AN OBJECT-ORIENTED PATTERN RECOGNITION SYSTEM

by

LAWRENCE MYKEL STOUT

B. A., University of Wisconsin, Madison, 1978

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1987

Approved by:

M. S. P. Lucas
Major Professor

ACKNOWLEDGEMENT

To my wife, Valerie.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A general purpose pattern recognition system is a key component in any comprehensive computer vision process. A software implementation of a pattern recognition system is useful for investigating the properties of various feature extraction and image classification techniques on a wide range of geometric objects. No simple definition of pattern recognition exists since it is a broad field with ill-defined boundaries. However, a primary goal of pattern recognition can be simply stated. This goal is to extract specific information from an input image that can be used by the computer to identify (classify) or understand the image. Pattern recognition is the task of understanding an object from its projected image [1].

Human vision is an automatic process that is often considered independently of intelligence. The field of artificial intelligence (AI) attempts to offer to the computer what the human brain accomplishes for its

vision process. The pattern recognition aspect of computer vision is an area where AI techniques will be of vital importance in the development of intelligent robots and autonomous vehicles. No future goals of a general purpose computer vision process can be imagined without the application of some AI techniques.

Despite all the media attention that AI has received, its major benefits as practical software are not yet realizable. Currently, AI systems are highly specific and largely inflexible. Various definitions of AI exist, but the essence of AI is the search for useful, approximate solutions to very hard problems. The complexity and lack of structure of AI problems implies that there are not absolute solutions to these problems. Consequently, the primary focus of AI research is the search for "adequate" solutions.

An important area of AI research is knowledge representation. To be useful, knowledge must be organized in such a fashion that it is of value to the human user and the computer software. Enclosing this pattern recognition system in a "flavor" structure offers the advantages of object-oriented programming,

and provides a useful method for representing knowledge. The flavor package is a general purpose software tool that can be used independently of this pattern recognition system.

This research was a software project. The software was written on the Electrical and Computer Engineering Department's VAX 11/750 (Digital Equipment Corporation), using a superset of the computer language LISP, called INTERLISP, developed at XEROX Corporation. Portability to XEROX's series of AI workstations is possible.

In PART II, the pattern recognition problem is discussed in detail. A computer vision system model is presented and its components are analyzed. The important algorithms used in the pattern recognition system are also examined, especially those involved in feature extraction and feature classification.

In PART III, the INTERLISP programming environment and object-oriented programming are described. The style of object-oriented programming chosen for this pattern recognition system is referred to as a "flavor"

system. All the flavor functions are analyzed in this section. The last section of this part contains a description of how the pattern recognition system is organized within the flavor system.

PART IV contains an analysis of system performance. Timing functions are analyzed to demonstrate how CPU times vary with respect to the area of an image being analyzed. Next, the invariance properties of the two feature extraction techniques are examined. The last section of this part contains results from two tests run on the pattern recognition system. Conclusions are drawn from the results of these tests.

Appendix I contains output from a sample session of the pattern recognition system. It shows the user how to get into the INTERLISP environment and run the software. Appendix II contains program listings for all the software used in the pattern recognition system, organized by file name.

II. PATTERN RECOGNITION

A. BACKGROUND

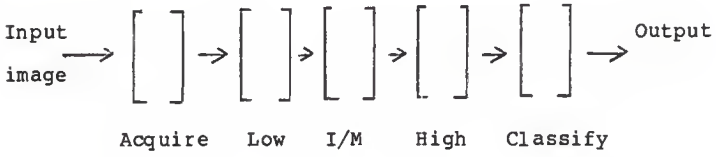
The advent of sufficiently fast computers and digital sensing devices has greatly expanded the efforts to create useful computer vision systems. The pattern recognition portion of the computer vision process begins after an image has been input and stored in the computer in digital form. This preliminary stage can be likened to human perception. This system assumes that the input images are stored in 2-dimensional arrays (matrices). All image analyses are based on only two dimensions. The number of elements in the array determines the resolution of the images. The number of bits available to any array element determines the gray-scale each element can have. The smallest element of an image corresponds to a single element in the array and is called a picture element or pixel. This system stores images in 32x32 pixel arrays. This number can easily be changed by the software developer. The gray-scale in this system is constrained to a binary one, where each pixel represents either an image pixel or a background pixel,

depending on whether it is inside or outside the region occupied by the object. The main function of this pattern recognition system is to provide a statistical analysis of the input image, usually concluding with a classification of the image.

B. GENERAL MODEL

A general model for a computer vision system is contained below.

Figure 1. Computer vision system model.



The five principle activities of this vision system are outlined below, and will be more thoroughly discussed in Sections C through H of PART II.

1. Data acquisition is the conversion of analog to digital information and storage of that information in data arrays.

2. Low-level image processing consists of noise reduction and various image enhancement techniques.
3. Intermediate-level image processing contains the techniques used to extract information from the image data arrays, usually statistical techniques.
4. High-level image processing seeks to find relations between individual patterns in an array, eg. image and scene understanding.
5. The classifier consists of decision-making algorithms and output.

This research assumes that activity 1 and parts of 2 have been previously accomplished by some combination of software and hardware. Activity 4 is not included in this pattern recognition system. Activity 5 consists of a classifier which attempts to match the unknown input images with a library of pre-processed images to identify what image class they belong to. Another option, which does not involve any library images, is also available and is called clustering.

The goal of each activity is to simplify the data from the previous one, so that there are fewer total pieces of information; but each individual piece

contains more useful or meaningful information. The type of information required is dependent on the type of problem being solved.

C. DATA ACQUISITION

Data acquisition is the first step in the computer vision process. Various types of digitizers are available, including high quality TV camera digitizers. A variety of sensors, including proximity and tactile ones, are also being used, especially for robotic applications. Faster A/D converters are being developed with the goal of real-time image acquisition.

Problems with various vision systems have often revolved around the camera and the video signal standards. These standards were developed for human eyes and are not always appropriate for computer vision. Human vision tends to be insensitive to absolute light intensity, slow variation in intensity and spatial accuracy. The eyes are well adapted to detection of local intensity gradients, while global differences in intensity are perceived only with high levels of contrast [2].

D. LOW-LEVEL IMAGE PROCESSING

Low-level image processing begins with techniques used to improve the overall quality of the image stored in the computer. These involve methods to decrease the noise in the images. For images which have been degraded or blurred, there are techniques to recover some of the original clarity using inverse or least-square filtering [3]. Image enhancement techniques are available which can alter an image so that it is more suitable than the original one for some specific application. These include gray-level transformations and histogram modifications [4]. Image smoothing techniques reduce errors introduced into the image by poor samplings or noisy transmission channels. These include neighborhood averaging and low-pass filtering [5]. Image sharpening techniques are used to highlight the edges of an image. Techniques include differentiation and high-pass filtering [6]. To obtain a binary image, various thresholding techniques are available. The functions in this pattern recognition system can begin analyzing the image from a binary array.

The functions in this section of the pattern

recognition system find and trace edges of images, calculate the area and center of area for each image, and count the number of holes in them.

After the arrays to be analyzed have been stored, the function Find.Image is called. The program listing for Find.Image is contained in Appendix II under the file name, TRACE. This function searches the input array row by row for one image pixel that has not been previously analyzed. This system assumes that background pixels will be represented by a period and image pixels will be represented by an "x". Since more than one image per array can be analyzed, something must be done to images that have already been processed. After the image has been analyzed, a function called Replace.Image.Points (in file CENTER) replaces all the image coordinate points with a sequence number. This sequence number is unique for each of the images in the array. Find.Image skips over any coordinate points that have integer values. When Find.Image is completed, it returns either the row and column coordinates of an edge point for a new image or a string containing an error message, indicating that there are no more new images in the array.

The next function called is Follow.Edge (file TRACE). Given the starting image edge point returned by Find.Image, this function traces the edge of the image until the starting point is reached again. For images which are not closed, like the letter "X", the algorithm traces over some image coordinate points more than once, but each image point occurs only once in the final edge list. At each image edge point, the search algorithm has an ordered sequence of directions in which to search for the next edge point. This list of directions is based on the previous direction in which an edge point was found. By adapting the list of directions in this fashion, the algorithm can guarantee that it will find the image with the largest external boundary. Two simple examples of this process are given below.

Figure 2. Edge tracing examples.

```
. . . . . . . . . . .
. 2 3 4 . . . 6 y' . .
. 1 x 5 . . . 5 y 1 .
. . x' 6 . . . 4 3 2 .
```

x' and y' represent the previous image point found, x and y are the present image points just located, and the numbers from 1 to 6 represent the sequence of directions which will be searched for the next edge point. The seventh direction will be x' or y'. The eighth direction is not considered since it was checked when either x' or y' was the present point (previous step).

Follow.Edge returns several important variables: the list of edge points, the number of edge points (perimeter), a list containing the Freeman chain code for the edge points and a boolean flag. The Freeman chain code provides an efficient way to store the image edge points [7]. Each of the eight possible directions from a coordinate point is assigned a unique integer.

Figure 3. Freeman code directions.

```
. . . . .
. 3 2 1 .
. 4 x 0 .
. 5 6 7 .
. . . . .
```

When an image point is found, the direction from the previous point to the present point is placed onto this list. Given the starting image coordinate point and the Freeman code, an identical image can be traced later. This code can also be used in future enhancements to search for straight edges and corners. The boolean flag is set to TRUE the first time an edge point is repeated. This flag is used to warn the user that the present image may not be closed. For example, the flag would be TRUE for the letter "X".

The next function called is Shell.Sort.Edge (file CENTER). This function sorts the edge coordinate points into numerical order, by row then by column, eg. (1 1) (1 2) (2 1) (2 2), where the order of these coordinates is row then column. The sorted list is returned in a one-dimensional array. This sorting is done so that a later function, Find.Image.Points, can more efficiently search for all the coordinate points in the interior of the image. Sorting the edge points allows Find.Image.Points to step through the edge array and know when to search between two edge points for interior points.

The next function called is Find.Image.Points. The purpose of this function is to search in an orderly fashion for all the coordinate points in the interior of an image. The algorithm takes two adjacent edge points from the sorted edge array, (these points are not necessarily physically adjacent) and checks to see if they are on the same row. If they are not, then one knows that there cannot be any interior points between them. If they are on the same row and not physically adjacent, then there is the possibility that there are interior points between them. A sequential search of all the points between the two edge points is made and all interior image points are identified. This row search is done column by column, from the smallest to the largest column. In addition to the list of interior points, lists containing the row and column numbers for each image point are kept. These lists will be used later to calculate the center of area for the image. The row list will contain a series of sublists, each containing the row number and the number of image points in that row. The column list will be less formatted because the search is organized on a row-by-row basis. The next row searched will always be one more numerically than the previous row, but column

numbers can vary, depending on the overall shape of the image.

While searching between two non-adjacent edge points in the same row, the algorithm is also keeping track of potential holes in the interior of the image. The first time a background pixel is encountered in the row search, the coordinate of the last image pixel before the hole coordinate is placed onto a list containing potential hole edge points. No other image pixels will be entered into the hole list for that row unless another image pixel is found between the two background pixels. None of these potential hole edge points may turn out to be actual hole edge points. Later, the function Find.No.Of.Holes will be called to calculate the number of actual holes in the image. At the conclusion of Find.Image.Points, the lists containing all the interior points, row and column quantities and potential hole edge points are returned to the calling routine.

The next function called is Calculate.Area (file SET_UP_SYSTEM). It adds the number of image edge points to the number of interior image points to obtain

the area of the image.

The next one called is Col.Sums. This function converts the list of image columns from Find.Image.Points into an ordered list similar to the row list returned from Find.Image.Points.

Calculate.Center.Area is the next function called (file SET_UP_SYSTEM). This function calculates the center of area for the image. The area and the ordered row and column lists are used to obtain this coordinate value.

Following that, the function Find.No.Of.Holes is called (file CENTER). This function is called only if the potential hole edge list returned from Find.Image.Points had at least one coordinate point in it. First, all the potential hole edge points which are also contained in the list of image edge points are removed. Then the function uses the remaining points as starting points in an edge search similar to the one contained in Follow.Edge. The main difference is that the minimum size of an edge that bounds the image is desired, instead of the maximum, as was the case in

Follow.Edge. The search algorithms are similar, except the ordered list of directions to be searched is shifted. After a hole has been successfully traced, all the coordinate points contained in that hole trace are removed from the original potential hole edge list. Then the process is repeated until there are no more coordinate points in the list of potential hole edge points. The function returns a list containing sublists, each with a hole number and the list of image coordinates surrounding that hole. The only restriction on this process is that all holes beginning near the left edge of an image must be separated from an edge pixel by at least one interior image pixel. No hole on the left edge of an image can have an edge pixel bounding it.

The last function called in the low-level image processing section is Replace.Image.Points (file CENTER). This function replaces all the image pixels in one image with a unique sequence number. This allows the user to quickly spot the image on a printout of the array, and lets the function Find.Image know that that particular image has already been processed.

E. INTERMEDIATE-LEVEL IMAGE PROCESSING

Intermediate-level image processing contains the algorithms used to extract feature measurements from the image. These measurements are later used to classify or cluster a set of input images. In general, most feature extraction techniques can be divided into two categories: decision-theoretic (statistical) or syntactic (linguistic). The syntactic approach will be discussed more thoroughly in PART II, Section F. The following paragraphs will discuss the invariances desired and examine the two feature extraction techniques chosen for this system.

The goal of statistical pattern recognition is to generate a finite set of measurements (feature vector) which characterize the image being analyzed. These measurements should be invariant to certain transformations.

Features are geometrical properties of an image which, when combined, lead to its identification [8]. Ideally, these features should exhibit two of the three isometries (invariances) of the mathematical entity known as an affine geometry group [9]. The first

property of this group is translational (including rotational) invariance. Two geometric objects exhibiting this type of isometry are called congruent. The second property is size (scale) invariance. Geometric objects with this isometry retain the concepts of angle and length ratio, but lose the concept of absolute length. The last invariance of the affine geometry group is one that is not desired in this pattern recognition system. This is invariance to shearing transformations, where the concept of angle is lost. In an affine geometry group, all parallelograms are equivalent.

The two feature extraction techniques chosen for this pattern recognition system exhibit the first two invariances. Although the invariances are not fully obeyed, the techniques are relatively insensitive to translational, rotational and scale transformations. The output of each feature extraction technique is an N-dimensional feature vector. Each feature vector can be mapped onto a single location in an N-dimensional feature space. The size of each of these dimensions is determined by the range of values of the corresponding feature vector element. By judicious choice of the

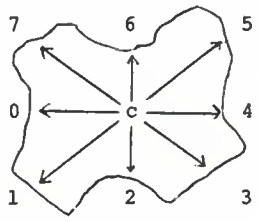
feature extraction technique, points from "similar" geometric objects will be in close proximity to each other in the N-dimensional feature space. Similarity in this pattern recognition system refers to overall shape. Correspondingly, geometric objects which are not similar should be widely separated from each other in the feature space. Many different measurements are available to calculate distances between points in the feature space. These will be discussed more thoroughly in PART II, Section J.

In general, the ability to classify geometric objects into M different categories (classes) is based on a division of the N-dimensional feature space into M partitions. The ability of the feature extraction technique to cluster similar geometric objects and separate dissimilar ones is a key component in the classification process. The size of these separations provides measurements of the overall quality of the feature extraction technique. The quality of the feature extraction technique is a fundamental limitation to the ability of the classifier to correctly identify geometric objects. Very little general mathematical theory is available to help in

choosing feature extraction techniques. The two techniques chosen for this pattern recognition system are the autoregressive model and the Zernicke moment techniques.

An autoregressive model is a parametric equation that expresses each sample of an ordered set of data samples as a linear combination of a specified number of previous samples from the set plus an error term [10]. The feature vector for this technique is obtained by solving a system of linear equations containing an ordered sequence of terms approximating the shape of the geometric object. These terms are obtained from prior knowledge of the edge points and the center of area for the geometric object. From this information, a series of N angularly equi-spaced rays are projected from the center of area outwards through the object boundary (Figure 4, where N = 8).

Figure 4. Image with angularly equi-spaced rays.



The algorithm contained in function Auto.Reg.Edge (file REG) checks each coordinate point in the list of edge points to find which ones lie on one of these rays or are closest to a ray. For convex objects, there will be only one edge point that is closest to each ray (Figure 4). For concave objects, there may be more than one edge point per ray (each ray may intersect the boundary of the image more than once), depending on how convoluted the object is. The distance from the center of area to the edge point closest to one of these rays is measured. An ordered set of these distances is used in the autoregressive model as an approximate shape descriptor. This ordered set of distances can be viewed as a periodic time series. The specific equation for the model is

$$(1) \quad R_t = A + \sum_{j=1}^{m} D_j R_{t-j} + B^{1/2} W_t$$

where $t = 1, 2, \ldots, M$ and $M > N$ [11].

The definitions of the parameters in equation (1) are:

R_t = current ray intersection length

R_{t-j} = ray intersection length detected j rays before the current R term (collectively called the lag terms)

D terms = unknown lag coefficients to be estimated from the observed time series

m = model order

$B^{1/2}$ = unknown constant to be estimated

$B^{1/2} W_t$ = current error, residual noise

A = unknown constant to be estimated

$\{W_t\}$ = a random sequence of independent, zero-mean samples with unit variance.

Since the variance of W_t is 1, the B term transforms the unit variance random variable W_t into a random variable of variance B.

The D terms are estimated from equation (1) by fitting the model to the the ordered sequence of ray intersection lengths (R terms), using a least-squares method to minimize the B term, which is a measure of

object shape noise [10]. The system of equations is

$$(2)\quad \begin{bmatrix} D_1 \\ D_2 \\ \vdots \\ D_m \end{bmatrix} = \begin{bmatrix} \sum_{t=1}^{N} R^2_{t-1} & \cdots & \sum R_{t-1}R_{t-m} \\ \sum R_{t-2}R_{t-1} & \cdots & \sum R_{t-2}R_{t-m} \\ \vdots & & \vdots \\ \sum R_{t-m}R_{t-1} & \cdots & \sum_{t=1}^{N} R^2_{t-m} \end{bmatrix}^{-1} \begin{bmatrix} \sum R_{t-1}R_t \\ \sum_{t=1}^{N} R_{t-2}R_t \\ \vdots \\ \sum R_{t-m}R_t \end{bmatrix}$$

The parameter A is found after equation (2) is solved, using the equation

$$(3)\quad A = R' * \left[1 - \sum_{j=1}^{m} D_j\right]$$

where R' is the mean ray intersection length.

The B terms can be estimated using

$$(4)\quad B = 1/N \sum_{t=1}^{N} \left[R_t - A - \sum_{j=1}^{m} D_j R_{t-j}\right]^2$$

where N is the number of ray intersections.

Since A is proportional to the mean ray intersection length, it is a measure of the overall size of the object. The larger an image is, the larger its value

for A will be. The B term is related to the boundary noise of the object, since its value is calculated from all the ray intersection lengths. The closer each ray length is to its neighbor's length, the smaller B will be (a circle will have a very small B). Therefore $A/B^{1/2}$ can be used as a measure of the object's shape signal-to-noise ratio. The D coefficients simulate the correlated boundary variations and can be used as shape descriptors.

Model parameters will remain approximately the same for scaled or rotated images. Rotation will only shift the ordered sequence of ray intersection lengths so that it starts at a different point. Scale changes will affect the absolute lengths of the rays but not their relative lengths. The primary differences will be due to the digital nature of the images. The only two parameters which are sensitive to size, A and B, are combined to provide a measure that is much more scale invariant. The output feature vector for the autoregressive technique is the set of D lag coefficients and the $A/B^{1/2}$ term.

The algorithm used to obtain the feature vector for

the autoregressive technique is contained in file REG. The top level routine is named Top.Auto.Regressive. The user is restricted to one of three choices for the two key parameters in the model. These two parameters are the number of rays that will be drawn from the center of area and the number of lag terms that will be used to determine the overall model parameters. The three choices are 1 lag component and 16 different rays, or 2 lag components and either 16 or 32 different rays. The restriction to 2 or less lag components was made arbitrarily. The restriction on the number of different rays allows for their slope values to be pre-computed.

The function Top.Auto.Regressive first calls Auto.Reg.Edge (file REG). Auto.Reg.Edge searches the edge list for ray intersection points or the closest edge points to a ray. First, the slope values for the rays are read in from a global variable. These slope values have already been calculated and are for rays in the first quadrant, from slope 0 to vertical. Since the number of rays is evenly divisible by 4, only one quadrant of values is required.

After these values are input, a loop is entered in which each edge point is checked against the list of ray slopes. Next, the numerator and denominator values for the slope, from the center of area to the edge point being considered, are calculated. The primary purpose of this function is to determine when an edge point crosses over or lands on one of the rays. The quadrant location and the location of the two rays whose slopes bound the slope of the present point are found. If the quadrant is different or the bounding ray slopes have changed from the previous point to the present point, a ray was crossed and an approximation to a ray intersection was found. An intersection is also found when the present edge point lands on a ray slope.

After an intersection has been found, its value will be placed into the list only if it differs from the previous intersection placed into the list. This is necessary for the case when an object has a straight line boundary. In such cases, certain orientations of the object may result in several adjacent edge points being mapped onto one ray slope intersection. This causes the resulting series of intersection lengths to

be longer than the series for the same object with a different orientation, biasing the model estimation. If a new intersection has been crossed, then the length from the center of area to the edge point is calculated. This length is placed into a list. After all the edge points have been analyzed, Auto.Reg.Edge returns the ordered list containing the intersection lengths and the number of items in this list. A graph of these values for a sample object (Figure 5) are contained in Figure 6.

Top.Auto.Regressive then calls the function Calculate.Sums. This function calculates the required elements of the matrix in equation (2). These terms are obtained by summing the appropriate combinations of the intersection lengths returned from Auto.Reg.Edge.

The last function called by Top.Auto.Regressive is Solve.Auto.Matrix. This function places the terms calculated in Calculate.Sums in the correct order in the matrix and sets up the solution array. The system of equations is then solved and the A and B parameters are calculated from the D terms. Top.Auto.Regressive returns the feature vector containing the m D

parameters and the $A/B^{1/2}$ term.

