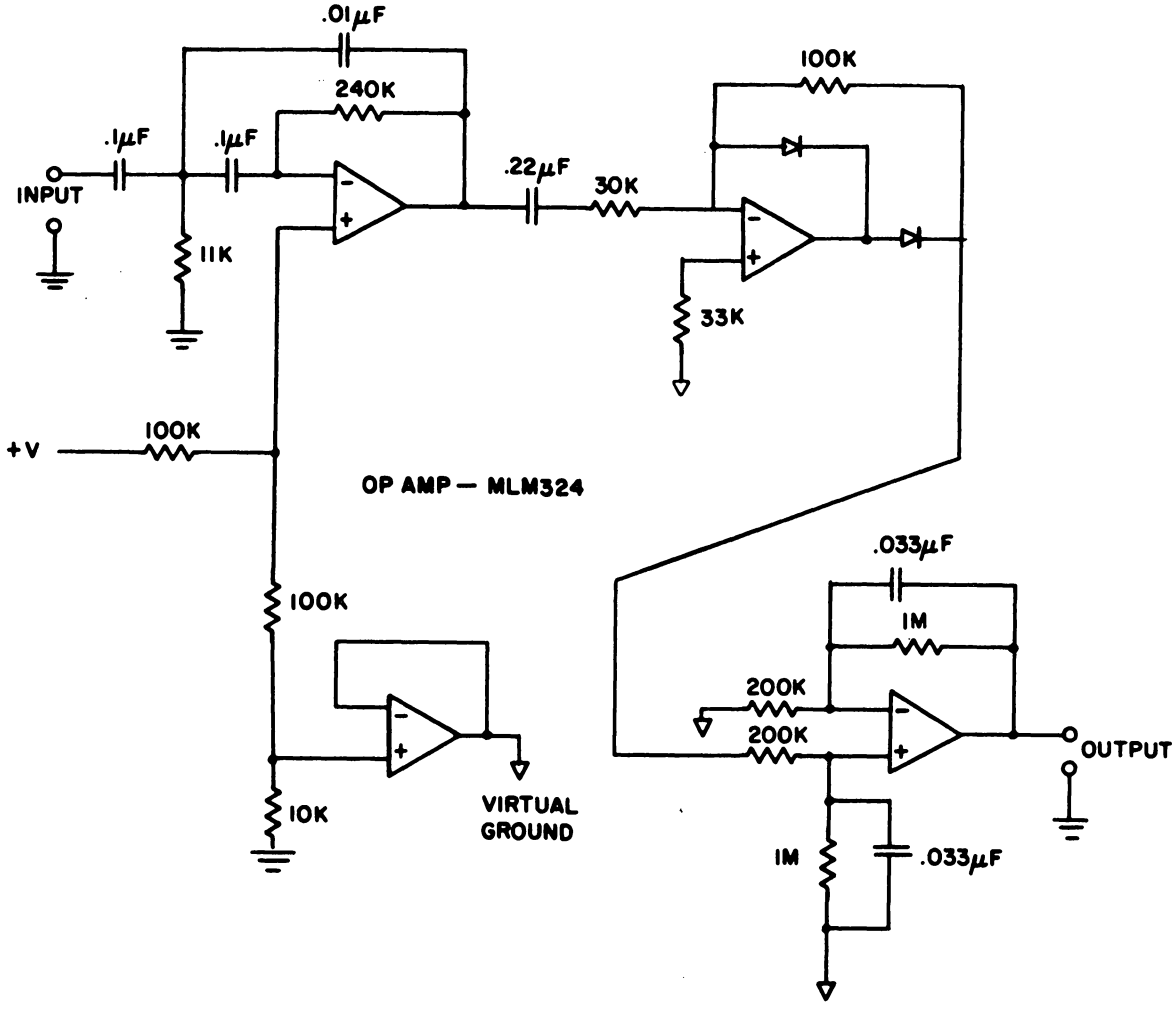


Figure 14 Schematic diagram of the signal processing circuit to convert the EMG signal amplitude to a proportional voltage

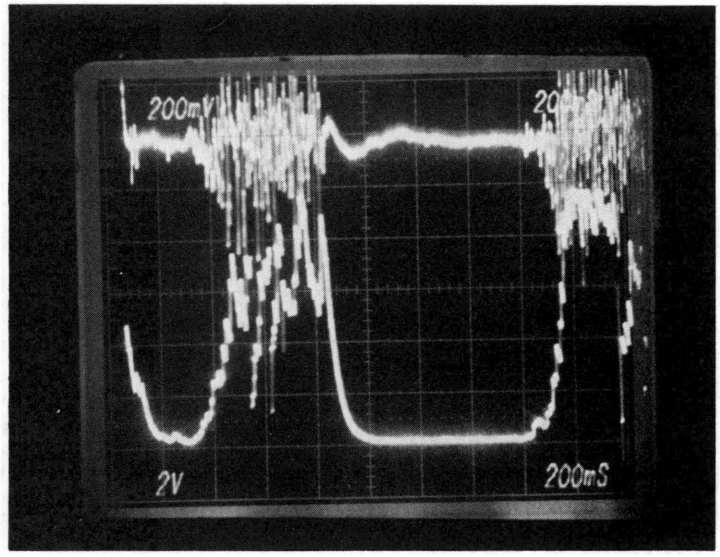


Figure 15 Active EMG electrode performance: Top trace is the output of the EMG signal amplifier and low pass filter, Bottom trace is the low pass filtered signal amplitude

possible and then tensed again to form the second contraction pulse.

The physical size and construction of the EMG electrode package is shown in Figure 16. The electrodes are fabricated from double sided printed circuit board stock with a tin coating. The ring around each electrode is grounded and provides the skin ground contact. In addition, the back conductor is also grounded so that the final electrode structure has each electrode surrounded by a ground plane for external noise shielding. The electrodes are about twice as large as commercially available equipment and provide equivalent sensitivity and gain.

F. Analog Multiplexer

At the time the hardware for this experiment was constructed the number of electrodes that would ultimately be used was unknown. As a result, the system was designed to use up to eight electrode pairs to monitor EMG signals. A multiplexer was necessary at some point in the signal path to combine the various channels. An obvious but rather hardware intensive solution is a digital multiplexer that would combine the outputs from eight A/D converters. The other logical method was to combine the analog signals just

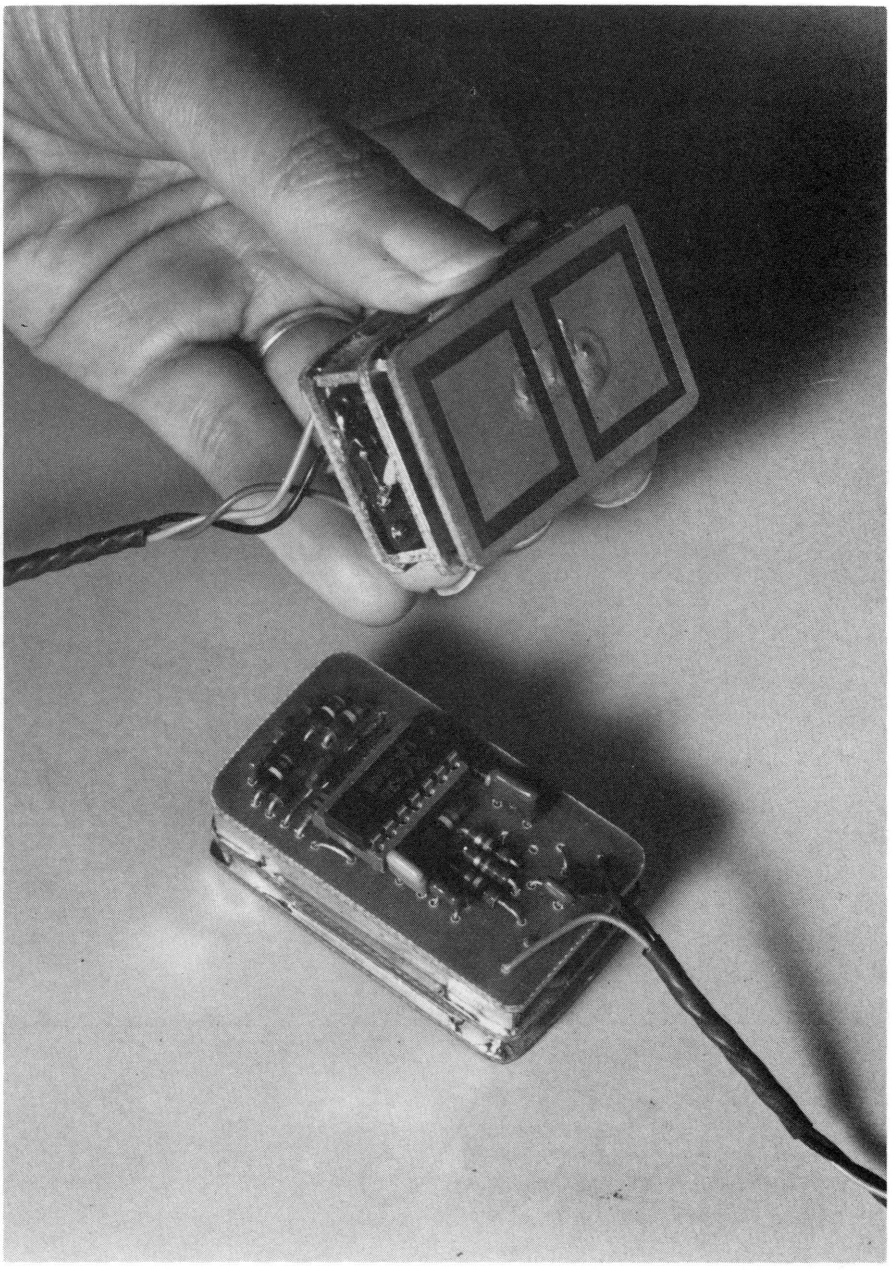


Figure 16 Photograph of the active EMG electrodes showing pickup plates and the signal processing circuitry

before the A/D converter.

At the time these issues were being contemplated, RCA introduced an eight-to-one CMOS analog multiplexer. After some initial experimentation, this approach was chosen. Figure 17 shows the rather simple circuitry necessary to perform this function. The CD4060 is a combination oscillator and multistage ripple counter. Four consecutive outputs are used to drive the analog multiplexer's (CD4051) address and inhibit inputs. Since the most significant bit drives the multiplexer's inhibit input, the output signal follows a pattern of switching between all eight inputs and of spending an equal amount of time with the output floating. When the output is floating, the 22k resistor pulls it down to ground. A typical multiplexer output signal is shown in Figure 18 with eight operating electrodes.

The multiplexer was designed to free run to save hardware and interconnections with the microcomputer. As a result of designing the A/D to free run, about ten conversions are completed at each channel voltage level before the multiplexer switches to the next one. The data conversion value at the center of the time interval for each channel is sorted out by a data input routine in the microcomputer. This is accomplished by timing the total interval that the data from the eight channels is available.

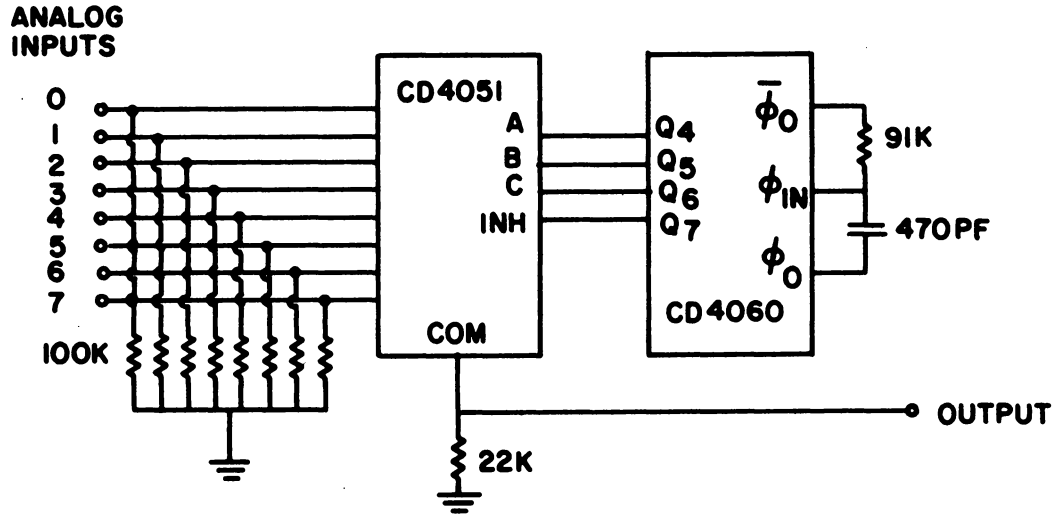


Figure 17 Schematic diagram of the analog multiplexer for time multiplexing the EMG electrodes output onto a common channel for input into the A/D converter

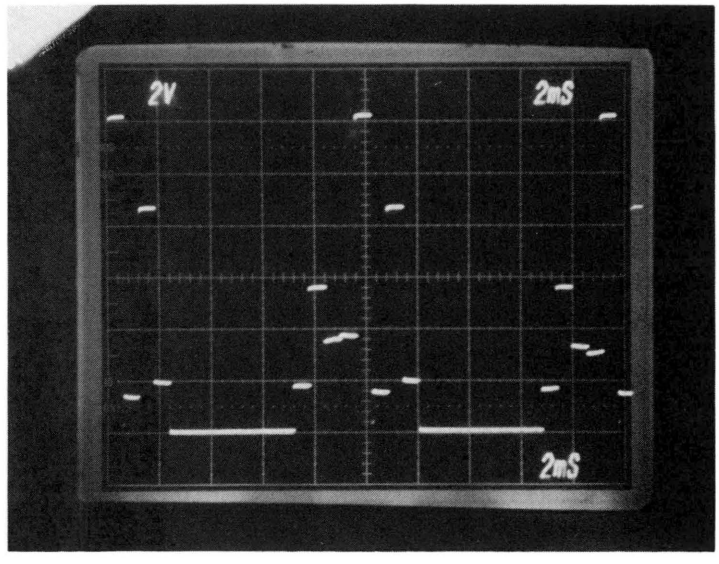


Figure 18 Output signal from the analog multiplexer that feeds the A/D converter

Then on the next cycle, when valid data first becomes available, a delay of one sixteenth of this total time interval provides a data value from the center of the first data channel. Additional delays of one eighth of the total time interval allow successive channels to be read.

IV. SYSTEM EVALUATION

Introduction

Section III discussed the computational system and its peripherals, which were constructed to evaluate this pattern recognition method of prosthesis control. This section discusses the system's operation and evaluates its performance: the assignment of an input vector to one or more of the six possible classes. The discussion will be divided into two parts: (1) external signal processing and internal microcomputer computations and (2) an evaluation of the system's performance.

A. System Operation

A differential EMG signal is induced on the plates of the active electrodes by changes in the chemical potentials within a contracted muscle beneath the plates. This signal is amplified and low pass filtered to remove noise above the 1500 Hz frequency content of the desired signal. The signal is then high pass filtered above 100 Hz to reduce the 60 Hz pickup and to eliminate the very low frequency (less than 10

Hz) noise created by slight movements of the electrodes (motion artifacts). After half wave rectifying the signal to obtain its amplitude envelope, the envelope signal is low pass filtered for smoothing. This final output signal from the active electrodes has a voltage proportional to the muscle contraction levels under the pick-up electrodes.

The muscle contraction level signal is time multiplexed with similar signals from the other electrodes and converted into a stream of parallel eight bit binary numbers for input into the microcomputer. Inside the microcomputer a subroutine interprets this data stream and extracts the values associated with the muscle contraction levels at the site of each electrode pair.

The set of data values sequentially obtained from the electrode complement is treated as a data vector. Four electrode pairs were employed in this evaluation; therefore, data vectors and the total data space will contain four dimensions. The ends of the data vectors were found to cluster when plotted in the data space. Each cluster was associated with a set of vectors obtained from a distinct limb position and it was also found that the clusters tended to elongate radially outward from the origin. Since a nearest neighbor pattern (cluster) recognition algorithm was to be implemented it was felt that the accuracy could be

improved and the size of the look-up table minimized by normalizing the data vectors and using the direction cosines instead of the vectors as they were input into the microcomputer.

The computations performed in the microcomputer start with the input of an offset vector, which may be obtained from a completely relaxed state. This allows the microcomputer to remove the constant offset values added to each vector component in the low pass filtering of the EMG amplitude signals. The offset vector is subtracted from all data vectors input into the microcomputer.

Before the pattern recognition program may be run, the look-up table data base must be created. In the table building mode, the microcomputer initially asks for a class number. The class (cluster) identifications may be arbitrarily chosen or identified as described in section II(C) if the servomotors are to be driven. After the class number (01 to FF hex) is specified the limb is positioned in the orientation to be associated with that class and the execute button depressed. The next data vector available from the A/D is input, has the offset vector subtracted from it, the difference normalized, and the magnitude compared to a threshold to determine if it is large enough. If the magnitude is too small, an error message is displayed and

the vector must be input again. If the magnitude is larger than the threshold, the classification code is stored in the look-up table followed by the components of the normalized vector. Vectors with the same class codes are stored sequentially in the look-up table. A class code of 00 (hex) indicates the end of the table. The table may be expanded by adding vectors to any class or reduced with a variety of methods to select the vectors to be eliminated.

Once the reference pattern template (look-up table) has been created, the program may be switched to one of several run modes. The basic choices are continuous and single vector classification. In either case, the computations are similar. An input routine obtains an input vector, subtracts the offset, and normalizes the result. The pattern classifier then takes this normalized input vector and computes the squared distance between itself and each of the vectors in the look-up table. For a single nearest neighbor, the class code associated with the closest vector in the look-up table is assigned to the input vector. For a k-nearest-neighbor classification, the k-nearest-neighbor class codes and distances are placed in a ballot, and after the entire table is searched, the class with the most votes is assigned to the input vector.

B. Classification Results

After the control and simulation hardware and software for an artificial limb was configured the remaining question to be answered was: how good is it? To answer this question an experiment was conducted in which the nearest neighbor classifier was trained by placing five vectors from each class in the look-up table and then was presented with 600 vectors, equally divided between six classes, for it to classify.

Before starting the classification experiment, the microcomputer software was configured to use a single nearest neighbor algorithm. Four electrodes were placed around a subject's right mid-forearm in roughly the four quadrants and were held in place with an ace bandage. Then five vectors were recorded for each of six limb configurations and entered into the look-up table. The limb configurations and microcomputer class codes used were the following: hand grasp 01 (hex), hand open 02 (hex), wrist flex 04 (hex), wrist extend 08 (hex), forearm pronate 10 (hex), and forearm supinate 20 (hex). During the experiment, the subject received immediate feedback of any errors, and as a result an improvement in accuracy was obtained that was

attributed to a learning process. The experiment was conducted by proceeding through the list of limb orientations both forwards and backwards and by requesting a position and then activating a single vector classification in the microcomputer.

The confusion matrix resulting from this total run of 600 vectors is shown in the matrix below. The vector labels in the left column indicate the pattern class for which a vector was requested. The numbers across each row are the precentage of time each class was chosen by the microcomputer classifier. The numbers in parenthesis are the standard deviations for the entries.

Confusion Matrix for the Total Test

	VECTOR CLASSIFIED AS:					
REQUESTED INPUT VECTOR:	GRASP (01)	OPEN (02)	FLEX (04)	EXTEND (08)	PRON (10)	SUPIN (20)
GRASP (01)	99(1)	0	0	0	0	1(1)
OPEN (02)	0	83(4)	0	0	17(4)	0
FLEX (04)	0	0	100	0	0	0
EXTEND (08)	3(2)	0	0	97(2)	0	0
PRONATE (10)	0	8(3)	14(3)	4(2)	63(5)	11(3)
SUPINATE (20)	3(2)	1(1)	0	0	18(4)	78(4)

These results compare favorably to those reported by Herberts et al. [26] who used a linear discriminant function as a classifier and positioned the electrodes for optimum pattern separation. In both cases the same limb

configurations were used. Before the training, Herberts obtained correct classification accuracies ranging from 57% to 100%, and the majority of the classes were identified correctly more than 95% of the time.

The effect of the limited training that occured during the course of the experiment can be seen by comparing the confusion matrix obtained for the last half of the experiment, shown in the matrix below, with that obtained for the entire experiment. As is evident in a comparison of the two tables, the classification of the first four movements (grasp, open, flex, and extend) did not improve substantially. However, the accuracy with which the last two classes were identified, i.e. pronate and supinate, did improve through the training process.

Confusion Matrix for the Last Half of the Test

	VECTOR CLASSIFIED AS:					
REQUESTED INPUT VECTOR:	GRASP (01)	OPEN (02)	FLEX (04)	EXTEND (08)	PRON (10)	SUPIN (20)
GRASP (01)	98(1)	0	0	0	0	2(1)
OPEN (02)	0	84(4)	0	0	16(4)	0
FLEX (04)	0	0	100	0	0	0
EXTEND (08)	0	0	0	100	0	0
PRONATE (10)	0	12(3)	6(2)	6(2)	72(4)	4(2)
SUPINATE (20)	4(2)	0	0	0	10(3)	86(3)

Further enlightenment as to why these particular results were obtained may be found from an examination of

the look-up table used by the microcomputer. This is shown in Figures 19 and 20 for two three-dimensional projections of the four dimensional data space. The vector codes are A-grasp, B-open, C-flex, D-extend, E-pronate, and F-supinate. As may be seen best in Figure 20, the vectors for classes A, B, C, and D formed reasonably tight groupings. Note that the operation of this classifier will place the unknown vector into the look-up table space (as is shown in the figures) and will pick the class of the closest vector. There was one "flyer" from class A (grasp) that is close to class D (extend) and class F (supinate) and is probably responsible for those 3% errors shown in the confusion matrix for the total test. Class E (pronate) and F (supinate) were the most diffuse clusters, which indicated that the unknown input vectors from these classes also had a corresponding spread. In addition, these diffuse clusters can sabotage the other classification results. Cluster B corresponding to hand open, contains two reasonably close vectors from cluster E (pronate). As a result, it is not suprising that a vector generated by a hand open movement could fall closer to one of the intruding class E vectors than one from the correct class B cluster. This is reflected in the confusion matrix as the 17% erroneous classification of forearm pronate for vectors generated by a hand open

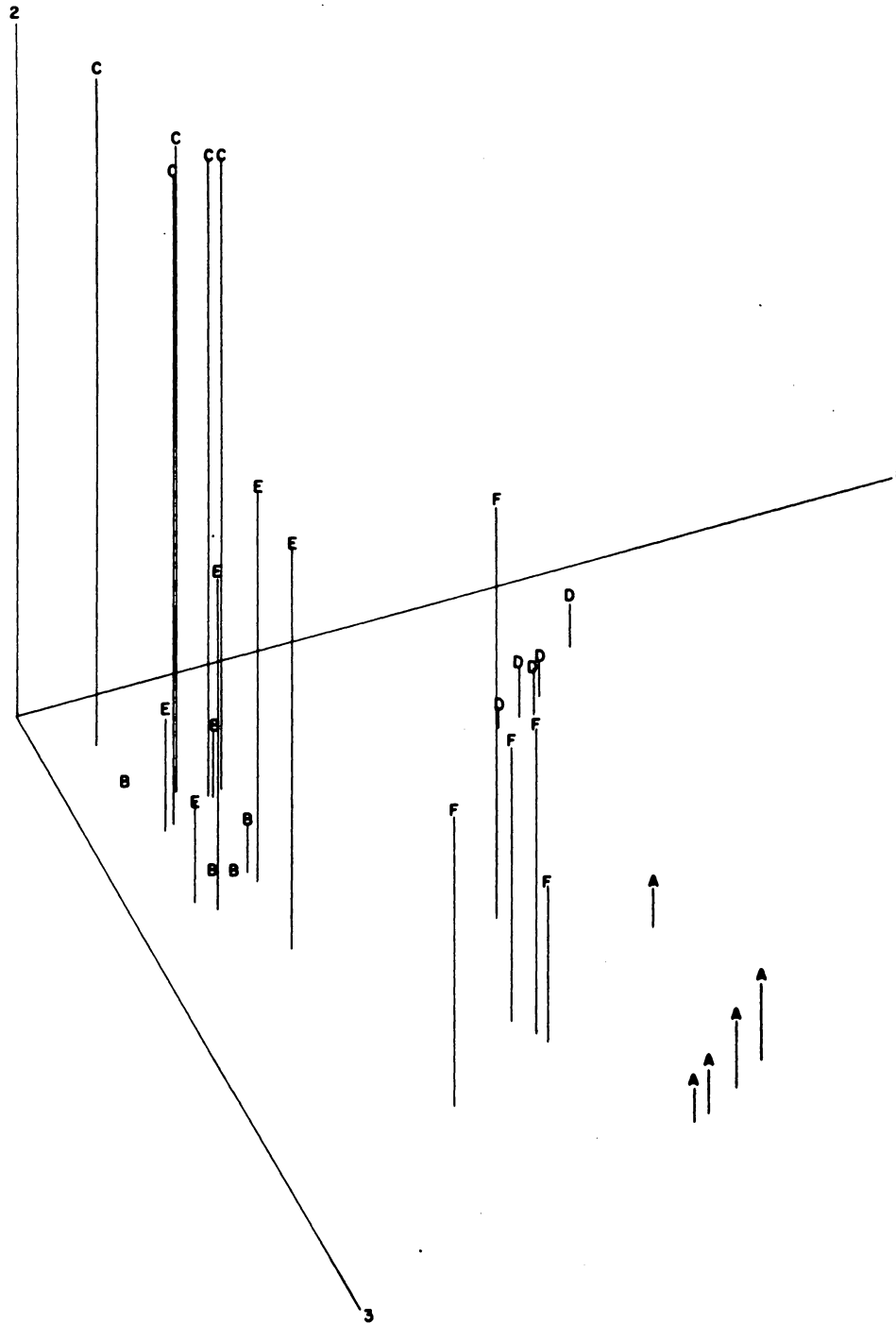


Figure 19 The system evaluation microcomputer look-up table projected onto the X1, X2, X3 dimension

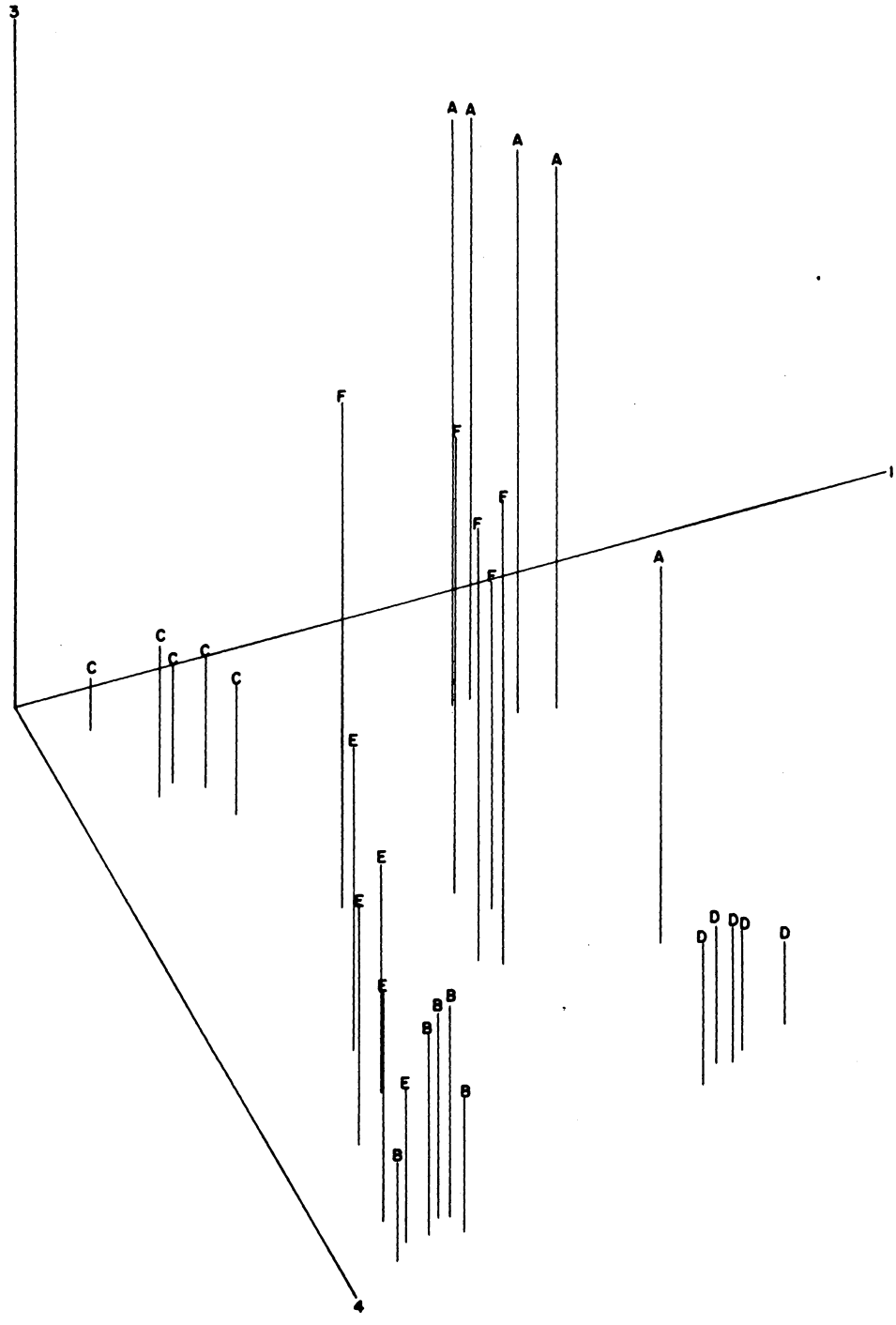


Figure 20 The system evaluation microcomputer look-up table projected onto the X1, X3, X4 dimension

motion.