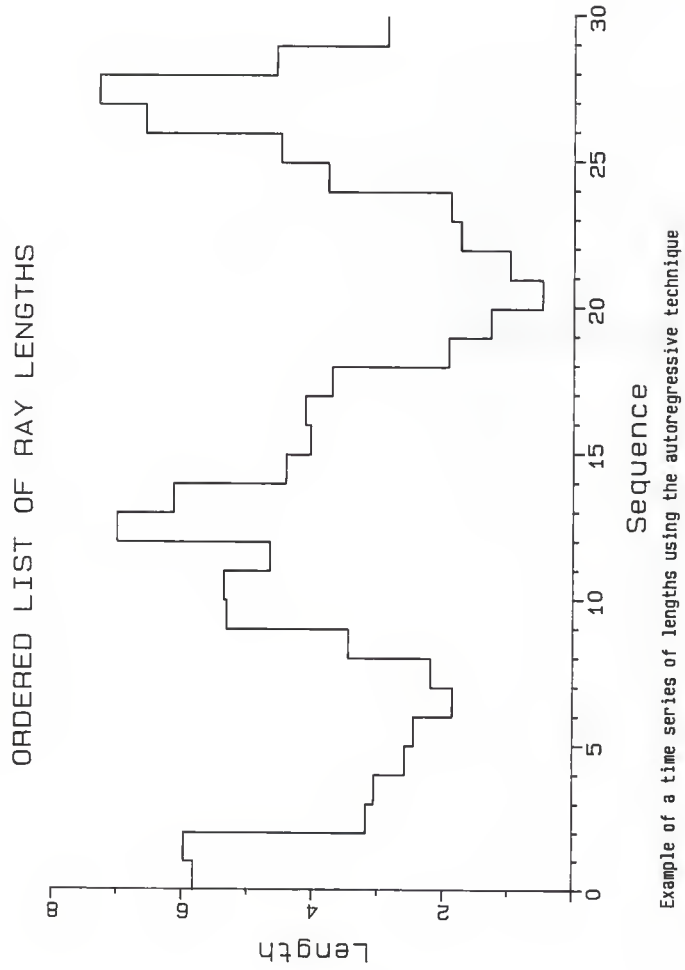
Figure 5 represents an array containing a complex image. This image is analyzed using the autoregressive feature extraction technique. The list of ray intersection lengths obtained from the autoregressive algorithm is then plotted to show what the function accomplishes. The plot is on the following page. Figure 5 shows only a portion of the 32x32 data arrays and is numbered for convenience. The area of this image is 68 and the center of area is (9.84 14.54), with the row number first, then column number. The approximate location of the center is marked with a "c". Image pixels are marked with an "x". The ray search begins at edge coordinate (5 16), which was the last edge pixel found in the edge search, and proceeds counter-clockwise.

Figure 5. Sample image.

columns

rows	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
4	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x	x	.
5	.	.	.	.	.	.	x	x	x	x	.	.	.	x	x	x	.
6	.	.	.	.	.	.	.	.	x	x	.	.	x	x	x	.	.
7	.	.	.	.	.	.	.	.	x	x	x	x	x	x	.	.	.
8	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x	.	.
9	.	.	.	.	.	.	.	.	.	.	x	x	x	x	x	.	.
10	.	.	.	.	.	.	.	.	.	.	.	.	c	x	x	x	.
11	.	.	.	.	.	.	.	.	x	x	x	x	x	x	.	.	.
12	.	.	.	.	.	.	.	.	x	x	x	x	x	.	.	.	.
13	.	.	.	.	.	.	.	x	x	x	x	x	x	x	.	x	.
14	.	.	.	.	.	.	.	.	.	.	x	x	x	.	.	x	.
15	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x	x	.
16	.	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x	.
17	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
18	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Figure 6. Ordered List of Ray Lengths



The second feature extraction technique used by this pattern recognition system is based on Zernicke moment invariants [12]. The first published paper on the use of image moment invariants for pattern recognition appeared in 1962 [13]. From the theory of statistical moments, a uniqueness theorem states that if f(x,y) is piecewise continuous and has nonzero values only in a finite part of the x-y plane, then moments of all orders exist, and the moment sequence (M_{pq}) is uniquely determined by f(x,y) and, conversely (M_{pq}) uniquely determines f(x,y) [14]. The problem is how to avoid having to calculate high orders of moments, yet still retain enough information to correctly classify an image.

In the analog world, given a continuous function of two variables x and y, any moment of order (p+q) can be described by the following equation

$$(5) \quad M_{pq} = \iint_R x^p y^q f(x,y)\, dx\, dy$$

where R is the region within which the object lies. The digital approximation to (5) is

$$(6) \quad M_{pq} = \sum_{x,y \text{ in } R} x^p y^q f(x,y)$$

If the images are restricted to binary ones, as in this system, f(x,y) is restricted to 1 or 0. In this case, equation (6) can be simplified using a discrete version of Green's Theorem [15]. This theorem relates a double integral (double summation) over a bound area to a line integral (single summation) over the boundary of the object. The application of this theorem to the moment equation yields the following;

$$(7) \quad M_{pq} = (1/n+1) \sum_{i=1}^{E} (x_i^{p+1} y_i^{q}) s_i$$

where $s_i = \text{sign}\ (y_i - y_{(i-1) \bmod E})$,
E is the number of edge points in the boundary and sign is the sign operation, returning either +1 or -1 [16].

This approximation yields a significant savings in the total number of mathematical operations required to obtain the moments. The long version for obtaining the moments, equation (5), is still available to the user.

The next operation on the moments is to centralize them. This operation makes them invariant to translation. The discrete equation for centralizing binary moments is

$$(8)\quad Mu_{pq} = \sum_{x,y \text{ in } R} (x-x')^p (y-y')^q$$

where x' and y' are the mean x and y values for the image.

To obtain scale invariance, the centralized moments are normalized. The general equation for normalizing the centralized moments is

$$(9)\quad Phi_{pq} = Mu_{pq}/Mu^a_{00}$$

where $a = 1 + [(p+q)/2]$ for $p+q > 1$.

Rotational invariance is difficult to achieve. Hu [13] derived rotational invariance by combining specific normalized, centralized moments. Another way to do this is to use the circular polynomials of Zernicke. These polynomials were developed by Zernicke while studying light diffraction from small aberrations in 1934 [17]. His polynomials have the property of being orthogonal over the interior of the unit circle and, unlike most others with this orthogonality, also contain basic invariant properties including rotational invariance. The polynomials have the following form, in two real variables,

(10) $V_{nl}(x,y) = R_{nl}(p)\exp(ilo)$

These functions are complete over the unit circle and satisfy the following equation

(11) $\iint dxdy[V_{nl}(x,y)]^{*}V_{mk}(x,y) = d_{mn}d_{kl}/(n+1)$

where the integration is over the unit circle $(x^2 + y^2) <= 1$

The last two terms in the numerator of the right half of the equation are Kronecker deltas.

Because of the nature of geometric objects, only the real portions of equation (10) are considered. By expansion, an equation is derived that can be used to calculate the Zernicke moments from the appropriate combinations of normalized and centralized moments. A complete description of the moments is contained in the comments of function Get.Zernicke.Moments in file MOMENT. A thorough derivation of this problem is contained in reference [17]. A maximum of 11 different Zernicke moments were used (up through fourth order moments) in this pattern recognition system.

The algorithms used to calculate the Zernicke moments are contained in the file MOMENT. The first

function called is Calculate.Zernicke.Moments. This function calls Calculate.Moments, which calculates the moments up through the fourth order from the list of edge points. Then, Central.Moments is called to normalize and centralize the moments. Finally, Get.Zernicke.Moments is called to calculate the Zernicke moments from the normalized and centralized ones. If the user desires to calculate the moments using both edge and interior points, then a different top function is called, Calculate.Zernicke.Long. This function calls a slightly different function, Calculate.Real.Moments, to obtain the moments from all the image points. After that, the other two functions described above are called. The output from either of these two routines is a 2, 6 or 11 element feature vector containing the Zernicke moments.

F. HIGH-LEVEL IMAGE PROCESSING

For some applications, the feature vector obtained from intermediate statistical processing is inadequate for thoroughly analyzing the image or scene. In 3-dimensional image analysis, high-level processing is especially important. What is most often required in such an analysis is a hierarchical description of the

scene. Structural understanding of a scene is not something most intermediate-level processing can accomplish. Syntactic (linguistic) analysis is much better suited to accomplishing the goals of high-level processing.

At its most basic level, a syntactic language consists of a set of symbols (alphabet) and a corresponding set of statements (production rules) that detail how the symbols can be manipulated in order to create higher order structures [18]. The symbols can be letters or simple geometric shapes. If one uses simple geometric shapes, the production rules can describe how the shapes will be put together to create more complicated geometric objects. By looking at the ordered sequence of symbols that was strung together (hierarchy), one has the basis for a pattern recognition system. After a new image is input, the algorithm (parser) tries to construct the same image from basic components (alphabet) using the production rules. Very complicated scenes can be analyzed using this technique. For simple geometric objects most intermediate-level statistical processes are faster and more efficient.

G. CLASSIFIER

The classifier is the last component in the computer vision process. It was introduced in PART II, Section E. Classification includes the decision-making algorithm that decides which image class the unknown object belongs to. The central assumption underlying the classification process is that a set of N-dimensional feature vectors from one type of geometric object (one image class) will be points in close proximity to each other in the N-dimensional feature space, based on some type of distance measure. Correspondingly, feature vectors from different image classes should be points widely separated in the feature space. A slight geometric distortion in an object from one image class should result in a small physical displacement of that image point relative to other points in the same class [19].

The N-dimensional feature space is partitioned into M sections for a classifier with M different image classes. Ideally, there should not be any overlap of partitions. Overlapping regions require more sophisticated classification techniques in order to separate the image classes.

The number of classes in a classification process is determined only by the particular type of problem being solved. Large numbers of classes should be avoided because each must be checked when the classifier is being run. This pattern recognition system allows the user to dynamically adapt the classifier by adding new classes or adding new images to previously defined classes.

Once the feature space has been partitioned, an appropriate distance measuring algorithm must be chosen. For a system with small feature element variability within each class and wide separation between different image classes, a simple linear Euclidean distance measure can be used. This system contains two metrics. One is the Minkowski metric of order s, where the the differences between the feature elements in the two feature vectors are raised to the s power and summed. The final result is raised to the 1/s power (s in this system is restricted to 1, 2 or 3). The other metric is the Chebychev, where the distance between two feature vectors is chosen to be the largest of the differences between the individual elements in the feature vector.

Next, the decision-making portion of the classifier examines the distances between the unknown image and each of the image classes and selects class membership based on the smallest distance. These distance metrics must be used cautiously because it is not always accurate to say that the farther away two points are in the feature space, the more dissimilar their corresponding geometric shapes are [20].

The function that contains the above metrics is Distance.Classifier in the file CLASSIFY. In this function, the distance (user-selected metric) between the unknown feature vector and one feature vector from each image class is calculated. An array containing these distances is returned for later use by the decision-making function Classify.Decision (file CLASSIFY). Another set of measurements is also calculated in Distance.Classifier. This set contains variance measurements for the distances between feature vectors. The sum of the variances for the differences used to calculate the final Euclidean distance is placed into an array, with one variance per image class. The user can use this measure to decide between two distances that are very close together.

Classify.Decision is called after Distance.Classifier is completed. This function searches the distance array for the two smallest distances and returns the corresponding array sequence numbers. The ratio of the two closest distances is also returned, giving the user an idea of the separation of the two closest image classes. The ratio of the same two array elements' variances is also returned.

A more sophisticated classifier is available to the user in this pattern recognition system. This technique transforms the N-dimensional feature space into a new N-dimensional feature space where the within-class variability has been reduced. The between-class distances are not greatly affected. A simple transformation can be obtained by weighting the individual elements in a feature vector differently. This is conceptually appealing, since all the elements in a feature vector rarely contribute equally to the overall image classification.

To obtain smaller within-class distances for those image classes with more than one feature vector, the

statistical variance of each element in the feature vector is calculated. Feature elements are then weighted inversely proportional to their variances. Therefore, a feature element from one image class which does not vary much, will be weighted larger than an element whose value is not as constant. Ideally, if an element remains constant, its weight should be infinite and all the rest of the weights should be zero. To test for class membership, only the one constant element needs to be checked. The problem with this approach is that the resulting classifier is often less efficient than the simple Distance.Classifier discussed previously, because only one element is used to determine class membership. To remedy this situation, any element with zero variance is arbitrarily reassigned a variance 20 times smaller than the next smallest variance. This allows all elements in the feature vector to be used when calculating distances. For a non-adaptive classifier of this type, the individual element weights for each class feature vector need to be calculated only once.

Mathematically, this technique yields feature element weights that represent the solution to the

minimization problem of the within-class distances [21]. A further constraint on the weights is added to disallow the transformation that merely shrinks the entire N-dimensional feature space. This shrinkage satisfies the overall criteria of decreasing the within-class distances but nothing has been gained. The equation to be minimized is

$$(12) \quad D^2 = \sum_{n=1}^{N} W_{nn}^2 (A_n - B_n)^2$$

where D is the distance, N is the number of feature vectors in that class, and A and B represent two feature vectors in the class.

This distance is to be minimized, not only for A and B, but for all images in the class. The added constraint is

$$(13) \quad \prod_{n=1}^{N} W_{nn} = 1$$

A complete derivation of this problem is outlined by Sebestyen [21]. The distance in the transformed N-dimensional feature space between point p in the unknown image and one class of images F is

$$(14)\quad D = \left(\prod_{p=1}^{N} a_p\right)^{2/N} \left[\sum_{n=1}^{N} (\{p_n - f'_n\}/a_n)^2 + N\right]$$

where a is the variance of an element, N is the number of elements in the feature vector, p is the unknown feature vector and f' is the sample mean feature vector.

The algorithm used to calculate these weights and distances is contained in function Weighted.Classifier in file CLASSIFY. Weighted.Classifier calls Weight.Library to calculate the variances of the feature elements for each class. Weight.Library returns a 1-dimensional array containing the means and variances for the feature elements in one image class. Weighted.Classifier calls Weight.Library once per image class and places all the return arrays into a 2-dimensional array. At the end of the function, the weights for each feature element in each class are calculated from the appropriate means and variances. The distance from the unknown feature vector to each image class is calculated (15) and returned. After the distances are calculated in Weighted.Classifier, Classify.Decision is called to decide on class membership. This is the same routine that was used

after Distance.Classifier was called. All the same quantities are calculated in Distance.Classifier as before except for the ratio of the distance variances.

The library of images used to partition the N-dimensional feature space is an important aspect of the classifier. Several functions in this pattern recognition system deal specifically with the image library. Both Distance.Classifier and Weighted.Classifier require that the library of images already be formatted. This means that the feature vectors for the library images must already have been calculated and placed into lists according to their class membership. Formatted library files can be loaded to and from storage using the two functions Create.Library.From.File and Store.Library.To.File. These are contained in the file SET_UP_SYSTEM. If no library files are available, or the user wants to start a new one, an adaptive classifier can be chosen. Using the function Add.To.Library (file SET_UP_SYSTEM), the user is prompted for the name of the image class. The function then places the new feature vector and new name into the formatted library. The same process is used to add an additional feature vector to an already

existing image class. The user has a lot of flexibility in designing the desired type of library and classifier.

The last function available to the user in the classifier is one that calculates four measures of classifier quality. All four measures are based on differences between within-class distances and between-class distances [22]. This function is called Quality.Of.Classifier (file CLASSIFY). Each of these measures requires the calculation of two matrices: the within-class scatter matrix and the between-class scatter matrix. Both are square matrices with dimensions equal to the number of elements in the feature vector. The between-class matrix can be represented as

$$(15) \quad B = \sum_{i=1}^{c} P_i (m_i - m)(m_i - m)^T$$

where c is the number of different image classes, P_i is the probability of occurrence of that particular image class (all probabilities in this system are assumed to be equal), m_i is a vector containing the mean feature elements for the ith

image class, and m is a vector containing the mixture sample mean (calculated from all the feature vectors from all the image classes).

The within-class matrix can be represented as

$$(16)\quad W = \sum_{i=1}^{c} P_i/n_i \sum_{k=1}^{n_i} (e_{ik} - m_i)(e_{ik} - m_i)^T$$

where c, P_i, and m_i are the same as in equation (15), n_i is the number of features in the feature vectors and e_{ik} is the kth feature vector in the ith image class.

The first, and simplest, quality measure (using the B and W matrices described above) is

$$(17)\quad Q_1 = \text{TRACE } (B + W)$$

where TRACE is the matrix operation of that name.

This is the only measure that can be calculated for image classes containing only one feature vector (W equals the zero matrix). The second measure is

$$(18)\quad Q_2 = \text{TRACE } (B)/\text{TRACE } (W)$$

This measure is a more realistic measure of classifier quality than the previous one because it is the ratio

of the two matrices' traces instead of the sum. It still does not account for the effect on the distance between classes caused by the correlation of feature vector elements. To avoid this problem, W can be altered by a suitable transformation U, so that the average covariance matrix of the transformed vector is the identity matrix (I) [23];

(19) $U^T W U = I$ therefore $U = W^{-1/2}$

The third measure of classifier quality is then

(20) Q_3 = TRACE $(W^{-1}B)$

The last, and most comprehensive measure of classifier quality is

(21) Q_4 = DETERMINANT (B + W)/ DETERMINANT (W)

Q_4 has the advantage over Q_3 in that it is less likely to select a set of features that allows good separation between only a few of the total number of classes, at the expense of all other class separations [24]. (All the matrix operations required to solve these equations are contained in the file MATRIX.) As the quality of the classifier increases, so should the absolute value of all four of these measures. In an adaptive system,

the user can use these measures to see how much the classifier is improving.

H. CLUSTER ANALYSIS

Cluster analysis is actually a form of classification that can be used when no library images are available or desired. Clustering is a technique used to group a set of feature vectors by a distance metric, and is a well-known analytical tool in statistics. Clustering can be used to test hypotheses concerning similarities and differences between groups of images; it is a tool for investigation.

The type of clustering available in this pattern recognition system is hierarchical clustering. The algorithm is contained in the function Clustering (file CLASSIFY). Before this function is called, the user must choose one of the two available feature extraction techniques: autoregressive or Zernicke moments. Then the user must choose the distance metric to be used in function Distance.Classifier (file CLASSIFY). The available ones are the Minkowski metrics of order 1, 2 or 3, or the Chebychev metric.

The main loop in Clustering continues until all the input feature vectors have been combined into one cluster. In each loop iteration, the distance between each feature vector and all other feature vectors is measured, and this value is stored in a 2-dimensional array. After all these distances have been calculated, the smallest one is located. The two feature vectors that correspond to this distance are then averaged and this new averaged vector replaces the first of the two vectors. The other feature vector is eliminated from the list. Each iteration of the outer loop reduces the number of feature vectors by one. Averaging is one of the simplest methods of combining the two feature vectors. Other more sophisticated techniques are possible [25].

The function Clustering returns an ordered list of sublists, each containing the pair of sequence numbers for the feature vectors that were clustered and the distance between them. The list is in reverse order, with the last two feature vectors clustered first. From this information, a dendogram can be plotted. A dendogram looks like a binary tree with each node containing exactly two offspring. The dendogram's X

axis contains the sequence number (name) of the feature vectors and the Y axis is distance. Examples are contained in PART IV.

III. THE PROGRAMMING ENVIRONMENT

A. BACKGROUND

In any large software system serious consideration must be made concerning methods for program development and the programming style to be used. Ideally, this should be done prior to writing any software. Key concerns are the computer hardware and the computer language to be used.

The use of the Electrical and Computer Engineering Department's VAX 11/750 was a practicable decision, given its availability. The choice of a superset of LISP, called INTERLISP, as the programming language was made primarily to allow for the development of a higher level software shell, using various concepts from the field of artificial intelligence (AI). Its availability on the VAX was also a key factor.

B. LISP

LISP (LISt Processing) was developed in the 1950's by John McCarthy at M.I.T. It was based largely on work done by Alonzo Church (on lambda calculus), that dealt with symbolic, as opposed to numerical

calculations [26].

LISP has a very simple basic structure in which all data is in the form of symbolic or s-expressions. S-expressions consist of two types of objects: (1) atoms, which are symbols, numbers or strings and (2) lists, which can contain other lists and/or atoms nested to a finite degree. The great nemesis of the LISP programmer, the parenthesis, is used to mark the beginning and end of each s-expression. There are no restrictions placed on these s-expressions, therefore it is possible to accomplish almost anything in the language, including self-modifying code and functions that can create and execute other functions. It is easy to create functions that take other functions as parameters. One important advantage of LISP is that the language makes no distinction between functions (executable code) and data. This characteristic gives LISP much of its power and richness. Its primary appeal, in constructing a pattern recognition system, is the ability to write higher-level structures (shells) that provide sophisticated controls to the software developer. An object-oriented programming structure was built and will be described in more

detail in PART III, Section G.

C. INTERLISP

INTERLISP (INTERactive LISP) is a superset of LISP developed at XEROX Corporation in the 1970's [27]. The version that runs on the VAX was written at the University of Southern California. INTERLISP has a very powerful and rich development environment. It contains the following important aspects:

1. a sophisticated structure editor, which helps keep track of nesting and parentheses,
2. an interactive debugger, which allows the developer to view, or change, stack contents,
3. a print facility, which outputs structured code, clearly showing the level of code nestings,
4. a program analysis package, which can be used to obtain a picture of the overall flow of control within a program,
5. forgiveness and automatic correction for some types of errors (especially spelling errors),
6. a mechanism for undoing the results of previous operations, which allows the developer to keep several different versions of the same function running simultaneously [28].

Unlike many other interactive languages, INTERLISP can also be compiled. Compiling improves the overall execution speed by approximately 35-fold. Another important aspect of the language is that variable bindings are deferred until execution time. This can save the developer time when prototyping new software. None of these benefits come without some cost. One disadvantage of INTERLISP is its complexity. It is a difficult language to learn to use well. Another disadvantage is the large amount of physical memory required and the corresponding reduction in execution speed. Along with the INTERLISP Reference Manual [IRM], another excellent source of information about the language is INTERLISP The Language and Its Usage [29].

D. PROGRAMMING STYLE

With a language as rich, powerful and unstructured as INTERLISP, certain guidelines must be established so that the finished software product is readable, understandable and maintainable. The following guidelines were used in the development of this pattern recognition system.

1. The use of recursion is limited, since it is often hard to follow what is being accomplished in such a procedure.

2. The number of global variables is minimized. Most of the variables used in this system are declared in a PROG statement which localizes them to the defining function.

3. A RETURN statement for the variable(s) to be returned from the function call is used. This is necessary when PROG is used, and allows the developer to clearly see the return values from the function.

4. CLISP (Conversational LISP) statements are used wherever possible for conditional statements and looping blocks. CLISP statements include IF-THEN-ELSE, DO-WHILE and DO-UNTIL. These produce a more readable code, especially for those less familiar with LISP.

5. All new function names defined by the developer have their first letters capitalized, and multiple words separated by periods, eg. Find.Image.Points. Each function accomplishes one set of calculations and their names are meaningful.

6. All functions previously defined by INTERLISP are fully capitalized, eg. PROG.

7. All user-defined variable names are in small

letters with different words separated by periods, eg. start.pt. Their names are also meaningful.

8. Suffixes at the end of variables are used to denote their data types when the variable name itself is not sufficient, eg. counter.i is an integer, difference.r is real, image.pixel.ch is a one letter character, extra.flag.b is a boolean flag (its value is either T or NIL), and begin.pt.c is a cartesian coordinate (row first then column).

9. An error message returned by a function is a string variable, so that the calling routine can quickly check for an error condition. No strings are returned by a function that successfully completes its task.

10. No self-modifying code or stack manipulations are used.

11. Input/output is limited to a few functions.

12. All software is well commented.

E. ARTIFICIAL INTELLIGENCE

INTERLISP is an excellent language to use for developing AI software. The application of AI to this pattern recognition system was a modest one. The area of AI of interest in this system was knowledge representation. The flavor style of object-oriented

programming used within it has several features that make it useful for knowledge representation. These will be discussed in more detail in PART III, Section F.

AI is a complex and controversial field. At the present time, it is in a transition phase between theoretical and practical applications. An AI system is using intelligence when it has the ability to focus on the most relevant knowledge and to ignore less relevant knowledge [30]. How its knowledge base is represented will play an important part in the overall AI system's ability to find adequate solutions to the problem being solved. Knowledge should be logically organized and easy to understand, access and update.

AI is built on the following three ideas:

1. knowledge of the domain of interest,
2. methods for operating on that knowledge,
3. control structures for choosing the appropriate methods for modifying the knowledge base [31].

The power of an AI system comes from the knowledge it has more than anything else.

Expert systems are one area of AI research that have achieved a fair amount of success in the past several years. They are computer programs that solve difficult problems requiring expertise, by utilizing facts and heuristics (rules of thumb) that the human expert would consider. Expert systems can be distinguished from the broad class of AI problems by the fact that they solve problems within a restricted domain and attain expert levels of performance [32].

Knowledge plays a key role in the development of an expert system. Expertise itself consists largely of knowledge about specific tasks or problem areas. The main problem for the expert system developer is how to locate and store knowledge, so that the computer can use it efficiently (knowledge engineering). There are four different paradigms (conceptual tools) used in developing expert systems, and they affect how knowledge will be represented. The first paradigm is the procedural or function-oriented approach. LISP is such a paradigm. The second is the rule-oriented approach, where the knowledge base consists of a list of IF-THEN rules used to arrive at a conclusion. The third is the data or access-oriented approach, where

the message is the key. The last paradigm is the object-oriented approach, of which the flavor system developed for this pattern recognition system is a member.

F. OBJECT-ORIENTED PROGRAMMING

Object-oriented programming is less a new way of programming than a new way of perceiving how a program is to be organized. Data objects and the functions that operate on them are linked together in a structure. Instead of passing data to a function, an object is requested to perform (message passing) some operation (function) on itself.

Each data object is a member of a certain class (flavor) of objects. All data objects within the same class share the same variables and procedures (methods). After creation, each data object is considered an instance of its class. Each class can have multiple instances, but each instance belongs to only one class. Calculations are accomplished by sending messages to an instance of a class and telling it what function (method) to execute.

A computer language that supports object-oriented programming should have the following attributes; information hiding, dynamic binding and inheritance [33]. The first language containing these elements was Smalltalk, developed at XEROX Corp.

Information hiding and data abstraction are used so that the developer need only focus on the important aspects of the software. A small set of functions is used to set up and run the interdependent modules in the object-oriented program. These functions are independent of the type of object classes and instances in existence. The developer does not need to know how the instance variables or methods are changed or invoked, only the names of the functions that accomplish these actions. The variables in one instance are not changed by methods acting on other instances within the same class. Not only is information hidden, but it is also protected.

Dynamic bindings of variables are required so that a method does not need to know what data types it will be acting on, before it is actually called. This is an aspect of data abstraction. An example of this would be

a stack routine with methods to POP and PUSH. Ideally, one does not want to have to specify different POP and PUSH routines for integer, real or string stacks [34].

Inheritance enables objects to be subclasses of other classes. A subclass has knowledge about (inherits) all the variables and methods of the class it belongs to. In this flavor system, one restriction added was that a subclass cannot change the value of a variable from a higher-level class. This allows for a modular design of the overall software system, combined with information hiding. Higher levels of classes will be more general, while class specialization will be accomplished at lower levels within the overall structure. Inheritance can be used to decrease code that must be shared between different objects.

One major disadvantage of object-oriented programming is the access time it adds when a function is invoked. Message passing is not as efficient as calling a procedure. Variable accessing and storage is also slower and more complicated. Another disadvantage is that a beginner cannot simply sit down and start programming in an object-oriented environment without

first learning the key functions that control the environment. More thought concerning the overall structure of the software must be accomplished before programming can actually begin. Despite these problems, the benefits of object-oriented programming in the development of large software systems considerably outweigh the disadvantages.

G. FLAVOR SOFTWARE

The object-oriented programming style developed for use in this pattern recognition system is called a flavor system. It can run independently of the pattern recognition system, and be used for general purpose object-oriented programming. Flavor style programming was introduced by Howard Cannon (one of the founders of Symbolics, Inc.) and is now available on several different types of LISP machines [35].

This flavor system consists of eight different functions: DEFFLAVOR, DEFMETHOD, MAKE.INSTANCE, SEND.MESSAGE, GET.VALUE, PUT.VALUE, PRINT.FLAVORS and PRINT.INSTANCE, all stored in file FLAVOR. DEFFLAVOR (DEFine FLAVOR) is the first function required. It needs three parameters; new flavor name (class), a list

of variables (and any default values) to be associated with this flavor, and the name of the parent flavor (for inheritance). To work correctly, the developer must create flavors from the top downward, ie. starting with the most general flavor. The first flavor that must be created in a system is VANILLA. This flavor contains variables and methods inherited by all other flavors in the system. The creation of the VANILLA flavor destroys any previously existing flavor environment, so the developer must be careful when it is used. DEFFLAVOR modifies the global variable flavor.environment which keeps track of existing flavors and instances. The developer never has to access flavor.environment directly. DEFFLAVOR also creates a new global variable which contains the list of variables and default values. This variable is used by MAKE.INSTANCE to create an instance of the flavor. DEFFLAVOR returns an error message if there was not at least one new variable associated with the new flavor.

DEFMETHOD (DEFine METHOD) is the second function required. All flavors must have at least one unique method associated with them. DEFMETHOD requires four parameters: the flavor name for the method, the method

name, the demon, and the actual method. There are three types of demons available: before, none and after. Neither the before or after demon methods are required. If the before demon is present, it will always be executed before the main method (no demon). If the after demon exists, it will be executed after the main method. This provides a way to hide information. In this pattern recognition system, the before demon was generally used to find the required parameters for the main method and place them into a properly ordered list. The after demon was generally used to place the return variables from the main method into their appropriate slots in the flavor environment. Using this approach simplifies the main method. DEFMETHOD uses a global variable which contains the list of methods for each flavor. Again, the developer never has to access this variable directly. DEFMETHOD returns an error message if the input flavor name has not already been defined by DEFFLAVOR.

Up to this point in the development, nothing has actually been created, only defined. To create an instance of a flavor requires a call to MAKE.INSTANCE. This function will work only if DEFFLAVOR and at least

one DEFMETHOD have been previously executed for the given flavor. It will also work only if an instance of the parent flavor has already been created. This means that before any other instance is made, an instance of VANILLA must be created. MAKE.INSTANCE requires the following variables: flavor name, name for the new instance, name of the parent instance, and any initial values for the flavor variables. The name of the parent instance is required so that inheritance can be strictly controlled, especially when there are multiple instances of the parent. Each instance of a new flavor will inherit only those variable values belonging to the parent instance. Only default flavor variables are inherited by all the instances of a flavor. This allows for complete data isolation between instances of a flavor. The altering of one flavor variable in one instance will not affect that variable value in any other instance of that flavor. The structure of a flavor is contained in a record called flavor which is at the end of the file FLAVOR. A flavor record consists of a list of methods, a list of variables and the name of its parent flavor. MAKE.INSTANCE returns an error if the parent instance is not in existence, if there are no methods associated with the flavor or if

the flavor has not been defined yet.

After an instance has been created, the methods associated (or inherited) with the flavor can be invoked. Invoking a method in an object-oriented system involves a process referred to as message passing. The function that accomplishes this is SEND.MESSAGE. This function requires four parameters: flavor name, instance name, name of the method to be invoked and a list of required parameters (if any). First, SEND.MESSAGE tries to find a before demon for the input method name. If one is found, the input variable list (if any), is passed into the before demon and it is executed. The return list from the before demon is placed into a variable called before.return. Next, the main method is invoked and passed either before.return, if the before demon exists, or the original variable list. The return from the main method is placed into the variable main.return. Finally, SEND.MESSAGE searches for an after demon and passes it main.return, if it exists. Either main.return or the return from the after demon is returned when SEND.MESSAGE is completed. Various error messages are returned if the flavor, instance or method

do not exist, or main.return contained an error.

Two functions, GET.VALUE and PUT.VALUE, deal exclusively with flavor variables. GET.VALUE returns a flag and the value of the variable requested. It requires three parameters: flavor name, instance name and the variable name. If no value for the variable is found on the specific instance variable list, then the default variable list defined in DEFFLAVOR is checked. Any variable with no value on either list is given a value of NIL. NIL is the default for all variables with no previously entered value. The flag returned by GET.VALUE is set to T only if the flavor, instance and variable names were legal. If the flag is NIL, then an error occurred. PUT.VALUE places a new value for a flavor variable into the list of one instance of that flavor. It requires four parameters: flavor name, instance name, variable name and new variable value. By requiring the instance name, isolation between instances of one flavor can be maintained. PUT.VALUE returns an error message if the flavor, instance or variable name was invalid. Any flavor can request the value of a variable it owns or inherits (GET.VALUE), but only the flavor owning the variable can change its

value (PUT.VALUE). This adds protection to the system.

PRINT.FLAVORS is used to output the present flavor environment. It prints out a list of flavors that have been defined and the names of all instances of those flavors. PRINT.INSTANCE is used to print out all the methods, variables and their values, both owned and inherited, belonging to one instance of a flavor. The required parameters are the flavor and instance names.

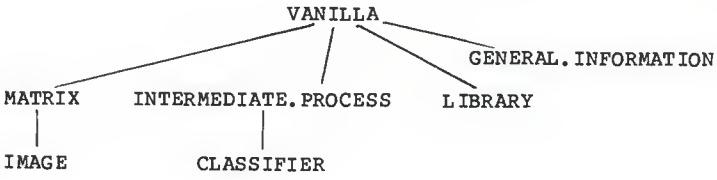
H. FLOW OF CONTROL

The basic functions that make up the pattern recognition system were discussed in PART II. This section will detail how the flavor system interacts with the pattern recognition functions to achieve an object-oriented pattern recognition system.

The top function for the entire system is Skeleton, which is contained in file START_SYSTEM. This is the function the user calls to start the entire process. The first information the user is prompted for is whether a session transcript is desired. This will allow the user to keep a printout of all the screen displays. After this, the function Load.System.Files

is called. This function loads all the compiled files required by the system. The names of these files are IN_OUTPUT.V, CENTER.V, TRACE.V, FLAVOR.V, MOMENT.V, REG.V, CLASSIFY.V, MATRIX.V and SET_UP_SYSTEM.V. All are contained in the VMS subdirectory <STOUT.COMPILED>. The next function called is Set.Up.Flavors. This is the key function for the flavor system. In it, all the flavors and methods for this system are defined. The flavor system used in this program is outlined below.

Figure 7. Organization of flavor system.



The flavor, MATRIX, contains information about the individual arrays; IMAGE contains information about the individual patterns in each array; LIBRARY contains the classifier image library; INTERMEDIATE.PROCESS contains the information about the autoregressive and Zernicke moment algorithms; and CLASSIFIER contains the

classifier and clustering algorithms. A complete description of these flavors is contained in the comments in Set.Up.Flavors in file SET_UP_SYSTEM.

After this environment is defined, instances of VANILLA, GENERAL.INFORMATION, INTERMEDIATE.PROCESS and CLASSIFIER are created. In this system, the name of the instance of VANILLA must be "VANILLA". PRINT.FLAVORS is then called to display the current flavor environment. Next, a message is sent to the method, Get.User.Variables. This method contains the user prompts for all the key system parameters. The information requested from the user includes the type of system desired, classifier or clustering, the type of intermediate process, the type of classifier, the name of the arrays to be input, and the name of the library (if any). A sample session is contained in APPENDIX 1.

If a valid list of arrays was loaded, then the analysis begins. If the user is going to classify geometric objects, then an instance of flavor LIBRARY is created. If a valid library file was loaded into the environment in Get.User.Variables, then the image

classification library is configured by a call to method Create.Library.From.File. Next, a loop is entered that continues until all the arrays loaded into the environment are analyzed.

For each new array, an instance of MATRIX is created. After an image pixel is found (method Find.Image), a new instance of IMAGE is created. This image is analyzed and the appropriate method of INTERMEDIATE.PROCESS is executed to obtain the feature vector. The next steps depend on what the user desires. If classification is desired, then the appropriate method from CLASSIFIER is called to classify. If clustering is desired, then all the feature vectors are placed into a list for the method, Clustering. After the classification analysis, the system outputs the results for the present image. If the classifier is adaptive, the user is requested to verify the classification of the image before the library is updated. This cycle is continued until all images on all the arrays are analyzed. At the conclusion of the program, the user can store the library in a file. This is especially useful for adaptive libraries.

IV. System Performance

A. CPU Function Times

To assist the user in making a choice of the type of feature extraction technique used in the pattern recognition system, various timing functions were used and analyzed. These results are especially useful when time is an important factor. CPU times for the low-level functions allow the user to analyze their relative complexities.

Several measurements of the pattern recognition system's performance can be made. This section will compare CPU times for the key methods (functions) in the low and intermediate-level image processing areas. The first two figures (Figures 8 and 9) are for methods in the low-level image processing section.

Figure 8 contains CPU times for the methods Follow.Edge, Shell.Sort.Edge and Find.Image.Points. Figure 9 contains CPU times for the methods Calculate.Area, Calculate.Center.Area, Col.Sums and Replace.Image.Points. The X axis is the length of one side of the square being analyzed by these low-level methods. The area of the square increases from 9 to

Figure 8. Low-level Functions - CPU Times (I)

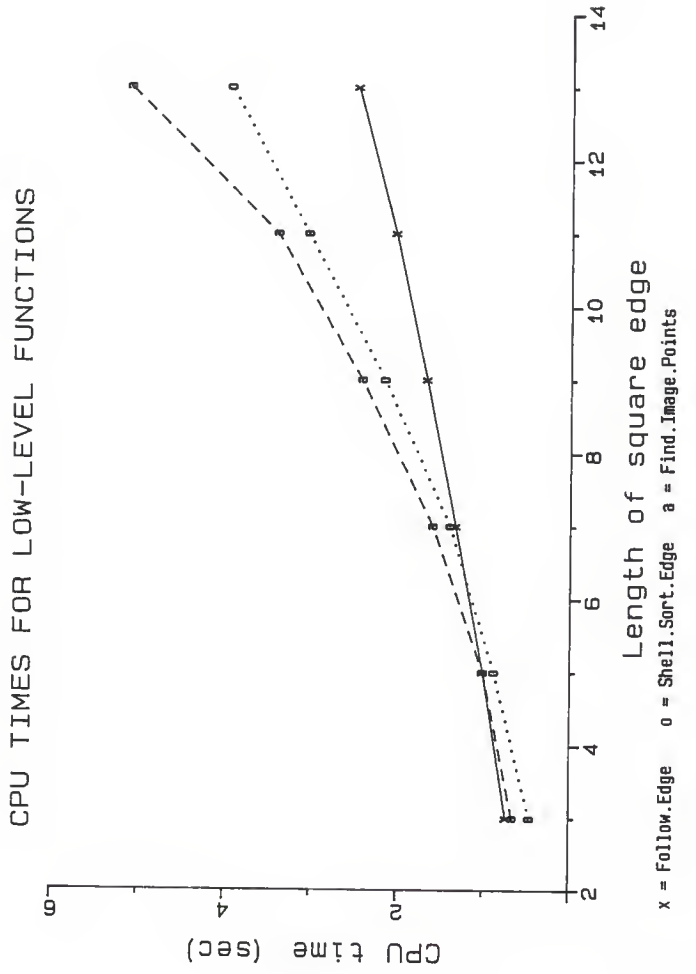
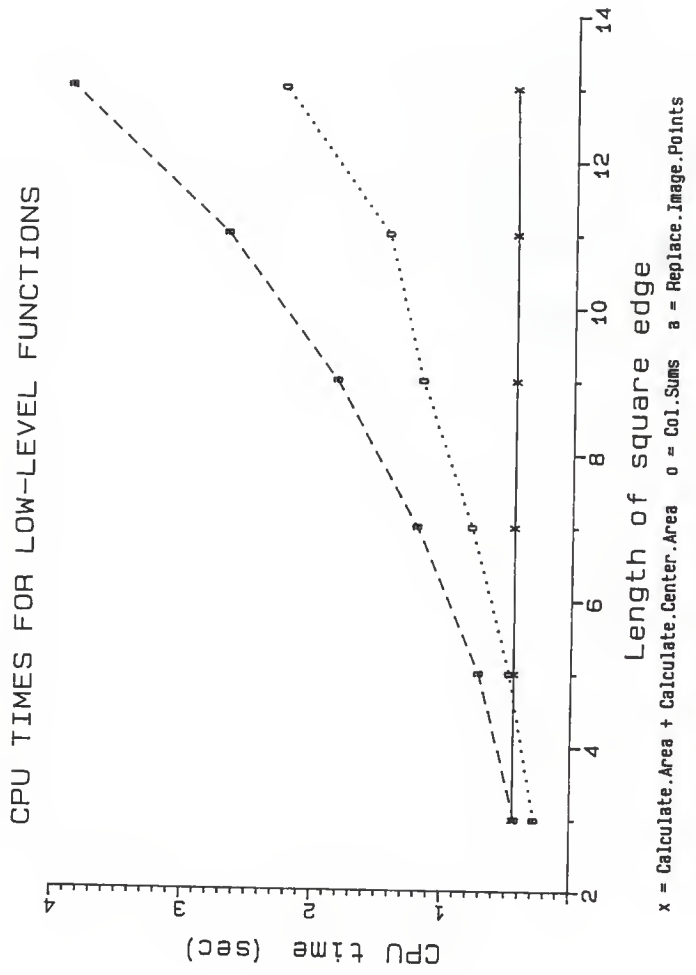


Figure 9. Low-level Functions - CPU Times (II)



169 pixels. Calculate.Area + Calculate.Center.Area shows very little change as the size of the square increases. These two methods multiply and divide several real numbers together. The method Col.Sums shows a nearly linear increase in time until the last square. The method Follow.Edge also shows a fairly steady rise, without the sudden jump at the end. The other three methods, Shell.Sort.Edge, Find.Image.Points and Replace.Image.Points, appear to increase with the square of the length (in proportion to the area). This is expected for Find.Image.Points and Replace.Image.Points, because they consider all image points. Shell.Sort.Edge sorts the edge points only, but the algorithm is not a linear function of the number of edge points. It is clear that Find.Image.Points will increasingly dominate CPU time as the size of the image grows.

The next set of figures (Figures 10, 11, 12, 13, and 14) measure CPU times for various intermediate-level image processing methods. These times can be useful to the system developer when time is a key consideration in the analysis process. Figure 10 compares the times for the two Zernicke moment feature extraction

techniques. The Zernicke technique using only the edge points increases linearly with time, while the one using all image points increases in proportion to the square of the length. When time is a critical factor and large images are involved, the technique using all the image points may have to be avoided, despite it being more accurate.