The results reported in this section show what may typically be expected from this application of a nearest neighbor pattern classifier to the myoelectric prosthesis control problem. There are obvious places for improvement. For example, by judicious positioning of the electrodes, the discrimination of vectors for the pattern classes could be improved, as was indicated by Herberts et al. In addition, a fine tuning of the look-up table could eliminate many of the errors caused by the intrusion of "bad" vectors from one class into the domain of another class.

V. ALGORITHM EVALUATION AND COMPARISONS

Introduction

The evaluation of pattern classifiers has traditionally implied recording two independent sets of data. A first set is used in setting up the internal classifier parameters. The classifier is then exercised with the second set to determine an error rate. The previous section provided an indication of the error rate with five vectors for each class template. In this section, an attempt will be made to determine the necessary size of the look-up table and to determine the resulting classification accuracy. The optimized nearest neighbor classifier will then be compared to a parametric (multivariate Gaussian) classifier to identify the relative measure of its merit.

A. Classification Data Sets

Two complete data sets of the EMG vectors were recorded for the limb configurations: hand grasp, hand open, wrist flex, wrist extend, forearm pronate, and forearm supinate. In addition a single set (128) of data vectors was recorded for the completely relaxed case so that the offset added by

the system might be removed. Four EMG active electrodes were positioned around the forearm (author's left), as was described in a previous section. The remaining four voltage inputs to the analog multiplexer were determined by resistive voltage dividers from the power supply. Thus, the last four voltage levels transmitted by the analog multiplexer remained at fixed values. All eight vector components (four active EMG signals plus four constant voltages) were recorded in 2708 EPROMS (128 vectors per chip). Two complete EPROMS were filled with vectors for each of the six limb configurations and a final EPROM was filled with vectors for a completely relaxed state so that system offset values could be determined and corrected.

The contents of all EPROMS were transferred to data files in a VAX 780 computer and merged into two independent test data files. Both test data files contained equal numbers of vectors (128) for the six pattern classes and the offset data. At this point, it was discovered that an error in the analog multiplexer had occasionally caused an incorrect vector to be recorded. These incorrect recordings were detected through the four trailing constant components in a vector having incorrect values. The questionable vectors were removed and the test data files then pruned down to 100 vectors in each class and in the offset case.

The test data files were stored in the Intel data format with sixteen data bytes (two complete vectors) per line. In most cases, the pruning was accomplished by deleting both vectors on the line with a bad vector. After they were pruned both test data files contained 700 vectors.

The two test data files were accessed by the pattern classifier programs for training and evaluation. This method of algorithm evaluation removed any possibility of subject learning, which had been observed in the microcomputer evaluations. The resulting classification scores were thus lower than those reported earlier.

B. Multivariate Normal Parametric Classifier

A minimum error rate parametric classifier [43,44] was constructed with the assumption that the probability density of each vector class could be "described" by a multivariate normal probability density of the form:

$$p(x) = \frac{1.}{(2\ pi)^{d/2}\ |C|^{1/2}} \exp[\ -(x-u)^{t}\ C^{-1}\ (x-u)/2\ ] \qquad (3)$$

where C is the covariance matrix, C^{-1} is its inverse, u is the class mean vector, and d is the dimension of the

distribution.

The covariance matrix may be estimated for each class from the training set (100 vectors per class) using:

$$C = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - u)(x_i - u)^t \quad . \qquad (4)$$

A minimum error rate classifier may be constructed by using a discriminant function of the form:

$$g(x) = \mathrm{Log}[p(x|w)] + \mathrm{Log}[P(w)] \quad . \qquad (5)$$

P(w) is the a priori probability of observing a particular class and p(x|w) is the conditional probability density of the vector x within that class. Note that in the case of equal a prior probabilities for the classes, as is in the case with the two test vector files, the second term in equation 5 remains at a constant value for all classes.

A minimum error rate discriminant function for each class may be constructed by combining equations 3 and 5.

$$g(x) = -(x^t C^{-1} x)/2 + (C^{-1} u)^t x - (u^t C^{-1} u)/2 - \mathrm{Log}[|C|]/2 - d\mathrm{Log}[2\pi]/2 - \mathrm{Log}[P(w)] \qquad (6)$$

To classify an unknown vector, the discriminant function

shown in equation 6 is computed for each class. The unknown vector is assigned to the class with the maximum value of the discriminant function. As a result of this procedure, the last two terms in equation 6 which are constants, may be neglected.

After constructing the discriminant functions from the first data file the classifier was presented with the vectors to be classified from the second file. The files were then reversed and training was accomplished on the second before the first was classified. The combined results of this experiment are presented as a confusion matrix shown below.

Confusion Matrix for Multivariate Normal Minimum Error Rate Classifier

		VECTOR CLASSIFIED AS:					
INPUT VECTOR CLASS:		GRASP	OPEN	FLEX	EXTEND	PRON	SUPIN
	GRASP	98	0	0	2	0	0
	OPEN	1	36	0	0	1.5	61.5
	FLEX	12.5	8.5	78.5	0	0	.5
	EXTEND	3	0	0	96.5	.5	0
	PRONATE	0	0	0	0	100	0
	SUPINATE	.5	25.5	0	0	0	74

The experiment was repeated (results shown on page 78) with the data vectors normalized to a unit length. This reproduced the data processing procedure utilized with the microcomputer based pattern recognition system as was

described in a previous section. The components of each vector became the direction cosines for that vector. The normalization process had the effect of reducing the dimensionality of the data system since all vectors were then constrained to lie on a four dimensional hyper-sphere. The normalized vectors were classified with the four dimensional coordinate system previously used.

Confusion Matrix for Multivariate Normal Minimum Error Rate Classifier Using Direction Cosines

		VECTOR CLASSIFIED AS:					
INPUT VECTOR CLASS:		GRASP	OPEN	FLEX	EXTEND	PRON	SUPIN
	GRASP	94.5	0	0	2.5	0	3
	OPEN	.5	28.5	0	0	1	70
	FLEX	0	0	100	0	0	0
	EXTEND	13	0	0	86	1	0
	PRONATE	0	0	0	0	100	0
	SUPINATE	.5	41.5	0	0	.5	57.5

C. Nearest Neighbor Classifier

A condensed nearest neighbor classifier [45] was constructed to determine a typical size for the reference list template that all unknown vectors are measured from and to compare its performance with that of the parametric classifier. The construction procedure for a condensed

nearest neighbor classifier proceeds as follows. The reference list is seeded with one of the unknown vectors in the training set. Then all of the data vectors, in turn, are classified by measuring the distance (geometrically in this case) from the unknown vector to all the vectors in the reference list. The class of the nearest vector in the reference list is assigned to the unknown. If the classification is incorrect, that data vector is then added to the reference list. The procedure of classifying all of the training set is repeated until either all data vectors can be classified correctly or all the training set has been transferred to the reference list. However, in all trials this procedure converged with less than 23% of the training set in the reference list.

As is illustrated in the literature [46], this type of classifier tends to pick the initial reference list vectors randomly in the data space but becomes more selective later as the decision boundary becomes better defined. The vectors chosen later tend to lie close to the decision boundaries. A further processing procedure was employed to prune the unneeded vectors from the reference list. In turn, each vector in the reference list was removed and the resulting list used to classify the entire training set. If the entire training set was not correctly classified, the removed

vector was placed back in the reference list. This pruning process reduced the reference list by 11% to 14%.

After the reference list was pruned, it was used as a template to classify the other data file (unknown vectors). The procedure was then repeated. The second data file was used to create the condensed reference list and the first data file was used for classifier evaluation. The combined results of training on one set and classifying the other for both data files is shown in the confusion matrix below.

Confusion Matrix Resulting from a Nearest Neighbor Classifier Using a Condensed Reference List

		VECTOR CLASSIFIED AS:					
INPUT VECTOR CLASS:		GRASP	OPEN	FLEX	EXTEND	PRON	SUPIN
	GRASP	91	4.5	0	3	0	1.5
	OPEN	.5	39.5	0	0	0	60
	FLEX	0	0	99.5	0	0	.5
	EXTEND	9	0	0	91	0	0
	PRONATE	1	6.5	0	2.5	89	1
	SUPINATE	.5	34.5	2	0	0	63

It was found that the number of vectors in the reference list tended to be low for the cases in which the classification accuracy was good. It was observed from studying the data files that the vectors for the classes hand open and forearm supinate tended to overlap in the data space and as a result were difficult for the algorithm to classify. This finding was also reflected in the relative

class populations in the reference list for the various classes. The reference lists constructed in classifying the two data files contained the following average numbers of vectors per class:

GRASP	6
OPEN	39.5
FLEX	1.5
EXTEND	6
PRONATE	5
SUPINATE	46

This provides a useful check on the reference list size needed for class separation. For easily distinguished clusters such as grasp, flex, extend, and pronate, a minimum number of vectors is required to produce good classification accuracies.

The classification process was repeated for the nearest neighbor classifier with normalized data vectors. The reference list and unknown data vectors in this case represent the direction cosines of the original data vectors. The reference table remained at approximately the same size as before, and as can be seen on page 82, the resulting confusion matrix was only slightly changed. The classifier employed in this case closely reproduces the procedure used in the microcomputer except that the reference list processing is more extensive. Classification

accuracy improved for four pattern classes through the use of normalized vectors and a nearest neighbor classifier. It is not clear that this represents a net gain in system performance since the two classes (open and supinate) that sustained a reduction in accuracy more than compensated for the improvements.

Confusion Matrix Resulting from a Nearest Neighbor Classifier Using Normalized Vectors and a Condensed Reference List

		VECTOR CLASSIFIED AS:					
INPUT VECTOR CLASS:		GRASP	OPEN	FLEX	EXTEND	PRON	SUPIN
	GRASP	91.5	3.5	0	4	0	1
	OPEN	.5	36	0	0	0	63.5
	FLEX	0	0	100	0	0	0
	EXTEND	4	0	0	96	0	0
	PRONATE	0	.5	0	1	96.5	2
	SUPINATE	.5	45.5	0	0	0	54

In all of the experimental evaluations on the vectors in the two data files, the correct classification accuracy for the hand open and forearm supinate classes was considerably below the accuracy for the other classes. It was felt that this result stemmed from a less than optimum electrode placement, which caused these two pattern classes to overlap in the data space. The X1 and X2 vector components tend to be large for both of these classes while the X2 and X3 components are small. In addition, the nearest

neighbor reference list contains a considerably larger number of entries for these two classes, which also indicates that there may be some difficulty in discriminating between the two classes.

The various classification procedures tested provide comparable results. The same data files were used for the training and testing of the classifiers described in this section. The work described in section IV used the microcomputer based classifier, which operated on and displayed the classification for vectors as they were generated and thus provided immediate feedback in the generation of new vectors. The major difference was that the effect of subject training was eliminated in the use of the stored vector data files.

VI. RECOMMENDATIONS AND CONCLUSIONS

A well planned experiment will naturally employ the scientific method and will proceed to experimental results and conclusions based on these results. However, a point often overlooked is that conclusions lead to new hypotheses and new experiments for another cycle of investigation. As the first cycle of experimentation on this project was being completed, problems were found that may be addressed in future projects.

The amplitude of EMG signals is a measure of the muscle contraction levels at the various electrode sites. This directly leads to a clustering in the multidimensional data input space of the input vectors recorded for similar motions. A nearest neighbor pattern classifier was chosen because of the inherent training capability it provided. As a result of the program structure, multidimensional control of demonstration servomotors became possible through the judicious labeling of the pattern classes. Total system performance is limited to a large extent by the length of the classifier look-up table and its effect on program cycle time. Related to this limitation was the necessity of keeping the control function simple; a more sophisticated approach would have increased memory requirements and

processor loading beyond the system's capability.

A possible solution to the system's performance limitations is a faster computational capability or a dual processor system having one processor dedicated to pattern classification while the other performs the physical control function and data input. Such a system also lends itself to experimentation with alternative classifier algorithms. In addition, a dual processor system would allow a more sophisticated hardware controller to be dedicated to reproducing the normal smooth motions of the original limb. A further argument in favor of a dual processor (or faster shared processor) is the natural division of tasks to be performed. The first is a pattern recognition task; the processor determines what the user's intended motion is from the detected EMG vectors. Then the pattern classification becomes the input to the hardware digital controller, which translates the stream of pattern classifications into fluid artificial limb motions.

The digital system hardware for this experiment was designed around a commercial single board computer. This hardware imposed constraints on data I/O that could be overcome by a redesigned dedicated system. Further improvements could be realized by including an arithmetic processor, in addition to the multiplier, in the total

system design. If additional throughput were required, a dedicated hardware system function could also perform the metric calculations for the nearest neighbor algorithm.

Data input requirements on the microcomputer were reduced by performing as much signal processing as possible in an analog fashion. Besides detecting and amplifying the input EMG signal, the active electrodes also band-limited the signal to reduce noise and low frequency electrode motion artifacts and they took absolute values and low pass filtered the output. This approach to the input signal conditioning was aimed specifically at the algorithm implemented on the microcomputer. Any other algorithm needing more input information than the EMG signal amplitude would require a redesign of the data input system.

An inconsistency exists between the results reported for the microcomputer based pattern classifier and the results obtained from the prerecorded data files. As was reported for the microcomputer based system, the patterns for pronate and supinate were difficult to resolve with the same accuracy as the other patterns. When various classifiers were applied to the prerecorded data files, the difficult patterns to classify were open and supinate. The reason for this pattern difference is probably related to electrode placement and the fact that the two experiments

were performed with different individuals.

In the results reported by Herberts et al., a linear combination discriminant function was applied to the EMG signal amplitudes from six sites. The various electrodes were repositioned until satisfactory pattern classification was obtained. In this experiment, the four electrodes were positioned around the four quadrants of the midforearm. A minimal ammount of repositioning was employed to optimize the correct pattern classification. In both cases, when the vectors were classified in the microcomputer and when data was recorded for the data files, the complete data sets were processed in one session to avoid the possibility of errors in replacing the electrodes.

Subject training was observed during the microcomputer based pattern classification. In addition, an observation was made that various other EMG patterns could be generated by motions different from those of the six classes employed. Although this approach would necessitate additional training, the classifiers would operate with equal ease on a one-for-one mapping of unorthodox but easily distinguished limb configurations for the desired classes.

In comparing the nearest neighbor classifier algorithm to a parametric type classifier, it was found that both resulted in comparable accuracies when the same data vectors

were used. This insensitivity to algorithm could be due to two factors. First, a nearest neighbor classifier will always have an error rate larger than a minimum rate Bayes classifier, but this does not preclude the nearest neighbor classifier from being almost as good as the Bayes classifier. Second, the minimum error rate parametric classifier application was based on the assumption that the probability density for each class could be described by a multivariate normal density. The assumption of a normal density is not precisely valid since the data space is bounded between 0 and 255 in all four dimensions and the components of the data vectors for each class were spread over a substantial fraction of this range.

Whether the normalized vectors (direction cosines) produced a better classification accuracy than the unnormalized vectors is clouded by the low accuracy with which the open and supinate classes were distinguished. However, if the results for these two classes are discounted, the normalized vectors apparently produced a better result. In the multivariate normal classifier, the number of classes for which classification accuracy improved is the same as the number showing a decline, but there is an apparent net gain in accuracy. In addition, the experiment with the normalized vectors and nearest neighbor classifier

produced a better classification accuracy for all classes, except in the discounted open and supinate cases, than did the nearest neighbor classifier and unnormalized vectors.

When the microcomputer based nearest neighbor classifier was evaluated, five vectors were placed into the reference list. This decision was based on previous qualitative evaluations of the classifier and a desire to keep the cycle time of the classifier as low as possible. The validity of this decision was confirmed during the algorithm evaluations when the condensed reference lists were found to average between two and six vectors for most classes. The exceptions were the difficult to separate hand open and forearm supinate motions, which were heavily represented in the reference list because of the criteria used to classify a vector correctly during training or to add that vector to the reference list.

VII. SUMMARY

The prosthesis controller described in this dissertation was developed on an applied research project. The controller represents a previously unexplored approach to the problem of controlling artifical limbs through the use of EMG pattern recognition via a nearest neighbor classifier. This solution to the problem--the use of a nearest neighbor classifier--inherently provides several features not found in existing prototypes, including patented models.

For example, the pattern classifier in existing EMG controlled prostheses and experimental devices consists of a hard wired resistor array or a set of vectors stored in read only memory. Such a classifier may not be changed except after a re-calibration period during which the necessary parameters are computed from an extensive reference data file. To change a nearest neighbor classifier, the look-up table is skewed and/or punished by the user's adding or subtracting selected vectors to improve the selection accuracy of the intended motion. The principal advantage is that a nearest neighbor based controller may be taught in the field to recognize an unanticipated highly specialized set of motions or trimmed to improve its classification

in selected areas.

Further benefits derived from a trainable classifier include user adaptation of the machine instead of to the machine. The error rate for a nearest neighbor classifier has experimentally been shown to be comparable to that of a minimum error rate classifier based on an assumed multivariate normal distribution function for the EMG vectors in each class.

VIII. LITERATURE CITED

1. P.J. Rasch and R.K. Burke, Kinesiology And Applied Anatomy. Philadelphia: Lea and Febiger, 1978.

2. R.B. Stein, "Peripheral Control of Movement," Physiological Reviews, Vol. 54, pp. 215-243, January 1974.

3. V.F. Harrison and O.A. Mortensen, "Identification and Voluntary Control of Single Motor Unit Activity in the Tibialis Anterior Muscle," Anatomical Record, Vol. 144, pp. 109-116, 1962.

4. O.C.J. Lippold, "The Relation Between Integrated Action Potentials in a Human Muscle and Its Isometric Tension," J. Physiol, Vol. 117, pp. 492-499, 1952.

5. B. Bigland and O.C.J. Lippold, "The Relation Between Force, Velocity and Integrated Electrical Activity in Human Muscles," J. Physiol, Vol. 123, pp. 214-224, 1954.

6. E.N. Zuniga and D.G.Simmons, "Nonlinear Relationship Between Averaged Electromyogram Potential and Muscle Tension in Normal Subjects," Archives of Physical Medicine and Rehabilitation, Vol. 50, pp. 613-620, November 1969.

7. A.H. Bottomley, "Myo-Electric Control of Powered Prosthesis," The Journal of Bone and Joint Surgery, Vol. 47B, pp. 411-415, August 1965.

8. A.H. Bottomley and T.K. Cowell, "An Artifical Hand Controlled by Nerves," New Scientist, Vol. 12, pp. 668-671, March 1964.

9. N. Hogan, "A Review of the Methods of Processing EMG for Use as a Proportional Control Signal," Biomedical Engineering, Vol. 11, pp. 81-86, March 1976.

10. R.N. Scott, "Myoelectric Control of Prostheses," Archives of Physical Medicine and Rehabilitation, Vol. 47, pp. 174-181, March 1966.

11. P. Herberts, "Myoelectric Signals in Control of Prostheses," Acta Orthopaedica Scandinavica, Supplementum No. 124, pp. 1-83, 1969.

12. C.K. Battye, A. Nightingale, and J. Whillis, "The Use of Myo-electric Current in the Operation of Prostheses," Journal of Bone and Joint Surgery, Vol. 37-B, pp.506, 1955.

13. D.S. McKenzie, "The Russian Myo-Electric Arm," Journal of Bone and Joint Surgery, Vol. 47-B, pp. 418-420, August 1965.

14. B. Popov, "The Bio-Electrically Controlled Prosthesis," Journal of Bone and Joint Surgery, Vol. 47-B, pp. 421-424, August 1965.

15. D.S. Childress and J.N. Billock, "Self-Containment and Self-Suspension of Externally Powered Prostheses for the Forearm," Bulletin of Prosthetics Research, Vol. 10, pp. 4-21, 1970.

16. C.H. Hoshall, W. Seamone, and R.L. Konigsbert, "Myoelectrically Controlled Prosthesis," U.S. Patent No. 3,735,425., May 1973.

17. V. Dunfield and E. Shwedyk, "Digital E.M.G. Processor," Medical and Biological Engineering and Computing, Vol. 16, pp. 745-751, November 1978.

18. F.R. Finley and R.W. Wirta, "Myocoder-Computer Study of Electromyographic Patterns," Archives of Physical Medicine and Rehabilitation, Vol. 48, pp. 20-24, January 1967.

19. F.R. Finley and R.W. Wirta, "Mycoder Studies of Multiple Myopotential Response," Archives of Physical Medicine and Rehabilitation, Vol. 48, pp. 598-601, November 1967.

20. F.R. Finley, "Pattern Recognition of Myoelectric Signals," Journal of the Canadian Physiotherapy Association, Vol. 21, pp. 19-24, February 1969.

21. F.R. Finley, R.W. Wirta, and K.A. Cody, "Muscle Synergies in Motor Performance," Archives of Physical Medicine and Rehabilitation, Vol. 49, pp. 655-660, November 1968.

22. R.W. Wirta, D.R. Taylor, and F.R. Finley, "Engineering Principles in the Control of External Power by Myoelectric Signals," Archivies of Physical Medicine and Rehabilitation, Vol. 49, pp. 294-296, May 1968.

23. P. Herberts, C. Almstrom, R. Kadefors, and P.D. Lawrence, "Hand Prosthesis Control Via Myoelectric Patterns," Acta Orthopaedica Scandinavica, Vol. 44, pp. 389-409, 1973.

24. P. Herberts, E. Kaiser, R. Magnusson, and I. Petersen, "Power Spectra of Myoelectric Signals in Muscles of Arm Amputees and Healthy Normal Controls," Acta Orthopaedica Scandinavica, Vol. 44, pp. 161-193, 1973.

25. C. Almstrom, "An Electronic Control System for a Prosthetic Hand With Six Degrees of Freedom," Research Laboratory of Medical Electronics Technical Report, 1:77, Goteborg, Sweden, 1977.

26. P. Herberts, C. Almstrom, and K. Caine, "Clinical Application Study of Multifunctional Prosthetic Hands," Journal of Bone and Joint Surgery, Vol. 60-B, pp. 552-560, November 1978.

27. G.N. Saridis and H.E. Stephanou, "Hierarchical Intelligent Control of a Prosthetic Arm," National Science Foundation Report No. PB 258 049, July 1976.

28. S.C. Jacobsen and R.W. Mann, "Control Systems for Artifical Arms," IEEE Conference on Man, Systems, and Cybernetics (Boston, Mass.), November 1973.

29. R.B. Jerard and S.C. Jacobsen, "Laboratory Evaluation of a Unified Theory for Simultaneous Multiple Axis Artifical Arm Control," ASME Paper No. 79-WA/Bio-8, December 1979.

30. A. Freedy, J.W. Aldrich, F.C. Hull, and J. Lyman, "Fundamental Studies of a High Performance Control System for Externally Powered Artifical Arms," Biotechnology Laboratory Technical Report No. 49, NTIS No. PB-216 983, November 1970.

31. J. Lyman, A. Freedy, and M. Sololonow, "Studies Toward a Pratical Computer-Aided Arm Prosthesis System," Bulletin of Prosthetics Research, pp. 213-225, Fall 1974.

32. A. Freedy, M. Solomonow, and J. Lyman, "A Microcomputer Based Arm Prosthesis with Two Channel Sensory Feedback," IEEE Decision and Control Conference (Houston, Texas), December 1975.

33. J. Lyman, A. Freedy, and M. Solomonow, "System Integration of Pattern Recognition, Adaptive Aided, Upper Limb Prosthesis," Mechanism and Machine Theory, Vol. 12, pp.503-514, 1977.

34. P.D. Lawrence and W.C. Lin, "Statistical Decision Making in the Real-Time Control of an Arm Aid for the Disabled," IEEE Transactions on Systems, Man, and Cybernatics, Vol. SMC-2, pp. 35-42, January 1972.

35. G.N. Saridis and M.A. Newman, "Upper Limb EMG Statistical Analysis," MidCon 79, Paper 10/1, 1979.

36. G.N. Saridis and T.P. Gootee, "E.M.G. Signal Pattern Analysis and Classification," AARL Report No. 37, March 1980.

37. D. Graupe and W.K. Cline, "Functional Separation of EMG Signals via ARMA Identification Methods for Prosthesis Control Purposes," IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-5, pp. 252-259, March 1975.

38. D. Graupe and W.J. Monlux, "Multifunctional Control of Artifical Upper Limbs Based on Parameter Identification of Myoelectric Signals," IEEE Decision and Control Conference (Houston, Texas), December 1975.

39. D. Graupe, "Multi-functional Control System for an Artifical Upper-Extremity Prosthesis for Below Elbow Amputees," U.S. Patent No. 4,030,141., June 1977.

40. D. Graupe, J. Magnussen, and A.A. Beex, "A Microprocessor System for Multifunctional Control of Upper-Limb Prostheses via Myoelectric Signal Identification," IEEE Transactions on Automatic Control, Vol. AC-23, pp. 538-544, August 1978.

41. D. Graupe and J. Salahi, "Limb Function Discrimination Performance Using EMG Parameter Identification," Proceedings of the Joint Automatic Control Conference (Denver, Colorado), pp. 426-428, June 1979.

42. J.E Anderson, Grant's Atlas of Anatomy. Baltimore: The Williams and Wilkins Company, 1978.

43. R.O. Duda and P.E. Hart, Pattern Classification and Scene Analysis. New York: Wiley-Interscience, 1973.

44. H.C. Andrews, Mathematical Techniques in Pattern Recognition. New York: Wiley-Interscience, 1972.

45. P.E. Hart, "The Condensed Nearest Neighbor Rule," IEEE Transactions on Information Theory, Vol. IT-14, pp. 515-516, May 1968.

46. I. Tomek, "Two Modifications of CNN," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-6, pp. 769-772, November 1976.