Figures 11 through 14 are CPU times for various combinations of ray and lag components in the autoregressive model feature extraction technique. Figure 11 shows the relationships between the size of a square and three different ray numbers, for one lag component. Figure 12 shows the same relationships, except using two lag components. From these two figures, one can see that the addition of rays does not significantly increase the CPU time for the autoregressive technique. Figure 13 shows the relationship between the number of lag components and the area of the image. For 32 ray slopes, increasing the number of lag components does not significantly increase the CPU time for the method. Figure 14 compares times for various combinations of lag components and ray slope quantities. From this plot,

Figure 10. Zernicke Moment Function - Cpu Times

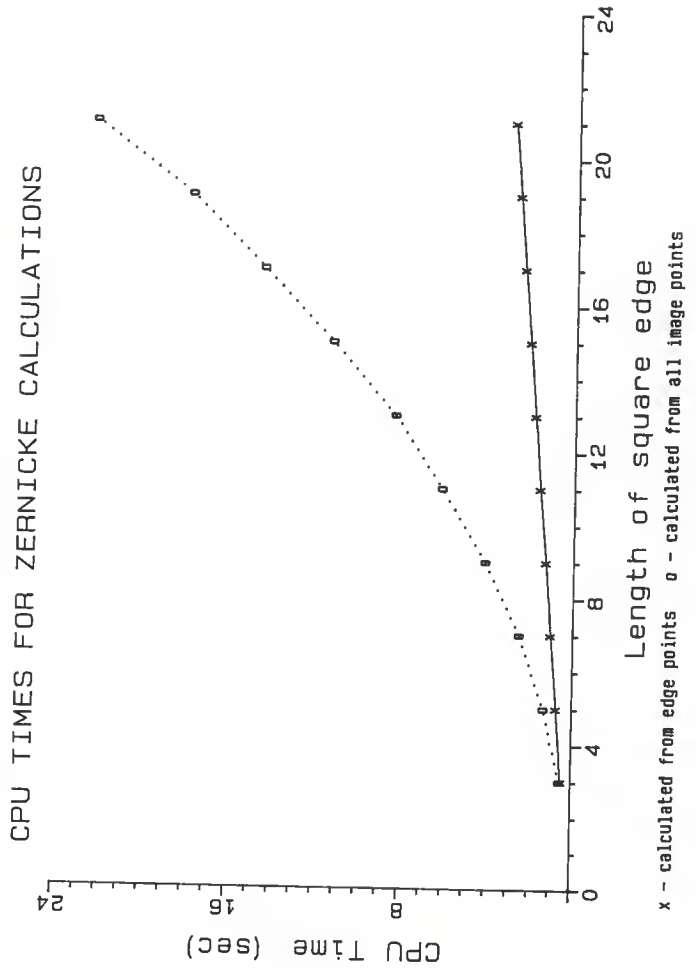


Figure 11. Autoregressive Model Function - CPU Times (I)

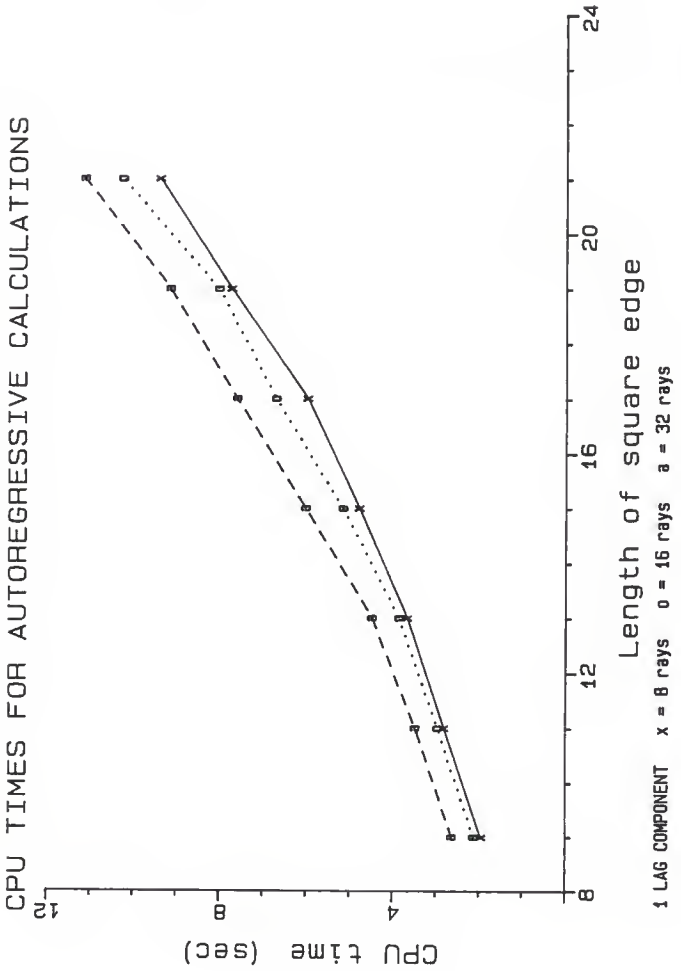


Figure 12. Autoregressive Model Function - CPU Times (II)

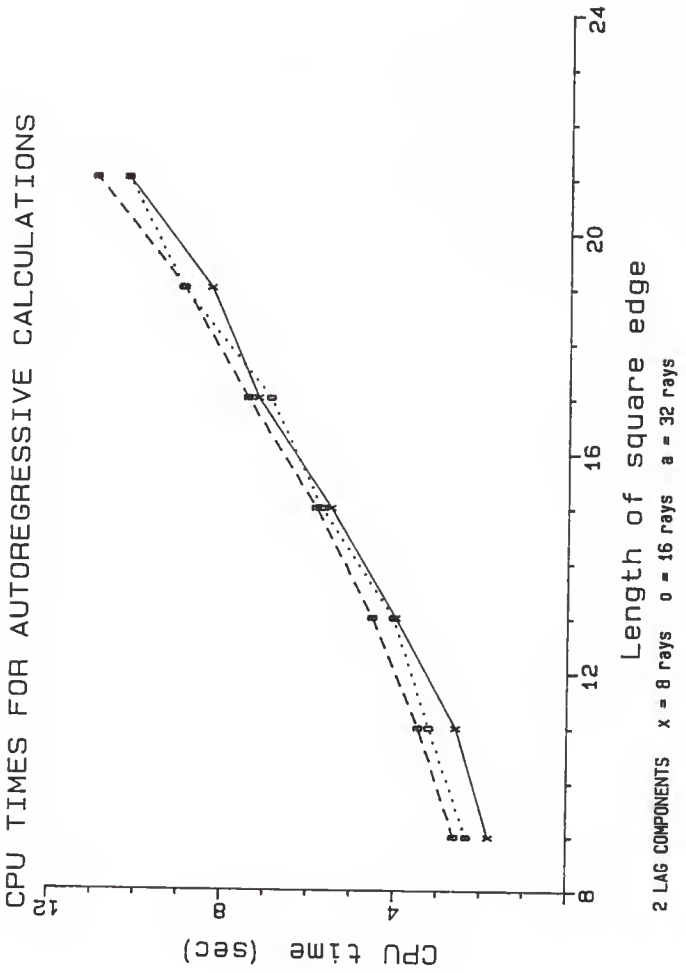


Figure 13. Autoregressive Model Function - CPU Times (III)

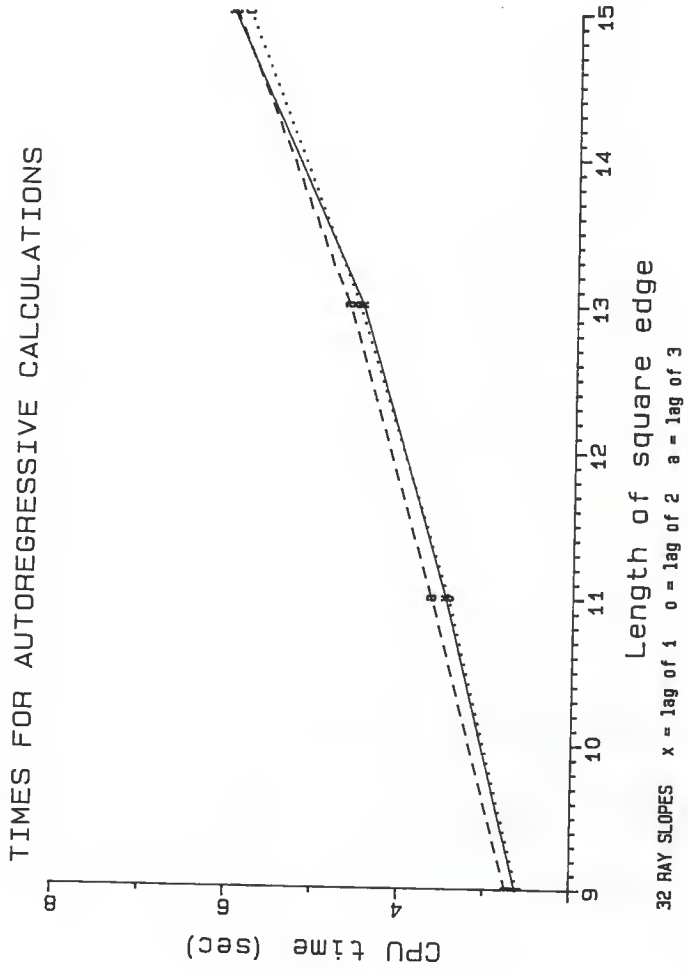
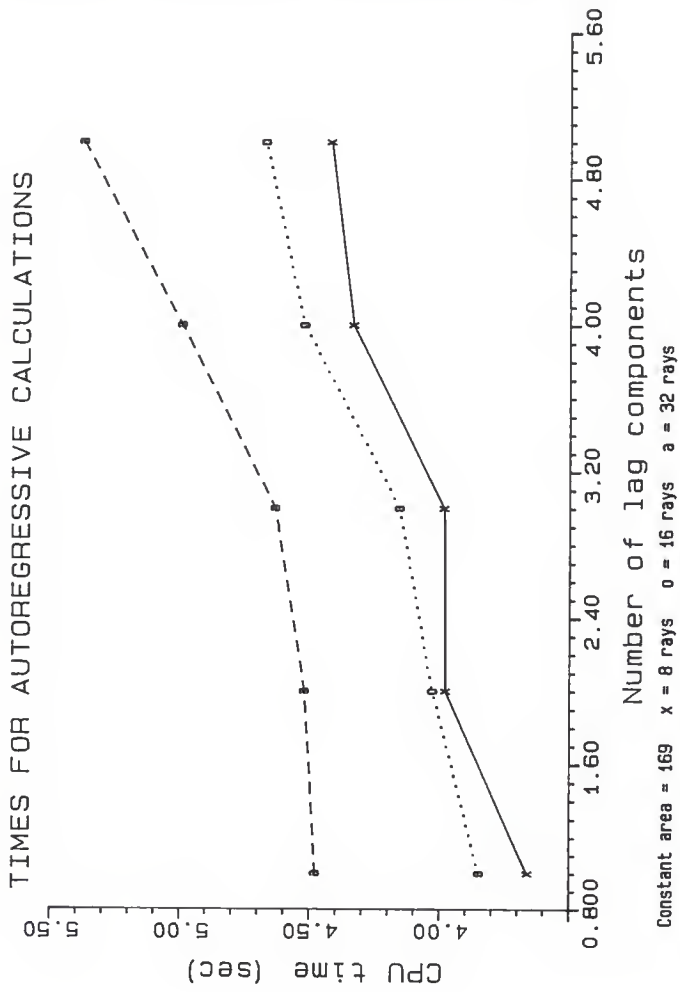


Figure 14. Autoregressive Model Function - CPU Times (IV)



one can see that there is a significant increase, at all lag values, when the number of ray slopes is increased from 8 or 16, to 32. These results from the autoregressive method leave the user with a lot of discretion for parameter selection. Comparing the CPU times with those from the Zernicke moment technique shows that the autoregressive technique uses less time than the long version (using all image pixels) of the Zernicke method, but more time than the short version (using edge points, only).

B. Invariance Properties

The invariance of the two feature extraction techniques to translational, scale and rotational transformations will determine their utility in this pattern recognition system. The translational and scale invariances will be investigated together.

Figures 15 through 20 apply to the autoregressive feature extraction technique. In each of these figures, the maximum variation will be calculated. This number is the percentage error of the feature element with the largest change on the plot. Error is the smallest value of that element divided by the

largest value. An error of zero means the feature element did not change. Scale invariance was calculated using a square with an area varying from 9 to 221. Rotational invariance was calculated using a square with an area equal to 100, rotated through 45 degrees, in 5 degree increments.

Figures 15 and 16 contain the feature elements for the model using one lag component. Only scale invariance was considered for one lag component. The maximum variation for the $A/B^{1/2}$ term is 12%, for the term calculated from 16 rays. Figure 16 contains the lag component for the feature vector and has a maximum variation of 1%, for the 16 ray term. For the simple distance classifier method, the largest feature element will have the most impact on the distance between two image feature vectors. The lag component has a value 20 to 30 times larger than the $A/B^{1/2}$ term, so the scale invariance error is close to 1%.

Figures 17 through 20 contain feature elements using two lagcomponents. Both scale and rotational invariances are considered in this example, and only 16 or 32 rays are used. The $A/B^{1/2}$ term is plotted in

Figure 15. Autoregressive Model Feature Element (I)

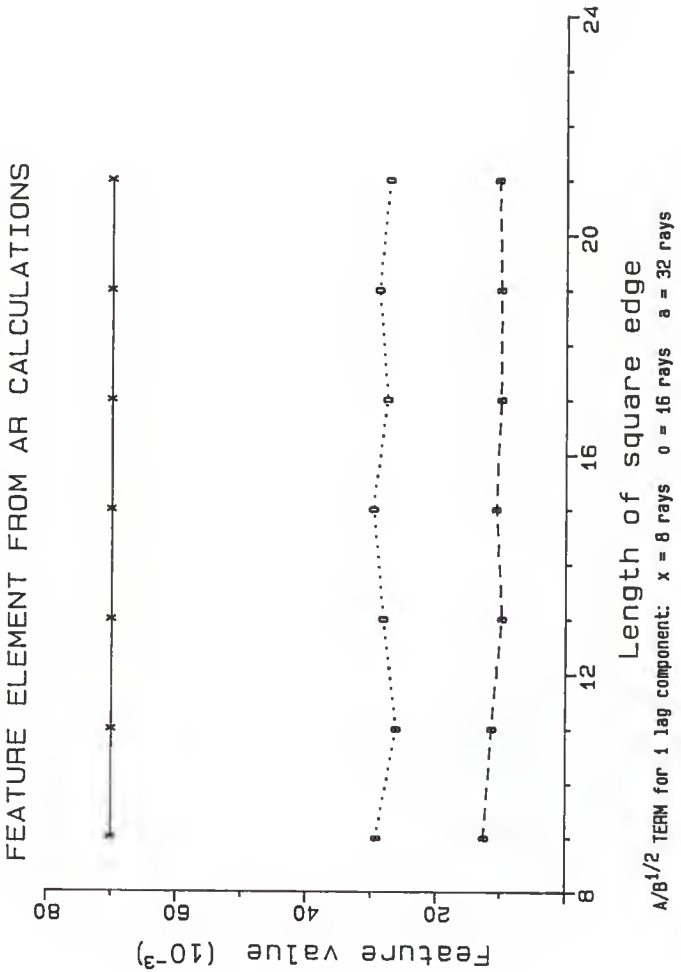


Figure 16. Autoregressive Model Feature Element (II)

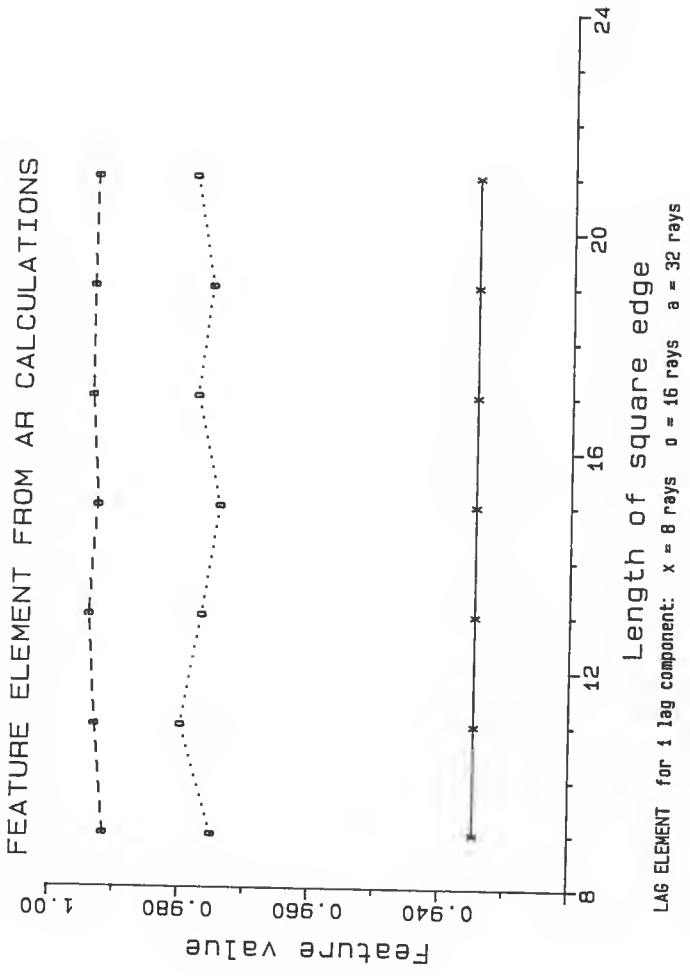


Figure 17. Autoregressive Model Feature Element (III)

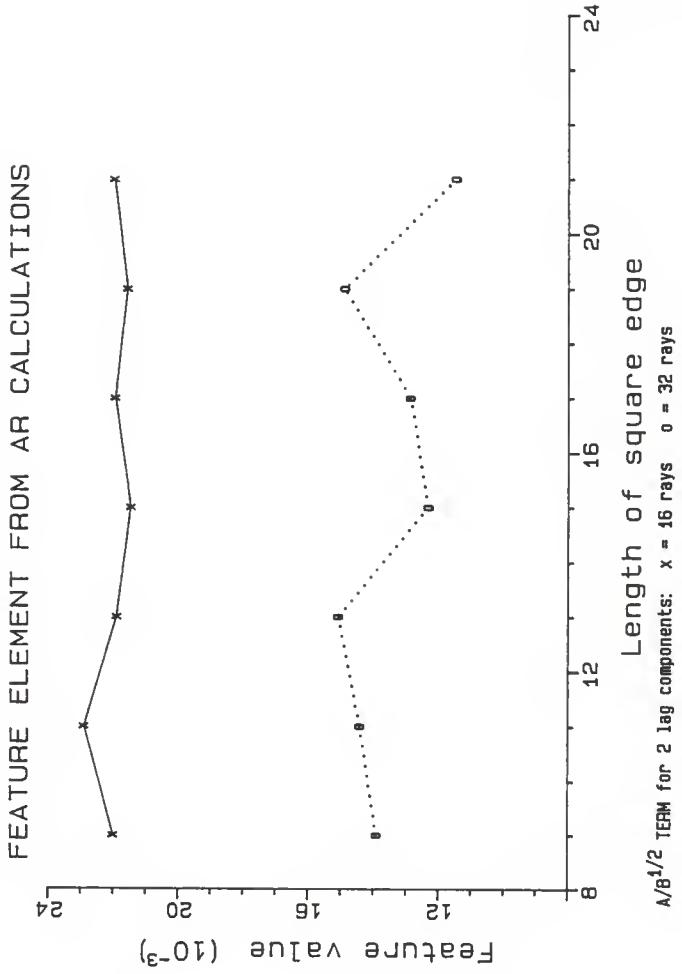
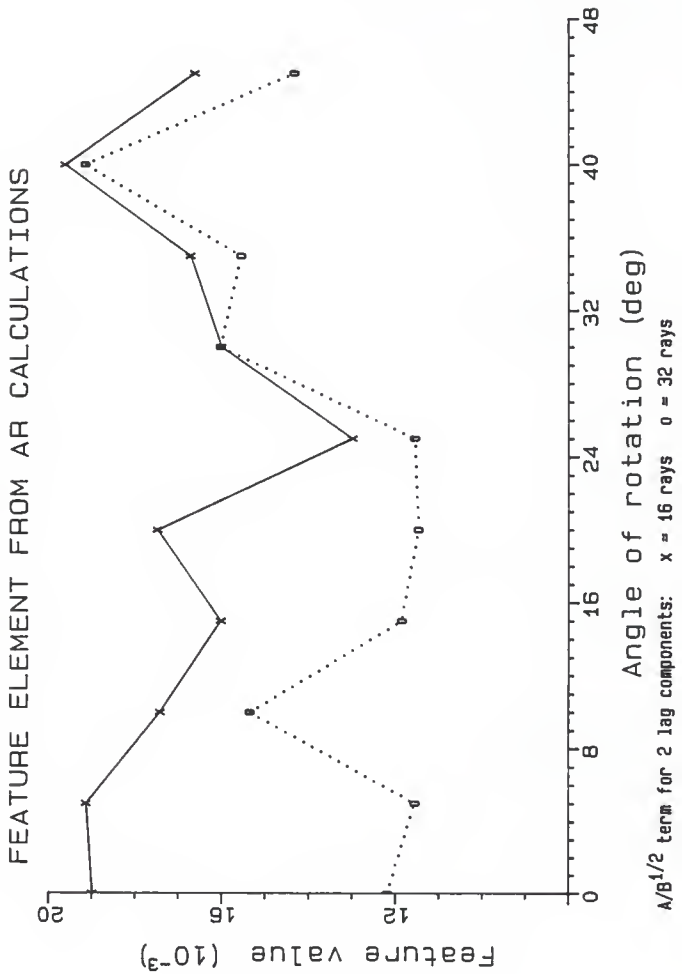


Figure 18. Autoregressive Model Feature Element (IV)



Figures 17 and 18. Its maximum variation is 19% for the scale changes and 40% for the rotational changes. Both occurred with the 32 ray example. Considering both 16 and 32 rays, the maximum variation is about twice as much for the rotated images as for the scaled images. Figures 19 and 20 plot the larger of the two lag components. The maximum variation for the scaled images is 22%. The maximum variation for the rotated images is 23%. Both occurred with the 32 ray example. This lag component is the largest of the three elements in the feature vector, so its variation will have the most impact on the overall error.

A few conclusions can be drawn from these examples. The autoregressive model technique offers better scale invariance than rotational invariance, for the set of images analyzed. Larger sized images will exhibit better rotational invariance than smaller ones. There also seems to be no real advantage to using 32 rays instead of 16, for images of the size analyzed here. For larger images, or more complicated ones, a larger number of rays will be an advantage.