IX. APPENDIX (COMPUTER PROGRAMS)

A. Microcomputer Based Classifier 8080 Source Listing

```
;       NEAREST NEIGHBOR CLASSIFICATION ROUTINE WITH SERVO DRIVER
;
;       COMPLETED 27 AUGUST 1980 BY LARRY WESTERMAN
;
;       THIS PROGRAM ALLOWS FOR K-NEAREST-NEIGHBOR CLASSIFICATION OF
;       MYOELECTRIC SIGNALS.  PROVISION IS PROVIDED TO BUILD AND
;       MODIFY A TABLE OF VECTORS TO USE IN THE CLASSIFICATION
;       PROCESS.  AFTER EACH SUCCESSFUL CLASSIFICATION, SERVO SET
;       POINTS ARE UPDATED AND OUTPUT TO THE SERVO CONTROLLERS.
;
;
;       EXECUTION SEQUENCE:
;
;               GO 800 (E/E)    INITIALIZE PROGRAM, CLEAR TABLE
;
;               USR I [OR]
;               GO 803 (E/E)            RESTART PROGRAM, NO CLEARING
;
;       KEY FUNCTIONS:
;
;         0     CONTINUOUS RUN, DISPLAY (INPUT - NULL)
;         1     CONTINUOUS RUN, DISPLAY NORMALIZED VECTOR
;         2     INPUT SINGLE VECTOR AND CLASSIFY, DISPLAY
;                (INPUT - NULL)
;         3     INPUT SINGLE VECTOR AND CLASSIFY, DISPLAY
;                NORMALIZED VECTOR
;         4     INPUT NEW NULL VECTOR
;         5     READ CASE NUMBER, INPUT VECTOR, ADD TO TABLE
;         6     READ CASE NUMBER, INPUT VECTOR, PUNISH TABLE
;                (IE, REMOVE VECTOR FURTHEST FROM CASE MEAN AND
;                ADD NEW VECTOR).  IF THE INPUT IS TOO SMALL,
;                THE PREVIOUS TABLE ENTRY IS NOT DELETED.
;         7     READ CASE NUMBER AND REMOVE FURTHEST VECTOR FROM CASE
;                MEAN FOR THAT CASE
;         8     REMOVE LAST TABLE ENTRY MADE
;         9     READ A CASE NUMBER AND DISPLAY THE NUMBER OF VECTORS
;                IN THE TABLE FOR THAT CASE
;         A     READ A CASE NUMBER AND DISPLAY THE FIRST TABLE ENTRY
;                FOR THAT CASE
;         B     DISPLAY NEXT TABLE ENTRY
;         C     REMOVE THE LAST TABLE ENTRY DISPLAYED
;         D     DISPLAY ADDRESS OF BOTTOM AND TOP OF TABLE
;         E     READ A PROM INTO MEMORY
;         F     PROGRAM A PROM
;
;
;       ERROR MESSAGES:
;
;               0       INPUT VECTOR TOO SMALL ON CLASSIFY ROUTINE
;               1       INPUT VECTOR TOO SMALL ON TABLE ADD
;               2       INPUT VECTOR TOO SMALL ON TABLE PUNISH
;               3       CASE NOT IN TABLE
;
$EJECT
;       THE FORMAT FOR THE CASE NUMBERS IS AS FOLLOWS:
;
;       BIT #   0       1       2       3       4       5       6       7
;               DECRE-  INCRE-  DECRE-  INCRE-  DECRE-  INCRE-  DECRE-  INCRE-
;               MENT    MENT    MENT    MENT    MENT    MENT    MENT    MENT
;               SERVO   SERVO   SERVO   SERVO   SERVO   SERVO   SERVO   SERVO
;               # 3     # 3     # 2     # 2     # 1     # 1     # 0     # 0
```

```
;
;       THE SET POINTS ARE INITIALIZED AT 01H.  AFTER EACH SUCCESSFUL CLASSIFICATION,
;       THE SET POINTS ARE UPDATED TO REFLECT EITHER AN ADDITION OR SUBTRACTION
;       FROM THE OLD POINT.  A 'ONE' IN ANY BIT OF THE CASE NUMBER WILL RESULT
;       IN THE ALTERATION OF THE OLD SET POINT ACCORDING TO THE FORMAT SHOWN
;       ABOVE.  OVERFLOW AND UNDERFLOW ARE AUTOMATICALLY PREVENTED DURING THE
;       UPDATE
;
;       AFTER THE CLASSIFICATION AND UPDATE, THE NEW SERVO SET POINTS ARE
;       OUTPUT TO THE SERVO CONTROLLERS.
;
$EJECT
;
;       CONSTANTS
;
CRITER  EQU     4       ;INPUT CRITERION.  AT LEAST ONE VECTOR ELEMENT
                        ;MUST EXCEED THIS LIMIT FOR FURTHER PROCESSING
                        ;ON INPUT
DEBOUN  EQU     6       ;NUMBER OF DEBOUNCE CYCLES FOR INPUT
INPLUS  EQU     5       ;LOWER LIMIT FOR WAVEFORM POSITIVE
INZERO  EQU     2       ;UPPER LIMIT FOR WAVEFORM GROUND
KNN     EQU     10      ;NUMBER OF NEAREST NEIGHBORS TO SEARCH FOR
NVECIN  EQU     8       ;NUMBER OF INPUT VECTORS TO READ AND AVERAGE
                        ;DURING INPUT ROUTINE
VECLEN  EQU     5       ;NUMBER OF ELEMENTS IN A VECTOR TABLE ENTRY
$EJECT
;
;
;       STORAGE BUFFERS
;
;
        ORG     3C10H   ;LOWER END OF RAM MEMORY
VKNN    EQU     $
        DS      1       ;STORAGE VARIABLE FOR NUMBER OF NEAREST NEIGHBORS TO SEARCH FOR
VCRIT   EQU     $
        DS      1       ;CRITERION LIMIT VALUE FOR INPUT VECTORS
SRVINC  EQU     $
        DS      8       ;STORAGE LOCATIONS FOR SERVO INCREMENTS
                        ;  FORMAT:      PLUS INCREMENT FOR SERVO # 0
                        ;               MINUS INCREMENT FOR SERVO # 0
                        ;               PLUS INCREMENT FOR SERVO # 1
                        ;               ...
                        ;               MINUS INCREMENT FOR SERVO # 3
                        ;  NOTE: ALL INCREMENTS ARE POSITIVE.  MINUS INCREMENTS
                        ;        ARE SUBTRACTED FROM THE SET POINT.
TSERVO  EQU     $
        DS      4       ;SET POINT STORAGE LOCATIONS FOR THE SERVOS
BALLOT  EQU     $
        DS      2*KNN+1 ;BALLOT FOR NEAR NEIGHBOR VOTE
                        ;  FORMAT:      CASE # I
                        ;               # VOTES FOR CASE I
                        ;               CASE # J
                        ;               # VOTES FOR CASE J
                        ;               ...
                        ;               # VOTES FOR CASE Z
                        ;               0 - SIGNALS END OF BALLOT
CASENO  EQU     $
        DS      1       ;CASE NUMBER
TPOINT  EQU     $
        DS      2       ;POINTER FOR DISPLAYING TABLE
TSMALL  EQU     $
```

```
        DS      3*KNN   ;K-NEAREST-NEIGHBOR LIST
                        ;  FORMAT:       MSB OF DIFFERENCE METRIC 1
                        ;                LSB OF DIFFERENCE METRIC 1
                        ;                CASE # FOR VECTOR 1
                        ;                MSB METRIC 2
                        ;                ...
                        ;                CASE # FOR K-TH VECTOR
VNULL   EQU     $
        DS      4       ;FOUR COMPONENTS OF NULL INPUT VECTOR
VINPUT  EQU     $
        DS      4*NVECIN        ;STORAGE FOR INPUT VECTORS
                        ;  FORMAT:       COMPONENT 1 OF VECTOR 1
                        ;                     "      2 OF   "     1
                        ;                     "      3 OF   "     1
                        ;                     "      4 OF   "     1
                        ;                COMPONENT 1 OF VECTOR 2
                        ;                ...
                        ;                COMPONENT 4 OF VECTOR NVECIN
VDIFF   EQU     $
        DS      4       ;STORAGE FOR DIFFERENCE VECTOR COMPONENTS
VNORM   EQU     $
        DS      4       ;STORAGE FOR NORMALIZED VECTOR COMPONENTS
VTABLE  EQU     $
        DS      1       ;STORAGE FOR VECTORS TO USE IN CLASSIFICATION
                        ;  FORMAT:       CASE NUMBER OF VECTOR 1
                        ;                COMPONENT 1 OF VECTOR 1
                        ;                     "      2 OF   "     1
                        ;                     "      3 OF   "     1
                        ;                     "      4 OF   "     1
                        ;                CASE NUMBER OF VECTOR 2
                        ;                COMPONENT 1 OF VECTOR 2
                        ;                ...
                        ;                COMPONENT 4 OF VECTOR N
                        ;                0 - SIGNALS END OF TABLE
                        ;  NOTE:
                        ;       ALL VECTORS WITH EQUAL CASE NUMBERS
                        ;       ARE STORED SEQUENTIALLY IN TABLE
$EJECT
;
;       EXTERNAL REFERENCES
;
DCON1   EQU     02EFH   ;MONITOR BLANKING ROUTINE
SPAO    EQU     04EAH   ;KEYBOARD INPUT ROUTINE
DBYT    EQU     02FDH   ;MONITOR DISPLAY ROUTINE
USRINT  EQU     3C02H   ;USER INTERRUPT ADDRESS
MONITR  EQU     00E7H   ;MONITOR FUNCTION INPUT
EXPR    EQU     03C3H   ;ROUTINE TO READ FUNCTION PARAMETERS
DELAY   EQU     0312H   ;1 MILLISECOND DELAY ROUTINE
HILO    EQU     03F0H   ;TEST FOR HIGH AND LOW ADDRESS EQUALITY
RAM     EQU     3C00H   ;TOP OF MEMORY PAGE ADDRESS
ERRORM  EQU     003BH   ;MONITOR ERROR ROUTINE
DREQC   EQU     0341H   ;DISPLAY MEMORY ROUTINE
KEYT    EQU     0529H   ;KEYBOARD CHARACTER READ
RSTRT   EQU     0069H   ;MONITOR RESTART SEQUENCE
$EJECT
;
;       THIS SECTION OF CODE REPLACES THE MONITOR BRANCH TABLE
;       AT THE BOTTOM OF PROM # 3
;
        ORG     0800H   ;STARTING ADDRESS OF PROM 3
READP:  JMP     START   ;ALL UNUSED MONITOR FUNCTIONS RESULT IN RESTART
```

```
WRITP:  JMP     RESTRT  ;EXCEPT THE ONE AT 0800H
PG08P:  JMP     PGM08   ;PROGRAM A PROM
CP08P:  JMP     RESTRT  ;UNUSED
TR08P:  JMP     TRN08   ;READ A PROM
MOVP:   JMP     RESTRT  ;UNUSED
HXAP:   JMP     RESTRT  ;UNUSED
SR8P:   JMP     RESTRT  ;UNUSED
SR16P:  JMP     RESTRT  ;UNUSED
$EJECT
;*************************************************
;*                                               *
;*      COMMAND INTERPRETER SEQUENCE : KNN       *
;*                                               *
;*************************************************
;
;       START-UP ROUTINE - BLANK TABLE AND SET UP USER INTERRUPT
;
START:  MVI     A,0     ;GET ZERO
        STA     VTABLE  ;BLANK FIRST TABLE POSITION
        MVI     A,KNN   ;SET UP THE DEFAULT NEAREST NEIGHBOR COUNT
        STA     VKNN    ;IN THE RAM MEMORY.  THIS MAY BE CHANGED
        MVI     A,CRITER        ;BY THE OPERATOR, AS WELL AS THE
        STA     VCRIT   ;CRITERION VALUE SET BY THIS STEP.
        MVI     A,10H   ;SET UP THE SERVO INCREMENT AND DECREMENT TABLE,
        MVI     B,12    ;AND INITIALIZE THE SET POINTS TO ONE
        LXI     H,SRVINC        ;GET THE POINTER TO THE SERVO TABLES
INIT:   MOV     M,A     ;PUT IN THE ONE
        INX     H       ;BUMP THE POINTER
        DCR     B       ;DROP THE COUNTER
        JNZ     INIT    ;KEEP SETTING UNTIL FINISHED
JUMP:   JMP     RESTRT  ;THIS WILL BE STORED AT THE USER INTERRUPT
RESTRT: LXI     D,USRINT        ;SET UP THE USER INTERRUPT FIRST
        LXI     H,JUMP  ;GET POINTER TO JUMP COMMAND
        MVI     B,3     ;TRANSFER 3 BYTES
        CALL    SHIFT   ;SHIFT THE BLOCK
        MVI     A,82H   ;SET UP THE I/O PORTS
        OUT     0EBH    ;TO NORMAL PATTERN
        MVI     A,7     ;SET HARDWARE MULTIPLY TO
        OUT     0EAH    ;8-BIT UNSIGNED MULTIPLY
;
;       RESTART - READ COMMAND AND EXECUTE
;
        EI              ;ENABLE THE INTERRUPTS
        MVI     A,17H   ;SET UP (A) AND
        MVI     E,0     ;(E) TO PREPARE FOR INPUT
        CALL    SPAO    ;GET COMMAND
        MOV     A,L     ;CHECK IF TWO DIGIT COMMAND
        CPI     10H     ;IF YES,
        JP      RESTRT  ;READ AGAIN
        PUSH    H       ;REMEMBER THE COMMAND,
        CALL    DBLK    ;CLEAR THE DISPLAY,
        POP     H       ;AND GET THE COMMAND BACK
;
;       NOW JUMP TO IMPLIED ROUTINE
;
        MOV     E,L     ;GET THE COMMAND NUMBER
        LXI     H,RESTRT        ;SET UP PSEUDO RETURN ADDRESS
        PUSH    H       ;TO SIMULATE A SUBROUTINE CALL
        LXI     H,CTABL ;GET BASE OF TABLE
        MVI     D,0     ;ADD (BASE + 2 * INDEX)
        DAD     D       ;TO THE (HL)
```

```
        DAD     D       ;REGISTER PAIR
        MOV     A,M     ;NOW GET THE BRANCH LOCATION
        INX     H       ;INTO THE
        MOV     H,M     ;(HL) REGISTER PAIR
        MOV     L,A     ;TO ALLOW A
        PCHL            ;JUMP TO THAT LOCATION
;
;       COMMAND BRANCH TABLE
;
CTABL:  DW      CONTO   ;CONTINUOUS RUN, INPUT DISPLAY
        DW      CONT1   ;CONTINUOUS RUN, NORMALIZED DISPLAY
        DW      SINGLO  ;SINGLE VECTOR, INPUT DISPLAY
        DW      SINGL1  ;SINGLE VECTOR, NORMALIZED DISPLAY
        DW      NULL    ;INPUT NEW NULL VECTOR
        DW      TABADD  ;ADD VECTOR TO TABLE
        DW      TABPUN  ;PUNISH THE TABLE
        DW      TABDEL  ;DELETE TABLE ENTRY
        DW      REMLST  ;REMOVE LAST ENTRY MADE IN TABLE
        DW      NUMBER  ;DISPLAY # VECTORS IN TABLE FOR GIVEN CASE
        DW      DISP1   ;DISPLAY FIRST TABLE ENTRY FOR GIVEN CASE
        DW      DISPN   ;DISPLAY NEXT TABLE ENTRY
        DW      REMDIS  ;REMOVE THE LAST TABLE ENTRY DISPLAYED
        DW      ADDRES  ;DISPLAY BOTTOM AND TOP ADDRESSES OF TABLE
        DW      PREAD   ;PROM READ ROUTINE
        DW      PBURN   ;PROM BURNING ROUTINE
$EJECT
;
;       THIS SEQUENCE CONDUCTS VECTOR INPUT AND CLASSIFICATION, WITH
;       EITHER CONTINUOUS RUN OR A SINGLE PASS THROUGH, AND EITHER
;       INPUT VECTOR OR NORMALIZED DISPLAY.
;
;       PSW IS PUSHED ONTO THE STACK WITH THE FOLLOWING MEANINGS:
;               CARRY -     SET   CONTINUOUS LOOP
;                           RESET SINGLE PASS
;               (A)   -     0     (INPUT - NULL) VECTOR DISPLAY
;                           1     NORMALIZED DISPLAY
;
;       CONTINUOUS RUN, INPUT DISPLAY
;
CONTO:  XRA     A       ;SET (A) TO ZERO
        STC             ;SET CARRY
        JMP     SPUSH   ;GO PUSH ONTO STACK
;
;       CONTINUOUS RUN, NORMALIZED DISPLAY
;
CONT1:  XRA     A       ;SET (A) TO
        INR     A       ;1 AND
        STC             ;SET CARRY
        JMP     SPUSH   ;GO PUSH ONTO STACK
;
;       SINGLE PASS, INPUT DISPLAY
;
SINGLO: XRA     A       ;RESET (A) AND CARRY
        JMP     SPUSH   ;GO PUSH ONTO STACK
;
;       SINGLE PASS, NORMALIZED DISPLAY
;
SINGL1: XRA     A       ;SET (A) TO
        INR     A       ;1 AND RESET CARRY
;
;       NOW PUSH FLAGS ONTO THE STACK AND
```

```
;       DO THE INPUT ROUTINE, CLASSIFY, AND DISPLAY
;
SPUSH:  PUSH    PSW     ;PUSH (A) AND CARRY ONTO STACK
        MVI     A,070H  ;SET THE TIMING MARKS
        OUT     0EAH    ;AND OUTPUT TO (EA)
        XRA     A       ;CLEAR THE MARKS
        OUT     0EAH    ;BY OUTPUTING NULL
        CALL    INPUT   ;GET INPUT VECTORS
        MVI     A,0     ;ERROR RETURN, SET ERROR FLAG TO
        JMP     ERROR   ;ZERO AND PROCESS THE ERROR
        CALL    CLASSN  ;FIND OUT WHAT CASE NUMBER
        MVI     A,070H  ;SET THE TIMING MARKS AGAIN
        OUT     0EAH    ;AND OUTPUT TO (EA)
        XRA     A       ;CLEAR THE MARKS
        OUT     0EAH    ;AND CLEAR THE PORT
        MOV     A,B     ;PUT CASE NUMBER IN
        STA     CASENO  ;STORAGE
        POP     PSW     ;RETRIEVE (A) BUT SAVE
        PUSH    PSW     ;THE CARRY BIT
        ORA     A       ;SEE IF (A)=0
        JNZ     DNORM   ;NO, DISPLAY NORM
        LXI     H,VINPUT        ;YES, (INPUT - NULL) IS TO BE SHOWN
        JMP     OUTPUT  ;GO DISPLAY
DNORM:  LXI     H,VNORM ;DISPLAY THE COMPONENTS AND CASE #
OUTPUT: CALL    DISPLA  ;ON THE CONSOLE
OUT1:   POP     PSW     ;NOW CHECK THE CARRY FLAG
        JC      SPUSH   ;CARRY SET, KEEP CHECKING
        RET             ;SINGLE PASS, DONE
;
;       REPLACE NULL VECTOR
;
NULL:   XRA     A       ;SET (A) TO ZERO
        STA     CASENO  ;SET CASE # TO ZERO
        STA     VNULL   ;AND BLANK NULL VECTOR
        CALL    INPUT   ;GET VECTOR
        MVI     B,4     ;SET (B) TO MOVE 4 BYTES
        XCHG            ;SWAP THE POINTERS
        CALL    SHIFT   ;(HL),(DE) POINTERS SET, SHIFT
        LXI     H,VNULL ;NOW DISPLAY NULL VECTOR
        JMP     DISPLA  ;WITH ZERO CASE # AND RETURN TO COMMAND LOOP
;
;       ADD VECTOR TO TABLE
;
TABADD: CALL    CASE    ;GET CASE #
        CALL    INPUT   ;GET INPUT VECTOR
        MVI     A,1     ;TOO SMALL, GO
        JMP     ERROR   ;DISPLAY ERROR AND RETURN TO COMMAND LOOP
TABADO: CALL    SEARCH  ;FIND THE CORRECT SLOT IN THE TABLE
        PUSH    D       ;SAVE (DE)
        MVI     L,0-VECLEN      ;CLEAR A SPOT IN THE TABLE
        CALL    SHUTTL  ;TO ACCOMODATE THE NEW VECTOR
        CALL    MOVE    ;PUT NORMALIZED VECTOR IN TABLE
        POP     H       ;GET POINTER
        INX     H       ;POINT TO FIRST COMPONENT
        JMP     DISPLA  ;DISPLAY NEW VECTOR AND RETURN TO COMMAND LOOP
;
;       PUNISH TABLE OR DELETE TABLE ENTRY
;
TABPUN: STC             ;SET CARRY TO ACT AS FLAG
        JMP     ALTER   ;GO AHEAD
TABDEL: XRA     A       ;RESET CARRY FOR FLAG
```

```
ALTER:  PUSH    PSW     ;SAVE CARRY BIT
        CALL    CASE    ;READ CASE #
        POP     PSW     ;SEE IF CARRY SET
        JNC     ALTER0  ;NO, GO FIND FURTHEST
;
;       READ VECTOR AND REMEMBER IT
;
        CALL    INPUT   ;READ VECTOR
        MVI     A,2     ;ERROR # 2 IF TOO SMALL
        JMP     ERROR   ;GO DISPLAY MESSAGE
        STC             ;SET CARRY AGAIN
ALTER0: PUSH    PSW     ;REMEMBER CARRY BIT
        CALL    GREAT   ;FIND FURTHEST VECTOR FROM MEAN
        POP     PSW     ;IF NO CARRY
        JNC     ALTER1  ;DELETE THIS ENTRY
        MVI     L,VECLEN        ;GET THE TABLE ENTRY LENGTH
        CALL    SHUTTL  ;REMOVE THE FURTHEST VECTOR
        JMP     TABADD  ;GO ADD THE NEW VECTOR INTO THE TABLE
;
;       REMOVE LAST ENTRY FROM TABLE
;
REMLST: CALL    SEARCH  ;FIND ENTRY IN TABLE
;
;       DISPLAY THE VECTOR BEING REMOVED
;
ALTER1: PUSH    D       ;SAVE THE ADDRESS
        INX     D       ;POINT TO THE FIRST VECTOR COMPONENT
        XCHG            ;PUT THE ADDRESS IN (HL)
        CALL    DISPLA  ;GO DISPLAY IT
        POP     D       ;GET THE ADDRESS BACK
        MVI     L,VECLEN        ;SET UP TO REMOVE ENTRY
        JMP     SHUTTL  ;GO BACK TO THE COMMAND LOOP AFTER DELETION
;
;       DISPLAY # VECTORS FOR GIVEN CASE
;
NUMBER: CALL    CASE    ;GET CASE #
        LXI     D,3FF7H ;DISPLAY
        CALL    DISPLC  ;CASE #
        PUSH    D       ;SAVE THE POINTER
        CALL    SEARCH  ;FIND THE CASE IN THE TABLE
        CALL    COUNT   ;COUNT THE VECTORS
        POP     D       ;AND DISPLAY THE
        MOV     A,B     ;THE COUNT
        JMP     DISPL1  ;AND RETURN TO COMMAND LOOP
;
;       DISPLAY FIRST TABLE ENTRY FOR GIVEN CLASS
;
DISP1:  CALL    CASE    ;READ CASE #
        CALL    SEARCH  ;FIND IN TABLE
        XCHG            ;PUT THE POINTER INTO (HL)
        SHLD    TPOINT  ;SAVE THE POINTER
        INX     H       ;BUMP TO POINT TO THE VECTOR COMPONENTS
        JMP     DISPLA  ;DISPLAY AND RETURN TO COMMAND LOOP
;
;       DISPLAY NEXT TABLE ENTRY
;
DISPN:  LHLD    TPOINT  ;GET THE OLD POINTER
        LXI     D,VECLEN        ;BUMP TO THE NEXT CASE
        DAD     D       ;POINT TO THE NEW
        SHLD    TPOINT  ;CASE, STORE BACK IN POINTER SPACE
        MOV     A,M     ;READ THE NUMBER
```

```
        STA     CASENO  ;AND STORE IT IN (CASENO)
        INX     H       ;POINT TO THE VECTOR COMPONENTS
        JMP     DISPLA  ;NOW DISPLAY IT, AND RETURN TO COMMAND LOOP
;
;       REMOVE THE LAST VECTOR DISPLAYED
;
REMDIS: LHLD    TPOINT  ;FIRST CORRECT THE TABLE POINTER FOR DISPLAY
        LXI     D,0-VECLEN      ;BY SUBTRACTING THE VECTOR LENGTH
        XCHG            ;SET UP THE POINTER IN (DE)
        DAD     D       ;FROM THE OLD POINTER
        SHLD    TPOINT  ;AND REPLACE THE POINTER
        JMP     ALTER1  ;AND GO DELETE FROM THE TABLE
;
;       DISPLAY TOP AND BOTTOM TABLE ADDRESSES
;
ADDRES: MVI     A,0     ;SET CASE NUMBER
        STA     CASENO  ;TO ZERO
        MOV     B,A     ;FIND END
        CALL    SEARCH  ;OF TABLE
        LXI     H,VINPUT+3      ;PREPARE POINTER TO MEMORY
        LXI     B,VTABLE        ;GET ADDRESS OF BEGINNING OF TABLE
        MOV     M,E     ;PUT BOTH
        DCX     H       ;ADDRESSES INTO
        MOV     M,D     ;THE STORAGE SPACES
        DCX     H       ;USED FOR
        MOV     M,C     ;THE INPUT
        DCX     H       ;VECTOR STORAGE
        MOV     M,B     ;THEN DISPLAY
        JMP     DISPLA  ;THE DATA AND RETURN TO THE COMMAND LOOP
;
;       READ A PROM INTO MEMORY
;
PREAD:  POP     H       ;RESET THE STACK
        MVI     A,99H   ;SET THE PARAMETERS FOR
        LXI     D,4     ;A MONITOR CALL
        JMP     MONITR  ;THEN GO TO THE MONITOR ROUTINE
;
;       GO TO THE PROM BURNING ROUTINE
;
PBURN:  POP     H       ;RESTORE STACK
        MVI     A,0A4H  ;GET PARAMETERS
        LXI     D,2     ;NECESSARY FOR MONITOR FUNCTION 2
        JMP     MONITR  ;GO PROCESS THE PROM BURN
$EJECT
;*************************
;*                       *
;*   SUBROUTINE AVERAG   *
;*                       *
;*************************
;
;       TAKE THE AVERAGE OF EACH OF THE FIRST 4 COMPONENTS OF
;       SPECIFIED VECTORS.  CAN AVERAGE IN PLACE.
;
;       INPUT   (B)     # VECTORS TO AVERAGE
;               (C)     LENGTH OF VECTORS (SPACINGS OF FIRST ELEMENTS)
;               (HL)    POINTER TO FIRST VECTOR
;               (DE)    POINTER TO STORAGE FOR AVERAGED COMPONENTS
;
;       OUTPUT  NONE
;
;       DESTROYS ALL REGISTERS
```

```
;
AVERAG: MVI     A,4     ;PREPARE COUNTER FOR 4 ELEMENTS
AVER0:  PUSH    PSW     ;REMEMBER THE COUNT
        PUSH    D       ;REMEMBER THE STORAGE POINTER
        PUSH    H       ;AND THE MEMORY POINTER
        PUSH    B       ;AND THE COUNTERS
        LXI     D,0     ;SET THE SUM TO ZERO
AVER1:  XRA     A       ;CLEAR (A) AND CARRY
        MOV     A,M     ;GET VECTOR COMPONENT
        ADD     E       ;ADD IN THE NEW VALUE TO THE SUM

        MOV     E,A     ;AND RESTORE IT
        JNC     AVER2   ;SEE IF OVERFLOW
        INR     D       ;YES, INCREMENT HIGH BYTE
AVER2:  MOV     A,B     ;SAVE THE COUNTER
        MVI     B,0     ;SET UP POINTER INCREMENT
        DAD     B       ;BUMP THE POINTER
        MOV     B,A     ;AND RESTORE THE COUNTER
        DCR     B       ;DECREMENT THE COUNT
        JNZ     AVER1   ;IF NOT ENOUGH VECTORS, KEEP ADDING
;
;       NOW THAT THE SUM IS FORMED, DO THE DIVISION FOR THE MEAN
;
        POP     B       ;GET BACK THE # OF VECTORS
        MOV     A,B     ;AND PUT IT INTO (A)
        XCHG            ;PUT THE SUM IN (HL)
AVER3:  ADD     A       ;SHIFT THE DIVISOR
        JZ      AVER4   ;IF A SINGLE BIT, GO AHEAD AND STORE
        JC      DIVIDE  ;IF NOT 2**N DIVIDE, GO DO LONG DIVISION
        DAD     H       ;SHIFT THE SUM
        JMP     AVER3   ;AND KEEP CHECKING THE DIVISOR
AVER4:  DAD     H       ;SHIFT ONE LAST TIME
        MOV     A,H     ;SAVE AVERAGE
AVER5:  POP     H       ;GET POINTERS BACK
        POP     D       ;FOR STORAGE
        STAX    D       ;STORE THE AVERAGE
        INX     D       ;BUMP POINTERS TO STORAGE
        INX     H       ;AND VECTOR LIST
        POP     PSW     ;GET COUNTER BACK,
        DCR     A       ;DROP IT AND SEE IF ZERO
        JNZ     AVER0   ;NOT DONE, LOOP AGAIN
        RET             ;DONE, EXIT FROM SUBROUTINE
;
;       DIVISOR WAS NOT POWER OF 2, SO LONG DIVISION
;
DIVIDE: RAR             ;RESTORE SHIFTED DIVISOR IN (A)
        RAR             ;BY SHIFTING RIGHT TWICE
        PUSH    B       ;SAVE THE COUNTER
;
;       PUT POSITIVE DIVISOR IN (BC)
;       PUT NEGATIVE DIVISOR IN (DE)
;       THEN DO REPEATED SUBTRACTS AND SHIFTS
;
        MOV     B,A     ;UPPER BYTE OF +
        MVI     A,0     ;LOWER BYTE OF +
        MOV     C,A     ;SET TO ZERO, AS WELL AS
        MOV     E,A     ;LOWER BYTE OF -
        SUB     B       ;MAKE NEGATIVE DIVISOR
        MOV     D,A     ;AND PUT INTO UPPER BYTE OF -
        XRA     A       ;RESTORE (A) TO 0
        INR     A       ;SET THE FLAG BIT
```

```
;
;       NOW DO THE SUBTRACTIONS
;
DIVID0: DAD     D       ;TRY SUBTRACTING
        JC      DIVID1  ;REMAINDER > DIVISOR, GO AHEAD TO SHIFT
        DAD     B       ;RESTORE SUM
        ORA     A       ;CLEAR CARRY
DIVID1: RAL             ;ROTATE IN THE NEW BIT
        JC      DIVID2  ;IF CARRY NOW, FINISHED
        DAD     H       ;SHIFT REMAINDER
        JMP     DIVID0  ;KEEP SUBTRACTING
DIVID2: POP     B       ;GET THE COUNTER BACK
        JMP     AVER5   ;GO BACK TO MAIN LOOP
$EJECT
;************************
;*                      *
;*   SUBROUTINE CASE    *
;*                      *
;************************
;
;       INPUT A CASE NUMBER FROM THE CONSOLE (2 DIGITS)
;
;       INPUT   NONE
;
;       OUTPUT  (CASENO)        CASE #
;
;       DESTROYS ALL REGISTERS
;
CASE:   CALL    DBLK    ;BLANK OUT THE DISPLAY
;
;       PUT 'CASE' IN THE DISPLAY
;
        MVI     B,4     ;SET UP TO MOVE 4 BYTES
        LXI     D,3FF2H ;FROM THE MEMORY TO THE
        LXI     H,CTABLE        ;DISPLAY LOCATIONS
        CALL    SHIFT   ;MOVE THE PATTERN
;
;       INPUT THE CASE #
;
        MVI     A,17H   ;SET UP (A) AND
        MVI     E,0     ;(E) TO PREPARE FOR INPUT
        CALL    SPA0    ;GET THE INPUT
        MOV     A,L     ;MOVE THE RESULT INTO
        STA     CASENO  ;(A) AND STORE IN CASE # STORAGE
        CALL    DBLK    ;ERASE THE DISPLAY
        RET             ;DONE WITH INPUT
;
;       'CASE'
;
CTABLE: DB      10000110B       ;'E'
        DB      10010010B       ;'S'
        DB      10001000B       ;'A'
        DB      11000110B       ;'C'
$EJECT
;************************
;*                      *
;*   SUBROUTINE CLASSN  *
;*                      *
;*                      *
;************************
;
```

```
;       CLASSIFY THE VECTOR IN VNORM BY THE K-NEAREST NEIGHBOR RULE,
;       UPDATE THE SET POINTS, AND OUTPUT THEM TO THE SERVO DRIVERS
;
;       INPUT   NONE
;
;       OUTPUT  (B)     NEAREST NEIGHBOR CASE #
;
;       DESTROYS ALL REGISTERS
;
CLASSN: MVI     B,0FFH  ;GET ONES INTO (B)
;
;       FIRST FILL THE K-NEAREST NEIGHBOR TABLE WITH ONES
;
        LXI     H,TSMALL        ;PREPARE TABLE POINTER
        LDA     VKNN    ;GET THE NUMBER OF NEIGHBORS
        MOV     C,A     ;REMEMBER IT IN (C)
        ADD     A       ;AND MULTIPLY BY THREE
        ADD     C       ;TO DETERMINE THE TABLE LENGTH
CLASS0: MOV     M,B     ;START FILLING THE TABLE
        INX     H       ;BUMP THE POINTER
        DCR     A       ;AND DROP THE COUNT
        JNZ     CLASS0  ;KEEP FILLING UNTIL DONE
;
;       NOW GO THROUGH THE VECTOR TABLE AND FORM THE DIFFERENCES
;       BETWEEN THE INPUT AND THE TABLE VECTORS, FINDING THE SMALLEST
;       K VALUES AND REMEMBERING THEIR CASE NUMBERS
;
        LXI     H,VTABLE        ;START AT THE BEGINNING OF THE TABLE
;
;       TEST TO SEE IF LOOPING IS FINISHED
;
CLASS1: MOV     A,M     ;GET VALUE FROM TABLE
        ANA     A       ;AND TEST IF ZERO
        JZ      VOTE    ;YES, GO TAKE VOTE
;
;       GET THE METRIC OF THE DIFFERENCE
;
        PUSH    PSW     ;SAVE THE CASE NUMBER
        LXI     D,VNORM ;POINT TO THE NORMALIZED VECTOR
        CALL    MINUS   ;CALCULATE THE DIFFERENCE METRIC
;
;       CHECK AGAINST THE K-NEAREST-NEIGHBOR TABLE
;
        POP     B       ;RETRIEVE THE CASE NUMBER
        PUSH    H       ;SAVE THE POINTER
        CALL    SMALL   ;NOW CHECK AGAINST THE TABLE
;
;       GO TO THE NEXT VECTOR UNTIL DONE
;
        POP     H       ;RETRIEVE VTABLE POINTER
        JMP     CLASS1  ;AND GO BACK
;
;       NOW WHEN DONE WITH SEARCHING THE TABLE, TAKE A VOTE
;
;       THE BALLOT LOOKS LIKE THIS:
;
;               (BALLOT)        CASE # I
;                               # VOTES FOR CASE I
;                               CASE # J
;                               # VOTES FOR CASE J
;                               ...
```

```
;                                   0 - TO INDICATE THE END OF THE BALLOT
;
VOTE:   LXI     H,BALLOT        ;FIRST CLEAR THE BALLOT
        MVI     B,0     ;PREPARE A ZERO
        LDA     VKNN    ;FIND THE BALLOT LENGTH
        MOV     C,A     ;REMEMBER IT IN (C)
        ADD     A       ;DOUBLE THE NEAR NEIGHBOR COUNT
        INR     A       ;AND ADD ONE FOR THE BALLOT LENGTH
VOTE0:  MOV     M,B     ;PUT IN ZERO
        INX     H       ;BUMP THE POINTER
        DCR     A       ;DROP THE COUNT
        JNZ     VOTE0   ;KEEP GOING UNTIL DONE
;
;       NOW GO THROUGH THE K-NEAREST NEIGHBORS
;       AND TAKE THE VOTE
;
        LXI     D,TSMALL+2      ;POINTER TO THE FIRST CASE #
VOTE1:  LXI     H,BALLOT        ;SET BALLOT POINTER
VOTE2:  MOV     A,M     ;FIRST SEE IF THE BALLOT IS EMPTY
        ORA     A       ;SET THE FLAGS AND
        LDAX    D       ;GET THE NEW CASE NUMBER IN CASE
        JZ      VOTE3   ;THE BALLOT IS EMPTY, IN WHICH CASE
        CMP     M       ;FILL IT.  OTHERWISE, CHECK TO SEE
        JZ      VOTE4   ;IF THE NEW CASE EQUALS THE BALLOT ENTRY.
        INX     H       ;NO, GO TO THE NEXT BALLOT POSITION
        INX     H       ;AND CHECK AGAIN
        JMP     VOTE2   ;TO SEE IF MATCH
VOTE3:  MOV     M,A     ;PUT IN THE CASE #
VOTE4:  INX     H       ;GO TO VOTE COUNT
        INR     M       ;AND INCREMENT IT
        INX     D       ;GO TO
        INX     D       ;THE NEXT
        INX     D       ;NEAR NEIGHBOR CASE
        DCR     C       ;DROP THE COUNT
        JNZ     VOTE1   ;NOT DONE, KEEP VOTING
;
;       NOW THAT THE BALLOT IS FULL, SEE
;       WHO GOT THE MOST VOTES
;
        LXI     H,BALLOT        ;GET THE BALLOT POINTER
VOTE5:  MOV     B,M     ;(B) HAS THE HIGHEST CASE #
        INX     H       ;BUMP POINTER
        MOV     C,M     ;(C) HAS # VOTES FOR THIS CASE
VOTE6:  INX     H       ;CHECK THE NEXT BALLOT POSITION
        MOV     A,M     ;FIRST SEE IF ZERO
        ANA     A       ;TO INDICATE BALLOT END
        JZ      SERVO   ;IF YES, UPDATE THE SERVO SET POINTS
        INX     H       ;NO, SEE IF HIGHER VOTE COUNT
        MOV     A,C     ;GET THE OLD VOTE COUNT
        CMP     M       ;AND COMPARE AGAINST THE NEW COUNT
        JP      VOTE6   ;NO, GO TO NEXT ENTRY
        DCX     H       ;YES, DROP BACK AND PUT
        JMP     VOTE5   ;NEW LEADER INTO WINNER'S SLOT
;
;       UPDATE THE SERVO SET POINTS AND OUTPUT THE NEW POINTS
;
SERVO:  PUSH    B       ;REMEMBER THE CASE NUMBER
        MVI     C,4     ;SET THE COUNTER FOR 4 SERVOS
        LXI     D,SRVINC        ;POINT TO THE SERVO INCREMENT TABLE
        LXI     H,TSERVO        ;AND THE SERVO SET POINT TABLE
SERVO0: MOV     A,B     ;DECODE THE CASE NUMBER
```

```
        RRC             ;CHECK THE BIT
        MOV     B,A     ;REMEMBER THE ROTATED CASE NUMBER
        JNC     SERVO2  ;BIT WAS ZERO SO NO ACTION ON THIS POINT
        LDAX    D       ;GET THE INCREMENT
        ADD     M       ;ADD IT TO THE OLD SET POINT
        JNC     SERVO1  ;ON OVERFLOW,
        MVI     A,0FFH  ;SET TO MAXIMUM EXCURSION
SERVO1: MOV     M,A     ;RESET THE SET POINT
SERVO2: INX     D       ;GO TO THE NEXT INCREMENT TABLE LOCATION
        MOV     A,B     ;CHECK THE NEXT BIT
        RRC             ;BY ROTATING IT INTO CARRY
        MOV     B,A     ;REMEMBER THE ROTATED CASE NUMBER
        JNC     SERVO4  ;IF ZERO AGAIN NO ACTION
        XCHG            ;SWAP THE POINTERS
        LDAX    D       ;GET THE OLD SET POINT
        SUB     M       ;SUBTRACT THE INCREMENT
        XCHG            ;REPLACE THE POINTERS
        JNC     SERVO3  ;ON UNDERFLOW
        MVI     A,1     ;RESET THE SERVO TO THE MINIMUM EXCURSION
SERVO3: MOV     M,A     ;AND RESET THE NEW SET POINT
SERVO4: INX     D       ;BUMP THE POINTER AGAIN
        INX     H       ;AS WELL AS THE SERVO POINTER
        DCR     C       ;DROP THE COUNTER AND KEEP CHECKING
        JNZ     SERVO0  ;UNTIL IT IS ZERO
        MVI     C,4     ;FINISHED WITH THE UPDATING
        MVI     B,0     ;NOW OUTPUT THE NEW SET POINTS
        LXI     H,TSERVO        ;POINT TO THE NEW POINTS
SERVO5: MOV     A,M     ;GET THE VALUE
        OUT     0E8H    ;PUT IT OUT TO THE PORT
        MOV     A,B     ;GET THE SERVO NUMBER
        OUT     0EAH    ;AND STROBE THE STORAGE LINE
        ORI     80H     ;TELL THE CONTROLLER TO OUTPUT THE PULSE
        OUT     0EAH    ;TO THE SERVO DRIVER
        ANI     7FH     ;THEN RESET THE
        OUT     0EAH    ;STROBE TO START THE OUTPUT PROCESS
        INX     H       ;BUMP THE TABLE POINTER
        MOV     A,B     ;AND THE SERVO COUNTER
        ADI     10H     ;TO THE NEXT SERVO NUMBER
        MOV     B,A     ;REMEMBER IT AGAIN
        DCR     C       ;DROP THE COUNT
        JNZ     SERVO5  ;IF NOT DONE, KEEP OUTPUTTING
        POP     B       ;GET THE CASE NUMBER BACK
        RET             ;AND RETURN TO THE MAIN LOOP
$EJECT
;************************
;*                      *
;*   SUBROUTINE COUNT   *
;*                      *
;************************
;
;       COUNT THE NUMBER OF ENTRIES IN THE TABLE FOR A GIVEN
;       CASE NUMBER
;
;       INPUT   (DE)    POINT TO FIRST CASE ENTRY IN TABLE
;               (CASENO)  CASE #
;
;       OUTPUT  (B)     # OF TABLE ENTRIES FOR CASE
;
;       DESTROYS (A),(HL)
;
COUNT:  PUSH    D       ;SAVE ADDRESS
```

```
        LXI     H,VECLEN        ;SET UP THE INCREMENT
        XCHG                    ;FOR TABLE POINTER
        MVI     B,0             ;SET COUNT TO ZERO
        LDA     CASENO          ;GET CASE #
COUNTO: CMP     M               ;AND BEGIN COMPARE AND COUNT
        JNZ     COUNT1          ;RETURN WHEN COUNTING DONE
        INR     B               ;NOT DONE, BUMP COUNT
        DAD     D               ;AND TABLE POINTER
        JMP     COUNTO          ;AND KEEP CHECKING
COUNT1: POP     D               ;RESTORE ADDRESS
        RET                     ;AND RETURN
$EJECT
;************************
;*                      *
;*   SUBROUTINE DBLK    *
;*                      *
;************************
;
;       BLANKS THE DISPLAY
;
;       INPUT   NONE
;
;       OUTPUT  NONE
;
;       DESTROYS ALL REGISTERS
;
DBLK:   MVI     A,0FFH  ;GET BLANK CHARACTER
        MVI     B,16    ;BLANK ENTIRE SCREEN
        CALL    DCON1   ;CALL MONITOR ROUTINE
        RET             ;RETURN
$EJECT
;*************************
;*                       *
;*   SUBROUTINE DISPLA   *
;*                       *
;*************************
;
;       DISPLAYS FOUR VALUES ON THE CONSOLE, WITH THE CASE NUMBER;
;       OR DISPLAYS CASE NUMBER OR (A) ALONE
;
;       INPUT   (HL)    POINTER TO FIRST VALUE TO DISPLAY
;
;       OUTPUT  (HL)    POINTS TO POSITION AFTER LAST VALUE DISPLAYED
;
;       DESTROYS ALL REGISTERS
;
DISPLA: LXI     D,3FFDH ;SET THE POINTER TO LEFT DIGIT IN DISPLAY
        MVI     C,4     ;COUNTER SET TO DISPLAY 4 VALUES
DISPLO: MOV     A,M     ;GET VALUE
        CALL    DBYT    ;OUTPUT IT
        INX     H       ;BUMP THE POINTER
        DCR     C       ;DROP THE COUNTER
        JNZ     DISPLO  ;KEEP DISPLAYING UNTIL DONE
;
;       ENTRY POINT FOR CASE NUMBER ONLY DISPLAY
;
DISPLC: LDA     CASENO  ;GET THE CASE #
;
;       ENTRY POINT FOR (A) DISPLAY ONLY
;
DISPL1: DCX     D       ;DROP THE ADDRESS COUNT
```

```
        DCX     D       ;TO LEAVE TWO BLANKS IN THE DISPLAY
        CALL    DBYT    ;NOW DISPLAY (A)
        RET             ;AND RETURN
$EJECT
;************************
;*                      *
;*   SUBROUTINE ERROR   *