The next set of plots (Figures 21 through 27)

Figure 19. Autoregressive Model Feature Element (V)

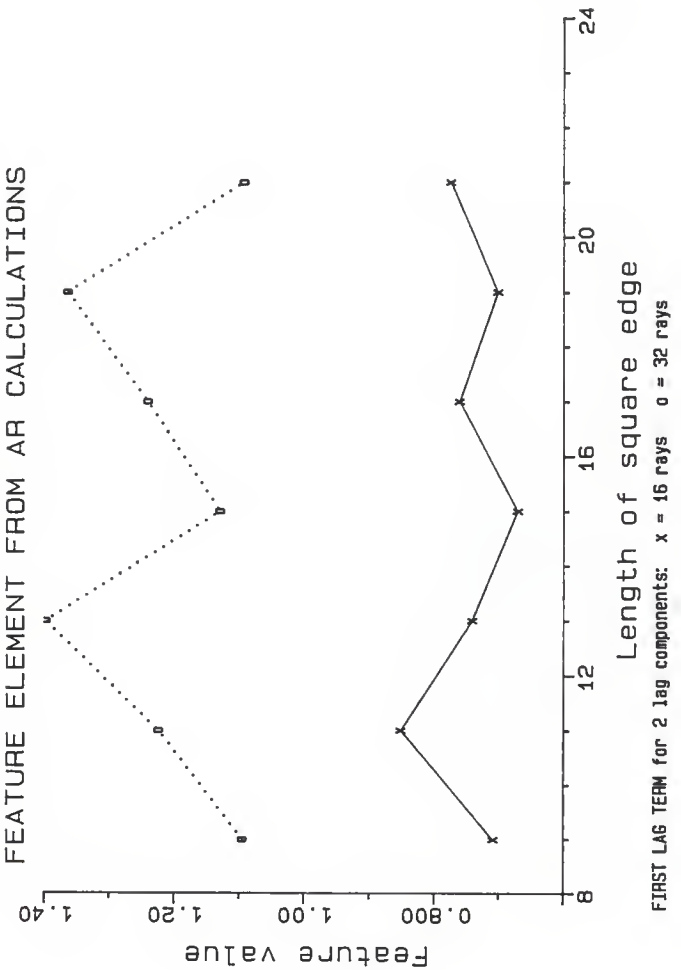
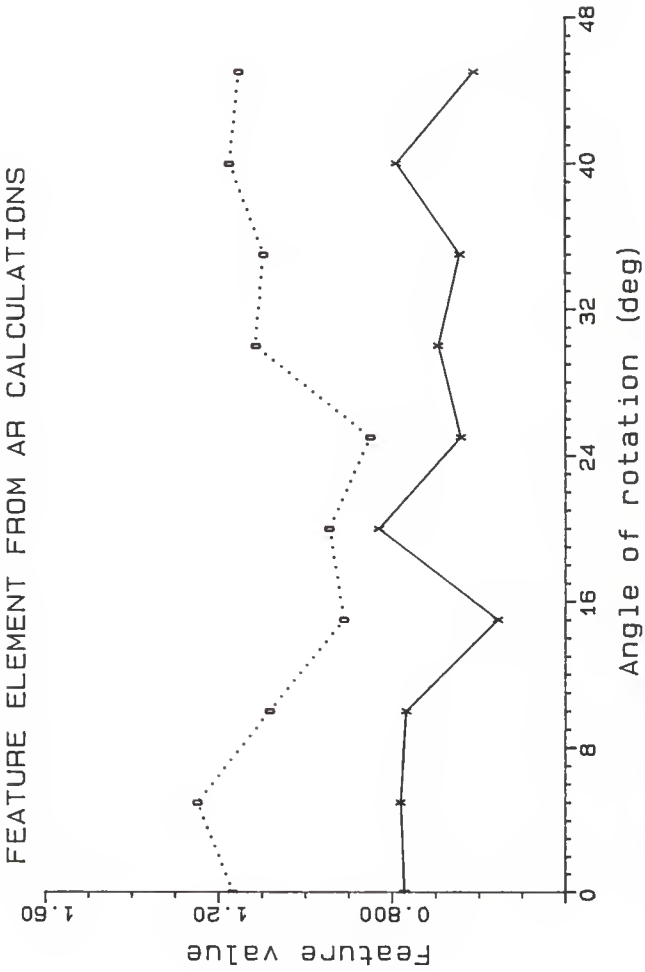


Figure 20. Autoregressive Model Feature Element (VI)



concerns the Zernicke moment feature extraction technique. Figures 21, 22 and 23 are plots of feature elements for this technique using all image points. Figures 21 and 22 are plots of one second order moment and one fourth order moment. The scaled version has a maximum variation of 6% for Z_{40}. All the rest of the second and third order moments for the scaled images were zero, so this technique exhibits excellent scale invariance. The rotated squares (Figure 22) with the same two moments have a maximum variation of 4% for Z_{40}. Figure 23 contains the plots for two second order and one third order moment for the rotated squares. These same three moments were all zero for the scaled squares. No variation measurements are possible because of the zero value for the 0 degree case. The values vary considerably, but they are all approximately 3 to 4 orders of magnitude smaller than the moments on Figure 22. From these results, one can see that this technique also exhibits good rotational invariance. This technique has better invariance characteristics than the autoregressive model technique, for the type of images analyzed.

The next set of plots (Figures 24 through 27)

Figure 21. Zernicke Moment Feature Element (I)

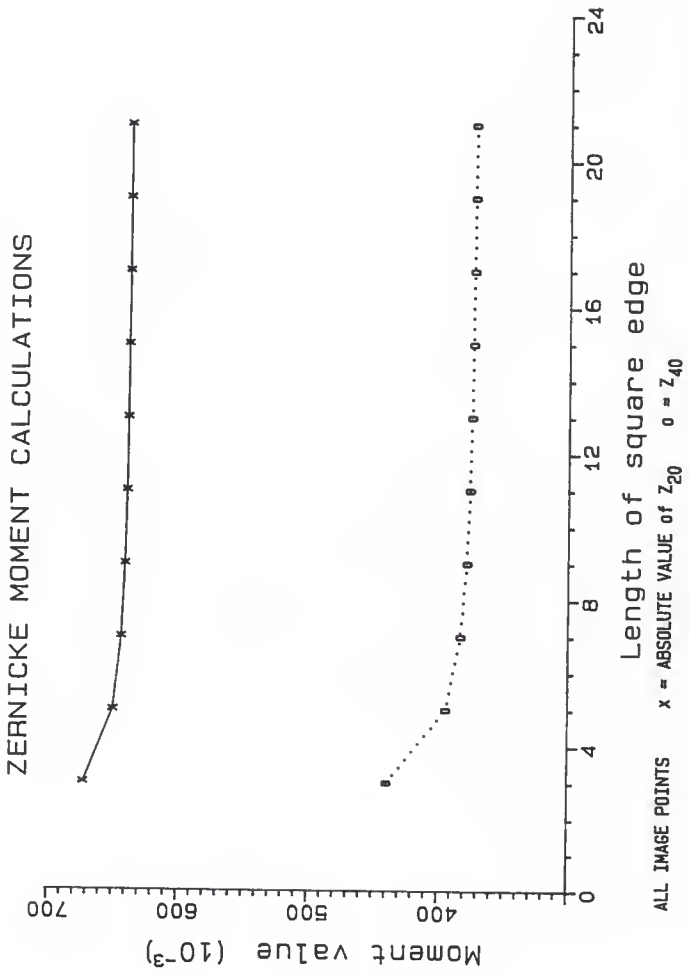


Figure 22. Zernicke Moment Feature Element (II)

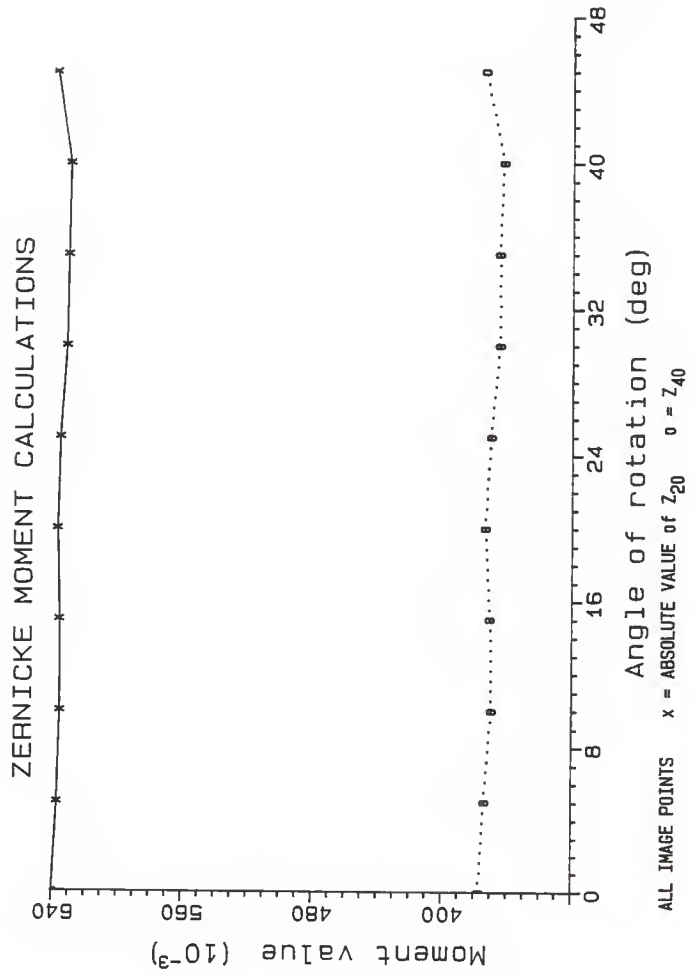
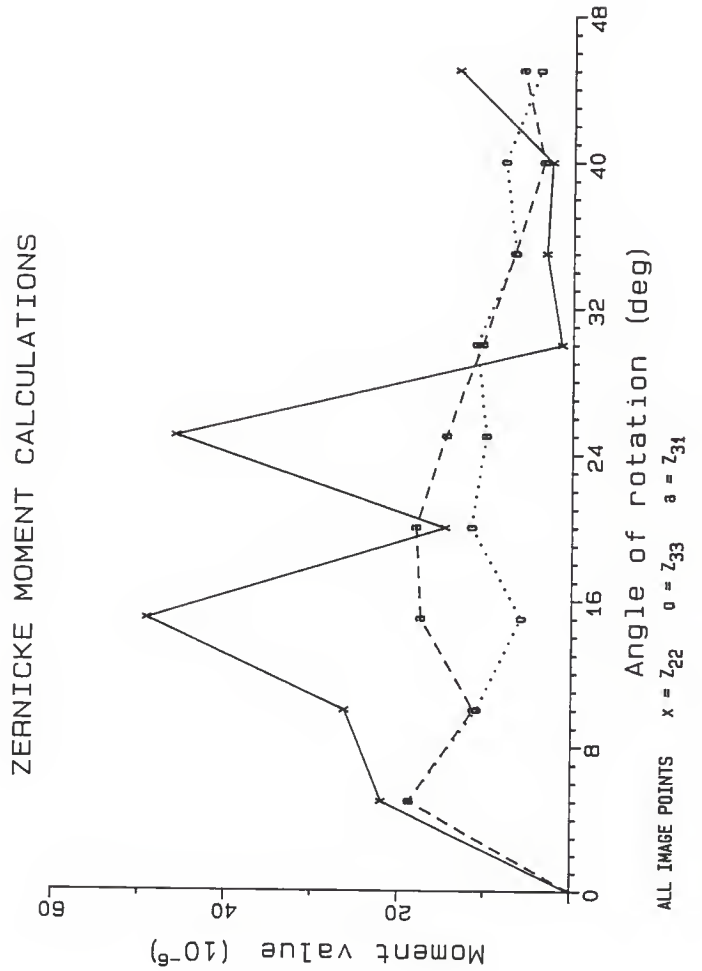


Figure 23. Zernicke Moment Feature Element (III)



concerns the Zernicke moment technique using just the edge points. As is expected, this technique is not as accurate as the one using all the image points. Figure 24 is a plot of the two largest moments, Z_{20} and Z_{40} for the scaled squares. Its maximum variation is 7.5% for Z_{40}. Figure 25 is a plot of the same two moments for the rotated images. Its maximum variation is much larger than indicated, because the points for the 45 degree rotation were omitted. The 45 degree rotation caused a large increase in the size of Z_{40}, from 0.6 to 9.4. The other moment has a maximum variation of 16%, including 45 degrees. Figures 26 and 27 are plots of the other second order moment and the two largest third order moments. Figure 26 depicts the scaled squares and has a maximum variation of 79% for Z_{22}. The rotated squares are shown in Figure 27. The same problem with the values for the 45 degree rotation also occurred here, for all three moments. Z_{31} went from 0.08 to 11.1. This figure shows that the rotational invariance worsens after 25 degrees, for these three moments.

These results clearly show that the Zernicke moment technique, using just the edge points, is the worst of

Figure 24. Zernicke Moment Feature Elements (I-A)

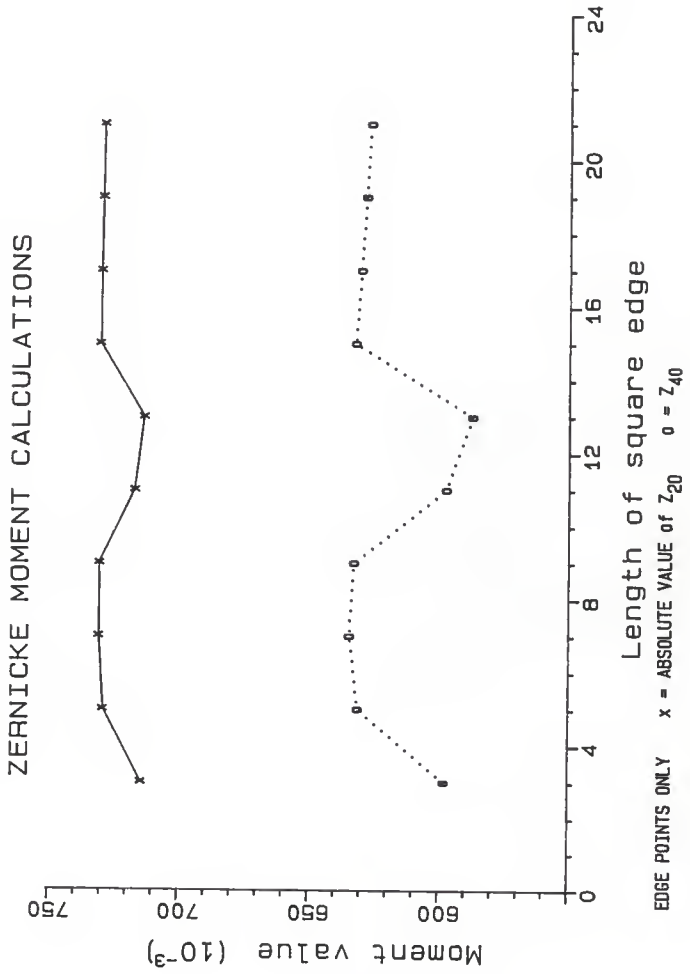


Figure 25. Zernicke Moment Feature Elements (II-A)

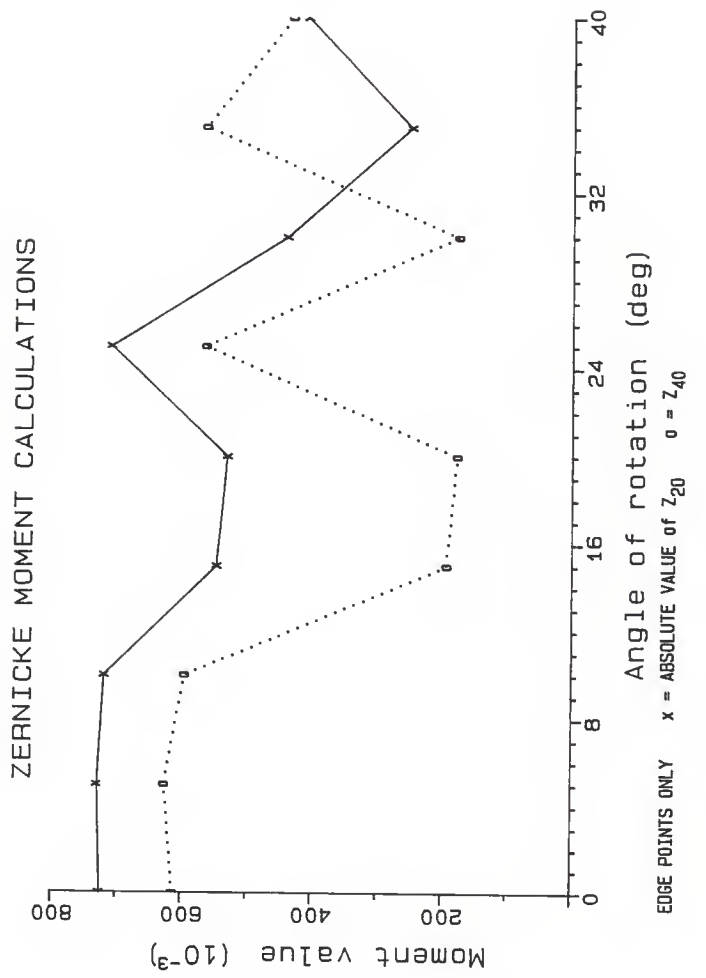


Figure 26. Zernicke Moment Feature Elements (III-A)

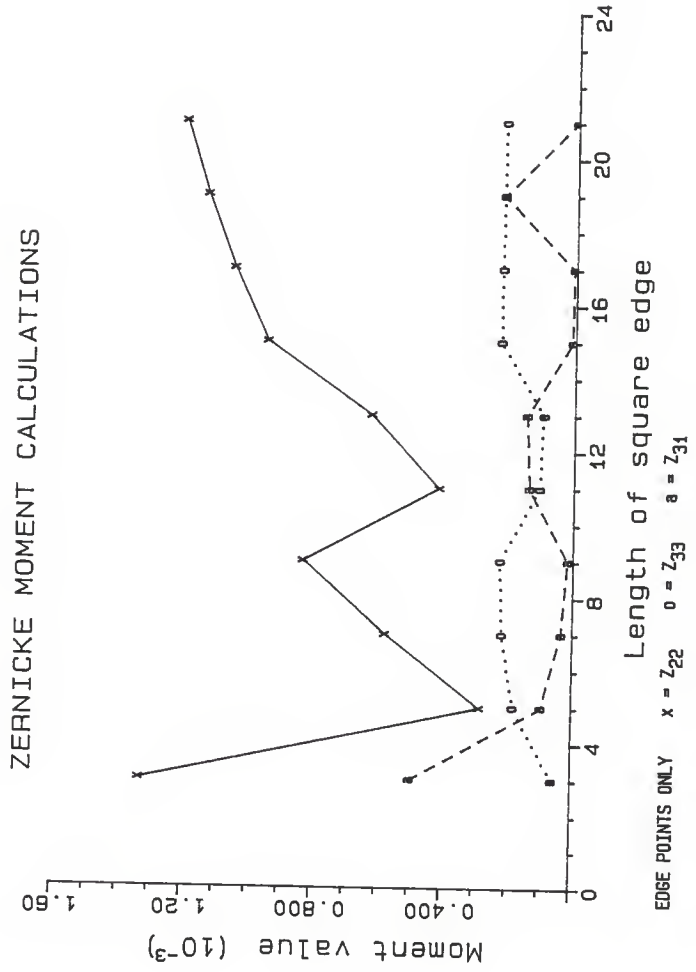
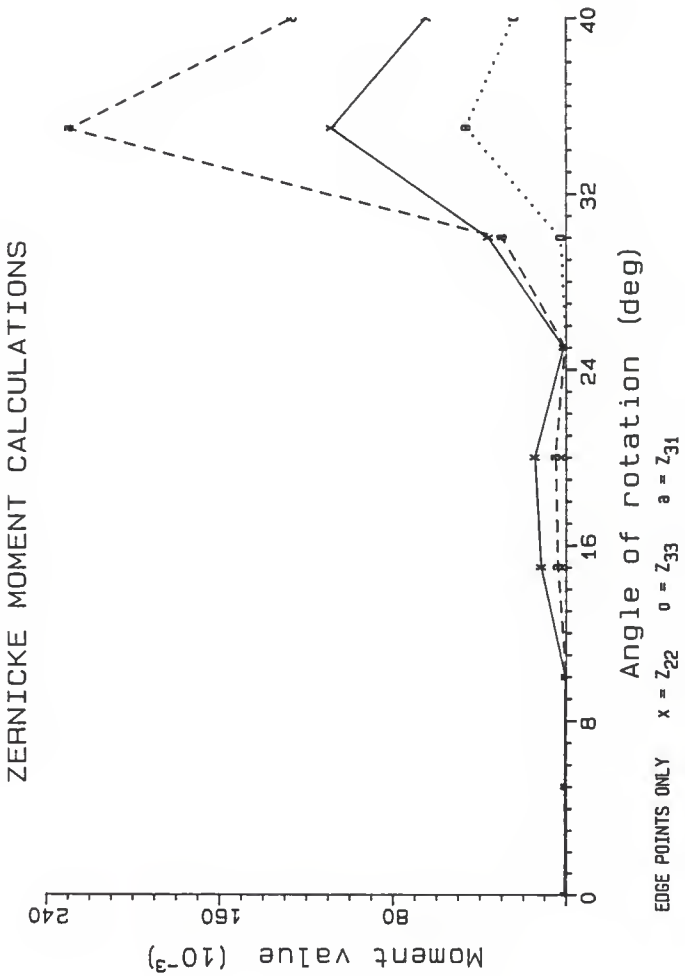


Figure 27. Zernicke Moment Feature Elements (IV-A)



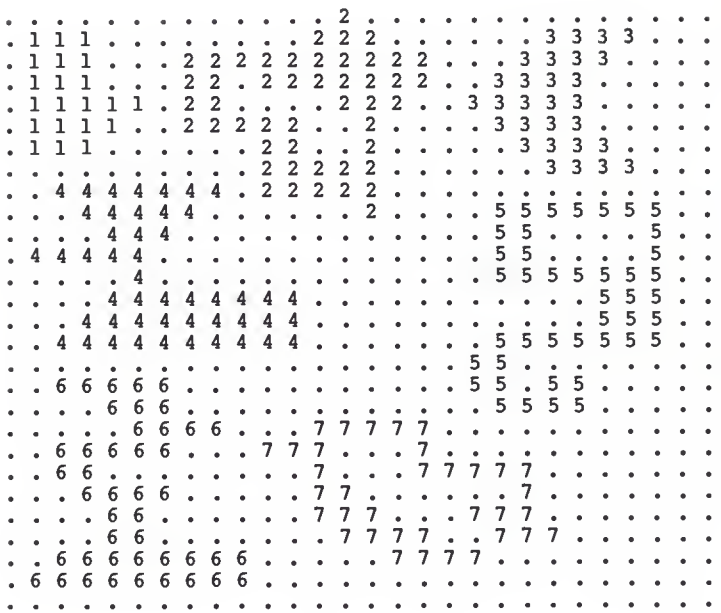
the three techniques, with respect to scale and especially rotational invariance. The Zernicke moment technique, using all the image points, is better than any of the autoregressive model examples. As one would expect, the CPU times for the three techniques are proportional to their quality. For images less than 200 pixels in area, the Zernicke technique, using all the image points, is the best one. For larger images, a time versus quality tradeoff must be made.

C. Clustering

Clustering groups a set of images by some distance measurement and organizes them in a tree-like structure (dendogram). As the distance increases, the images are more and more dissimilar. The sample images that were clustered are contained in the array depicted in Figure 28. The numbers underneath the X axis in Figures 29 and 30 correspond to the numbers of the images in Figure 28.

Figures 29 and 30 are dendograms obtained from data from the clustering algorithm. These examples demonstrate the feature extraction techniques and the simple distance classifier algorithm. Figure 29

Figure 28. Set of images to cluster.



contains the clusters obtained from the Zernicke moment technique, using all the image points. The simple distance classifier was used and the distance metric was the Chebychev one. The Chebychev metric sets the distance between two images to the largest difference between the feature vector elements. Figure 30 contains clusters obtained from the autoregressive technique, using 2 lag components and 32 rays. The same classifier was used, but the distance metric was the Euclidean distance squared. The differences in the two dendograms highlight the fact that each feature extraction technique and each distance metric obtain unique similarities and dissimilarities within a set of images. No general guidelines are available to assist the user in making the "proper" choice of methods.