;*                      *
;************************
;
;       DISPLAY ERROR MESSAGE
;
;       INPUT   (A)     ERROR #
;
;       OUTPUT  NONE
;
;       DESTROYS (B),(C),(D),(E),(H),(L)
;
;       AFTER DISPLAY IS FINISHED
;               IF (A) > 0, RETURN
;               IF (A) = 0, JUMP TO OUT1
;
ERROR:  PUSH    PSW     ;REMEMBER (A)
        CALL    DBLK    ;BLANK THE DISPLAY
        LXI     H,EMSG  ;WRITE 'ERROR'
        LXI     D,3FF9H ;INTO DISPLAY MEMORY
        MVI     B,5     ;TOTAL OF 5 LETTERS
        CALL    SHIFT   ;MOVE THE CHARACTERS
        POP     PSW     ;NOW RECALL (A)
        PUSH    PSW     ;BUT KEEP IT ON THE STACK
        LXI     D,3FF7H ;WRITE THE ERROR #
        CALL    DBYT    ;ON THE DISPLAY
        POP     PSW     ;CHECK (A) ONE MORE TIME
        ANA     A       ;IF NON-ZERO
        RNZ             ;RETURN, OTHERWISE
        JMP     OUT1    ;GO BACK TO THE INPUT SEQUENCE
;
;       ERROR MESSAGE
;
EMSG:   DB      10101111B       ;'R'
        DB      10100011B       ;'O'
        DB      10101111B       ;'R'
        DB      10101111B       ;'R'
        DB      10000110B       ;'E'
$EJECT
;************************
;*                      *
;*   SUBROUTINE FINDO   *
;*                      *
;************************
;
;       DEBOUNCES THE ZERO LEVEL ON INPUT
;
;       INPUT   NONE
;
;       OUTPUT  NONE
;
;       DESTROYS (A),(B)
;
FINDO:  MVI     B,DEBOUN        ;SET (B) TO DEBOUNCE COUNT
FINDO1: IN      0E9H    ;READ VALUE FROM A/D
```

```
        CPI     INZERO  ;IS VALUE < INPUT LIMIT?
        JNC     FINDO   ;NO, RESET AND CHECK AGAIN
        DCR     B       ;YES, COUNT DOWN FOR DEBOUNCE
        JNZ     FINDO1  ;KEEP COUNTING UNTIL DONE
        RET             ;RETURN WHEN COUNT = 0
$EJECT
;************************
;*                      *
;*   SUBROUTINE FIND1   *
;*                      *
;************************
;
;       DEBOUNCES THE PLUS LEVEL ON INPUT
;
;       INPUT   NONE
;
;       OUTPUT  NONE
;
;       DESTROYS (A),(B)
;
FIND1:  MVI     B,DEBOUN        ;SET (B) TO DEBOUNCE COUNT
FIND11: IN      0E9H    ;READ VALUE FROM A/D
        CPI     INPLUS  ;DOES IT EQUAL OR EXCEED LIMIT
        JC      FIND1   ;NO, RESET AND CHECK AGAIN
        DCR     B       ;YES, COUNT DOWN FOR DEBOUNCE
        JNZ     FIND11  ;KEEP COUNTING UNTIL DONE
        RET             ;RETURN AFTER COUNT = 0
$EJECT
;************************
;*                      *
;*   SUBROUTINE GREAT   *
;*                      *
;************************
;
;       FIND THE VECTOR FURTHEST FROM THE MEAN OF A GIVEN CASE IN THE
;       TABLE, STORE AT VINPUT
;
;       INPUT   (CASENO)        CASE #
;
;       OUTPUT  (DE)    POINTS TO TABLE ENTRY FURTHEST FROM MEAN
;
;       DESTROYS ALL REGISTERS
;
GREAT:  CALL    SEARCH  ;FIND VECTORS IN TABLE
        LDAX    D       ;FIND OUT IF CASE # IS
        ANA     A       ;NOT IN TABLE
        JNZ     GREATO  ;ENTRY NOT ZERO, THUS IN TABLE, GO AVERAGE
        MVI     A,3     ;NOT IN TABLE, DISPLAY ERROR 3
        CALL    ERROR   ;ON CONSOLE, THEN
        POP     H       ;REMOVE 'CALL GREAT' ADDRESS FROM STACK
        POP     PSW     ;AND THE PSW FLAGS
        RET             ;AND RETURN TO COMMAND LOOP
;
;       CALCULATE AVERAGE OF TABLE VECTORS
;
GREATO: PUSH    D       ;REMEMBER PLACE IN TABLE
        CALL    COUNT   ;FIND OUT HOW MANY ENTRIES IN TABLE
        MVI     C,VECLEN        ;SET UP PARAMETERS
        LXI     H,VINPUT        ;FOR AVERAGING
        XCHG            ;AND STORING IN VINPUT
        PUSH    D       ;REMEMBER THE ADDRESS
```

```
        PUSH    D       ;TWICE ON THE STACK
        INX     H       ;POINT TO THE FIRST COMPONENT
        CALL    AVERAG  ;NOW CALCULATE THE AVERAGE
        POP     D       ;GET THE VINPUT ADDRESS BACK
        PUSH    D       ;BUT REMEMBER IT AGAIN
        CALL    METRIC  ;GET THE METRIC OF THE AVERAGE VECTOR
        POP     B       ;SET THE POINTERS TO STORE THE
        POP     H       ;NORMALIZED VALUES AT VINPUT
        CALL    NORMAL  ;AND GO NORMALIZE IT
;
;       NOW FIND THE VECTOR WITH THE GREATEST DISTANCE FROM THIS
;       AVERAGE.  FIRST SET UP THE STACK AS FOLLOWS:
;
;                       ...
;                       POINTER TO FURTHEST VECTOR
;          (BOTTOM)     METRIC TO FURTHEST VECTOR
;
        POP     H       ;GET ADDRESS OF FIRST ENTRY
        PUSH    H       ;AND SET UP THE
        LXI     B,0     ;STACK AS
        PUSH    B       ;DESCRIBED ABOVE
;
;       NOW CALCULATE METRIC OF DIFFERENCE
;
GREAT1: LXI     D,VINPUT        ;USE THE INPUT VECTOR STORAGE
        PUSH    H       ;REMEMBER THE TABLE POINTER
        CALL    MINUS   ;GET THE METRIC OF THE DIFFERENCE
        POP     H       ;GET THE TABLE POINTER BACK
;
;       CHECK THE NEW METRIC AGAINST THE OLD AND SWAP IF NECESSARY
;
        POP     B       ;CHECK NEW METRIC AGAINST OLD
        PUSH    B       ;SAVE METRIC ON STACK
        MOV     A,B     ;SEE IF NEW IS GREATER
        CMP     D       ;THAN OLD, MSB'S FIRST
        JC      GREAT2  ;YES, REPLACE WITH NEW VALUE
        JNZ     GREAT3  ;NO, IF NOT EQUAL SKIP TO NEXT VECTOR
        MOV     A,C     ;EQUAL, CHECK LSB
        CMP     E       ;IS NEW > OLD?
        JNC     GREAT3  ;NO, GO TO NEXT COMPONENT
GREAT2: POP     B       ;YES, CLEAR THE TABLE AND
        POP     B       ;PUT THE NEW ADDRESS
        PUSH    H       ;AND THE NEW
        PUSH    D       ;METRIC ON THE STACK
GREAT3: LXI     D,VECLEN        ;POINT TO NEW VECTOR
        DAD     D       ;THEN CHECK IF DONE LOOKING
        MOV     A,M     ;IF ENTRY IS
        ANA     A       ;ZERO, THEN END
        JZ      GREAT4  ;OF TABLE
        LDA     CASENO  ;CASE # BEING
        CMP     M       ;CHECKED
        JZ      GREAT1  ;SAME, KEEP CHECKING
;
;       CHECKING DONE, RESTORE STACK AND RETURN
;
GREAT4: POP     B       ;RESTORE STACK
        POP     D       ;AND PUT ADDRESS OF FURTHEST VECTOR
        RET             ;IN (DE) AND RETURN
$EJECT
;************************
;*                      *
```

```
;*   SUBROUTINE INPUT   *
;*                      *
;************************
;
;       THIS ROUTINE PERFORMS THE FOLLOWING FUNCTIONS:
;
;               (1)     TIME THE WAVEFORM PATTERN
;               (2)     READ (NVECIN) INPUT VECTORS INTO THE VINPUT TABLE
;               (3)     AVERAGE THE VECTORS, LEAVE AVERAGE VALUES IN FIRST
;                       4 POSITIONS OF THE VINPUT TABLE
;               (4)     IF THE NULL VECTOR WAS ZERO, RETURN
;               (5)     SUBTRACT OUT THE NULL VECTOR
;               (6)     DETERMINE IF CRITERION EXCEEDED.  IF NOT, GO
;                       BACK TO THE FIRST RETURN
;               (7)     NORMALIZE THE VECTOR
;               (8)     RETURN TO THE SECOND RETURN
;
;
;       CALLING PROCEDURE FOR NORMAL INPUT:
;
;       LABEL:          CALL INPUT
;       LABEL+3:        (ERROR RETURN IF CRITERION NOT MET)
;       LABEL+8:        (NORMAL RETURN)
;
;
;       CALLING PROCEDURE FOR NULL VECTOR READ
;
;       LABEL:          CALL INPUT
;       LABEL+3:        (NORMAL RETURN)
;
;       INPUT   NONE
;
;       OUTPUT  (AVERAGED) INPUT VECTOR AT VINPUT
;
;       DESTROYS ALL REGISTERS
;
INPUT:
;
;       FIRST TIME THE INPUT WAVEFORM
;
        MVI     A,5     ;START THE
        OUT     0EAH    ;A/D CONVERTER
        CALL    FINDO   ;DEBOUNCE THE GROUND STATE
        CALL    FIND1   ;LOOK FOR THE SIGNAL
;
;       NOW MEASURE THE SIGNAL LENGTH, USING A STANDARD
;       LOOP LENGTH OF 52 MACHINE CYCLES
;
        LXI     H,0     ;INITIALIZE THE COUNT
        MVI     C,INZERO        ;SET UP THE ZERO LIMIT
TIMERO: MVI     B,DEBOUN        ;SET UP THE DEBOUNCE COUNT
        MOV     B,B     ;DUMMY TO PAD THE LOOP LENGTH
        NOP             ;DUMMY
TIMER1: INX     H       ;BUMP LENGTH COUNT
        IN      0E9H    ;READ VALUE
        CMP     C       ;IS IT > ZERO LIMIT
        JNC     TIMERO  ;YES, RESET COUNTER
        DCR     B       ;NO, COUNT FOR DEBOUNCE
        JNZ     TIMER1  ;NOT LONG ENOUGH, KEEP COUNTING
;
;       NOW SET UP HALF AND FULL LENGTHS BY DIVIDING BY 16
```

```
;       AND B AND STORING IN (E) AND (D)
;
        DAD     H       ;MULTIPLY BY 16
        DAD     H
        DAD     H
        DAD     H
        MOV     A,H     ;THEN SHIFT RIGHT 8 PLACES AND STORE
        DAD     H       ;MULTIPLY BY 2
        DCR     H       ;LOWER BY 1 LOOP LENGTH
        MOV     D,H     ;AND REMEMBER AGAIN
        SUI     DEBOUN  ;SUBTRACT OUT THE DEBOUNCE LOOP COUNT FROM
        MOV     E,A     ;THE HALF CYCLE LENGTH
;
;       NOW SET UP TO INPUT THE VECTORS
;
        LXI     H,VINPUT        ;GET ADDRESS OF INPUT TABLE
        MVI     B,NVECIN        ;SET UP COUNT FOR # INPUT VECTORS
INPUT1: PUSH    B       ;AND SAVE ON STACK
        CALL    SAMPLE  ;GET ONE VECTOR
        POP     B       ;RECALL COUNT
        DCR     B       ;COUNT DOWN ONE
        JNZ     INPUT1  ;NOT DONE, GO BACK
;
;       NOW AVERAGE THE VECTORS
;
        MVI     C,4     ;VECTORS ARE 4 BYTES LONG
        MVI     B,NVECIN        ;AVERAGE (NVECIN) VECTORS
        LXI     D,VINPUT        ;POINT TO THE INPUT TABLE
        MOV     H,D     ;SET UP THE STORAGE POINTER
        MOV     L,E     ;IN (HL)
        CALL    AVERAGE ;GET THE AVERAGE
;
;       SEE IF THE NULL VECTOR IS ZERO.  IF SO, THIS WAS A CALL TO
;       READ THE NULL VECTOR AND NORMALIZING IS NOT CALLED FOR
;
        LXI     D,VINPUT        ;GET POINTER TO AVERAGE VALUES
        LXI     H,VNULL ;GET NULL VECTOR POINTER
        MOV     A,M     ;GET FIRST NULL ELEMENT
        ANA     A       ;SEE IF ZERO
        RZ              ;IF YES, RETURN
;
;       NOT A NULL VECTOR CALL, SO SUBTRACT OUT THE OLD NULL VECTOR
;
        MVI     B,4     ;SET UP COUNTER
OFFSTO: LDAX    D       ;GET VECTOR COMPONENT
        SUB     M       ;SUBTRACT NULL COMPONENT
        JNC     OFFST1  ;IF < 0
        XRA     A       ;SET TO 0
OFFST1: STAX    D       ;STORE BACK
        INX     H       ;BUMP BOTH
        INX     D       ;STORAGE POINTERS
        DCR     B       ;DROP COUNTER
        JNZ     OFFSTO  ;KEEP GOING IF NOT DONE
;
;       SEE IF CRITERION EXCEEDED
;
        LXI     H,VINPUT        ;LOOK AT INPUT VECTOR
        LDA     VCRIT   ;USE CRITERION VALUE
        MVI     D,4     ;LOOK AT ALL 4 COMPONENTS
CRITRN: CMP     M       ;COMPARE COMPONENT AGAINST CRITERION
        JC      PASSED  ;EXCEEDS CRITERION, GO AHEAD
```

```
        INX     H       ;BUMP POINTER
        DCR     D       ;DROP COUNT
        JNZ     CRITRN  ;NOT DONE, CHECK ANOTHER
        RET             ;FAILED TO MEET CRITERION, ERROR RETURN
;
;       CALCULATE METRIC OF VECTOR
;
PASSED: LXI     D,VINPUT        ;GET POINTER
        CALL    METRIC  ;GO CALCULATE METRIC
;
;       NORMALIZE THE COMPONENTS
;
        LXI     H,VINPUT        ;GET POINTER TO SHIFTED INPUT VECTOR
        LXI     B,VNORM ;GET POINTER FOR NORMALIZED COMPONENT STORAGE
        CALL    NORMAL  ;GO NORMALIZE
;
;       NOW RETURN TO SECOND RETURN
;
        POP     H       ;GET VALUE FROM STACK
        LXI     D,5     ;BUMP OVER INITIAL RETURN
        DAD     D       ;TO RETURN PAST IT
        PCHL            ;NOW RETURN
$EJECT
;*************************
;*                       *
;*   SUBROUTINE METRIC   *
;*                       *
;*************************
;
;       CALCULATES THE SUM OF SQUARES OF FOUR COMPONENTS OF A VECTOR
;
;       INPUT   (DE)    POINTER TO THE FIRST COMPONENT
;
;       OUTPUT  (DE)    DOUBLE LENGTH RESULT, 1/4 * SUM ( V(I)**2 )
;
;       DESTROYS ALL REGISTERS
;
;       TOTAL LOOP LENGTH - 738 MACHINE CYCLES
;
METRIC: LDAX    D       ;GET MULTIPLIER
        OUT     0E8H    ;OUTPUT TO MULTIPLIER
        MVI     A,1     ;LOAD WITH
        OUT     0EAH    ;STROBE 1
        INR     A       ;SET UP FOR
        OUT     0EAH    ;STROBE 2
        LDAX    D       ;GET MULTIPLIER AGAIN
        OUT     0E8H    ;AND OUTPUT TO MULTIPLICAND
        XRA     A       ;RESET (A) AND CARRY
        OUT     0EAH    ;START MULTIPLY
        MVI     A,4     ;SET UP TO READ
        OUT     0EAH    ;MSB OF RESULT
        IN      0E9H    ;READ FROM PORT
        MOV     H,A     ;SET UP ANSWER IN (HL)
        MVI     A,3     ;SET UP TO READ
        OUT     0EAH    ;LSB OF RESULT
        IN      0E9H    ;READ IT AND
        MOV     L,A     ;PUT IT INTO THE ANSWER
        INX     D       ;GO TO THE NEXT COMPONENT
        LDAX    D       ;REPEAT THE ABOVE CODE FOR THIS COMPONENT
        OUT     0E8H    ;LOAD MULTIPLIER
        MVI     A,1
```

```
OUT     0EAH
INR     A
OUT     0EAH
LDAX    D
OUT     0E8H
XRA     A       ;START MULTIPLY
OUT     0EAH
MVI     A,4
OUT     0EAH
IN      0E9H
MOV     B,A     ;NOW PUT RESULT IN (BC)
MVI     A,3
OUT     0EAH
IN      0E9H
MOV     C,A     ;SECOND PART OF ANSWER SET
DAD     B       ;ADD THE TWO PARTS
MOV     A,H     ;AND DIVIDE THE RESULT BY 2
RAR             ;BY ROTATING
MOV     H,A     ;MSB FIRST
MOV     A,L     ;AND THEN
RAR             ;SHUTTLING THE CARRY
MOV     L,A     ;BIT INTO THE LSB
PUSH    H       ;REMEMBER THIS HALF ON THE STACK
INX     D       ;GO TO THE NEXT COMPONENT
LDAX    D       ;NOW DO THE THIRD COMPONENT LIKE THE FIRST
OUT     0E8H    ;LOAD THE MULTIPLIER
MVI     A,1
OUT     0EAH
INR     A
OUT     0EAH
LDAX    D
OUT     0E8H
XRA     A
OUT     0EAH
MVI     A,4
OUT     0EAH
IN      0E9H    ;GET RESULT
MOV     H,A     ;AND STORE IN (HL) AS BEFORE
MVI     A,3
OUT     0EAH
IN      0E9H
MOV     L,A     ;THIRD PART OF RESULT IN (HL)
INX     D       ;GO TO THE NEXT COMPONENT
LDAX    D       ;ONCE MORE, THIS TIME LIKE THE SECOND
OUT     0E8H    ;COMPONENT WAS HANDLED
MVI     A,1
OUT     0EAH
INR     A
OUT     0EAH
LDAX    D
OUT     0E8H
XRA     A
OUT     0EAH
MVI     A,4
OUT     0EAH
IN      0E9H
MOV     B,A     ;PUT RESULT IN (BC)
MVI     A,3
OUT     0EAH
IN      0E9H
MOV     C,A     ;FOURTH PART OF ANSWER IN (BC)
```

```
        DAD     B       ;ADD THIRD AND FOURTH PARTS OF ANSWER
        MOV     A,H     ;AND DIVIDE BY 2
        RAR             ;BY ROTATING
        MOV     H,A     ;(H) AND (L)
        MOV     A,L     ;AS WAS DONE
        RAR             ;BEFORE ONE BIT TO THE
        MOV     L,A     ;RIGHT
        POP     B       ;GET THE FIRST HALF OF THE ANSWER BACK
        XRA     A       ;CLEAR CARRY
        DAD     B       ;ADD THE TWO HALVES
        MOV     A,H     ;AND DIVIDE BY TWO
        RAR             ;ONE MORE
        MOV     D,A     ;TIME BY
        MOV     A,L     ;ROTATING
        RAR             ;(H) AND (L) AGAIN
        MOV     E,A     ;ONE BIT TO THE RIGHT
        RET             ;DONE, RETURN
$EJECT
;************************
;*                      *
;*   SUBROUTINE MINUS   *
;*                      *
;************************
;
;       FORMS THE DIFFERENCE BETWEEN INPUT VECTOR AND TABLE VECTOR
;       AND CALCULATES THE METRIC
;
;       INPUT   (HL)    TABLE POINTER
;               (DE)    POINTER TO THE INPUT VECTOR
;
;       OUTPUT  (DE)    METRIC OF DIFFERENCE VECTOR
;               (HL)    POINTS TO NEXT VECTOR ELEMENT
;
;       DESTROYS ALL REGISTERS
;
MINUS:  MVI     A,4     ;PREPARE TO PROCESS FOUR COMPONENTS
        INX     H       ;POINT TO THE FIRST VECTOR COMPONENT
;
;       FORM THE DIFFERENCE AND TAKE ABSOLUTE VALUE
;
        LXI     B,VDIFF ;STORE AT VDIFF
MINUS0: PUSH    PSW     ;SAVE THE COUNTER
        XRA     A       ;CLEAR THE CARRY BIT
        LDAX    D       ;GET VALUE
        SUB     M       ;AND SUBTRACT THE TABLE COMPONENT
        JNC     MINUS1  ;IF VALUE < 0
        CMA             ;TAKE THE ABSOLUTE VALUE
        INR     A       ;2'S COMPLEMENT
MINUS1: STAX    B       ;STORE THE DIFFERENCE COMPONENT
        INX     H       ;BUMP ALL
        INX     B       ;THREE
        INX     D       ;POINTERS
        POP     PSW     ;RECALL THE COUNTER,
        DCR     A       ;DROP IT
        JNZ     MINUS0  ;NOT DONE, KEEP COUNTING
;
;       NOW CALCULATE THE METRIC
;
        PUSH    H       ;REMEMBER THE POINTER TO THE CASE #
        LXI     D,VDIFF ;GET THE POINTER AND
        CALL    METRIC  ;TAKE THE METRIC
```

```
        POP     H       ;RETRIEVE THE CASE # POINTER
        RET             ;AND RETURN
$EJECT
;************************
;*                      *
;*   SUBROUTINE MOVE    *
;*                      *
;************************
;
;       MOVE A NORMALIZED VECTOR ONTO THE TABLE WITH ITS CASE #
;
;       INPUT   (DE)    POINTER TO SLOT IN TABLE
;               (CASENO)        CASE #
;
;       OUTPUT  NONE
;
;       DESTROYS ALL REGISTERS
;
MOVE:   LXI     H,VNORM ;GET POINTER TO NORMALIZED VECTOR
        LDA     CASENO  ;GET THE CASE NUMBER
        STAX    D       ;AND PUT IT INTO THE TABLE
        INX     D       ;BUMP TO THE FIRST VECTOR COMPONENT POSITION
        MVI     B,4     ;ALLOW FOR 4 VECTOR COMPONENTS
        CALL    SHIFT   ;MOVE THE VECTOR
        RET             ;RETURN
$EJECT
;**************************
;*                        *
;*   SUBROUTINE NORMAL    *
;*                        *
;**************************
;
;       NORMALIZE 4 COMPONENTS OF A VECTOR, USING A SUPPLIED METRIC
;       AND A LOOKUP TABLE FOR THE NORMALIZATION FACTOR.  CAN WORK IN
;       PLACE
;
;       INPUT   (BC)    STORAGE POINTER FOR NORMALIZED COMPONENTS
;               (DE)    METRIC OF INPUT VECTOR
;               (HL)    POINTER TO INPUT COMPONENTS
;
;       OUTPUT  NONE
;
;       DESTROYS ALL REGISTERS
;
NORMAL: PUSH    B       ;SAVE THE STORAGE POINER
;
;       SHIFT THE METRIC TO NORMALIZED FORM [IN WHICH THE LEFT 2 BITS
;       ARE (01), (10), OR (11)] AND COUNT THE SHIFTS REQUIRED
;
        XCHG            ;PUT THE METRIC IN (HL) AND SAVE THE POINTER
        MVI     B,1     ;START THE COUNT AT 1
NORM0:  MVI     A,0C0H  ;SET (A) TO (11000000)
        ANA     H       ;CHECK LEFT 2 BITS OF H
        JNZ     NORM1   ;IF NOT ZERO, METRIC IS IN STANDARD FORM
        DAD     H       ;IF ZERO, SHIFT (HL) OVER
        DAD     H       ;TWO SPACES
        INR     B       ;KEEP COUNT OF THE SHIFT
        JMP     NORM0   ;AND CHECK AGAIN
;
;       NOW GET THE LEFT 6 BITS OF (H) AND DETERMINE THE APPROXIMATE
;       MAGNITUDE TO INDEX THE LOOKUP TABLE
```

```
;
NORM1:  MVI     A,0FCH  ;SET (A) TO (11111100)
        ANA     H       ;GET LEFT 6 BITS OF (H)
        RRC             ;SHIFT RIGHT TWO PLACES
        RRC             ;TO REDUCE TO RANGE 16-63
        LXI     H,NTABLE-16     ;GET INITIAL TABLE POINTER - 16
        ADD     L       ;ADD IN THE DISPLACEMENT
        MOV     L,A     ;TO (HL) POINTER
        JNC     NORM2   ;BUMP (H) IF NECESSARY
        INR     H       ;BECAUSE OF OVERFLOW FROM (L)
;
;       NOW GET THE NORMALIZING FACTOR AND MULTIPLY ALL THE COMPONENTS
;
NORM2:  MOV     L,M     ;GET FACTOR
        MOV     H,B     ;REMEMBER THE SHIFT COUNT
        MVI     C,4     ;SET THE COUNTER FOR 4 COMPONENTS
NORM3:  LDAX    D       ;GET COMPONENT
        OUT     0E8H    ;SEND TO MULTIPLIER
        MVI     A,1     ;STROBE 1 TO LOAD
        OUT     0EAH    ;INTO MULTIPLIER
        INR     A       ;SET STROBE 2
        OUT     0EAH    ;TO PREPARE MULTIPLICAND
        MOV     A,L     ;GET FACTOR
        OUT     0E8H    ;AND SEND IT
        XRA     A       ;CLEAR A TO PREPARE MULTIPLY
        OUT     0EAH    ;START THE MULTIPLY
        PUSH    H       ;SAVE THE COUNTER AND FACTOR
        MVI     A,4     ;SET UP TO READ THE RESULT
        OUT     0EAH    ;MSB OF ANSWER FIRST
        IN      0E9H    ;READ THE MSB
        MOV     H,A     ;SET UP THE ANSWER IN (HL)
        MVI     A,3     ;PREPARE FOR THE
        OUT     0EAH    ;LSB OF THE ANSWER
        IN      0E9H    ;READ IT
        MOV     L,A     ;AND FINISH (HL)
;
;       NOW SHIFT TO COMPENSATE FOR PRIOR SHIFTING TO NORMALIZE METRIC
;
NORM4:  DCR     B       ;DROP COUNT
        JZ      NORM5   ;IF ZERO, DONE SHIFTING
        DAD     H       ;SHIFT ANSWER
        JMP     NORM4   ;AND KEEP COUNTING
;
;       STORE THE COMPONENT BACK INTO MEMORY
;
NORM5:  MOV     A,H     ;SAVE HIGH BYTE, WHICH IS DESIRED COMPONENT
        POP     H       ;GET THE COUNTER BACK
        XTHL            ;GET THE POINTER FROM THE STACK
        MOV     M,A     ;STORE THE NORMALIZED COMPONENT
        DCR     C       ;DROP THE COUNT
        JZ      NORM6   ;IF DONE, GO POP THE STACK
        INX     H       ;INCREMENT THE STORAGE POINTER
        XTHL            ;AND PUT IT BACK ONTO THE STACK
        INX     D       ;INCREMENT THE INPUT POINTER
        MOV     B,H     ;RESTORE THE SHIFT COUNT
        JMP     NORM3   ;GO GET THE NEXT COMPONENT
;
;       DONE, RESTORE THE STACK AND RETURN
;
NORM6:  POP     B       ;GET THE VALUE FROM THE STACK
        RET             ;AND GO BACK TO THE CALL
```

```
;
;       THIS IS THE NORMALIZATION FACTOR TABLE.  THE FACTOR IS SET SO
;       AS TO MAKE EACH (NORMALIZED) METRIC AS CLOSE TO (0111 1111
;       0000 0000) AS POSSIBLE.  THIS MEANS THAT FOUR EQUAL
;       COMPONENTS WOULD EACH BE SET TO (0111 1111) = 127, AND A
;       SINGLE NON-ZERO COMPONENT VECTOR WOULD BE SET TO (1111 1110),
;       THUS AVOIDING ANY OVERFLOW IN THE METRIC OR THE COMPONENTS.
;
NTABLE:
;       16 TO 23
        DB      246,239,233,227,221,216,211,207
;       24 TO 31
        DB      203,199,195,192,188,185,182,179
;       32 TO 39
        DB      176,174,171,169,167,164,162,160
;       40 TO 47
        DB      158,156,154,153,151,149,148,146
;       48 TO 55
        DB      145,143,142,140,139,138,136,135
;       56 TO 63
        DB      134,133,132,131,130,129,128,127
$EJECT
;*************************
;*                       *
;*   SUBROUTINE PGM08    *
;*                       *
;*************************
;
;       DRIVER PROGRAM FOR PROMPT 80 PROM PROGRAMMER
;
;       (SEE MONITOR LISTING PAGE 54)
;
PGM08:  INR     C
        CALL    EXPR
        DI
        MVI     A,80H
        OUT     0EBH
        MVI     A,70H
        OUT     0E6H
        MVI     A,0E0H
        OUT     0EAH
        MVI     A,7DH
        OUT     0E6H
        POP     B
        POP     D
        POP     H
        CALL    ADT08
        PUSH    H
        PUSH    B
        PUSH    H
        MVI     H,64H
        XTHL
PG80:   POP     PSW
        POP     B
        POP     H
        PUSH    H
        PUSH    B
        PUSH    PSW
PG81:   MOV     A,C
        OUT     0EBH
        MOV     A,B
```

```
        CMA
        ANI     3
        ORI     0E0H
        OUT     0EAH
        MVI     A,0CH
        OUT     0EBH
        MOV     A,M
        OUT     0E9H
        PUSH    PSW
        POP     PSW
        MVI     A,0EH
        OUT     0EBH
        CALL    DELAY
        MVI     A,0FH
        OUT     0EBH
        MVI     A,0DH
        OUT     0EBH
        INX     B
        CALL    HILO
        JNC     PGB1
        XTHL
        DCR     H
        XTHL
        JNZ     PGB0
        POP     PSW
        POP     B
        POP     H
        CALL    CPB0
        RET
;
;       COMPARE PROM WITH MEMORY (SEE MONITOR PAGE 56)
;
CPB0:   CALL    RDPGB
        PUSH    B
        MOV     B,A
        XRA     M
        MOV     A,B
        POP     B
        JZ      CPB1
        PUSH    H
        PUSH    D
        PUSH    B
        PUSH    PSW
        PUSH    H
        LHLD    RAM
        MVI     L,0E9H
        MOV     A,M
        DCR     A
        POP     H
        JZ      ERRORM
        XCHG
        LHLD    RAM
        MVI     L,0F3H
        XCHG
        CALL    DREGC
        MOV     A,M
        CALL    DBYT
        XCHG
        CALL    KEYT
        XCHG
        CPI     0EH
```

```
        JZ      RSTRT
        CPI     7FH
        JZ      RSTRT
        CPI     OFFH
        JNZ     ERRORM
        POP     PSW
        MVI     E,OEFH
        CALL    DBYT
        CALL    KEYT
        CPI     OEH
        JZ      RSTRT
        CPI     7FH
        JZ      RSTRT
        CPI     OFFH
        JNZ     ERRORM
        POP     B
        POP     D
        POP     H
CP81:   INX     B
        CALL    HILO
        JNC     CP80
        RET
;
;       VERIFY PROM ADDRESS PARAMETERS (SEE MONITOR PAGE 62)
;
ADTOB:  MOV     A,B
        ANI     OFCH
        JNZ     ERRORM
        MOV     A,C
        ANI     OFH
        JNZ     ERRORM
        MOV     A,E
        SUB     L
        ANI     OFH
        CPI     OFH
        JNZ     ERRORM
ADT1:   MOV     A,E
        SUB     L
        MOV     A,D
        SBB     H
        SBI     4
        JP      ERRORM
        PUSH    H
        MOV     A,H
        CMA
        MOV     H,A
        MOV     A,L
        CMA
        MOV     L,A
        INX     H
        DAD     D
        DAD     B
        MOV     A,H
        ANI     OFCH
        JNZ     ERRORM
        LHLD    RAM
        MVI     L,OEBH
        XRA     A
        MOV     M,A
        POP     H
        RET
```

```
$EJECT
;*************************
;*                       *
;*   SUBROUTINE SAMPLE   *
;*                       *
;*************************
;
;       INPUTS FOUR VECTOR ELEMENTS AND STORES THEM
;
;       INPUT   (D)     CONTAINS FULL SIGNAL COUNT
;               (E)     HALF SIGNAL COUNT (DIMINISHED)
;               (HL)    POINTER FOR STORAGE
;
;       OUTPUT  (HL) POINTS TO NEXT EMPTY SPACE
;
;       DESTROYS (A),(B),(C)
;
;       INPUT LOOP TIMING SET TO 52 MACHINE CYCLES, PADDDED WHERE NEEDED
;
SAMPLE: CALL    FIND0   ;FIRST DEBOUNCE THE GROUND STATE
        CALL    FIND1   ;THEN DEBOUNCE THE SIGNAL
;
;       NOW WE KNOW THAT THIS IS THE LEADING EDGE OF THE SIGNAL
;       GET FOUR VALUES FROM THE WAVEFORM AND STORE
;
        MOV     B,E     ;SET UP THE COUNT FOR THE HALF-LENGTH
        MVI     C,4     ;PREPARE COUNTER FOR 4 READS
SAMP1:  IN      0E9H    ;READ THE VALUE
        DCR     B       ;COUNT DOWN
        JZ      SAMP2   ;CHECK IF DONE
        INR     A       ;DUMMY TO PAD LOOP LENGTH
        INR     A       ;DUMMY
        NOP             ;DUMMY
        JMP     SAMP1   ;GO READ AGAIN
SAMP2:  MOV     M,A     ;STORE THE VALUE
        INX     H       ;BUMP THE MEMORY POINTER
        MOV     B,D     ;GET THE FULL-LENGTH COUNT
        DCR     C       ;DROP THE COUNTER
        JNZ     SAMP1   ;NOT DONE, KEEP INPUTTING
        RET             ;DONE WITH INPUTTING
$EJECT
;*************************
;*                       *
;*   SUBROUTINE SEARCH   *
;*                       *
;*************************
;
;       SEARCH THE VECTOR TABLE FOR A GIVEN CASE #
;
;       INPUT   (CASENO) HAS CASE #
;
;       OUTPUT  (DE)    POINTER TO FIRST ELEMENT OF THAT CASE VECTOR,
;                       OR END OF TABLE IF NOT FOUND
;
;       DESTROYS ALL REGISTERS
;
SEARCH: LDA     CASENO  ;GET THE CASE #
        MOV     B,A     ;SAVE IT
        LXI     D,VECLEN        ;SET THE LENGTH ADDER
        LXI     H,VTABLE        ;POINT TO THE FIRST VECTOR
SEAR0:  MOV     A,M     ;SEE IF END OF TABLE
```

```
        ORA     A       ;SET THE FLAGS
        JZ      SEAR1   ;AND GO BACK IF END OF TABLE
        CMP     B       ; SEE IF CORRECT CASE NUMBER
        JZ      SEAR1   ;YES, GO BACK
        DAD     D       ;BUMP TO THE NEXT VECTOR
        JMP     SEARO   ;AND KEEP LOOKING
SEAR1:  XCHG            ;SET UP (DE) TO POINT TO CASE
        RET             ;AND RETURN TO THE CALL
$EJECT
;*************************
;*                       *
;*   SUBROUTINE SHIFT    *
;*                       *
;*************************
;
;       MOVE A BLOCK OF MEMORY
;
;       INPUT   (B)     # BYTES TO MOVE
;               (DE)    "TO" POINTER
;               (HL)    "FROM" POINTER
;
;       OUTPUT  NONE
;
;       DESTROYS (A),(B),(DE),(HL)
;
SHIFT:  MOV     A,M     ;GET THE BYTE
        STAX    D       ;STORE IT
        INX     H       ;BUMP BOTH POINTERS
        INX     D       ;TO MEMORY
        DCR     B       ;AND DROP THE COUNT
        JNZ     SHIFT   ;NOT DONE, KEEP SHIFTING
        RET             ;DONE, GO BACK
$EJECT
;*************************
;*                       *
;*   SUBROUTINE SHUTTL   *
;*                       *
;*************************
;
;       MOVE MEMORY DOWN TO FILL A GAP IN A TABLE, OR UP TO CREATE
;       A GAP IN THE TABLE
;
;       INPUT   (DE)    POINTER TO LOWEST ELEMENT OF THE GAP
;               (L)     LENGTH OF GAP, > 0 TO FILL, < 0 TO CREATE
;
;       OUTPUT  (DE)    POINTS TO END OF TABLE FOR FILL, OR
;                       BEGINNING OF GAP FOR EXPAND
;
;       DESTROYS (A),(B),(C),(H)
;
SHUTTL: PUSH    D       ;SAVE THE POINTER
        MVI     H,0     ;SET (H) TO ZERO
        PUSH    H       ;AND SAVE (L)
        MOV     A,H     ;PREPARE THE L VALUE
        SUB     L       ;BY CHECKING FOR MINUS
        JM      SHUTLO  ;(L) > 0, FILL GAP
        MOV     L,H     ;(L) < 0, FIND END OF TABLE INSTEAD
SHUTLO: DAD     D       ;PREPARE UPPER ADDRESS
SHUTL1: MOV     A,M     ;GET THE ELEMENT
        STAX    D       ;STORE DOWN IN MEMORY
        ANA     A       ;SEE IF ELEMENT IS ZERO
```

```
        JZ      SHUTL3  ;IF YES, SEE IF DONE WITH SHUTTLING
        MVI     C,VECLEN        ;IF NOT, MOVE ONE VECTOR DOWN
SHUTL2: INX     D       ;BUMP POINTERS FIRST
        INX     H       ;TO BOTH SLOTS
        DCR     C       ;NOW COUNT THE LENGTH
        JZ      SHUTL1  ;WHEN DONE, SEE IF END OF TABLE
        MOV     A,M     ;SHUTTLE MEMORY
        STAX    D       ;DOWN
        JMP     SHUTL2  ;AND KEEP COUNTING
;
;       NOW SEE IF DOWN, IN WHICH CASE RETURN, OR UP, IN WHICH
;       CASE EXPAND THE GAP
;
SHUTL3: POP     H       ;RECALL THE ORIGINAL (L) VALUE
        POP     B       ;AND THE ORIGINAL POINTER
        MOV     A,H     ;THEN SET UP THE TEST AGAIN
        SUB     L       ;TAKE THE NEGATIVE AND
        RM              ;RETURN IF IT WAS ORIGINALLY PLUS
;
;       CALCULATE THE NEW END ADDRESS AND LENGTH TO MOVE
;       AND MOVE THE TABLE UP
;
        PUSH    B       ;SAVE THE ORIGINAL POINTER
        MOV     L,A     ;POSITIVE VALUE INTO (L)
        DAD     D       ;AND ADD THE END OF TABLE POINTER
        XCHG            ;SAVE THE END
        PUSH    H       ;TABLE POINTER
        MOV     A,C     ;TAKE THE
        CMA             ;2'S COMPLEMENT
        MOV     C,A     ;OF THE INITIAL
        MOV     A,B     ;ADDRESS AND
        CMA             ;SUBTRACT
        MOV     B,A     ;IT FROM
        INX     B       ;THE FINAL
        DAD     B       ;ADDRESS OF THE TABLE
        INX     H       ;ALLOW FOR ONE EXTRA POSITION
        POP     B       ;NOW (HL) HAS LENGTH COUNT
SHUTL4: LDAX    B       ;(BC) HAS LOW POINTER,
        STAX    D       ;(DE) HAS HIGH POINTER
        DCX     B       ;BUMP BOTH POINTERS
        DCX     D       ;IN THE TABLE
        DCX     H       ;DROP THE COUNT
        MOV     A,H     ;THEN CHECK IF
        ORA     L       ;THE COUNT IS
        JNZ     SHUTL4  ;ZERO, IF NOT, KEEP SHIFTING
        POP     D       ;RESTORE THE POINTER
        RET             ;IF YES, RETURN
$EJECT
;*************************
;*                       *
;*   SUBROUTINE SMALL    *
;*                       *
;*************************
;
;       PLACE AN ENTRY INTO THE LIST OF THE K SMALLEST METRIC VALUES
;
;       INPUT   (DE)    VALUE OF METRIC
;               (B)     CASE # OF ENTRY
;
;       OUTPUT  NONE
;
```

```
;       DESTROYS (A),(C),(HL)
;
;       THIS ROUTINE TAKES (20 + 127 * KNN) MACHINE CYCLES
;
SMALL:  LDA     VKNN    ;GET THE NUMBER OF NEAREST NEIGHBORS
        MOV     C,A     ;SET UP THE COUNTER
        LXI     H,TSMALL        ;GET THE POINTER TO THE SMALL TABLE
;
;       THIS LOOP LOOKS AT THE (DOUBLE LENGTH) TABLE ENTRY AND COMPARES
;       IT TO THE NEW METRIC.  IF THE NEW ENTRY IS SMALLER, IT IS
;       SWAPPED INTO THE TABLE.
;
SMALL0: MOV     A,D     ;GET MSB OF ENTRY
        CMP     M       ;AND COMPARE IT TO THE TABLE ENTRY
        JZ      SMALL3  ;TABLE ENTRY EQUAL, CHECK LSB
        JNC     SMALL1  ;TABLE ENTRY SMALLER, NO SWAP
        MOV     D,M     ;NEW ENTRY SMALLER,
        MOV     M,A     ;SWAP ENTRY AND TABLE VALUE
        INX     H       ;BUMP THE MEMORY POINTER
        MOV     A,M     ;SWAP THE LSB
        MOV     M,E     ;OF ENTRY
        MOV     E,A     ;AND TABLE VALUE
        INX     H       ;BUMP THE MEMORY POINTER AGAIN
        MOV     A,M     ;AND SWAP THE CASE #
        MOV     M,B     ;OF THE NEW ENTRY AND THE
        MOV     B,A     ;TABLE ENTRY
        JMP     SMALL4  ;CHECK THE NEXT POSITION
SMALL1: INX     H       ;BUMP THE POINTER
        MOV     A,A     ;DUMMY TO PAD LOOP LENGTH
        MOV     A,M     ;DUMMY
SMALL2: INX     H       ;BUMP THE POINTER
        JP      $+3     ;DUMMY TO PAD THE LOOP
        JP      $+3     ;DUMMY
        JMP     SMALL4  ;END OF LOOP
SMALL3: INX     H       ;ENTRIES WERE EQUAL, CHECK LSB
        MOV     A,E     ;GET NEW VALUE
        CMP     M       ;AND COMPARE TO TABLE VALUE
        JNC     SMALL2  ;IF TABLE SMALLER OR EQUAL, NO SWAP
        MOV     E,M     ;NEW ENTRY SMALLER
        MOV     M,A     ;SO SWAP VALUES
        INX     H       ;BUMP THE POINTER
        MOV     A,M     ;AND SWAP THE
        MOV     M,B     ;CASE NUMBERS OF ENTRY
        MOV     B,A     ;AND TABLE
        MOV     A,M     ;DUMMY TO PAD LOOP
        XRA     A       ;DUMMY
SMALL4: INX     H       ;BUMP POINTER TO NEXT TABLE POSITION
;
;       THE FIRST COMPARISON AND SWAP IS COMPLETE.
;       THE ABOVE CODE IS REPLICATED FOR EACH OF THE
;       NEAREST NEIGHBORS
;
        DCR     C       ;BUMP THE COUNT OF NEIGHBORS
        JNZ     SMALL0  ;NOT DONE, LOOP BACK
        RET             ;DONE, RETURN
$EJECT
;*************************
;*                       *
;*   SUBROUTINE TRN08    *
;*                       *
;*************************
```

```
;
;       TRANSFER CONTENTS OF PROM TO MEMORY
;
;       (SEE MONITOR LISTING PAGE 57)
;
TRNOB:  INR     C
        CALL    EXPR
        POP     B
        POP     D
        POP     H
        CALL    ADT1
TNBO:   CALL    RDPOB
        MOV     M,A
        INX     B
        CALL    HILO
        JNC     TNBO
        RET
;
;       READ PROM CONTENTS (SEE MONITOR PAGE 69)
;
RDPOB:  DI
        MVI     A,70H
        OUT     0E6H
        MVI     A,82H
        OUT     0EBH
        MVI     A,7DH
        OUT     0E6H
        MOV     A,C
        OUT     0E8H
        MOV     A,B
        CMA
        ANI     3
        ORI     0E0H
        OUT     0EAH
        MVI     A,0AH
        OUT     0EBH
        IN      0E9H
        PUSH    PSW
        MVI     A,0BH
        OUT     0EBH
        POP     PSW
        EI
        RET
$EJECT
        END
```