D. Example Runs

Two different sets of images were used to thoroughly test this pattern recognition system. The first test involved classifying three different types of geometric shapes, rectangle, triangle and arrow. The arrow was chosen because it is a combination of a rectangle and triangle. The area of the images varied from 26 pixels to 180. Rotations from 0 degrees to 45

Figure 29. Dendogram- Zernicke

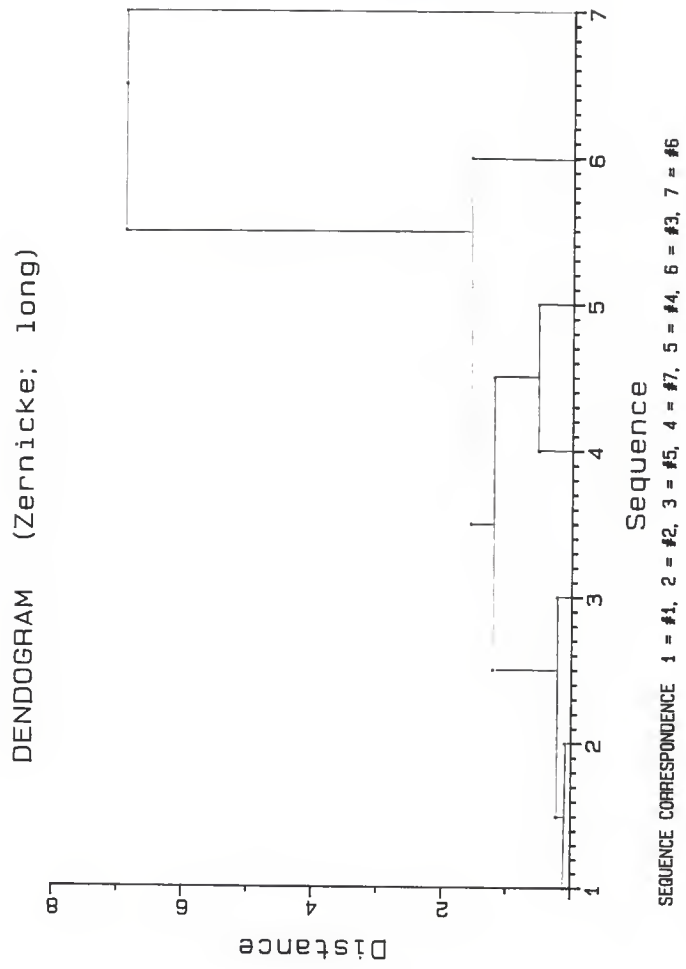
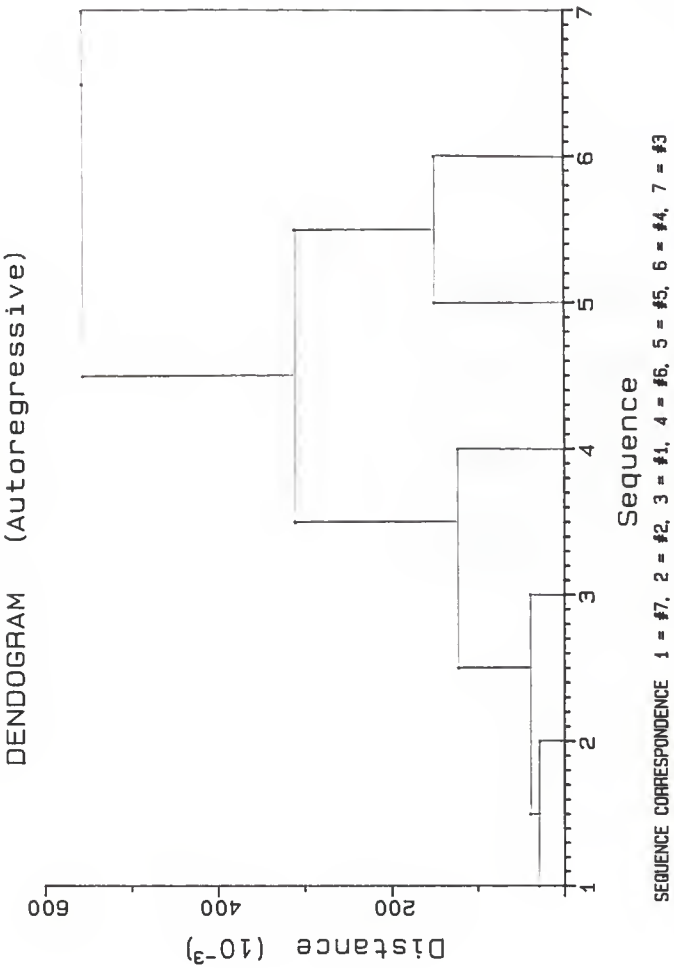


Figure 30. Dendogram - Autoregressive



degrees were also included. Many different combinations of techniques were run and the results are listed in Table 1. A total of 15 images were used. The first three images were used to set up the classifier, therefore only 12 decisions were made by the system. Three categories of answers are used, the answer is correct, the answer is wrong, the answer is wrong but the second closest choice is correct. For the adaptive classifier, only second order moments were possible because of real number precision restrictions with INTERLISP on the VAX. The distance metric used for all the simple classifier examples is the Euclidean distance squared.

Table 1. Results from Test 1.

Technique	Classifier	Right	Wrong	2nd
Zernicke (all, 2 moments)	Adaptive	8	0	4
Zernicke (edge, 2 moments)	Adaptive	4	3	5
Autoreg. (2 lag, 32 rays)	Adaptive	6	3	3
Autoreg. (2 lag, 16 rays)	Adaptive	5	3	4
Zernicke (all, 2 moments)	Non-adapt.	9	0	3
Zernicke (all, 6 moments)	Non-adapt.	9	0	3
Zernicke (all, 11 moments)	Non-adapt.	7	1	4
Zernicke (edge, 2 moments)	Non-adapt.	5	6	1
Zernicke (edge, 6 moments)	Non-adapt.	5	6	1
Zernicke (edge, 11 moments)	Non-adapt.	5	6	1
Autoreg. (2 lag, 32 rays)	Non-adapt.	4	3	5
Autoreg. (2 lag, 16 rays)	Non-adapt.	3	4	5

The most important conclusion from Test 1 is that the adaptive classifier, with its varying feature weights, is less effective than the simple distance classifier. To analyze these results in more detail, the four measurements of classifier quality were calculated after every three images were input. This allows one to check class separations for the adaptive

classifier. Table 2 shows the value of quality measure Q_4 for the adaptive methods listed in Table 1, after the number of images indicated have been input.

Table 2. Q_4 Values for Adaptive Classifiers

Technique	9	12	15
Zernicke (all, 2 moments)	33.7	29.65	10.418
Zernicke (edge, 2 moments)	11.5	1.31	1.11
Autoreg. (2 lag, 32 rays)	147.5	168.2	7.65
Autoreg. (2 lag, 16 rays)	13.0	16.7	10.7

From these results, one can see that part of the reason that the adaptive classifiers are doing poorly is that the between class separations are decreasing, and within class separations are growing. This type of classifier does not work well for sets of images with overlapping partitions. No improvement is obtained as the number of images increases.

Another important result of this test is that the Zernicke moment technique, using all the image points, provides the best feature vector for classification

purposes. This was expected, after the analysis of the scaled and rotated squares. Also, the autoregressive technique is shown to be more efficient than the Zernicke technique, using just the edge points. Not much improvement was found by increasing the number of rays from 16 to 32. The last conclusion that can be made about this test is that the amount of information contained in the third and fourth order moments for the Zernicke technique is not great. For the non-adaptive classifier, reducing the number of moments did not affect the classification success much.

The second test of this pattern recognition system used a set of block capital letters, "A", "E", and "I". The area of the letters was not changed much but their height and width were. Edges were also smeared, as if noise had degraded the original letters. Some of the letters were also placed on their sides. Results from this test were similar to the above test, and are contained in Table 3. No record was kept of answers that were wrong but had the second guess correct. A total of 15 images were used, with the first three forming the classifier.

Table 3. Results from Test 2.

Technique	Classifier	Right	Wrong
Zernicke (all, 11 moments)	Adaptive	7	6
Zernicke (edge, 11 moments)	Adaptive	6	7
Zernicke (all, 2 moments)	Adaptive	8	5
Zernicke (edge, 11 moments)	Adaptive	6	7
Autoreg. (2 lag, 32 rays)	Adaptive	5	8
Autoreg. (2 lag, 16 rays)	Adaptive	8	5
Zernicke (all, 2 moments)	Non-adapt.	10	3
Zernicke (all, 11 moments)	Non-adapt.	11	2
Zernicke (edge, 2 moments)	Non-adapt.	8	5
Zernicke (edge, 11 moments)	Non-adapt.	7	6
Autoreg. (2 lag, 32 rays)	Non-adapt.	6	7
Autoreg. (2 lag, 16 rays)	Non-adapt.	3	10

The results are similar to those from Test 1, but the adaptive classifier performs even worse than before, for the Zernicke technique. The adaptive classifier is helpful in the autoregressive technique. Once again, the Zernicke technique, using all the image points, provided the best feature vector for classification purposes. The higher order moments

contributed little, or nothing to the classification process. Classification actually got worse with the higher order moments. This is realistic because higher order moments are likely be more sensitive to noise. The classifier quality measure, Q_4, was calculated in the middle and the end of this test. In all adaptive cases, its value decreased. This is an indicator that the image classes were overlapping again. In this test the Zernicke technique, using just the edge points, was more successful than the autoregressive technique. No clear advantage between 16 and 32 rays can be noted, since 16 rays was better for the adaptive classifier and 32 rays was better for the non-adaptive classifier.

E. Summary

A comprehensive pattern recognition system was developed in software, using INTERLISP, that allows the user to input, store, analyze, extract features, and classify any 2-dimensional, binary geometric object. The user can choose between two different statistical feature extraction techniques (autoregressive model or Zernicke moments), and two different types of classifiers (simple or weighted). Several different distance metrics are available for the simple distance

classifier. Libraries of images, containing the image classes used in the classification process, can be input from a file, built dynamically (adaptively), and stored back into a file. Measurements of the quality of the library image classes can be calculated, and are especially useful when an adaptive classifier is used. If no library of images is available or desirable, the user can analyze the similarities and dissimilarities between a set of input images using a technique called clustering. The user chooses among these options in response to a series of system prompts.

After running a series of images through the pattern recognition system, the following conclusions were made. The Zernicke moment technique, using all the image points, provided the best feature vector for classification purposes. It was the most invariant to translational, scale and rotational transformations. No clear second best technique was found. CPU time will be an important factor for large images (greater than 500 pixels), especially for the Zernicke technique using all the image points.

The weighted classifier, although more sophisticated

than the simple distance classifier, did not prove to be the more accurate of the two. When there is a possibility of having blurred or noisy images in each image class, the simple, non-adaptive distance classifier may prove to be a more accurate classifier than the weighted one.

The object-oriented flavor system developed independently of the pattern recognition system provided an excellent software development environment. A set of eight INTERLISP functions comprise this flavor system. These eight functions can be used independently of the pattern recognition software, allowing the user to develop other systems utilizing the features of object-oriented programming. Object-oriented programming provides the user with data abstraction, information hiding and protection, late variable bindings and inheritance. In software systems, such as this pattern recognition one, object-oriented programming provides an excellent way to represent, manipulate and display system knowledge (knowledge representation). This type of environment, because of its modularity, enables the developer to easily modify and/or add new functions (methods) to the

system, without having to change a lot of code. The overall structure of the system is more visible to the user.

Future enhancements for this system could include more feature extraction techniques and more sophisticated classifiers. A more sophisticated classifier is needed for overlapping image regions. A good, high-level image processing section would assist the classifier in making choices between two, close image classes. Syntactic techniques would allow this pattern recognition system to identify lines, edges and corners. More sophisticated techniques would allow the system to analyze the relationship between objects. These techniques will be necessary if 3-dimensional image processing is desired. Adding an interface to a camera would allow the system to be used directly in image processing. A more distant, future enhancement would take advantage of a multi-processor programming environment to calculate the Zernicke moments more rapidly. The algorithm used in the function is ideal for division among several processors, and would make a real-time system possible.

REFERENCES

[1] G. L. Simons, Introducing Artificial Intelligence. Manchester, Great Britain: The National Computing Centre Limited, 1984, p. 132.

[2] P. Dunbar, "Machine Vision," Byte, vol. 11, pp. 161-173, January 1986.

[3] R. C. Gonzalez and P. Wintz, Digital Image Processing. Reading, MA: Addison-Wesley Publishing Company, 1983, pp. 199-217.

[4] ---, pp. 115-135.

[5] ---, pp. 136-150.

[6] ---, pp. 154-165.

[7] H. Freeman, "Computer Processing of Line-Drawing Images," Computing Surveys, vol. 6, no. 1, March 1974.

[8] E. A. Lord and C. B. Wilson, The Mathematical Description of Shape and Form. New York, NY: John Wiley and Sons, 1984, p. 220.

[9] ---, pp. 34-36.

[10] S. R. Dubois and F. H. Glanz, "An Autoregressive Approach to Two-Dimensional Shape Classification," IEEE Transactions of Pattern Analysis and Machine Intelligence, vol. PAMI-8, pp. 55-66, January 1986.

[11] R. L. Kashyap and R. Chellappa, "Stochastic models for closed boundary analysis: Representation and reconstruction," IEEE Transactions on Information Theory, vol. IT-27, pp. 627-637, 1981.

[12] M. R. Teague, "Image analysis via the general theory of moments," Journal of the Optical Society of America, vol. 70, pp. 920-930, August 1980.

[13] M. K. Hu, "Visual Pattern Recognition by Moment Invariants," IRE Transactions of Information Theory, vol. IT-8, pp. 179-187, 1962.

[14] R. C. Gonzalez and P. Wintz, Digital Image Processing. Reading, MA: Addison-Wesley Publishing Company, 1983, p. 356.

[15] W. F. Trench, Advanced Calculus. New York, NY: Harper and Row, Publishers, 1978, pp. 652-656.

[16] D. Cyganski and J. A. Orr, "Applications of Tensor Theory to Object Recognition and Orientation Determination," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-7, pp. 671-672, November 1985.

[17] M. Born and E. Wolf, Principles of Optics. New York, NY: Pergamon Press, 1975, pp. 459-468, 767-772.

[18] R. C. Gonzalez and P. Wintz, Digital Image Processing. Reading, MA: Addison-Wesley Publishing Company, 1983, p. 366.

[19] P. A. Devijver and J. Kittler, Pattern Recognition: A Statistical Approach. Englewood Cliffs, NJ: Prentice/Hall International, 1982, pp. 9-11.

[20] ---, p. 10.

[21] G. S. Sebestyen, Decision-Making Processes in Pattern Recognition. New York, NY: The Macmillan Company, 1962, pp. 17-25.

[22] P. A. Devijver and J. Kittler, Pattern Recognition: A Statistical Approach. Englewood Cliffs, NJ: Prentice/Hall International, 1982, pp. 233-239.

[23] ---, p. 236.

[24] ---, p. 239.

[25] J. Zupan, Clustering of Large Data Sets. Letchworth, Great Britain: John Wiley and Sons, 1982.

[26] A. Church, The Calculi of Lambda-Conversion. Princeton, NJ: Princeton University Press, 1941.

[27] Xerox Corporation, INTERLISP Reference Manual. Xerox Corporation, 1983.

[28] D. Partridge, Artificial Intelligence: Applications in the Future of Software Engineering. New York, NY: John Wiley and Sons, 1986, pp. 180-181.

[29] S. H. Kaisler, INTERLISP The Language and Its Usage. New York, NY: John Wiley and Sons, 1986.

[30] D. Partridge, Artificial Intelligence: Applications in the Future of Software Engineering. New York, NY: John Wiley and Sons, 1986, p. 25.

[31] F. Hayes-Roth, Building Expert Systems. Reading, MA: Addison-Wesley Publishing Company, 1983, pp. 10-100.

[32] R. H. Michaelson, D. Michie and A. Boulanger, "Technology of Expert Systems," Byte, vol. 10, pp. 303-312, April 1985.

[33] G. A. Pascoe, "Elements of Object-Oriented Programming," Byte, vol. 11, pp. 139-144, August 1986.

[34] ---, p. 140.

[35] Symbolics, Incorporated, FLAV Objects, Message Passing, and Flavors. Symbolics, Incorporated, February 1984.

Appendix I. Sample Session

NOTE: This program does not run well on the graphics terminals because of their continuous scrolling. For better results, use the other terminals and press CONTROL NO-SCROLL. This eliminates continuous scrolling for these terminals.

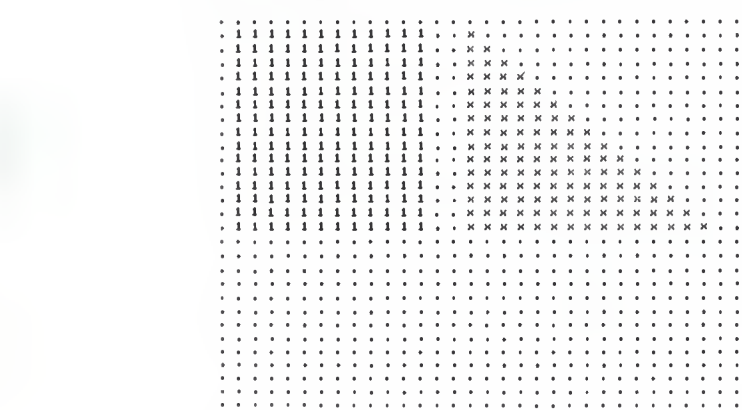
SAMPLE SESSION OF THE PATTERN RECOGNITION SYSTEM

```
% ILISP -NOREET

_(LOAD 'BEGIN)
File Created: 8-APR-87 10:45:42
BEGINCOMS

EE1:<STOUT.COMPILED>FORMAT.;15
File Created: 8-APR-87 10:13:42
FORMATCOMS

EE1:<STOUT.COMPILED>START_SYSTEM.;14
compiled on  8-APR-87 13:10:48
START_SYSTEMCOMS


To begin type  (Skeleton)   after the BEGIN file
name is output and the underline prompt character is output.

EE1:<STOUT>BEGIN.;8
_(Skeleton)
WELCOME TO AN OBJECT-ORIENTED PATTERN RECOGNITION SYSTEM.

Do you want a file of this session? (DEFAULT yes)  nno


LOADING SYSTEM FILES...Please standby.
compiled on  4-MAR-87 13:33:26
IN_OUTPUTCOMS
compiled on  4-APR-87 20:45:33
CENTERCOMS
compiled on  4-APR-87 21:46:28
TRACECOMS
compiled on  2-MAR-87 21:17:11
FLAVORCOMS
compiled on  2-APR-87 20:22:55
MOMENTCOMS
compiled on 10-MAR-87 15:15:58
REDCOMS
compiled on  2-APR-87 20:47:36
CLASSIFYCOMS
compiled on  1-APR-87 10:28:22
MATRIXCOMS
compiled on  5-APR-87 13:07:19
SET_UP_SYSTEMCOMS

Creating initial FLAVOR environment.  Please standby.
```

```
The following FLAVOR environment has been created:


The present flavor environment for this program consists of
the following flavors and instances:

FLAVOR - CLASSIFIER has 1 instance(s) in existence.
         The list of instance names is -
         (CLASSIFIER.1.INTERMEDIATE.PROCESS.1)

FLAVOR - INTERMEDIATE.PROCESS has 1 instance(s) in existence.
         The list of instance names is -
         (INTERMEDIATE.PROCESS.1)

FLAVOR - IMAGE has 0 instance(s) in existence.
FLAVOR - MATRIX has 0 instance(s) in existence.
FLAVOR - LIBRARY has 0 instance(s) in existence.
FLAVOR - GENERAL.INFORMATION has 1 instance(s) in existence.
         The list of instance names is - (GENERAL.INFORMATION)

FLAVOR - VANILLA has 1 instance(s) in existence.
         The list of instance names is - (VANILLA)


What do you want to do with your input images?  Select 1 or 2.

1. Find closest match to a set of library images. (DEFAULT)
2. Cluster the images.

1


All data files must be in the <STOUT.DATA> directory.

The following is a list of files in <STOUT.DATA>.


(EE1:<STOUT.DATA>ALPHA.#3 EE1:<STOUT.DATA>CLUSTER.#6 EE1:<STOUT.DATA>SAMPLE.#2
EE1:<STOUT.DATA>SHAPE.#3 EE1:<STOUT.DATA>SQUARE_19.#2
EE1:<STOUT.DATA>TEST_PATTERN32.#3 EE1:<STOUT.DATA>TEST_SQUARES.#10
EE1:<STOUT.DATA>TEST_SQ_ROT.#7)
```

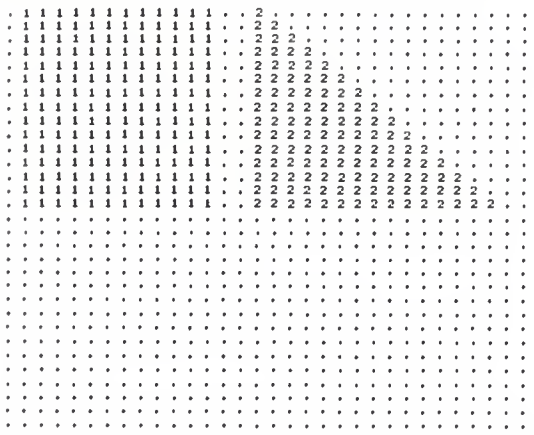
```
Do you have any files containing matrices with images
to be analyzed? These matrices should not have been previously
formatted within the INTERLISP environment by the Store.Pattern.Matrix
method.  (DEFAULT yes)  yes


Input the list of file names containing the unformatted matrices.  Begin
with a left parenthesis and end with a right paranthesis only if more
than one file name is to be input.  Continue on more than one line
if necessary.

SAMPLE

WHAT NAME WOULD YOU LIKE TO GIVE THIS PATTERN MATRIX?
ONE

Do you have any files containing matrices with images
to be analyzed? These matrices should already have been previously
formatted within the INTERLISP environment by the Store.Pattern.Matrix
method.  (DEFAULT no)  no


Which technique to you want to use to form the feature vector?
Select 1, 2 or 3.

1. Auto-regressive
2. Zernicke moments with edge points. (DEFAULT)
3. Zernicke moments using edge and interior points.

1

What parameters do you want for the auto-regressive technique?
Select 1, 2 or 3.

1. 1 log components and 16 slopes.
2. 2 log components, 16 slopes. (DEFAULT)
3. 2 log components, 32 slopes.

2

Which library of images do you want to compare yours
with? Select 1 or 2.

1. Formatted standard library. (DEFAULT)
2. One of my formatted libraries in subdirectory LIBRARY.
3. No library! create it dynamically as images are analyzed!
The adaptive classifier is the only possible classifier.

3
```

```
Do you want to calculate measures of classifier
quality? (DEFAULT was)  no

Analyzing IMAGE.1.MATRIX.1  Please standby.

Calculating the area and center of area for IMAGE.1.MATRIX.1  Please standby.

Creating feature vector for IMAGE.1.MATRIX.1  Please standby.

IMAGE.1.MATRIX.1 has been analyzed.  The following matrix contains this
image,
which is numbered 1
```



```
Pausing 20 seconds to allow for identification of the above image.
```

```
IMAGE.1.MATRIX.1 has an area = 180
center of area = (9.0 7.5)
no of holes = 0


No issues in the library. Answer NO to the next question
if this is an adaptive classifier.

Was the above classification of the issue correct?  (DEFAULT yes) no

Input the correct name for the issue.
SQUARE


Select one of the below to output the variable values for the current
instance of the flavor mentioned.

1. None.
2. IMAGE) includes MATRIX.  (DEFAULT)
3. CLASSIFIER) includes INTERMEDIATE.PROCESS.
4. LIBRARY.
5. All 3 of the above.

1
Analyzing IMAGE.2.MATRIX.1  Please standby.


Calculating the area and center of area for IMAGE.2.MATRIX.1  Please standby.


Creating feature vector for IMAGE.2.MATRIX.1  Please standby.


Applying CLASSIFIER to IMAGE.2.MATRIX.1  Please standby.


IMAGE.2.MATRIX.1 has been analyzed.  The following matrix contains this
issue,
which is numbered 2
```



```
Pausine 20 seconds to ellow for identification of the above ieaee.

1MAGE.2.MATRIX.1 has an area = 120
center of area = (11.3333 20.6667)
no of holes = 0

Of the followine list of library ieaees: (SQUARE) 1MAGE.2.MATRIX.1
 is eost sieilar to SQUARE


Was the above classification of the ieaee correct?  (DEFAULT yes) no

Inrut the correct naee for the ieaee.
TRIANGLE
```

```
Select one of the below to output the variable values for the current
instance of the flavor mentioned.

1. None.
2. IMAGE: includes MATRIX.  (DEFAULT)
3. CLASSIFIER: includes INTERMEDIATE,PROCESS.
4. LIBRARY.
5. All 3 of the above.

1

Do you want to store the present library in a file? (DEFAULT no)  yes

The following are the current library files in subdirectory LIBRARY:

(EE:<STOUT.LIBRARY>STANDARD_LIBRARY.;4)

Input the name for the new library file to be
stored in the LIBRARY subdirectory.

NEW_LIBRARY

Are you completely done? (DEFAULT yes)  yes

_(LOGOUT T)
```