B. Parametric Classifier

```
C       PARAMETRIC CLASSIFIER FOR EMG VECTORS
C       WRITTEN BY D.C. DENING  9/9/81
C
C       TRAINING DONE ON FIRST DATA FILE
C       EVALUATION DONE ON SECOND FILE
C
C
C       VARIABLES:
C       IDATA (999,9)    CONTAINS THE INPUT DATA POINTS, PLUS CASE LABEL
C                        FOR EACH VECTOR IN ELEMENT 9 OF EACH ROW
C       NVECS            NUMBER OF DATA VECTORS
C       NCASES           NUMBER OF CASES
C       NOFFSET          CASE NUMBER OF ZERO INPUT CASE
C       AVERAGE          MEAN VECTORS FOR EACH CASE
C       COVAR            COVARIANCE MATRICES FOR EACH CASE
C       A                COVARIANCE MATRIX FOR ENSEMBLE ABOUT ZERO INPUT POINT
C       GRANDM           GRAND MEAN VECTOR FOR ENSEMBLE
C       GRANDC           COVARIANCE MATRIX FOR ENSEMBLE ABOUT ORIGIN
C
C
C
      LOGICAL FLAGPRINT
      DIMENSION GRANDM(8),GRANDC(8,8),TITLE(10)
      DIMENSION AVERAGE(26,9),COVAR(26,8,8),A(8,8),S(8,8),X(4),INPT(4)
      DIMENSION IDATA(999,9),OFFSET(8),CONF(6,6),WD(6)
      DIMENSION GR(4,4),OP(4,4),FL(4,4),EX(4,4),PR(4,4),SU(4,4)
      DIMENSION GRM(4),OPM(4),FLM(4),EXM(4),PRM(4),SUM(4),OFSTM(4)
      DIMENSION GRINV(4,4),OPINV(4,4),FLINV(4,4),EXINV(4,4),PRINV(4,4),
     *          SUINV(4,4)
      COMMON IDATA,NVECS,NCASES,NOFFSET,NELEM,AVERAGE
      COMMON /TITLE/ TITLE
      COMMON /FLAGS/ FLAGPRINT
C
C       INPUT DATA AND CONTROL PARAMETERS
C
      CALL INPUT
C
C       INITIALIZE MEANS AND COVARIANCES MATRICES
C
      DO 300 I=1,26
              DO 200 J=1,8
                      AVERAGE(I,J)=0.0
                      DO 100 K=1,8
                              COVAR(I,J,K)=0.0
100                   CONTINUE
200           CONTINUE
              AVERAGE(I,9)=0.0
300   CONTINUE
C
C       ZERO OUT GRAND MEAN AND COVARIANCE
C
      DO 1100 I=1,8
              GRANDM(I)=0.0
              DO 1000 J=1,8
                      GRANDC(I,J)=0.0
1000          CONTINUE
1100  CONTINUE
C
C       CALCULATE MEANS
C
      DO 500 I=1,NVECS
```

```
                K=IDATA(I,9)
                DO 400 J=1,8
                        AVERAGE(K,J)=AVERAGE(K,J)+FLOAT(IDATA(I,J))
                        GRANDM(J)=GRANDM(J)+FLOAT(IDATA(I,J))
400             CONTINUE
                AVERAGE(K,9)=AVERAGE(K,9)+1.0
500     CONTINUE
        DO 700 I=1,NCASES
                IF (AVERAGE(I,9).NE.0.0) THEN
                        DO 600 J=1,8
                              AVERAGE(I,J)=AVERAGE(I,J)/AVERAGE(I,9)
600                     CONTINUE
                ENDIF
700     CONTINUE
C
C         CALCULATE COVARIANCE MATRIX FOR EACH CASE
C
        DO 900 I=1,NVECS
                N=IDATA(I,9)
                DO 800 J=1,8
                DO 800 K=1,8
                        COVAR(N,J,K)=COVAR(N,J,K)+(FLOAT(IDATA(I,J))
     1                  -AVERAGE(N,J))*(FLOAT(IDATA(I,K))-AVERAGE(N,K))
800             CONTINUE
900     CONTINUE
        DO 1200 I=1,8
                GRANDM(I)=GRANDM(I)/FLOAT(NVECS)
1200    CONTINUE
        DO 1400 I=1,NVECS
              DO 1300 J=1,8
                DO 1300 K=1,J
                        GRANDC(J,K)=GRANDC(J,K)+(FLOAT(IDATA(I,J))-
     1                  GRANDM(J))*(FLOAT(IDATA(I,K))-GRANDM(K))
                        GRANDC(K,J)=GRANDC(J,K)
1300            CONTINUE
1400    CONTINUE
C
C         NOW DIVIDE BY (# VECTORS-1) TO ESTIMATE COVARIANCE
C
        DO 1600 I=1,NCASES
                IF (AVERAGE(I,9).GT.1.0) THEN
                        DO 1500 J=1,8
                        DO 1500 K=1,8
                             COVAR(I,J,K)=COVAR(I,J,K)/(AVERAGE(I,9)-1.0)
1500                    CONTINUE
                ENDIF
1600    CONTINUE
        DO 1800 I=1,8
              OFFSET(I)=AVERAGE(NOFFSET,I)
                DO 1700 J=1,8
                        GRANDC(I,J)=GRANDC(I,J)/FLOAT(NVECS-1)
1700            CONTINUE
1800    CONTINUE
C
C         PRINT INPUT DATA, MEANS, COVARIANCE MATRICES,
C          GRAND MEAN, GRAND COVARIANCE, EIGENVALUES, AND EIGENVECTORS
C
        IF (.NOT.FLAGPRINT) GO TO 2175
        WRITE (1,9012) TITLE
        DO 2000 I=1,NVECS
                IF (I.EQ.1) THEN
```

```
                WRITE (1,9000) (K,K=1,8),IFIX(AVERAGE(IDATA(I,9),9)),
     1                (IDATA(I,J),J=1,8)
              ELSE IF (IDATA(I,9).NE.IDATA(I-1,9)) THEN
                WRITE (1,9001) IDATA(I,9),IFIX(AVERAGE(IDATA(I,9),9)),
     1                (IDATA(I,J),J=1,8)
              ELSE
                        WRITE (1,9002) (IDATA(I,J),J=1,8)
              ENDIF
2000  CONTINUE
      DO 2100 I=1,NCASES
              IF (((I+1)/2)*2.EQ.I+1) WRITE (1,9003)TITLE
              WRITE (1,9004) I,(AVERAGE(I,J),J=1,8)
              WRITE (1,9005) ((COVAR(I,J,K),K=1,8),J=1,8)
2100  CONTINUE
      WRITE (1,9003) TITLE
      WRITE (1,9006) NCASES,(GRANDM(I),I=1,8)
      WRITE (1,9007) (AVERAGE(NOFFSET,I),I=1,8)
      WRITE (1,9008) ((GRANDC(I,J),J=1,8),I=1,8)
2175  CONTINUE
9000  FORMAT (1X,'DATA VECTORS AS READ'//27X,'VARIABLE #'//12X,
     1 8(I1,5X)//1X,'CASE # 1:     # VECTORS =',I3//11X,8(I3,3X))
9001  FORMAT (/1X,'CASE # ',I2,':     # VECTORS = ',I3//11X,8(I3,3X))
9002  FORMAT (11X,8(I3,3X))
9003  FORMAT ('1',10A4)
9004  FORMAT (1X,'CASE # ',I2///6X,'AVERAGE VECTOR :   ',8(F6.2,4X)///
     1 6X,'COVARIANCE MATRIX'//)
9005  FORMAT (8(11X,8(F8.2,3X)//)///)
9006  FORMAT (1X,'GRAND MEAN FOR ',I2,' CASES :   ',8(F6.2,4X)///)
9007  FORMAT (1X,'OFFSET VECTOR :   ',8(F6.2,4X)///)
9008  FORMAT (1X,'GRAND COVARIANCE MATRIX'//8(11X,8(F8.2,3X)//)///)
9009  FORMAT (1X,'COVARIANCE MATRIX'//4(11X,4(F8.2,3X)//)///)
9010  FORMAT (1X,'INVERSE OF COVARIENCE'//4(14X,(4(E12.4,3X)/)////))
9011  FORMAT (1X,'DETERMINATE FOR CASE #',I3,'  =',E12.4/)
9012  FORMAT (1X,10A4//)
      DO 2200 I=1,4
      GRM(I)=AVERAGE(1,I)-AVERAGE(NOFFSET,I)
      IF(GRM(I).LT.0.) GRM(I)=0.
      OPM(I)=AVERAGE(2,I)-AVERAGE(NOFFSET,I)
      IF(OPM(I).LT.0.) OPM(I)=0.
      FLM(I)=AVERAGE(3,I)-AVERAGE(NOFFSET,I)
      IF(FLM(I).LT.0.) FLM(I)=0.
      EXM(I)=AVERAGE(4,I)-AVERAGE(NOFFSET,I)
      IF(EXM(I).LT.0.) EXM(I)=0.
      PRM(I)=AVERAGE(5,I)-AVERAGE(NOFFSET,I)
      IF(PRM(I).LT.0.) PRM(I)=0.
      SUM(I)=AVERAGE(6,I)-AVERAGE(NOFFSET,I)
      IF(SUM(I).LT.0.) SUM(I)=0.
      OFSTM(I)=AVERAGE(7,I)
        DO 2200 J=1,4
        GR(I,J)=COVAR(1,I,J)
        OP(I,J)=COVAR(2,I,J)
        FL(I,J)=COVAR(3,I,J)
        EX(I,J)=COVAR(4,I,J)
        PR(I,J)=COVAR(5,I,J)
        SU(I,J)=COVAR(6,I,J)
 2200 CONTINUE
      CALL INVMAT(GR,GRINV,4,DGR)
      CALL INVMAT(OP,OPINV,4,DOP)
      CALL INVMAT(FL,FLINV,4,DFL)
      CALL INVMAT(EX,EXINV,4,DEX)
      CALL INVMAT(PR,PRINV,4,DPR)
```

```
      CALL INVMAT(SU,SUINV,4,DSU)
      IF(.NOT.FLAGPRINT) GO TO 2250
      K=1
         WRITE (1,9004) K,(GRM(I),I=1,4)
         WRITE (1,9009) ((GR(I,J),J=1,4),I=1,4)
         WRITE (1,9011) K,DGR
         WRITE (1,9010) ((GRINV(I,J),J=1,4),I=1,4)
      K=K+1
         WRITE (1,9004) K,(OPM(I),I=1,4)
         WRITE (1,9009) ((OP(I,J),J=1,4),I=1,4)
         WRITE (1,9011) K,DOP
         WRITE (1,9010) ((OPINV(I,J),J=1,4),I=1,4)
      K=K+1
         WRITE (1,9004) K,(FLM(I),I=1,4)
         WRITE (1,9009) ((FL(I,J),J=1,4),I=1,4)
         WRITE (1,9011) K,DFL
         WRITE (1,9010) ((FLINV(I,J),J=1,4),I=1,4)
      K=K+1
         WRITE (1,9004) K,(EXM(I),I=1,4)
         WRITE (1,9009) ((EX(I,J),J=1,4),I=1,4)
         WRITE (1,9011) K,DEX
         WRITE (1,9010) ((EXINV(I,J),J=1,4),I=1,4)
      K=K+1
         WRITE (1,9004) K,(PRM(I),I=1,4)
         WRITE (1,9009) ((PR(I,J),J=1,4),I=1,4)
         WRITE (1,9011) K,DPR
         WRITE (1,9010) ((PRINV(I,J),J=1,4),I=1,4)
      K=K+1
         WRITE (1,9004) K,(SUM(I),I=1,4)
         WRITE (1,9009) ((SU(I,J),J=1,4),I=1,4)
         WRITE (1,9011) K,DSU
         WRITE (1,9010) ((SUINV(I,J),J=1,4),I=1,4)
 2250 CONTINUE
      CALL WCONST(GRM,GRINV,DGR,WGR)
      CALL WCONST(OPM,OPINV,DOP,WOP)
      CALL WCONST(FLM,FLINV,DFL,WFL)
      CALL WCONST(EXM,EXINV,DEX,WEX)
      CALL WCONST(PRM,PRINV,DPR,WPR)
      CALL WCONST(SUM,SUINV,DSU,WSU)
 2300 WRITE(6,9100)
      READ (6,9101,ERR=2300)(INPT(I),I=1,4)
      DO 2400 I=1,4
      X(I)=FLOAT(INPT(I))-OFSTM(I)
 2400 CONTINUE
      WRITE (6,9103)(X(I),I=1,4)
      IF(X(1).LT.0.)GO TO 2500
      CALL DISC(X,GRINV,GRM,WGR,GRDIS)
      CALL DISC(X,OPINV,OPM,WOP,OPDIS)
      CALL DISC(X,FLINV,FLM,WFL,FLDIS)
      CALL DISC(X,EXINV,EXM,WEX,EXDIS)
      CALL DISC(X,PRINV,PRM,WPR,PRDIS)
      CALL DISC(X,SUINV,SUM,WSU,SUDIS)
      WRITE (6,9102) GRDIS,OPDIS,FLDIS,EXDIS,PRDIS,SUDIS
      GO TO 2300
 9100 FORMAT(//1X,'INPUT DATA VECTOR TO CLASSIFY (4 COMPONENTS)..')
 9101 FORMAT(4I3)
 9102 FORMAT(/1X,'GRASP DISCRIMINANT   =  ',E12.4,
     *       /1X,'OPEN DISCRIMINANT    =  ',E12.4,
     *       /1X,'FLEX DISCRIMINANT    =  ',E12.4,
     *       /1X,'EXTEND DISCRIMINANT  =  ',E12.4,
     *       /1X,'PRONATE DISCRIMINANT =  ',E12.4,
```

```
     *       /1X,'SUPINATE DISCRIMINANT = ',E12.4)
9103  FORMAT(/1X,'INPUT MINUS OFFSET = ',4F8.2)
9104  FORMAT(//1X,'COMPUTED CLASSIFIER PARAMETERS FROM ',10A4,
     *       /1X,'NOW READY TO INPUT VECTORS TO CLASSIFY'/)
9105  FORMAT(///1X,'CONFUSION MATRIX FROM DATA SET ',10A4)
9106  FORMAT(///30X,'DATA CLASSIFIED AS',/28X,
     *       '  GRASP    OPEN    FLEX   EXTEND  PRONATE SUPINATE',/1X,
     *       'INPUT VECTOR CLASS: GRASP    ',6(F6.0,2X),/21X,'OPEN     ',
     *       6(F6.0,2X),/21X,'FLEX     ',6(F6.0,2X),/21X,'EXTEND   ',
     *       6(F6.0,2X),/21X,'PRONATE  ',6(F6.0,2X),/21X,'SUPINATE',
     *       6(F6.0,2X))
 2500 CONTINUE
      DO 2550 I=1,6
      DO 2550 J=1,6
      CONF(I,J)=0.
 2550 CONTINUE
      WRITE (1,9104)TITLE
      WRITE (6,9104)TITLE
      CALL INPUT
      DO 2600 I=1,NVECS
      IF(IDATA(I,9).EQ.NOFFSET) GO TO 2600
      DO 2700 K=1,4
      X(K)=FLOAT(IDATA(I,K))-OFSTM(K)
      IF(X(K).LT.0.) X(K)=0.
 2700 CONTINUE
      CALL DISC(X,GRINV,GRM,WGR,WD(1))
      CALL DISC(X,OPINV,OPM,WOP,WD(2))
      CALL DISC(X,FLINV,FLM,WFL,WD(3))
      CALL DISC(X,EXINV,EXM,WEX,WD(4))
      CALL DISC(X,PRINV,PRM,WPR,WD(5))
      CALL DISC(X,SUINV,SUM,WSU,WD(6))
C
C     FIND MAXIMUM PROBABILITY
C
      DUMMY=WD(1)
      ICASE=1
      DO 2800 J=1,6
      IF(DUMMY.LT.WD(J)) ICASE=J
      IF(DUMMY.LT.WD(J)) DUMMY=WD(J)
 2800 CONTINUE
      IDCASE=IDATA(I,9)
      CONF(IDCASE,ICASE)=CONF(IDCASE,ICASE)+1.
 2600 CONTINUE
      WRITE(1,9105)TITLE
      WRITE(1,9106)((CONF(I,J),J=1,6),I=1,6)
      WRITE(6,9105)TITLE
      WRITE(6,9106)((CONF(I,J),J=1,6),I=1,6)
      STOP
      END
C
C-----MATRIX INVERSE----------------------------------------
C
C                  -1
C     MATRIX W = A
C
C     A = INPUT MATRIX,  W = OUTPUT MATRIX
C     IS = DIMENSION, D = DETERMINATE
C
C
C     **** NOTE *** INPUT AND OUTPUT IN PACKED FORMAT
C
```

```
C
C
      SUBROUTINE INVMAT(A,W,IS,D)
      DIMENSION A(1),W(1),IP(400),ICI(20),T(20)
      DO 15 I=1,IS*IS
   15 W(I)=A(I)
      DO 10 I=1,IS
      ICI(I)=I
   10 IP(I)=0
      DET=1.
      DO 80 I=1,IS
      X=0.
      DO 25 J=1,IS
      IF(IP(J).NE.0)GO TO 25
      DO 20 K=1,IS
      IF(IP(K).NE.0)GO TO 20
      IF(ABS(X).GE.ABS(W(J+(K-1)*IS))) GOTO 20
      X=W(J+(K-1)*IS)
      IR=J
      IC=K
   20 CONTINUE
   25 CONTINUE
      IF(ABS(X).GT.1.E-18) GOTO 30
      DET=0.
      GOTO 35
   30 DET=X*DET
      IP(IR)=1
      IF(IR.EQ.IC)GO TO 50
      DO 40 J=1,IS
      Z=W(J+(IR-1)*IS)
      W(J+(IR-1)*IS)=W(J+(IC-1)*IS)
   40 W(J+(IC-1)*IS)=Z
      DET=-DET
      IT=ICI(IR)
      ICI(IR)=ICI(IC)
      ICI(IC)=IT
   50 DO 65 J=1,IS
      IF(J.EQ.IR)GO TO 65
      DO 60 K=1,IS
      IF(K.EQ.IR)GO TO 60
      W(J+(K-1)*IS)=W(J+(K-1)*IS)-W(J+(IR-1)*IS)*W(IR+(K-1)*IS)/X
   60 CONTINUE
   65 CONTINUE
      W(IR+(IR-1)*IS)=-X
      DO 70 J=1,IS
      W(J+(IR-1)*IS)=W(J+(IR-1)*IS)/X
   70 W(IR+(J-1)*IS)=-W(IR+(J-1)*IS)/X
   80 CONTINUE
      DO 110 I=1,IS
   90 IR=ICI(I)
      IF(IR.EQ.I)GO TO 110
      DO 100 J=1,IS
      Z=W(IR+(J-1)*IS)
      W(IR+(J-1)*IS)=W(I+(J-1)*IS)
  100 W(I+(J-1)*IS)=Z
      IT=ICI(I)
      ICI(I)=ICI(IR)
      ICI(IR)=IT
      GO TO 90
  110 CONTINUE
   35 D=DET
```

```
      RETURN
      END
C
C
C     COMPUTE THE PRODUCT OF A MATRIX AND A VECTOR
C
C           P = A*X
C
C     DIMENSION ASSUMED TO BE 4*4
C
C
      SUBROUTINE AX(A,X,P)
      DIMENSION A(4,4),X(4),P(4)
      IX=4
      IY=4
      DO 10 N=1,IX
      P(N)=0.
   10 CONTINUE
      DO 20 I=1,IX
      DO 20 J=1,IY
      P(I)=A(I,J)*X(J)+P(I)
   20 CONTINUE
      RETURN
      END
C
C       INPUT THE DATA FROM A FILE
C
      SUBROUTINE INPUT
      CHARACTER*40 FILENAME
      LOGICAL ANSWER,FLAGPRINT
      DIMENSION IDATA(999,9),NVECT(26),TITLE(10)
      COMMON IDATA,NVECS,NCASES,NOFFSET,NELEM
      COMMON /TITLE/TITLE
      COMMON /FLAGS/FLAGPRINT
      DATA FLAGPRINT/.FALSE./
C
C       GET FILE NAME AND OPEN FILE
C
100   WRITE (6,1000)
      READ (6,1001) FILENAME
      OPEN (UNIT=2,NAME=FILENAME,READONLY,TYPE='OLD',ERR=110)
      GO TO 130
C
C       FILE OPENING ERROR
C
110   WRITE (6,1002)
120   CLOSE (UNIT=2,ERR=100)
      GO TO 100
C
C       FIND # CASES (1-26)
C
130   WRITE (6,1003)
      READ (6,1004) NCASES
      IF ((NCASES.GT.0).AND.(NCASES.LT.27)) GO TO 140
      WRITE (6,1005)
      GO TO 130
C
C       FIND # VECTORS IN EACH CASE
C
140   NVECS=0
      DO 160 I=1,NCASES
```

```
150           WRITE (6,1006) I
              READ (6,1004) NVECT(I)
              IF (NVECT(I).EQ.-99) GO TO 140
              IF ((NVECT(I).LT.1).OR.(NVECT(I).GT.130)) GO TO 150
              NVECS=NVECS+NVECT(I)
160   CONTINUE
      IF (NVECS.LT.999) GO TO 170
      WRITE (6,1007)
      GO TO 140
C
C       ATTEMPT TO READ THE DATA FILE
C
170   READ (2,1008,ERR=180,END=190) ((IDATA(I,J),J=1,8),I=1,NVECS)
      WRITE (6,1009)
      GO TO 200
180   WRITE (6,1010)
      GO TO 120
190   WRITE (6,1011) I
      READ (6,1012) ANSWER
      IF (ANSWER.EQ.'R') GO TO 180
200   CLOSE (UNIT=2,ERR=210)
210   IX=0
      DO 230 I=1,NCASES
              DO 220 J=1,NVECT(I)
                      IX=IX+1
                      IDATA(IX,9)=I
220           CONTINUE
230   CONTINUE
C
C       FIND OUT # ACTIVE ELEMENTS PER VECTOR
C
240   WRITE (6,1015)
      READ (6,1004) NELEM
      IF ((NELEM.LT.1).OR.(NELEM.GT.8)) GO TO 240
C
C       FIND OFFSET CASE NUMBER
C
      WRITE (6,1013)
250   READ (6,1004) NOFFSET
      IF ((NOFFSET.GT.0).AND.(NOFFSET.LE.NCASES)) GO TO 260
      WRITE (6,1014) NCASES
      GO TO 240
260   WRITE (6,1020)
      READ (6,1022) TITLE
      WRITE (6,1016)
      READ (6,1012) ANSWER
      IF (ANSWER.EQ.'Y') FLAGPRINT=.TRUE.
      RETURN
1000  FORMAT (1X,'INPUT DATA FILE NAME : '$)
1001  FORMAT (A40)
1002  FORMAT (1X,'FILE OPENING ERROR')
1003  FORMAT (1X,'INPUT # OF CASES (0<N<27): '$)
1004  FORMAT (I10)
1005  FORMAT (1X,'# CASES OUTSIDE RANGE')
1006  FORMAT (5X,'CASE #',I2,'  # VECTORS = '$)
1007  FORMAT (1X,'TOO MANY VECTORS (MUST BE <501)')
1008  FORMAT (10X,16Z2)
1009  FORMAT (1X,'DATA READ')
1010  FORMAT (1X,'FILE READING ERROR, FILE CLOSED')
1011  FORMAT (1X,'PREMATURE END OF FILE ON VECTOR ',I3/1X,
     1 'ACCEPT OR RECYCLE (A OR R)?'$)
```

```
1012  FORMAT (A1)
1013  FORMAT (1X,'WHICH CASE IS NULL INPUT?'$)
1014  FORMAT (1X,'**MUST BE BETWEEN 1 AND ',I2,'**')
1015  FORMAT (1X,'HOW MANY ELEMENTS ARE ACTIVE?'$)
1016  FORMAT (1X,'OUTPUT OPTIONS (ANSWER Y OR N)'/5X,'DATA PRINTOUT: '$)
1020  FORMAT (1X,'TITLE FOR OUTPUT: '$)
1022  FORMAT (10A4)
      END
C
C       NORMALIZE THE VECTOR (W,X,Y,Z)
C
      SUBROUTINE NORMAL(W,X,Y,Z)
      WXYZ=SQRT(W**2+X**2+Y**2+Z**2)
      IF (WXYZ.EQ.0.0) RETURN
      W=W/WXYZ
      X=X/WXYZ
      Y=Y/WXYZ
      Z=Z/WXYZ
      RETURN
      END
C
C             T
C   COMPUTE Y X FOR 4 DIMENSIONAL VECTORS
C
C     OUTPUT IS SUM
C
C
      SUBROUTINE YTX(SUM,Y,X)
      DIMENSION Y(4),X(4)
      SUM=0.
      DO 10 I=1,4
      SUM=SUM+Y(I)*X(I)
 10   CONTINUE
      RETURN
      END
C
C
C      COMPUTE THE CONSTANT TERM FOR THE DISCRIMINANT FUNCTION
C
C                T      -1
C   W = -1/2 * U * SIGMA  * U  - 1/2 * LOG(DET)
C
C     U = MEAN VECTOR FOR CASE
C
C         -1
C     SIGMA = COVARIANCE MATRIX INVERSE
C
C     DET = DETERMINANT OF THE COVARIANCE MATRIX
C
C
      SUBROUTINE WCONST(U,COVINV,DET,W)
      DIMENSION U(4),COVINV(4,4),P(4)
      CALL AX(COVINV,U,P)
      CALL YTX(SUM,U,P)
      W=-SUM/2.-ALOG10(DET)/2.
      RETURN
      END
C
C
C  DISCRIMINANT FUNCTION EVALUATION
C
```

```
C
      SUBROUTINE DISC(X,COVINV,U,WCONST,W)
      DIMENSION X(4),COVINV(4,4),U(4),P(4)
      CALL AX(COVINV,X,P)
      CALL YTX(C,P,X)
      CALL AX(COVINV,U,P)
      CALL YTX(TOT,P,X)
      W=-C/2.+TOT+WCONST
      RETURN
      END
```

C. Parametric Classifier with Normalized Vectors

```
C       PARAMETRIC CLASSIFIER FOR NORMALIZED EMG VECTORS
C
C       WRITTEN BY D.C. DENING  9/10/81
C
C       TRAINING IS DONE OF FIRST DATA FILE INPUT
C       EVALUATION IS DONE ON SECOND DATA FILE
C
C       (DATA FILE VECTORS ARE UNNORMALIZED)
C       (NORMALIZATION IS DONE INTERNALLY TO THE PROGRAM)
C
C       VARIABLES:
C       IDATA (999,9)   CONTAINS THE INPUT DATA POINTS, PLUS CASE LABEL
C                       FOR EACH VECTOR IN ELEMENT 9 OF EACH ROW
C       NVECS           NUMBER OF DATA VECTORS
C       NCASES          NUMBER OF CASES
C       NOFFSET         CASE NUMBER OF ZERO INPUT CASE
C       AVERAGE         MEAN VECTORS FOR EACH CASE
C       COVAR           COVARIANCE MATRICES FOR EACH CASE
C       A               COVARIANCE MATRIX FOR ENSEMBLE ABOUT ZERO INPUT POINT
C       GRANDM          GRAND MEAN VECTOR FOR ENSEMBLE
C       GRANDC          COVARIANCE MATRIX FOR ENSEMBLE ABOUT ORIGIN
C
C
C
      LOGICAL FLAGPRINT
      DIMENSION GRANDM(8),GRANDC(8,8),TITLE(10)
      DIMENSION AVERAGE(26,9),COVAR(26,8,8),A(8,8),S(8,8),X(4),INPT(4)
      DIMENSION IDATA(999,9),OFFSET(8),CONF(6,6),WD(6),DATA(999,9)
      DIMENSION GR(4,4),GP(4,4),FL(4,4),EX(4,4),PR(4,4),SU(4,4)
      DIMENSION GRM(4),GPM(4),FLM(4),EXM(4),PRM(4),SUM(4),OFSTM(4)
      DIMENSION GRINV(4,4),GPINV(4,4),FLINV(4,4),EXINV(4,4),PRINV(4,4),
     *          SUINV(4,4)
      COMMON IDATA,NVECS,NCASES,NOFFSET,NELEM
      COMMON /TITLE/ TITLE
      COMMON /FLAGS/ FLAGPRINT
C
C       INPUT DATA AND CONTROL PARAMETERS
C
      CALL INPUT
C
C       COMPUTE THE DC OFFSET IN EACH VECTOR FROM THE NULL CASE
C
      DO 30 I=1,4
      OFSTM(I)=0.
 30   CONTINUE
      NNULL=0
      DO 40 I=1,NVECS
      IF(IDATA(I,9).EQ.NOFFSET) GO TO 35
      GO TO 45
 35   NNULL=NNULL+1
      DO 45 J=1,4
      OFSTM(J)=OFSTM(J)+FLOAT(IDATA(I,J))
 45   CONTINUE
 40   CONTINUE
      WRITE(6,9013) NNULL
      DO 46 I=1,4
      OFSTM(I)=OFSTM(I)/FLOAT(NNULL)
 46   CONTINUE
C
C
C       NORMALIZE INPUT VECTORS..... DATA IS NORMALIZED IDATA
```

```
C
      DO 50 I=1,NVECS
      DO 60 J=1,4
      DATA(I,J)=FLOAT(IDATA(I,J))-OFSTM(J)
 60   CONTINUE
      CALL NORMAL(DATA(I,1),DATA(I,2),DATA(I,3),DATA(I,4))
 50   CONTINUE
C
C       INITIALIZE MEANS AND COVARIANCES MATRICES
C
      DO 300 I=1,26
              DO 200 J=1,8
                      AVERAGE(I,J)=0.0
                      DO 100 K=1,8
                              COVAR(I,J,K)=0.0
100                   CONTINUE
200           CONTINUE
              AVERAGE(I,9)=0.0
300   CONTINUE
C
C       ZERO OUT GRAND MEAN AND COVARIANCE
C
      DO 1100 I=1,8
              GRANDM(I)=0.0
              DO 1000 J=1,8
                      GRANDC(I,J)=0.0
1000          CONTINUE
1100  CONTINUE
C
C       CALCULATE MEANS
C
      DO 500 I=1,NVECS
              K=IDATA(I,9)
              DO 400 J=1,8
                      AVERAGE(K,J)=AVERAGE(K,J)+DATA(I,J)
                      GRANDM(J)=GRANDM(J)+DATA(I,J)
400           CONTINUE
              AVERAGE(K,9)=AVERAGE(K,9)+1.0
500   CONTINUE
      DO 700 I=1,NCASES
              IF (AVERAGE(I,9).NE.0.0) THEN
                      DO 600 J=1,8
                             AVERAGE(I,J)=AVERAGE(I,J)/AVERAGE(I,9)
600                   CONTINUE
              ENDIF
700   CONTINUE
C
C       CALCULATE COVARIANCE MATRIX FOR EACH CASE
C
      DO 900 I=1,NVECS
              N=IDATA(I,9)
              DO 800 J=1,8
              DO 800 K=1,8
                      COVAR(N,J,K)=COVAR(N,J,K)+(DATA(I,J)
     1                -AVERAGE(N,J))*(DATA(I,K)-AVERAGE(N,K))
800           CONTINUE
900   CONTINUE
      DO 1200 I=1,8
              GRANDM(I)=GRANDM(I)/FLOAT(NVECS)
1200  CONTINUE
      DO 1400 I=1,NVECS
```

```
            DO 1300 J=1,8
              DO 1300 K=1,J
                      QRANDC(J,K)=QRANDC(J,K)+(DATA(I,J)-
     1                QRANDM(J))*(DATA(I,K)-QRANDM(K))
                      QRANDC(K,J)=QRANDC(J,K)
1300          CONTINUE
1400  CONTINUE
C
C       NOW DIVIDE BY (# VECTORS-1) TO ESTIMATE COVARIANCE
C
      DO 1600 I=1,NCASES
              IF (AVERAGE(I,9).GT.1.0) THEN
                      DO 1500 J=1,8
                      DO 1500 K=1,8
                           COVAR(I,J,K)=COVAR(I,J,K)/(AVERAGE(I,9)-1.0)
1500                  CONTINUE
              ENDIF
1600  CONTINUE
      DO 1800 I=1,8
            OFFSET(I)=AVERAGE(NOFFSET,I)
              DO 1700 J=1,8
                      QRANDC(I,J)=QRANDC(I,J)/FLOAT(NVECS-1)
1700          CONTINUE
1800  CONTINUE
C
C       PRINT INPUT DATA, MEANS, COVARIANCE MATRICES,
C        QRAND MEAN, QRAND COVARIANCE, EIGENVALUES, AND EIGENVECTORS
C
      IF (.NOT.FLAGPRINT) GO TO 2175
      WRITE (1,9012) TITLE
      DO 2000 I=1,NVECS
              IF (I.EQ.1) THEN
                 WRITE (1,9000) (K,K=1,8),IFIX(AVERAGE(IDATA(I,9),9)),
     1                (IDATA(I,J),J=1,8)
              ELSE IF (IDATA(I,9).NE.IDATA(I-1,9)) THEN
                 WRITE (1,9001) IDATA(I,9),IFIX(AVERAGE(IDATA(I,9),9)),
     1                (IDATA(I,J),J=1,8)
              ELSE
                      WRITE (1,9002) (IDATA(I,J),J=1,8)
              ENDIF
2000  CONTINUE
      DO 2100 I=1,NCASES
              IF (((I+1)/2)*2.EQ.I+1) WRITE (1,9003)TITLE
              WRITE (1,9004) I,(AVERAGE(I,J),J=1,8)
              WRITE (1,9005) ((COVAR(I,J,K),K=1,8),J=1,8)
2100  CONTINUE
      WRITE (1,9003) TITLE
      WRITE (1,9006) NCASES,(QRANDM(I),I=1,8)
      WRITE (1,9007) (AVERAGE(NOFFSET,I),I=1,8)
      WRITE (1,9008) ((QRANDC(I,J),J=1,8),I=1,8)
2175  CONTINUE
9000  FORMAT (1X,'DATA VECTORS AS READ'//27X,'VARIABLE #'//12X,
     1 8(I1,5X)//1X,'CASE # 1:     # VECTORS =',I3//11X,8(I3,3X))
9001  FORMAT (/1X,'CASE # ',I2,':     # VECTORS = ',I3//11X,8(I3,3X))
9002  FORMAT (11X,8(I3,3X))
9003  FORMAT ('1',10A4)
9004  FORMAT (1X,'CASE # ',I2///6X,'AVERAGE VECTOR :   ',8(F6.2,4X)///
     1 6X,'COVARIANCE MATRIX'//)
9005  FORMAT (8(11X,8(F8.2,3X)//)///)
9006  FORMAT (1X,'QRAND MEAN FOR ',I2,' CASES :   ',8(F6.2,4X)///)
9007  FORMAT (1X,'OFFSET VECTOR :   ',8(F6.2,4X)///)
```

```
9008  FORMAT (1X,'GRAND COVARIANCE MATRIX'//8(11X,8(F8.2,3X)//)////)
9009  FORMAT (1X,'COVARIANCE MATRIX'//4(11X,4(F8.2,3X)//)////)
9010  FORMAT (1X,'INVERSE OF COVARIENCE'//4(14X,(4(E12.4,3X)/)////))
9011  FORMAT (1X,'DETERMINATE FOR CASE #',I3,'  =',E12.4/)
9012  FORMAT (1X,10A4//)
9013  FORMAT (1X,'NUMBER OF NULL CASE VECTORS = ',I4)
      DO 2200 I=1,4
      GRM(I)=AVERAGE(1,I)
      OPM(I)=AVERAGE(2,I)
      FLM(I)=AVERAGE(3,I)
      EXM(I)=AVERAGE(4,I)
      PRM(I)=AVERAGE(5,I)
      SUM(I)=AVERAGE(6,I)
        DO 2200 J=1,4
        GR(I,J)=COVAR(1,I,J)
        OP(I,J)=COVAR(2,I,J)
        FL(I,J)=COVAR(3,I,J)
        EX(I,J)=COVAR(4,I,J)
        PR(I,J)=COVAR(5,I,J)
        SU(I,J)=COVAR(6,I,J)
 2200 CONTINUE
      CALL INVMAT(GR,GRINV,4,DGR)
      CALL INVMAT(OP,OPINV,4,DOP)
      CALL INVMAT(FL,FLINV,4,DFL)
      CALL INVMAT(EX,EXINV,4,DEX)
      CALL INVMAT(PR,PRINV,4,DPR)
      CALL INVMAT(SU,SUINV,4,DSU)
      IF(.NOT.FLAGPRINT) GO TO 2250
      K=1
         WRITE (1,9004) K,(GRM(I),I=1,4)
         WRITE (1,9009) ((GR(I,J),J=1,4),I=1,4)
         WRITE (1,9011) K,DGR
         WRITE (1,9010) ((GRINV(I,J),J=1,4),I=1,4)
      K=K+1
         WRITE (1,9004) K,(OPM(I),I=1,4)
         WRITE (1,9009) ((OP(I,J),J=1,4),I=1,4)
         WRITE (1,9011) K,DOP
         WRITE (1,9010) ((OPINV(I,J),J=1,4),I=1,4)
      K=K+1
         WRITE (1,9004) K,(FLM(I),I=1,4)
         WRITE (1,9009) ((FL(I,J),J=1,4),I=1,4)
         WRITE (1,9011) K,DFL
         WRITE (1,9010) ((FLINV(I,J),J=1,4),I=1,4)
      K=K+1
         WRITE (1,9004) K,(EXM(I),I=1,4)
         WRITE (1,9009) ((EX(I,J),J=1,4),I=1,4)
         WRITE (1,9011) K,DEX
         WRITE (1,9010) ((EXINV(I,J),J=1,4),I=1,4)
      K=K+1
         WRITE (1,9004) K,(PRM(I),I=1,4)
         WRITE (1,9009) ((PR(I,J),J=1,4),I=1,4)
         WRITE (1,9011) K,DPR
         WRITE (1,9010) ((PRINV(I,J),J=1,4),I=1,4)
      K=K+1
         WRITE (1,9004) K,(SUM(I),I=1,4)
         WRITE (1,9009) ((SU(I,J),J=1,4),I=1,4)
         WRITE (1,9011) K,DSU
         WRITE (1,9010) ((SUINV(I,J),J=1,4),I=1,4)
 2250 CONTINUE
      CALL WCONST(GRM,GRINV,DGR,WGR)
      CALL WCONST(OPM,OPINV,DOP,WOP)
```

```
      CALL WCONST(FLM,FLINV,DFL,WFL)
      CALL WCONST(EXM,EXINV,DEX,WEX)
      CALL WCONST(PRM,PRINV,DPR,WPR)
      CALL WCONST(SUM,SUINV,DSU,WSU)
 2300 WRITE(6,9100)
      READ (6,9101,ERR=2300)(INPT(I),I=1,4)
      DO 2400 I=1,4
      X(I)=FLOAT(INPT(I))-OFSTM(I)
 2400 CONTINUE
      WRITE (6,9103)(X(I),I=1,4)
      IF(X(1).LT.0.)GO TO 2500
      CALL DISC(X,GRINV,GRM,WGR,GRDIS)
      CALL DISC(X,OPINV,OPM,WOP,OPDIS)
      CALL DISC(X,FLINV,FLM,WFL,FLDIS)
      CALL DISC(X,EXINV,EXM,WEX,EXDIS)
      CALL DISC(X,PRINV,PRM,WPR,PRDIS)
      CALL DISC(X,SUINV,SUM,WSU,SUDIS)
      WRITE (6,9102) GRDIS,OPDIS,FLDIS,EXDIS,PRDIS,SUDIS
      GO TO 2300
9100  FORMAT(//1X,'INPUT DATA VECTOR TO CLASSIFY (4 COMPONENTS)..')
9101  FORMAT(4I3)
9102  FORMAT(/1X,'GRASP DISCRIMINANT   =  ',E12.4,
     *       /1X,'OPEN DISCRIMINANT    =  ',E12.4,
     *       /1X,'FLEX DISCRIMINANT    =  ',E12.4,
     *       /1X,'EXTEND DISCRIMINANT  =  ',E12.4,
     *       /1X,'PRONATE DISCRIMINANT =  ',E12.4,
     *       /1X,'SUPINATE DISCRIMINANT = ',E12.4)
9103  FORMAT(/1X,'INPUT MINUS OFFSET = ',4F8.2)
9104  FORMAT(//1X,'COMPUTED NORMALIZED VECTOR CLASSIFIER PARAMETERS',
     * ' FROM ',10A4,/1X,'NOW READY TO INPUT VECTORS TO CLASSIFY'/)
9105  FORMAT(///1X,'CONFUSION MATRIX FROM NORMALIZED DATA SET ',10A4)
9106  FORMAT(///30X,'DATA CLASSIFIED AS',/28X,
     *       '  GRASP   OPEN    FLEX   EXTEND  PRONATE SUPINATE',/1X,
     *      'INPUT VECTOR CLASS: GRASP   ',6(F6.0,2X),/21X,'OPEN    ',
     *      6(F6.0,2X),/21X,'FLEX    ',6(F6.0,2X),/21X,'EXTEND  ',
     *      6(F6.0,2X),/21X,'PRONATE ',6(F6.0,2X),/21X,'SUPINATE',
     *      6(F6.0,2X))
 2500 CONTINUE
      DO 2550 I=1,6
      DO 2550 J=1,6
      CONF(I,J)=0.
 2550 CONTINUE
      WRITE (1,9104)TITLE
      WRITE (6,9104)TITLE
      CALL INPUT
      DO 2600 I=1,NVECS
      IF(IDATA(I,9).EQ.NOFFSET) GO TO 2600
      DO 2700 K=1,4
      X(K)=FLOAT(IDATA(I,K))-OFSTM(K)
      IF(X(K).LT.0.) X(K)=0.
 2700 CONTINUE
      CALL NORMAL(X(1),X(2),X(3),X(4))
      CALL DISC(X,GRINV,GRM,WGR,WD(1))
      CALL DISC(X,OPINV,OPM,WOP,WD(2))
      CALL DISC(X,FLINV,FLM,WFL,WD(3))
      CALL DISC(X,EXINV,EXM,WEX,WD(4))
      CALL DISC(X,PRINV,PRM,WPR,WD(5))
      CALL DISC(X,SUINV,SUM,WSU,WD(6))
C
C     FIND MAXIMUM PROBABILITY
C
```

```
      DUMMY=WD(1)
      ICASE=1
      DO 2800 J=1,6
      IF(DUMMY.LT.WD(J)) ICASE=J
      IF(DUMMY.LT.WD(J)) DUMMY=WD(J)
 2800 CONTINUE
      IDCASE=IDATA(I,9)
      CONF(IDCASE,ICASE)=CONF(IDCASE,ICASE)+1.
 2600 CONTINUE
      WRITE(1,9105)TITLE
      WRITE(1,9106)((CONF(I,J),J=1,6),I=1,6)
      WRITE(6,9105)TITLE
      WRITE(6,9106)((CONF(I,J),J=1,6),I=1,6)
      STOP
      END
C
C-----MATRIX INVERSE-----------------------------------
C
C                  -1
C     MATRIX W = A
C
C     A = INPUT MATRIX,   W = OUTPUT MATRIX
C     IS = DIMENSION,  D = DETERMINATE
C
C
C     **** NOTE *** INPUT AND OUTPUT IN PACKED FORMAT
C
C
C
      SUBROUTINE INVMAT(A,W,IS,D)
      DIMENSION A(1),W(1),IP(400),ICI(20),T(20)
      DO 15 I=1,IS*IS
   15 W(I)=A(I)
      DO 10 I=1,IS
      ICI(I)=I
   10 IP(I)=0
      DET=1.
      DO 80 I=1,IS
      X=0.
      DO 25 J=1,IS
      IF(IP(J).NE.0)GO TO 25
      DO 20 K=1,IS
      IF(IP(K).NE.0)GO TO 20
      IF(ABS(X).GE.ABS(W(J+(K-1)*IS))) GOTO 20
      X=W(J+(K-1)*IS)
      IR=J
      IC=K
   20 CONTINUE
   25 CONTINUE
      IF(ABS(X).GT.1.E-18) GOTO 30
      DET=0.
      GOTO 35
   30 DET=X*DET
      IP(IR)=1
      IF(IR.EQ.IC)GO TO 50
      DO 40 J=1,IS
      Z=W(J+(IR-1)*IS)
      W(J+(IR-1)*IS)=W(J+(IC-1)*IS)
   40 W(J+(IC-1)*IS)=Z
      DET=-DET
      IT=ICI(IR)
```

```
      ICI(IR)=ICI(IC)
      ICI(IC)=IT
   50 DO 65 J=1,IS
      IF(J.EQ.IR)GO TO 65
      DO 60 K=1,IS
      IF(K.EQ.IR)GO TO 60
      W(J+(K-1)*IS)=W(J+(K-1)*IS)-W(J+(IR-1)*IS)*W(IR+(K-1)*IS)/X
   60 CONTINUE
   65 CONTINUE
      W(IR+(IR-1)*IS)=-X
      DO 70 J=1,IS
      W(J+(IR-1)*IS)=W(J+(IR-1)*IS)/X
   70 W(IR+(J-1)*IS)=-W(IR+(J-1)*IS)/X
   80 CONTINUE
      DO 110 I=1,IS
   90 IR=ICI(I)
      IF(IR.EQ.I)GO TO 110
      DO 100 J=1,IS
      Z=W(IR+(J-1)*IS)
      W(IR+(J-1)*IS)=W(I+(J-1)*IS)
  100 W(I+(J-1)*IS)=Z
      IT=ICI(I)
      ICI(I)=ICI(IR)
      ICI(IR)=IT
      GO TO 90
  110 CONTINUE
   35 D=DET
      RETURN
      END
C
C
C     COMPUTE THE PRODUCT OF A MATRIX AND A VECTOR
C
C           P = A*X
C
C     DIMENSION ASSUMED TO BE 4*4
C
C
      SUBROUTINE AX(A,X,P)
      DIMENSION A(4,4),X(4),P(4)
      IX=4
      IY=4
      DO 10 N=1,IX
      P(N)=0.
   10 CONTINUE
      DO 20 I=1,IX
      DO 20 J=1,IY
      P(I)=A(I,J)*X(J)+P(I)
   20 CONTINUE
      RETURN
      END
C
C       INPUT THE DATA FROM A FILE
C
      SUBROUTINE INPUT
      CHARACTER*40 FILENAME
      LOGICAL ANSWER,FLAGPRINT
      DIMENSION IDATA(999,9),NVECT(26),TITLE(10)
      COMMON IDATA,NVECS,NCASES,NOFFSET,NELEM
      COMMON /TITLE/TITLE
      COMMON /FLAGS/FLAGPRINT
```

```
      DATA FLAGPRINT/.FALSE./
C
C       GET FILE NAME AND OPEN FILE
C
100   WRITE (6,1000)
      READ (6,1001) FILENAME
      OPEN (UNIT=2,NAME=FILENAME,READONLY,TYPE='OLD',ERR=110)
      GO TO 130
C
C       FILE OPENING ERROR
C
110   WRITE (6,1002)
120   CLOSE (UNIT=2,ERR=100)
      GO TO 100
C
C       FIND # CASES (1-26)
C
130   WRITE (6,1003)
      READ (6,1004) NCASES
      IF ((NCASES.GT.0).AND.(NCASES.LT.27)) GO TO 140
      WRITE (6,1005)
      GO TO 130
C
C       FIND # VECTORS IN EACH CASE
C
140   NVECS=0
      DO 160 I=1,NCASES
150           WRITE (6,1006) I
              READ (6,1004) NVECT(I)
              IF (NVECT(I).EQ.-99) GO TO 140
              IF ((NVECT(I).LT.1).OR.(NVECT(I).GT.130)) GO TO 150
              NVECS=NVECS+NVECT(I)
160   CONTINUE
      IF (NVECS.LT.999) GO TO 170
      WRITE (6,1007)
      GO TO 140
C
C       ATTEMPT TO READ THE DATA FILE
C
170   READ (2,1008,ERR=180,END=190) ((IDATA(I,J),J=1,8),I=1,NVECS)
      WRITE (6,1009)
      GO TO 200
180   WRITE (6,1010)
      GO TO 120
190   WRITE (6,1011) I
      READ (6,1012) ANSWER
      IF (ANSWER.EQ.'R') GO TO 180
200   CLOSE (UNIT=2,ERR=210)
210   IX=0
      DO 230 I=1,NCASES
              DO 220 J=1,NVECT(I)
                      IX=IX+1
                      IDATA(IX,9)=I
220           CONTINUE
230   CONTINUE
C
C       FIND OUT # ACTIVE ELEMENTS PER VECTOR
C
240   WRITE (6,1015)
      READ (6,1004) NELEM
      IF ((NELEM.LT.1).OR.(NELEM.GT.8)) GO TO 240
```

```
C
C       FIND OFFSET CASE NUMBER
C
      WRITE (6,1013)
250   READ (6,1004) NOFFSET
      IF ((NOFFSET.GT.0).AND.(NOFFSET.LE.NCASES)) GO TO 260
      WRITE (6,1014) NCASES
      GO TO 240
260   WRITE (6,1020)
      READ (6,1022) TITLE
      WRITE (6,1016)
      READ (6,1012) ANSWER
      IF (ANSWER.EQ.'Y') FLAGPRINT=.TRUE.
      RETURN
1000  FORMAT (1X,'INPUT DATA FILE NAME : '$)
1001  FORMAT (A40)
1002  FORMAT (1X,'FILE OPENING ERROR')
1003  FORMAT (1X,'INPUT # OF CASES (0<N<27): '$)
1004  FORMAT (I10)
1005  FORMAT (1X,'# CASES OUTSIDE RANGE')
1006  FORMAT (5X,'CASE #',I2,'  # VECTORS = '$)
1007  FORMAT (1X,'TOO MANY VECTORS (MUST BE <501)')
1008  FORMAT (10X,16Z2)
1009  FORMAT (1X,'DATA READ')
1010  FORMAT (1X,'FILE READING ERROR, FILE CLOSED')
1011  FORMAT (1X,'PREMATURE END OF FILE ON VECTOR ',I3/1X,
     1 'ACCEPT OR RECYCLE (A OR R)?'$)
1012  FORMAT (A1)
1013  FORMAT (1X,'WHICH CASE IS NULL INPUT?'$)
1014  FORMAT (1X,'**MUST BE BETWEEN 1 AND ',I2,'**')
1015  FORMAT (1X,'HOW MANY ELEMENTS ARE ACTIVE?'$)
1016  FORMAT (1X,'OUTPUT OPTIONS (ANSWER Y OR N)'/5X,'DATA PRINTOUT: '$)
1020  FORMAT (1X,'TITLE FOR OUTPUT: '$)
1022  FORMAT (10A4)
      END
C
C       NORMALIZE THE VECTOR (W,X,Y,Z)
C
      SUBROUTINE NORMAL(W,X,Y,Z)
      WXYZ=SQRT(W**2+X**2+Y**2+Z**2)
      IF (WXYZ.EQ.0.0) RETURN
      W=W/WXYZ
      X=X/WXYZ
      Y=Y/WXYZ
      Z=Z/WXYZ
      RETURN
      END
C
C            T
C   COMPUTE Y X FOR 4 DIMENSIONAL VECTORS
C
C     OUTPUT IS SUM
C
C
      SUBROUTINE YTX(SUM,Y,X)
      DIMENSION Y(4),X(4)
      SUM=0.
      DO 10 I=1,4
      SUM=SUM+Y(I)*X(I)
 10   CONTINUE
      RETURN
```

```
      END
C
C
C      COMPUTE THE CONSTANT TERM FOR THE DISCRIMINANT FUNCTION
C
C                  T      -1
C   W = -1/2 * U * SIGMA  * U  - 1/2 * LOG(DET)
C
C     U = MEAN VECTOR FOR CASE
C
C         -1
C     SIGMA = COVARIANCE MATRIX INVERSE
C
C     DET = DETERMINANT OF THE COVARIANCE MATRIX
C
C
      SUBROUTINE WCONST(U,COVINV,DET,W)
      DIMENSION U(4),COVINV(4,4),P(4)
      CALL AX(COVINV,U,P)
      CALL YTX(SUM,U,P)
      W=-SUM/2.-ALOG10(DET)/2.
      RETURN
      END
C
C
C  DISCRIMINANT FUNCTION EVALUATION
C
C
      SUBROUTINE DISC(X,COVINV,U,WCONST,W)
      DIMENSION X(4),COVINV(4,4),U(4),P(4)
      CALL AX(COVINV,X,P)
      CALL YTX(C,P,X)
      CALL AX(COVINV,U,P)
      CALL YTX(TOT,P,X)
      W=-C/2.+TOT+WCONST
      RETURN
      END
```