APPENDIX II-A. File BEGIN

```
(FILECREATED " 8-APR-87 10:45:42" EE:<STOUT>BEGIN.;8 671

      changes to:  (VARS BEGINCONS)

      previous date: " 8-APR-87 10:20:46" EE:<STOUT>BEGIN.;7)


(PRETTYCOMPRINT BEGINCONS)

(RPAQQ BEGINCONS ((FILES EE:<STOUT.COMPILED>FORMAT EE:<STOUT.COMPILED>START_SYSTEM.V)
                  (P (PRINTOUT T T
      'To begin type  (Skeleton)   after the BEGIN file
needs output and the underline prompt character is output.'
                                 T T))))
(FILESLOAD EE:<STOUT.COMPILED>FORMAT EE:<STOUT.COMPILED>START_SYSTEM.V)
(PRINTOUT T T
      'To begin type  (Skeleton)   after the BEGIN file
needs output and the underline prompt character is output.'
         T T)
(DECLARE: DONTCOPY
  (FILEMAP (NIL)))
STOP
```

APPENDIX II-B. File FORMAT

```
(FILECREATED " 8-APR-87 10:13:42" EE:<STOUT>FORMAT.;2 1230

      changes to:  (VARS FORMATCOMS COPYRIGHTFLG FILELINELENGTH LINESPERPAGE)

      previous date: "13-MAR-87 15:40:19" EE:<STOUT.COMPILED>FORMAT.;14)


(PRETTYCOMPRINT FORMATCOMS)

(RPAQQ FORMATCOMS [(VARS FONTCHANGEFLG COPYRIGHTFLG CLISPFLG CLISPIFTRANFLG CLISPIFYPACKFLG LCASEFLG
                         #CAREFULCOLUMNS DONTPAUSE GCGAG FILELINELENGTH LINESPERPAGE)
                  (I.S.OPRS SUM PRODUCT)
                  (DECLARE: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS (ADDVARS (MLAMA)
                                                                                       (MLAML)
                                                                                       (LAMA]

(RPAQQ FONTCHANGEFLG NIL)

(RPAQQ COPYRIGHTFLG NEVER)

(RPAQQ CLISPFLG TYPE-IN)

(RPAQQ CLISPIFTRANFLG T)

(RPAQQ CLISPIFYPACKFLG NIL)

(RPAQQ LCASEFLG NIL)

(RPAQQ #CAREFULCOLUMNS 25)

(RPAQQ DONTPAUSE T)

(RPAQQ GCGAG NIL)

(RPAQQ FILELINELENGTH 123)

(RPAQQ LINESPERPAGE 50)
(DECLARE: EVAL@COMPILE
[I.S.OPR (QUOTE SUM)
         (QUOTE (SETQ ##VAL (FPLUS ##VAL BODY)))
         (QUOTE (FIRST (SETQ ##VAL 0]
[I.S.OPR (QUOTE PRODUCT)
         (QUOTE (SETQ ##VAL (FTIMES ##VAL BODY)))
         (QUOTE (FIRST (SETQ ##VAL 1]
)
(DECLARE: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS

(ADDTOVAR MLAMA )

(ADDTOVAR MLAML )

(ADDTOVAR LAMA )
)
(DECLARE: DONTCOPY
  (FILEMAP (NIL)))
STOP
```

APPENDIX II-C. File CENTER

(FILECREATED " 7-APR-87 14:53:55" EE:<STOUT>CENTER.;3 26754

previous date: " 7-APR-87 14:48:54" EE:<STOUT>CENTER.;2)

(PRETTYCOMPRINT CENTERCOMS)

```
(RPAQQ CENTERCOMS ((* CALLED BY: User. PURPOSE: creates a counter whose value is maintained on the stack frame and can
                      have multiple versions running independently. Used to keep track of the number of images found on
                      one matrix. REQUIRED VARIABLES: name of the particular instance of the counter to be created.
                      RETURNS: a stack pointer when the counter is created, all other times when called using APPLY, it
                      returns an integer value. CALLS: none.)
                   (FNS Make.Counter)
                   (* CALLED BY: METHOD of FLAVOR IMAGE. PURPOSE: numerically sorts the list of edge points by row then by
                      column. Useful to simplify the area calculations. REQUIRED VARIABLES: list of edge points and the
                      number of edge points. RETURNS: an array containing the sorted edge points from lowest to highest.
                      CALLS: Swap.Sort)
                   (FNS Shell.Sort.Edge)
                   (* CALLED BY: Shell.Sort.Edge. PURPOSE: Swaps two elements in an array that are numerically out of
                      sequence. REQUIRED VARIABLES: the array containing the coordinates, pointer into the array for the
                      first point and the size of the gap between the two points to be swapped. RETURNS: NIL. CALLS: none.)
                   (FNS Swap.Sort)
                   (* CALLED BY: METHOD of FLAVOR IMAGE. PURPOSE: An important function because it does several different
                      things. It finds the row and column sums necessary to calculate the center of area for the image in
                      the matrix by counting the number of image pixels in each row and column of the matrix. It also
                      places all the interior points into a list and finds potential hole points. REQUIRED VARIABLES:
                      matrix name; an array containing all the sorted image edge points; the number of edge points in the
                      sorted array and the background pixel character. RETURNS: list containing row and column sums, where
                      each row entry is a list of row number and quantity. Column sums are formatted by a later call to
                      Col.Sums. Also returns a possible list of hole points in hole.list and a list containing all the
                      interior image points and the number of interior points. CALLS: Check.For.Image.Points.)
                   (FNS Find.Image.Points)
                   (* CALLED BY: Find.Image.Points. PURPOSE: Loops through a row in the matrix checking for image points.
                      The row search is bounded by the values of first.coord.c and next.coord.c from Find.Image.Points.
                      Also keeps track of potential hole points. REQUIRED VARIABLES: none. All variables used in this
                      routine are obtained from Find.Image.Points. RETURNS: nothing. Returns when the row search
                      perameters are exhausted. CALLS: Find.Value.)
                   (FNS Check.For.Image.Points)
                   (* CALLED BY: METHOD of FLAVOR IMAGE. PURPOSE: Sets the edge and interior points of an image to a
                      numeric value which is one larger than the current number of images already processed in the input
                      matrix. REQUIRED VARIABLES: matrix name, list of edge points and interior points and the numeric
                      value to be written on the matrix for this image. RETURNS: updated matrix with the image displayed
                      with numeric values. CALLS: Set.Value.)
                   (FNS Replace.Image.Points)
                   (* CALLED BY: METHOD of FLAVOR IMAGE. PURPOSE: finishes the job of calculating the column sums. Called
                      after Find.Image.Points to format the column sums in list form, since the row sums are already
                      properly formatted. REQUIRED VARIABLES: column list containing all the column numbers for image
                      coordinates. RETURNS: column list containing column numbers and quantities. CALLS: none.)
                   (FNS Col.Sums)
```

```
                (* CALLED BY: METHOD of FLAVOR IMAGE. PURPOSE: calculates the numerator for the center of area of the
                   image by multiplying the row/col number by the quantity for each entry and summing all of them.
                   REQUIRED VARIABLES: properly formatted list of row or column sums ie. individual lists of row/col
                   number and quantity. RETURNS: numerator for the center of area calculations. CALLS: none.)
                (FNS Calculate.Center)
                (* CALLED BY: METHOD of FLAVOR IMAGE. PURPOSE: calculates the number of holes in a given image.
                   REQUIRED VARIABLES: matrix name, list of exterior edge points, list of potential hole points
                   generated by Find.Image.Points, the size of one dimension of the matrix and the character value of
                   an image point. RETURNS: the number of holes in the image and the edge points for each hole. CALLS:
                   Follow.Hole.Edge)
                (FNS Find.No.Of.Holes)
                (* CALLED BY: Find.No.Of.Holes. PURPOSE: attempts to trace around a hole to see if it is completely
                   enclosed by the image. REQUIRED VARIABLES: matrix name, the starting location of a potential hole
                   edge point, the size of one dimension of the matrix and the character value of an image pixel. The
                   routine is similar to Follow.Edge. RETURNS: list of edge points for the hole. CALLS: Next.Coord and
                   Get.Next.Hole.Dir.)
                (FNS Follow.Hole.Edge)
                (* CALLED BY: Follow.Hole.Edge. PURPOSE: calculates the sequence of directions to be searched to try
                   and locate the next hole edge point. Unlike tracing the exterior edge, this routine attempts to find
                   the minimum size of the hole. REQUIRED VARIABLES: previous direction in which a hole edge point was
                   found. RETURNS: ordered list of directions for the search routine. CALLS: none.)
                (FNS Get.Next.Hole.Dir)))
```

C-2

```
(* CALLED BY: User. PURPOSE: creates a counter whose value is maintained on the stack frame and can have multiple versions
running independently. Used to keep track of the number of images found on one matrix. REQUIRED VARIABLES: name of the
particular instance of the counter to be created. RETURNS: a stack pointer when the counter is created, all other times
when called using APPLY, it returns an integer value. CALLS: none.)

(DEFINEQ

                                                                                                        [1]

(Make.Counter
  [LAMBDA (count.1)                                          (* edited: '13-FEB-87 08:53')
    (FUNCTION [LAMBDA NIL
                (PROG1 count.1 (SETQ count.1 (ADD1 count.1]
              (count.1])
)


(* CALLED BY: METHOD of FLAVOR IMAGE. PURPOSE: numerically sorts the list of edge points by row then by column. Useful to
simplify the area calculations. REQUIRED VARIABLES: list of edge points and the number of edge points. RETURNS: an array
containing the sorted edge points from lowest to highest. CALLS: Swap.Sort)
```

```
(DEFINEQ

(Shell.Sort.Edge                                                    [2]
  [LAMBDA (coord.list.l no.coord.i)                    (* edited: '27-FEB-87 11:34')

          (* * KEY VARIABLES: array.counter.i is the value of the array element presently being compared.
          -
          gap.size.i is the integer value added to array.counter.i to find the second array element in the above
          comparison. -
          -
          done.b is a boolean that when TRUE indicates that the coordinates have been sorted.
          -
          sorted.array is the array that eventually contains the sorted list of edge points from lowest to
          highest.)


     (PROG ((array.counter.i 1)
            (gap.size.i (IQUOTIENT no.coord.i 6))
            (done.b NIL)
            (sorted.array (ARRAY no.coord.i 0 NIL)))

          (* * Place the coordinate list into the sorted array.)


          [MAPCAR coord.list.l (QUOTE (LAMBDA (temp)
                                              (SETA sorted.array array.counter.i temp)
                                              (ADD array.counter.i 1]

          (* * Sort the coordinate points using a version of QUICKSORT, first by row then by column.)


          (IF (ILESSP gap.size.i 1)
              THEN (SETQ gap.size.i 1))                    (* min gap size is 1)
          (DO (SETQ done.b T)

          (* * Loop until the array is sorted.)


              (SETQ array.counter.i 1)

          (* * Compare the row numbers of two array elements, the first at location array.counter.i, the second
          at array.counter.i plus gap.size.i. IF the two elements are out of order THEN swap them by calling
          Swap.Sort.)
```

```
            (DO [IF [IGREATERP (CAR (ELT sorted.array array.counter.i))
                               (CAR (ELT sorted.array (IPLUS array.counter.i gap.size.i]
                        THEN (SETQ done.b (Swap.Sort sorted.array array.counter.i gap.size.i))

        (* * ELSEIF the two elements are in the same row THEN compare their columns and swap if appropriate.)


                ELSEIF [EQ (CAR (ELT sorted.array array.counter.i))
                           (CAR (ELT sorted.array (IPLUS array.counter.i gap.size.i]
                       THEN (IF [IGREATERP (CADR (ELT sorted.array array.counter.i]
                                           (CADR (ELT sorted.array (IPLUS array.counter.i gap.size.i]
                                THEN (SETQ done.b (Swap.Sort sorted.array array.counter.i gap.size.i]

        (* * Increment the array pointer and loop back.)


              (ADD array.counter.i 1) REPEATUNTIL (IGREATERP (IPLUS array.counter.i gap.size.i)
                                                             no.coord.i)))

        (* * Divide the gap size by 2 and begin the next loop.)


            (IF (IGREATERP gap.size.i 1)
                THEN (SETQ gap.size.i (IQUOTIENT gap.size.i 2))
                     (SETQ done.b NIL)
                     (IF (ILESSP gap.size.i 1)
                         THEN (SETQ gap.size.i 1)))
           REPEATUNTIL (EQ done.b T))
        (RETURN (LIST sorted.array))
)


(* CALLED BY: Shell.Sort.Edge. PURPOSE: Swaps two elements in an array that are numerically out of sequence. REQUIRED
VARIABLES: the array containing the coordinates, pointer into the array for the first point and the size of the gap between
 the two points to be swapped. RETURNS: NIL. CALLS: none.)

(DEFINEQ

                                                                                                        [3]

(Swap.Sort
  [LAMBDA (coord.array counter.i gap.i)                    (* edited: " 8-JUL-86 10:38")

        (* * KEY VARIABLES: temp.c is a temporary variable used to hold one of the coordinates to be swapped.)
```

```
    (PROG ((temp.c (ELT coord.array counter.i)))

          (* * Using a temporary variable swap two array elements which are out of order.)


          (SETA coord.array counter.i (ELT coord.array (IPLUS counter.i gap.i)))
          (SETA coord.array (IPLUS counter.i gap.i)
                temp.c)
          (RETURN NIL))
)


(* CALLED BY: METHOD of FLAVOR IMAGE. PURPOSE: An important function because it does several different things. It finds the
 row and column sues necessary to calculate the center of area for the image in the matrix by counting the number of image
pixels in each row and column of the matrix. It also places all the interior points into a list and finds potential hole
points. REQUIRED VARIABLES: matrix name, an array containing all the sorted image edge points, the number of edge points in
 the sorted array and the background pixel character. RETURNS: list containing row and column sums, where each row entry is
 a list of row number and quantity. Column sums are formatted by a later call to Col.Sums. Also returns a possible list of
hole points in hole.list and a list containing all the interior image points and the number of interior points. CALLS:
Check.For.Image.Points.)

(DEFINEQ
```