D. Nearest Neighbor Classifier

```
C       NEAREST NEIGHBOR EMG VECTOR CLASSIFIER
C
C       WRITTEN BY D.C. DENING   9/15/81
C
C       REFERENCE TABLE SETUP IS DONE USING THE FIRST DATA FILE
C
C       EVALUATION IS DONE USING THE SECOND DATA FILE
C
C
C       VARIABLES:
C       IDATA (999,9)    CONTAINS THE INPUT DATA POINTS, PLUS CASE LABEL
C                        FOR EACH VECTOR IN ELEMENT 9 OF EACH ROW
C       NVECS            NUMBER OF DATA VECTORS
C       NCASES           NUMBER OF CASES
C       NOFFSET          CASE NUMBER OF ZERO INPUT CASE
C
C
C
      LOGICAL FLAGPRINT
      DIMENSION TITLE(10)
      DIMENSION OFSTM(4),X(4),TEMP(9)
      DIMENSION IDATA(999,9),CONF(6,6),DATA(999,9),REF(999,9)
      COMMON IDATA,NVECS,NCASES,NOFFSET,NELEM
      COMMON /TITLE/ TITLE
      COMMON /FLAGS/ FLAGPRINT
C
C       INPUT DATA AND CONTROL PARAMETERS
C
      CALL INPUT
C
C       COMPUTE THE DC OFFSET IN EACH VECTOR FROM THE NULL CASE
C
      DO 30 I=1,4
      OFSTM(I)=0.
 30   CONTINUE
      NNULL=0
      DO 40 I=1,NVECS
      IF(IDATA(I,9).NE.NOFFSET) GO TO 45
 35   NNULL=NNULL+1
      DO 45 J=1,4
      OFSTM(J)=OFSTM(J)+FLOAT(IDATA(I,J))
 45   CONTINUE
 40   CONTINUE
      WRITE(6,9000) NNULL
      DO 46 I=1,4
      OFSTM(I)=OFSTM(I)/FLOAT(NNULL)
 46   CONTINUE
C
C
C       REMOVE OFFSET FROM INPUT VECTORS..... DATA IS CONVERTED IDATA
C
      DO 50 I=1,NVECS
      DATA(I,9)=FLOAT(IDATA(I,9))
      DO 60 J=1,4
      DATA(I,J)=FLOAT(IDATA(I,J))-OFSTM(J)
      DATA(I,J+4)=FLOAT(IDATA(I,J+4))
 60   CONTINUE
      DATA(I,8)=FLOAT(I)
 50   CONTINUE
C
C       SEED ONE VECTOR INTO THE REFERENCE TABLE
```

```
C
      NREF=1
      DO 100 I=1,9
      REF(1,I)=DATA(1,I)
100   CONTINUE
      NREFOLD=NREF
      DO 300 I=1,NVECS
      IF(IIFIX(DATA(I,9)).EQ.NOFFSET) GO TO 300
         DO 150 J=1,4
         X(J)=DATA(I,J)
150      CONTINUE
      CALL CLASS(REF,X,ICLASS,NREF)
      IF(IIFIX(DATA(I,9)).EQ.ICLASS) GO TO 300
         DO 200 J=1,9
         REF(NREF+1,J)=DATA(I,J)
200      CONTINUE
         NREF=NREF+1
300   CONTINUE
      WRITE(1,9001)NREF
      WRITE(6,9001)NREF
      IF(NREFOLD.LT.NREF)GO TO 100
      WRITE(1,9003)
      WRITE(1,9005)(REF(I,8),I=1,NREF)
      WRITE(6,9003)
      WRITE(6,9005)(REF(I,8),I=1,NREF)
C
C
C     CONDENSE NEAREST NEIGHBOR REFERENCE LIST
C
C
      DO 500 I=1,NREF
         DO 350 J=1,9
         TEMP(J)=REF(I,J)
         IF (J.GT.4) GO TO 350
         REF(I,J)=1000.
350      CONTINUE
      DO 400 K=1,NVECS
      IF(IIFIX(DATA(K,9)).EQ.NOFFSET) GO TO 400
         DO 450 L=1,4
         X(L)=DATA(K,L)
450      CONTINUE
      CALL CLASS(REF,X,ICLASS,NREF)
      IF(IIFIX(DATA(K,9)).EQ.ICLASS) GO TO 400
         DO 470 N=1,9
         REF(I,N)=TEMP(N)
470      CONTINUE
         GO TO 500
400   CONTINUE
500   CONTINUE
      JREF=NREF
C
C     WIPE OUT UNNEEDED REFERENCE VECTORS
C
      DO 800 I=1,NREF
      IF(REF(I,1).LT.1000.) GO TO 800
         DO 700 K=I+1,NREF
            DO 600 N=1,9
            REF(K-1,N)=REF(K,N)
600         CONTINUE
700      CONTINUE
      JREF=JREF-1
```

```
800   CONTINUE
      NREF=JREF
      WRITE(1,9002)NREF
      WRITE(1,9004)
      WRITE(1,9005)(REF(I,8),I=1,NREF)
      WRITE(1,9006)TITLE
      WRITE(6,9002)NREF
      WRITE(6,9004)
      WRITE(6,9005)(REF(I,8),I=1,NREF)
      WRITE(6,9006)TITLE
C
C
C       GET UNKNOWN DATA FILE
C
C
      CALL INPUT
      WRITE(1,9007)TITLE
      WRITE(6,9007)TITLE
C
C         ZERO CONFUSION MATRIX
C
      DO 1000 I=1,6
      DO 1000 J=1,6
      CONF(I,J)=0.
1000  CONTINUE
C
C
C          CLASSIFY THE INPUT VECTORS
C
C
      DO 1100 I=1,NVECS
      IF(IDATA(I,9).EQ.NOFFSET) GO TO 1100
        DO 1150 N=1,4
        X(N)=FLOAT(IDATA(I,N))-OFSTM(N)
1150    CONTINUE
      IDCASE=IDATA(I,9)
      CALL CLASS(REF,X,ICLASS,NREF)
      CONF(IDCASE,ICLASS)=CONF(IDCASE,ICLASS)+1.0
1100  CONTINUE
      WRITE(1,9008)((CONF(I,J),J=1,6),I=1,6)
      WRITE(6,9008)((CONF(I,J),J=1,6),I=1,6)
9000  FORMAT(/1X,'NUMBER OF NULL VECTORS = ',I4)
9001  FORMAT(/1X,'NUMBER OF VECTORS IN REFERENCE TABLE = ',I4)
9002  FORMAT(////1X,'REFERENCE TABLE CONDENSED DOWN TO ',I4,' VECTORS')
9003  FORMAT(//1X,'REFERENCE TABLE AFTER ITERATIONS (VECTOR ID #S)')
9004  FORMAT(///1X,'REFERENCE TABLE AFTER CONDENSATION (VECTOR ID #S)')
9005  FORMAT(10(2X,F5.0))
9006  FORMAT(/////1X,'NEAREST NEIGHBOR LOOK-UP TABLE CREATED WITH FILE
     * ',10A4)
9007  FORMAT(///1X,'WILL NOW CLASSIFY THE UNKNOWN FILE ',10A4)
9008  FORMAT(///30X,'DATA CLASSIFIED AS',/28X,
     *       '  GRASP    OPEN     FLEX    EXTEND  PRONATE  SUPINATE',/1X,
     *       'INPUT VECTOR CLASS: GRASP    ',6(F6.0,2X),/21X,'OPEN     ',
     *       6(F6.0,2X),/21X,'FLEX     ',6(F6.0,2X),/21X,'EXTEND   ',
     *       6(F6.0,2X),/21X,'PRONATE  ',6(F6.0,2X),/21X,'SUPINATE',
     *       6(F6.0,2X))
      STOP
      END
C
C         INPUT THE DATA FROM A FILE
C
```

```
      SUBROUTINE INPUT
      CHARACTER*40 FILENAME
      LOGICAL ANSWER,FLAGPRINT
      DIMENSION IDATA(999,9),NVECT(26),TITLE(10)
      COMMON IDATA,NVECS,NCASES,NOFFSET,NELEM
      COMMON /TITLE/TITLE
      COMMON /FLAGS/FLAGPRINT
      DATA FLAGPRINT/.FALSE./
C
C       GET FILE NAME AND OPEN FILE
C
100   WRITE (6,1000)
      READ (6,1001) FILENAME
      OPEN (UNIT=2,NAME=FILENAME,READONLY,TYPE='OLD',ERR=110)
      GO TO 130
C
C       FILE OPENING ERROR
C
110   WRITE (6,1002)
120   CLOSE (UNIT=2,ERR=100)
      GO TO 100
C
C       FIND # CASES (1-26)
C
130   WRITE (6,1003)
      READ (6,1004) NCASES
      IF ((NCASES.GT.0).AND.(NCASES.LT.27)) GO TO 140
      WRITE (6,1005)
      GO TO 130
C
C       FIND # VECTORS IN EACH CASE
C
140   NVECS=0
      DO 160 I=1,NCASES
150           WRITE (6,1006) I
              READ (6,1004) NVECT(I)
              IF (NVECT(I).EQ.-99) GO TO 140
              IF ((NVECT(I).LT.1).OR.(NVECT(I).GT.130)) GO TO 150
              NVECS=NVECS+NVECT(I)
160   CONTINUE
      IF (NVECS.LT.999) GO TO 170
      WRITE (6,1007)
      GO TO 140
C
C       ATTEMPT TO READ THE DATA FILE
C
170   READ (2,1008,ERR=180,END=190) ((IDATA(I,J),J=1,8),I=1,NVECS)
      WRITE (6,1009)
      GO TO 200
180   WRITE (6,1010)
      GO TO 120
190   WRITE (6,1011) I
      READ (6,1012) ANSWER
      IF (ANSWER.EQ.'R') GO TO 180
200   CLOSE (UNIT=2,ERR=210)
210   IX=0
      DO 230 I=1,NCASES
              DO 220 J=1,NVECT(I)
                      IX=IX+1
                      IDATA(IX,9)=I
220           CONTINUE
```

```
230   CONTINUE
C
C       FIND OUT # ACTIVE ELEMENTS PER VECTOR
C
240   WRITE (6,1015)
      READ (6,1004) NELEM
      IF ((NELEM.LT.1).OR.(NELEM.GT.8)) GO TO 240
C
C       FIND OFFSET CASE NUMBER
C
      WRITE (6,1013)
250   READ (6,1004) NOFFSET
      IF ((NOFFSET.GT.0).AND.(NOFFSET.LE.NCASES)) GO TO 260
      WRITE (6,1014) NCASES
      GO TO 240
260   WRITE (6,1020)
      READ (6,1022) TITLE
      WRITE (6,1016)
      READ (6,1012) ANSWER
      IF (ANSWER.EQ.'Y') FLAGPRINT=.TRUE.
      RETURN
1000  FORMAT (1X,'INPUT DATA FILE NAME : '$)
1001  FORMAT (A40)
1002  FORMAT (1X,'FILE OPENING ERROR')
1003  FORMAT (1X,'INPUT # OF CASES (0<N<27): '$)
1004  FORMAT (I10)
1005  FORMAT (1X,'# CASES OUTSIDE RANGE')
1006  FORMAT (5X,'CASE #',I2,'  # VECTORS = '$)
1007  FORMAT (1X,'TOO MANY VECTORS (MUST BE <501)')
1008  FORMAT (10X,16Z2)
1009  FORMAT (1X,'DATA READ')
1010  FORMAT (1X,'FILE READING ERROR, FILE CLOSED')
1011  FORMAT (1X,'PREMATURE END OF FILE ON VECTOR ',I3/1X,
     1 'ACCEPT OR RECYCLE (A OR R)?'$)
1012  FORMAT (A1)
1013  FORMAT (1X,'WHICH CASE IS NULL INPUT?'$)
1014  FORMAT (1X,'**MUST BE BETWEEN 1 AND ',I2,'**')
1015  FORMAT (1X,'HOW MANY ELEMENTS ARE ACTIVE?'$)
1016  FORMAT (1X,'OUTPUT OPTIONS (ANSWER Y OR N)'/5X,'DATA PRINTOUT: '$)
1020  FORMAT (1X,'TITLE FOR OUTPUT: '$)
1022  FORMAT (10A4)
      END
C
C
C       FIND THE NEAREST NEIGHBOR IN 'DATA' OF 'X'
C
C
      SUBROUTINE CLASS(DATA,X,ICLASS,NVECS)
      DIMENSION DATA(999,9),X(4)
      D=SQRT((DATA(1,1)-X(1))**2 + (DATA(1,2)-X(2))**2 +
     *       (DATA(1,3)-X(3))**2 + (DATA(1,4)-X(4))**2)
      ICLASS=IIFIX(DATA(1,9))
      DO 100 I=1,NVECS
      TEMPD=SQRT((DATA(I,1)-X(1))**2 + (DATA(I,2)-X(2))**2 +
     *           (DATA(I,3)-X(3))**2 + (DATA(I,4)-X(4))**2)
      IF(TEMPD.LT.D) ICLASS=IIFIX(DATA(I,9))
      IF(TEMPD.LT.D) D=TEMPD
100   CONTINUE
      RETURN
      END
C
```

```
C       NORMALIZE THE VECTOR (W, X, Y, Z)
C
      SUBROUTINE NORMAL(W, X, Y, Z)
      WXYZ=SQRT(W**2+X**2+Y**2+Z**2)
      IF (WXYZ.EQ.0.0) RETURN
      W=W/WXYZ
      X=X/WXYZ
      Y=Y/WXYZ
      Z=Z/WXYZ
      RETURN
      END
```

E. Nearest Neighbor Classifier with Normalized Vectors

```
C       NORMALIZED VECTOR NEAREST NEIGHBOR CLASSIFIER
C
C       WRITTEN BY D.C. DENING  9/15/81
C
C       TRAINING IS DONE WITH FIRST DATA FILE INPUT
C       EVALUATION IS DONE USING THE SECOND FILE
C
C       (EMG VECTORS IN DATA FILES ARE NOT NORMALIZED)
C       (NORMALIZATION IS ACCOMPLISHED INTERNALLY IN THE PROGRAM)
C
C       VARIABLES:
C       IDATA (999,9)    CONTAINS THE INPUT DATA POINTS, PLUS CASE LABEL
C                        FOR EACH VECTOR IN ELEMENT 9 OF EACH ROW
C       NVECS            NUMBER OF DATA VECTORS
C       NCASES           NUMBER OF CASES
C       NOFFSET          CASE NUMBER OF ZERO INPUT CASE
C
C
C
      LOGICAL FLAGPRINT
      DIMENSION TITLE(10)
      DIMENSION OFSTM(4),X(4),TEMP(9)
      DIMENSION IDATA(999,9),CONF(6,6),DATA(999,9),REF(999,9)
      COMMON IDATA,NVECS,NCASES,NOFFSET,NELEM
      COMMON /TITLE/ TITLE
      COMMON /FLAGS/ FLAGPRINT
C
C       INPUT DATA AND CONTROL PARAMETERS
C
      CALL INPUT
C
C       COMPUTE THE DC OFFSET IN EACH VECTOR FROM THE NULL CASE
C
      DO 30 I=1,4
      OFSTM(I)=0.
 30   CONTINUE
      NNULL=0
      DO 40 I=1,NVECS
      IF(IDATA(I,9).NE.NOFFSET) GO TO 45
 35   NNULL=NNULL+1
      DO 45 J=1,4
      OFSTM(J)=OFSTM(J)+FLOAT(IDATA(I,J))
 45   CONTINUE
 40   CONTINUE
      WRITE(6,9000) NNULL
      DO 46 I=1,4
      OFSTM(I)=OFSTM(I)/FLOAT(NNULL)
 46   CONTINUE
C
C
C       REMOVE OFFSET FROM INPUT VECTORS AND NORMALIZE ....
C               ..... DATA IS CONVERTED IDATA
C
      DO 50 I=1,NVECS
      DATA(I,9)=FLOAT(IDATA(I,9))
      DO 60 J=1,4
      DATA(I,J)=FLOAT(IDATA(I,J))-OFSTM(J)
      DATA(I,J+4)=FLOAT(IDATA(I,J+4))
 60   CONTINUE
      DATA(I,8)=FLOAT(I)
      CALL NORMAL(DATA(I,1),DATA(I,2),DATA(I,3),DATA(I,4))
```

```
 50   CONTINUE
C
C       SEED ONE VECTOR INTO THE REFERENCE TABLE
C
      NREF=1
      DO 100 I=1,9
      REF(1,I)=DATA(1,I)
100   CONTINUE
      NREFOLD=NREF
      DO 300 I=1,NVECS
      IF(IIFIX(DATA(I,9)).EQ.NOFFSET) GO TO 300
         DO 150 J=1,4
         X(J)=DATA(I,J)
150      CONTINUE
      CALL CLASS(REF,X,ICLASS,NREF)
      IF(IIFIX(DATA(I,9)).EQ.ICLASS) GO TO 300
         DO 200 J=1,9
         REF(NREF+1,J)=DATA(I,J)
200      CONTINUE
         NREF=NREF+1
300   CONTINUE
      WRITE(1,9001)NREF
      WRITE(6,9001)NREF
      IF(NREFOLD.LT.NREF)GO TO 100
      WRITE(1,9003)
      WRITE(1,9005)(REF(I,8),I=1,NREF)
      WRITE(6,9003)
      WRITE(6,9005)(REF(I,8),I=1,NREF)
C
C
C     CONDENSE NEAREST NEIGHBOR REFERENCE LIST
C
C
      DO 500 I=1,NREF
         DO 350 J=1,9
         TEMP(J)=REF(I,J)
         IF (J.GT.4) GO TO 350
         REF(I,J)=1000.
350      CONTINUE
      DO 400 K=1,NVECS
      IF(IIFIX(DATA(K,9)).EQ.NOFFSET) GO TO 400
         DO 450 L=1,4
         X(L)=DATA(K,L)
450      CONTINUE
      CALL CLASS(REF,X,ICLASS,NREF)
      IF(IIFIX(DATA(K,9)).EQ.ICLASS) GO TO 400
         DO 470 N=1,9
         REF(I,N)=TEMP(N)
470      CONTINUE
         GO TO 500
400   CONTINUE
500   CONTINUE
      JREF=NREF
C
C     WIPE OUT UNNEEDED REFERENCE VECTORS
C
      DO 800 I=1,NREF
      IF(REF(I,1).LT.1000.) GO TO 800
         DO 700 K=I+1,NREF
            DO 600 N=1,9
            REF(K-1,N)=REF(K,N)
```

```
600         CONTINUE
700       CONTINUE
      JREF=JREF-1
800   CONTINUE
      NREF=JREF
      WRITE(1,9002)NREF
      WRITE(1,9004)
      WRITE(1,9005)(REF(I,8),I=1,NREF)
      WRITE(1,9006)TITLE
      WRITE(6,9002)NREF
      WRITE(6,9004)
      WRITE(6,9005)(REF(I,8),I=1,NREF)
      WRITE(6,9006)TITLE
C
C
C       GET UNKNOWN DATA FILE
C
C
      CALL INPUT
      WRITE(1,9007)TITLE
      WRITE(6,9007)TITLE
C
C         ZERO CONFUSION MATRIX
C
      DO 1000 I=1,6
      DO 1000 J=1,6
      CONF(I,J)=0.
1000  CONTINUE
C
C
C          CLASSIFY THE INPUT VECTORS
C
C
      DO 1100 I=1,NVECS
      IF(IDATA(I,9).EQ.NOFFSET) GO TO 1100
        DO 1150 N=1,4
        X(N)=FLOAT(IDATA(I,N))-OFSTM(N)
1150    CONTINUE
      CALL NORMAL(X(1),X(2),X(3),X(4))
      IDCASE=IDATA(I,9)
      CALL CLASS(REF,X,ICLASS,NREF)
      CONF(IDCASE,ICLASS)=CONF(IDCASE,ICLASS)+1.0
1100  CONTINUE
      WRITE(1,9008)((CONF(I,J),J=1,6),I=1,6)
      WRITE(6,9008)((CONF(I,J),J=1,6),I=1,6)
9000  FORMAT(/1X,'NUMBER OF NULL VECTORS = ',I4)
9001  FORMAT(/1X,'NUMBER OF NORMALIZED VECTORS IN REFERENCE TABLE = '
     *       ,I4)
9002  FORMAT(////1X,'REFERENCE TABLE CONDENSED DOWN TO ',I4,' VECTORS')
9003  FORMAT(//1X,'REFERENCE TABLE AFTER ITERATIONS (VECTOR ID #S)')
9004  FORMAT(///1X,'REFERENCE TABLE AFTER CONDENSATION (VECTOR ID #S)')
9005  FORMAT(10(2X,F5.0))
9006  FORMAT(/////1X,'NORMALIZED NEAREST NEIGHBOR LOOK-UP TABLE CREATED
     * WITH FILE ',10A4)
9007  FORMAT(///1X,'WILL NOW CLASSIFY THE UNKNOWN FILE ',10A4)
9008  FORMAT(///19X,'NORMALIZED DATA CLASSIFIED AS',/28X,
     *       '  GRASP    OPEN     FLEX   EXTEND  PRONATE  SUPINATE',/1X,
     *       'INPUT VECTOR CLASS: GRASP    ',6(F6.0,2X),/21X,'OPEN     ',
     *       6(F6.0,2X),/21X,'FLEX     ',6(F6.0,2X),/21X,'EXTEND   ',
     *       6(F6.0,2X),/21X,'PRONATE  ',6(F6.0,2X),/21X,'SUPINATE',
     *       6(F6.0,2X))
```

```
      STOP
      END
C
C       INPUT THE DATA FROM A FILE
C
      SUBROUTINE INPUT
      CHARACTER*40 FILENAME
      LOGICAL ANSWER,FLAGPRINT
      DIMENSION IDATA(999,9),NVECT(26),TITLE(10)
      COMMON IDATA,NVECS,NCASES,NOFFSET,NELEM
      COMMON /TITLE/TITLE
      COMMON /FLAGS/FLAGPRINT
      DATA FLAGPRINT/.FALSE./
C
C       GET FILE NAME AND OPEN FILE
C
100   WRITE (6,1000)
      READ (6,1001) FILENAME
      OPEN (UNIT=2,NAME=FILENAME,READONLY,TYPE='OLD',ERR=110)
      GO TO 130
C
C       FILE OPENING ERROR
C
110   WRITE (6,1002)
120   CLOSE (UNIT=2,ERR=100)
      GO TO 100
C
C       FIND # CASES (1-26)
C
130   WRITE (6,1003)
      READ (6,1004) NCASES
      IF ((NCASES.GT.0).AND.(NCASES.LT.27)) GO TO 140
      WRITE (6,1005)
      GO TO 130
C
C       FIND # VECTORS IN EACH CASE
C
140   NVECS=0
      DO 160 I=1,NCASES
150           WRITE (6,1006) I
              READ (6,1004) NVECT(I)
              IF (NVECT(I).EQ.-99) GO TO 140
              IF ((NVECT(I).LT.1).OR.(NVECT(I).GT.130)) GO TO 150
              NVECS=NVECS+NVECT(I)
160   CONTINUE
      IF (NVECS.LT.999) GO TO 170
      WRITE (6,1007)
      GO TO 140
C
C       ATTEMPT TO READ THE DATA FILE
C
170   READ (2,1008,ERR=180,END=190) ((IDATA(I,J),J=1,8),I=1,NVECS)
      WRITE (6,1009)
      GO TO 200
180   WRITE (6,1010)
      GO TO 120
190   WRITE (6,1011) I
      READ (6,1012) ANSWER
      IF (ANSWER.EQ.'R') GO TO 180
200   CLOSE (UNIT=2,ERR=210)
210   IX=0
```

```
      DO 230 I=1,NCASES
              DO 220 J=1,NVECT(I)
                      IX=IX+1
                      IDATA(IX,9)=I
220           CONTINUE
230   CONTINUE
C
C       FIND OUT # ACTIVE ELEMENTS PER VECTOR
C
240   WRITE (6,1015)
      READ (6,1004) NELEM
      IF ((NELEM.LT.1).OR.(NELEM.GT.8)) GO TO 240
C
C       FIND OFFSET CASE NUMBER
C
      WRITE (6,1013)
250   READ (6,1004) NOFFSET
      IF ((NOFFSET.GT.0).AND.(NOFFSET.LE.NCASES)) GO TO 260
      WRITE (6,1014) NCASES
      GO TO 240
260   WRITE (6,1020)
      READ (6,1022) TITLE
      WRITE (6,1016)
      READ (6,1012) ANSWER
      IF (ANSWER.EQ.'Y') FLAGPRINT=.TRUE.
      RETURN
1000  FORMAT (1X,'INPUT DATA FILE NAME : '$)
1001  FORMAT (A40)
1002  FORMAT (1X,'FILE OPENING ERROR')
1003  FORMAT (1X,'INPUT # OF CASES (0<N<27): '$)
1004  FORMAT (I10)
1005  FORMAT (1X,'# CASES OUTSIDE RANGE')
1006  FORMAT (5X,'CASE #',I2,'  # VECTORS = '$)
1007  FORMAT (1X,'TOO MANY VECTORS (MUST BE <501)')
1008  FORMAT (10X,16Z2)
1009  FORMAT (1X,'DATA READ')
1010  FORMAT (1X,'FILE READING ERROR, FILE CLOSED')
1011  FORMAT (1X,'PREMATURE END OF FILE ON VECTOR ',I3/1X,
     1 'ACCEPT OR RECYCLE (A OR R)?'$)
1012  FORMAT (A1)
1013  FORMAT (1X,'WHICH CASE IS NULL INPUT?'$)
1014  FORMAT (1X,'**MUST BE BETWEEN 1 AND ',I2,'**')
1015  FORMAT (1X,'HOW MANY ELEMENTS ARE ACTIVE?'$)
1016  FORMAT (1X,'OUTPUT OPTIONS (ANSWER Y OR N)'/5X,'DATA PRINTOUT: '$)
1020  FORMAT (1X,'TITLE FOR OUTPUT: '$)
1022  FORMAT (10A4)
      END
C
C
C       FIND THE NEAREST NEIGHBOR IN 'DATA' OF 'X'
C
C
      SUBROUTINE CLASS(DATA,X,ICLASS,NVECS)
      DIMENSION DATA(999,9),X(4)
      D=SQRT((DATA(1,1)-X(1))**2 + (DATA(1,2)-X(2))**2 +
     *       (DATA(1,3)-X(3))**2 + (DATA(1,4)-X(4))**2)
      ICLASS=IIFIX(DATA(1,9))
      DO 100 I=1,NVECS
      TEMPD=SQRT((DATA(I,1)-X(1))**2 + (DATA(I,2)-X(2))**2 +
     *           (DATA(I,3)-X(3))**2 + (DATA(I,4)-X(4))**2)
      IF(TEMPD.LT.D) ICLASS=IIFIX(DATA(I,9))
```

```
      IF(TEMPD.LT.D) D=TEMPD
100   CONTINUE
      RETURN
      END
C
C       NORMALIZE THE VECTOR (W,X,Y,Z)
C
      SUBROUTINE NORMAL(W,X,Y,Z)
      WXYZ=SQRT(W**2+X**2+Y**2+Z**2)
      IF (WXYZ.EQ.0.0) RETURN
      W=W/WXYZ
      X=X/WXYZ
      Y=Y/WXYZ
      Z=Z/WXYZ
      RETURN
      END
```

The vita has been removed from the scanned document

PROSTHESIS CONTROL USING A NEAREST NEIGHBOR ELECTROMYOGRAPHIC PATTERN CLASSIFIER

by

David Charles Dening

(ABSTRACT)

A prosthesis control strategy using a nearest neighbor electromyographic pattern classifier was investigated with both a real time microprocessor-based controller and off-line computational facilities. Four active electrodes for myoelectric signal amplitude detection were interfaced with a microcomputer for data logging and pattern classification. A nearest neighbor algorithm correctly identified arm motions as belonging to one of six pattern classes from 72 percent to 100 percent of the time. There were five vectors for each class in the look-up table.

The nearest neighbor pattern classifier was compared to a minimum error rate Bayes classifier under the assumption that the probability densities were distributed as a multivariate normal distribution. Comparable error rates were obtained with the same data vectors.

A condensed nearest neighbor classifier was constructed to determine what minimum number of vectors was necessary in the look-up table. This minimum number of vectors ranged from two to six for the majority of the classes. Larger numbers of vectors were placed in the look-up table for classes that were more difficult to classify.