C-5

[4]

```
(Find.Image.Points
  [LAMBDA (sort.array matrix.name no.edge.pts.i background.pixel.ch)
                                                  (* edited: " 4-APR-87 20:38")

          (* * KEY VARIABLES: first.ptr.i is a pointer into sort.array and represents the lower bound array
          element presently being processed. -
          -
          next.ptr.i is a pointer into sort.array and represents the upper bound array element being processed.
          -
          first.coord.c is the coordinate value of the array element pointed to by first.ptr.i.
          -
          next.coord.c is the coordinate value of the array element pointed to by next.ptr.i.
          -
          col.list is a list of columns that contain image points. -
          -
          row.counter.i is a counter for the number of image points in a particular row. -
          -
          hole.list is a list of points that may be hole points. -
          -
          hole.flag.b is a boolean that when TRUE means a non-image point has been found between two other image
          points in the same row. -
          -
          list.of.interior.pts is a list of points in the interior of the image. -
```

```
        -
        no.interior.pts is the nueber of points in list.of.interior.Pts. -
        -
        new.col.i Is the value of the column of the next point beins checked in a row. -
        -
        teep.ch Is e teeporarw character which is the value of the eatrix at a particular row and column.
        -
        old.row.I Is the row nueber of the current row beins processed. -
        -
        row.sums.l is a list containins row nuabers and ouantities and will be used to calculate the center of
        area. -
        -
        counter.I is a Inteaer pointer that when added to the coluen nueber of first.coord.c represents the
        coluen of the current coordinate point beins investisated. -
        -
        push.flas.b is a boolean that is TRUE when coordinate iease points need to be placed on the column and
        row lists. Used to aet the proper results when Check.For.Iease.Points is first called.)


    (PROG ((first.ptr.i 1)
           (next.ptr.i 2)
           (first.coord.c (ELT sort.arraw i))
           (next.coord.c (ELT sort.arraw 2))
           (col.list NIL)
           (row.counter.i i)
           (hole.list NIL)
           (hole.flas.b NIL)
           (list.of.interior.Pts NIL)
           (no.interior.pts.i 0)
           new.col.I teep.ch old.row.i row.sums.l counter.i push.flas.b)

          (* * Loop until all points in sorted.arraw have been processed. First initialize row inforaation.)


          (DO

          (* * Put the row and column information onto their lists and aet values for first and next.coord.c.
          Search throush sorted.arraw until these two coordinates are on the same row.)


              (DO (PUSH col.list (CADR first.coord.c))
                  (PUSH row.sues.l (LIST (CAR first.coord.c)
                                         i))
                  (SETQ first.coord.c (ELT sort.arraw (ADD first.ptr.i 1)))
                  (IF (NEQ first.ptr.i no.edse.Pts.i)
                      THEN (SETQ next.coord.c (ELT sort.arraw (ADD next.ptr.i 1]
                 WHILE (AND (NEQ (CAR first.coord.c)
                                 (CAR next.coord.c))
                            (NEQ first.ptr.i no.edse.pts.i)))
```

```
        (SETQ old.row.i (CAR first.coord.c))

    (% * IF the two coordinates are in the same row THEN search for interior points.)

        [IF (AND (EQ (CAR first.coord.c)
                     (CAR next.coord.c))
                 (NEQ first.ptr.i no.edge.pts.i))
            THEN

    (% * Initialize search parameters.)

                 (DO (SETQ counter.i 1)
                     (SETQ push.flag.b T)

    (% * Call Check.For.Lease.Points which loops while next.coord.c is still on the present row being
    processed.)

                     (Check.For.Lease.Points)

    (% * Completed the search between two edge coordinates in the same row so move the array pointers and
    check for completion. IF not completed then reset first and next.coord.c and see if a row change has
    occured. IF not then loop back.)

                     (SETQ first.ptr.i next.ptr.i)
                     [IF (NEQ first.ptr.i no.edge.pts.i)
                         THEN (SETQ first.coord.c (ELT sort.array first.ptr.i))
                              (SETQ next.coord.c (ELT sort.array (ADD next.ptr.i 1]
                   REPEATWHILE (AND (EQ (CAR first.coord.c)
                                        (CAR next.coord.c))
                                    (EQ (CAR first.coord.c)
                                        old.row.i)
                                    (NEQ first.ptr.i no.edge.pts.i]

    (% * End of a row so finish processing the information by incrementing pointers and no.interior.pts.i
    and PUSHing the new row and column information onto the appropriate lists. IF not completed THEN reset
    first and next.coord.c and loop back.)

        (PUSH row.sues.l (LIST (CAR first.coord.c)
                               row.counter.i))
        (PUSH col.list (CADR first.coord.c))
        (SETQ first.ptr.i next.ptr.i)
        (IF (NEQ first.ptr.i no.edge.pts.i)
            THEN (SETQ row.counter.i 1)
                 (ADD next.ptr.i 1)
```

```
                    (SETQ first.coord.c (ELT sort.array first.ptr.i))
                    (SETQ next.coord.c (ELT sort.array next.ptr.i)))
               REPEATUNTIL (EQ first.ptr.i no.edge.pts.i))
          (RETURN (LIST list.of.interior.pts no.interior.pts.i col.list row.sums.l hole.list])
)


(* CALLED BY: Find.Image.Points. PURPOSE: Loops through a row in the matrix checking for image points. The row search is
bounded by the values of first.coord.c and next.coord.c from Find.Image.Points. Also keeps track of potential hole points.
REQUIRED VARIABLES: none. All variables used in this routine are obtained from Find.Image.Points. RETURNS: nothing. Returns
 when the row search parameters are exhausted. CALLS: Find.Value.)

(DEFINEQ


(Check.For.Image.Points                                                    [5]
  (LAMBDA NIL                                    (* edited: " 4-APR-87 20:39")

          (* * first and next.coord.c are on the same row so check all the points between them to see if they are
          image, background or possible hole points.)

    (DO (IF push.flag.b
            THEN (PUSH col.list (CADR first.coord.c))
                 (ADD row.counter.i 1)
                 (SETQ push.flag.b NIL))

          (* * Set the column number for the current coordinate being checked.)


        (SETQ new.col.i (IPLUS (CADR first.coord.c)
                               counter.i))

          (* * Call to find the value of the coordinate at the current location being checked.)


        (SETQ temp.ch (Find.Value matrix.name (CAR first.coord.c)
                                  new.col.i))

          (* * See if the coordinate is an image point that has not been previously processed.
          IF it is a new image point THEN process the information.)


        (IF (AND (NEQ temp.ch background.pixel.ch)
                 (NOT (NUMBERP temp.ch)))
            THEN (PUSH col.list new.col.i)
                 (ADD row.counter.i 1)
                 (ADD no.interior.pts.i 1)
```

```
                (PUSH list.of.interior.pts (LIST (CAR first.coord.c)
                                                 new.col.i))

          (* * Toggle the hole flag. The hole flag is set so that the first background coordinate found between
          two image coordinates is placed in hole.list.)


                (IF (EQ hole.flag.b T)
                    THEN (SETQ hole.flag.b NIL))

          (* * Now have a possible hole point so PUSH the information onto the list.)


          ELSEIF (EQ hole.flag.b NIL)
            THEN (PUSH hole.list (LIST (CAR first.coord.c)
                                       (SUB1 new.col.i)))
                 (SETQ hole.flag.b T))
        (ADD counter.i 1) WHILE (NEQ (CADR next.coord.c)
                                     (1PLUS (CADR first.coord.c)
                                            counter.i)))

          (* * Used only if the above loop was not entered at all when Check.For.Image.Points was called.)


    (IF push.flag.b
        THEN (PUSH col.list (IPLUS (CADR first.coord.c)
                                   (SUB1 counter.i)))
             (IF (EQ next.ptr.i no.edge.pts.i)
                 THEN (RPLACA col.list (ADD (CAR col.list)
                                            1)))
             (ADD row.counter.i 1))
)


(* CALLED BY: METHOD of FLAVOR IMAGE. PURPOSE: Sets the edge and interior points of an image to a numeric value which is
one larger than the current number of images already processed in the input matrix. REQUIRED VARIABLES: matrix name, list
of edge points and interior points and the numeric value to be written on the matrix for this image. RETURNS: updated
matrix with the image displayed with numeric values. CALLS: Set.Value.)

(DEFINEQ


(Replace.Image.Points                                                                    [6]
  [LAMBDA (matrix.name edge.list interior.pts.list image.counter.i)
                                                          (* edited: " 1-APR-87 19:13")
```

```
        (* * KEY VARIABLE: full.list is a list of all image points, edge and interior points.)


    (PROG ((full.list (APPEND edge.list interior.pts.list))
           (image.counter.i (1MOD (SUB1 image.counter.i)
                                  10)))

          (* * Loop through all the points in the list and set their values in the matrix to the appropriate
          numeric value.)


          (MAPC full.list (QUOTE (LAMBDA (temp.c)
                                         (Set.Value matrix.name (CAR temp.c)
                                                    (CADR temp.c)
                                                    image.counter.i])
)


(* CALLED BY: METHOD of FLAVOR IMAGE. PURPOSE: finishes the job of calculating the column sums. Called after
Find.Image.Points to format the column sums in list form, since the row sums are already properly formatted. REQUIRED
VARIABLES: column list containing all the column numbers for image coordinates. RETURNS: column list containing column
numbers and quantities. CALLS: none.)

(DEFINEQ

                                                                                                                [7]

(Col.Sums
  (LAMBDA (col.list.1)                                          (* edited: "27-FEB-87 11:36")

          (* * KEY VARIABLES: counter.i is an integer counter for the number of points with a particular column
          number. -
          -
          sum.cols.1 is a list of column numbers and quantities. -
          -
          present.col.i is the present column being processed.)


    (PROG ((counter.i 1)
           (sum.cols.1 NIL)
           present.col.1)

          (* * Sort the column list from lowest to highest.)
```

```
        (SETQ col.list.1 (SORT col.list.1 (QUOTE GREATERP)))
        (SETQ present.col.i (CAR col.list.1))

        (* * Mapping function loops until all points have been processed for column number.
        Counts the number of occurences for each column.)


        [MAPC (CDR col.list.1)
              (QUOTE (LAMBDA (temp)

        (* * IF still in the same column THEN count the new point ELSE go to the next column.)


                             (IF (EQ temp present.col.i)
                                 THEN (ADD counter.i 1)
                               ELSE (PUSH sum.cols.1 (LIST present.col.i counter.i))
                                    (SETQ counter.i 1)
                                    (SETQ present.col.i temp)
        (RETURN (LIST (PUSH sum.cols.1 (LIST present.col.i counter.i])
)


(* CALLED BY: METHOD of FLAVOR IMAGE. PURPOSE: calculates the numerator for the center of area of the image by multiplying
the row/col number by the quantity for each entry and summing all of them. REQUIRED VARIABLES: properly formatted list of
row or column sums ie. individual lists of row/col number and quantity. RETURNS: numerator for the center of area
calculations. CALLS: none.)

(DEFINEQ


(Calculate.Center                                                                                          [8]
  [LAMBDA (sum.list.1)                                    (* edited: "27-FEB-87 13:43")

        (* * KEY VARIABLE: center.coord.r is a real number representing either the row or column value for the
        center of area, before the value is divided by the total area.)


    (PROG ((center.coord.r 0))

        (* * Mapping function loops through the list multiplying the row/col number by the number of points in
        that particular row/col. Sums all these together.)


        [MAPC sum.list.1 (QUOTE (LAMBDA (temp.c)
                                         (ADD center.coord.r (ITIMES (CAR temp.c)
                                                                     (CADR temp.c]
        (RETURN (LIST center.coord.r))
)
```

```
(* CALLED BY: METHOD of FLAVOR IMAGE. PURPOSE: calculates the number of holes in a given lease. REQUIRED VARIABLES: matrix
name; list of exterior edge points; list of potential hole points generated by Find.Lease.Points; the size of one dimension
 of the matrix and the character value of an image point. RETURNS: the number of holes in the lease and the edge points for
 each hole. CALLS: Follow.Hole.Edge)

(DEFINEQ


(Find.No.Of.Holes                                                                                                    [9]
  [LAMBDA (matrix.name edge.pts.l hole.pts.l size.of.array.i image.pixel.ch)
                                                        (* edited: '11-MAR-87 09:33')

          (* * KEY VARIABLES: hole.no.i is the integer number of holes found in the image.
          -
          search.list.l is a list of potential hole points after all points which are also edge points and all
          points found in previous holes have been eliminated. -
          -
          hole.info.l is a list containing the number of holes and the edge points of each hole.
          -
          one.hole.l is a list containing edge points for one hole.)


    (PROG ((hole.no.i 0)
           (search.list.l (LDIFFERENCE hole.pts.l edge.pts.l))
           (hole.info.l NIL)
           one.hole.l)

          (* * Trace the edge around one hole by calling Follow.Hole.Edge.)


          (DO (SETQ one.hole.l (Follow.Hole.Edge matrix.name (CAR search.list.l)
                                                 size.of.array.i image.pixel.ch))
              (ADD hole.no.i 1)

          (* * Process the hole information.)


              (PUSH hole.info.l (LIST hole.no.i one.hole.l))

         (* * Strip off any hole edge points just found from the list of potential hole points.)
```

```
            (SETQ search.list.1 (LDIFFERENCE search.list.1 one.hole.1)) WHILE (LISTP search.list.1))
          (RETURN (LIST hole.info.1])
)


(* CALLED BY: Find.No.Of.Holes. PURPOSE: attempts to trace around a hole to see if it is completely enclosed by the image.
REQUIRED VARIABLES: matrix name, the starting location of a potential hole edge point, the size of one dimension of the
matrix and the character value of an image pixel. The routine is similar to Follow.Edge. RETURNS: list of edge points for
the hole. CALLS: Next.Coord and Get.Next.Hole.Dir.)

(DEFINEQ

                                                                                                                [10]
(Follow.Hole.Edge
  [LAMBDA (matrix.name begin.pt.c size.of.array.i image.pixel.ch)
                                                        (* edited: '11-MAR-87 10:09')

          (* * KEY VARIABLES: next.dir.i is an ordered list of directions to search for the next hole edge point.
          -
          stop.flag.b is a boolean that when TRUE means the next hole edge point has been found.
          -
          edge.pts.l is a list of edge points for the hole. -
          -
          temp.c is the next trial hole edge point. -
          -
          temp.dir.i is the the direction the previous hole edge point was found in.)


    (PROG ((next.dir.i (QUOTE (7 6 5 4 3 2)))
           (stop.flag.b NIL)
           (edge.pts.l (CONS begin.pt.c NIL))
           temp.c temp.dir.i)

          (* * Loop until the coordinates for the next potential hole edge point have been found.)


          (DO (FOR direction.i IN next.dir.i
                 DO

          (* * Call Next.Coord to find the coordinates for the next point in a given direction.)


                   (SETQ temp.c (Next.Coord direction.i (CAR edge.pts.l)
                                            matrix.name size.of.array.i image.pixel.ch))
```

```
        (* * IF the next point is a hole edge point THEN process it.)


                (IF (LISTP temp.c)
                    THEN (SETQ stop.flag.b T)
                         (PUSH edge.pts.1 temp.c)
                         (SETQ temp.dir.i direction.i))
              UNTIL (EQ stop.flag.b T))
            (SETQ stop.flag.b NIL)

        (* * Get the next ordered list of directions to be searched by calling Get.Next.Hole.Dir.)


            (SETQ next.dir.i (Get.Next.Hole.Dir temp.dir.i)) REPEATUNTIL (EQUAL (CAR edge.pts.1)
                                                                                begin.pt.c))
        (RETURN (COR edge.pts.1))
)


(* CALLED BY: Follow.Hole.Edge. PURPOSE: calculates the sequence of directions to be searched to try and locate the next
hole edge point. Unlike tracing the exterior edge, this routine attempts to find the minimum size of the hole. REQUIRED
VARIABLES: previous direction in which a hole edge point was found. RETURNS: ordered list of directions for the search
routine. CALLS: none.)

(DEFINEQ

                                                                                                                [11]

(Get.Next.Hole.Dir
  (LAMBDA (last.dir.i)                                    (* edited: '17-JUL-86 11:18')
    (SELECTQ last.dir.i
             (0 (QUOTE (1 0 7 6 5 4)))
             (1 (QUOTE (3 2 1 0 7 6 5)))
             (2 (QUOTE (3 2 1 0 7 6)))
             (3 (QUOTE (5 4 3 2 1 0 7)))
             (4 (QUOTE (5 4 3 2 1 0)))
             (5 (QUOTE (7 6 5 4 3 2 1)))
             (6 (QUOTE (7 6 5 4 3 2)))
             (7 (QUOTE (1 0 7 6 5 4 3)))
             NIL)
)
(DECLARE: DONTCOPY
  (FILEMAP (NIL (6117 6322 (Make.Counter 6127 . 6320)) (6648 9284 (Shell.Sort.Edge 6658 . 9282)) (9591 10129 (Swap.Sort 9601
 . 10127)) (11021 15920 (Find.Image.Points 11031 . 15918)) (16362 18398 (Check.For.Image.Points 16372 . 18396)) (18839
19528 (Replace.Image.Points 18849 . 19526)) (19930 21099 (Col.Sues 19930 . 21097)) (21494 22167 (Calculate.Center 21504 .
22165)) (22581 23885 (Find.No.Of.Holes 22591 . 23883)) (24311 25921 (Follow.Hole.Edge 24321 . 25919)) (26312 26732 (
Get.Next.Hole.Dir 26322 . 26730)))))
STOP
```

APPENDIX II-D. File CLASSIFY

```
(FILECREATED " 7-APR-87 15:12:07" EE:<STOUT>CLASSIFY.;3 41562

      previous date: " 7-APR-87 15:03:48" EE:<STOUT>CLASSIFY.;2)


(PRETTYCOMPRINT CLASSIFYCOMS)

(RPAQQ CLASSIFYCOMS ((* CALLED BY: METHOD of FLAVOR CLASSIFIER. PURPOSE: makes the decision of which library class the
                        unknown image belongs to based on distance measure produced by the distance functions. Also gives
                        a measure of how good the decision is. REQUIRED VARIABLES: number of classes of library images,
                        distance array from one of the two distance classifiers, a flag indicating whether variance
                        statistics were taken by the distance measuring functions and the variance array if this statistic
                        was calculated. RETURNS: the sequence number of the closest image class, the distance to the
                        closest class, sequence number of the second closest class, the distance to the second closest
                        class, the ratio of the closest and second closest image class distances and if variance
                        statistics were taken, the ratio of the closest and second closest image class distance variances.
                        CALLS: none.)
                       (FNS Classify.Decision)
                       (* CALLED BY: METHOD of FLAVOR CLASSIFIER. PURPOSE: calculates the Euclidean distance from an unknown
                        pattern and a library of known images. Can also calculate the Minkowski metric to any order for a
                        distance measure or the Chebychev metric, which is simply the largest distance in the feature
                        vector. Can also calculate the variance of the distance calculations. REQUIRED VARIABLES: a list
                        of image array patterns for each class in the library; they must be formed by INTERLISP argument
                        LIST, the number of different library classes, the feature array of the unknown image, the number
                        of elements in each feature vector, , type of distance measure requested, the power to be used in
                        measuring the Euclidean distance for the Minkowski metric and a flag that when TRUE indicates that
                        variance statistics will be taken on the distance measurements. RETURNS: an array containing the
                        sum of the distances from each of the elements in the unknown image feature vector to each of the
                        library image class feature vectors. CALLS: none.)
                       (FNS Distance.Classifier)
                       (* CALLED BY: METHOD of FLAVOR CLASSIFIER. PURPOSE: calculates the weighted Euclidean distance from
                        the unknown image feature vector to each of the library image class feature vectors. REQUIRED
                        VARIABLES: a list of lists, each containing all the image feature vectors for one library image
                        class, each list must be formed by the INTERLISP argument LIST, the number of different library
                        image classes, the number of elements in the feature vector, the feature vector for the unknown
                        image, , a Boolean flag to indicate that the weight vectors have already been calculated and the
                        previously calculated mean and variance matrices if done.flag is TRUE. RETURNS: an array
                        containing the weighted sum of the distances from each of the elements in the unknown image
                        feature vector to each of the library image class feature vectors, the mean and variance matrices
                        used in calculating the weights and the value of the flag that indicates the weight matrices have
                        already been calculated. If only one image in a class then returns simple Euclidean distance and
                        no mean or variance values for that class. CALLS: Weight.Library or Distance.Classifier, Matrix
                        and Find.Value)
                       (FNS Weighted.Classifier)
                       (* CALLED BY: Weighted.Classifier. PURPOSE: calculates a weight for each feature element in a list of
                        library images belonging to the same class, based on the variance of the individual feature
                        elements. REQUIRED VARIABLES: the list of common images; must be input using the INTERLISP
                        function LIST, the number of images in the library class, an array to store the mean values, the
                        number of elements in the feature vector and an array to store the variance values. RETURNS: the
```

```
      mean and variance arrays. CALLS: none.)
(FNS Weight.Library)
(* CALLED BY: METHOD of FLAVOR CLASSIFIER. PURPOSE: calculates four increasingly precise measures of
   the quality of the classifier scheme used to obtain the library images. These involve comparing
   how close each image is to the others in its class and how far away each class is to all other
   classes. REQUIRED VARIABLES: list of lists, each containing the set of image class feature vectors
   for that particular library class, the number of different library classes and the number of
   elements in the feature vector. RETURNS: the four classifier measures: one through four. CALLS:
   Matrix, Find.Value, Set.Value, Inverse.Matrix, Find.Determinant, Matrix.Multiplication,
   Matrix.Addition and Find.Within.Class.Scatter.)
(FNS Quality.Of.Classifier)
(* CALLED BY: Quality.Of.Classifier. PURPOSE: Calculates the within.class.matrix. REQUIRED VARIABLES:
   none. All variables used in this routine are obtained directly from the calling routine. RETURNS:
   within.class.matrix. CALLS: Find.Value, Set.Value, Matrix.Multiplication, Matrix.Addition.)
(FNS Find.Within.Class.Scatter)
(* CALLED BY: METHOD of FLAVOR CLASSIFIER. PURPOSE: Clusters a set of image feature vectors based
   upon a measure of the closeness of each pair of images. REQUIRED VARIABLES: a list of image
   feature vectors, the number of images to be clustered, the number of elements in each feature
   vector, the type of distance measurement to be used and the power for the Minkowski metric.
   RETURNS: a list of lists, each containing the distance between the two closest image feature
   vectors and the sequence number of the two closest image vectors, for that particular iteration.
   CALLS: Matrix, Find.Value, Set.Value and Distance.Classifier.)
(FNS Clustering)))
```


(* CALLED BY: METHOD of FLAVOR CLASSIFIER. PURPOSE: makes the decision of which library class the unknown image belongs to based on distance measure produced by the distance functions. Also gives a measure of how good the decision is. REQUIRED VARIABLES: number of classes of library images, distance array from one of the two distance classifiers, a flag indicating whether variance statistics were taken by the distance measuring functions and the variance array if this statistic was calculated. RETURNS: the sequence number of the closest image class, the distance to the closest class, sequence number of the second closest class, the distance to the second closest class, the ratio of the closest and second closest image class distances and if variance statistics were taken, the ratio of the closest and second closest image class distance variances. CALLS: none.)

```
(DEFINEQ

(Classify.Decision                                                    [1]
  [LAMBDA (no.in.library,i distance.array extra.flag,b variance.array)
                                          (* edited: '25-FEB-87 16:25')

          (* * KEY VARIABLES: fuzzy.variance,r is a measurement of how 'good' the decision was of which class the
          unknown image belongs to. It is the ratio of the second closest distance variance and the closest
          distance variance. The larger this ratio is the the larger the variance of the second closest array is,
          and therefore the better the decision probably is; the more separation between the two closest classes.
          -
          min.distance,r is the distance between the unknown feature image and the closest library image class
          array. -
```

```
        -
        which.min.i is the sequence number of the closest library image class array. -
        -
        next.min.dist.r is the distance between the unknown feature image and the second closest library image
        class array. -
        -
        which.next.min.i is the sequence number of the second closest library image class array.
        -
        temp.dist.r is a temporary variable for storing the distance from one image class to the unknown image
        feature vector. -
        -
        fuzziness.r is a measurement of how "good" the decision was of which class the unknown image belongs
        to. It is the ratio of the second closest distance and the closest distance The larger this ratio is
        the the larger the larger the separation of these two classes and therefore the better the decision
        probably is.)


    (PROG ((fuzzy.variance.r NIL)
           min.distance.r which.min.i next.min.dist.r which.next.min.i temp.dist.r fuzziness.r)

          (* * Find the closest library image class to the image feature; this is the smallest value in
          distance.array. Also want to find the second smallest distance to get a measure of how good the
          separation was between the two closest library classes.)


          (SETQ min.distance.r (ELT distance.array 1))
          (SETQ which.min.i 1)
          (SETQ next.min.dist.r 9.999999E+27)

          (* * Loop to find the two smallest distance values.)


          (FOR i FROM 2 TO no.in.library.i
             DO (SETQ temp.dist.r (ELT distance.array i))
                (IF (FLESSP temp.dist.r min.distance.r)
                    THEN (SETQ next.min.dist.r min.distance.r)
                         (SETQ which.next.min.i which.min.i)
                         (SETQ min.distance.r temp.dist.r)
                         (SETQ which.min.i i)
                  ELSEIF (FLESSP temp.dist.r next.min.dist.r)
                    THEN (SETQ next.min.dist.r temp.dist.r)
                         (SETQ which.next.min.i i)))

          (* * Set the fuzziness.r parameter.)
```

```
        (SETQ fuzziness.r (FQUOTIENT next.min.dist.r min.distance.r))

        (* * IF variances were taken of the distances then calculate another fuzziness factor.)


        [IF extra.flag.b
            THEN (SETQ fuzzy.variance.r (FQUOTIENT (ELT variance.array which.next.min.i)
                                                   (ELT variance.array which.min.i]
        (RETURN (LIST which.min.i min.distance.r which.next.min.i next.min.dist.r fuzziness.r fuzzy.variance.r])
)


(* CALLED BY: METHOD of FLAVOR CLASSIFIER. PURPOSE: calculates the Euclidean distance from an unknown pattern and a library
 of known images. Can also calculate the Minkowski metric to any order for a distance measure or the Chebychev metric,
which is simply the largest distance in the feature vector. Can also calculate the variance of the distance calculations.
REQUIRED VARIABLES: a list of image array patterns for each class in the library; they must be formed by INTERLISP argument
 LIST, the number of different library classes, the feature array of the unknown image, the number of elements in each
feature vector, , type of distance measure requested, the power to be used in measuring the Euclidean distance for the
Minkowski metric and a flag that when TRUE indicates that variance statistics will be taken on the distance measurements.
RETURNS: an array containing the sum of the distances from each of the elements in the unknown image feature vector to each
 of the library image class feature vectors. CALLS: none.)

(DEFINEQ

                                                                                                        [2]

(Distance.Classifier
  [LAMBDA (library.list no.in.list.i feature.array no.features.i type.of.distance.ch power.i extra.flag.b)
                                                        (* edited: "31-MAR-87 16:17")

          (* * KEY VARIABLES: library.array is set to each one of the library image class feature vectors until
          library.list is exhausted. -
          -
          difference.array contains the difference between each element of feature.array and the corresponding
          element in one of the library feature arrays. This difference may be to any power
          1 to n. -
          -
          distance.array contains the sum of the distances from each library image class vector to the unknown
          image feature vector. -
          -
          mean.array contains the mean value for the all the feature differences, calculated from
          difference.array, one value for each library image. -
          -
          variance.array contains the variance for all the feature distances calculated from the square of the
          distance between each element in distance.array and the mean.array value. -
          -
          deviation.r is the distance between one array element and the mean array element.)
```

```
(PROG ((library.array (ARRAY no.features.i NIL NIL))
       (difference.array (ARRAY no.features.i NIL NIL))
       (distance.array (ARRAY no.in.list.i NIL NIL))
       (mean.array (ARRAY no.in.list.i NIL NIL))
       (variance.array (ARRAY no.in.list.i NIL NIL))
       deviation.r)

      (* * Loop until all library lease arrays have been processed.)


      [FOR i FROM 1 TO no.in.list.i
         DO

      (* * Get the next library lease and calculate the feature differences.)


            (SETQ library.array (POP library.list))

      (* * Make sure that library.array is not a list of common library leases. IF it is THEN take just the
      first lease in that library class.)


            (IF (LISTP library.array)
                THEN (SETQ library.array (POP library.array)))
            [FOR J FROM 1 TO no.features.i DO (SETA difference.array J (ABS (DIFFERENCE (ELT feature.array J)
                                                                                        (ELT library.array J]

      (* * IF not Chebychev THEN check what the order is for the Minkowski metric. This metric takes the
      appropriate power of each element in the difference.array and sums them. Finally that sum is raised to
      the 1/power for the final distance measure.)


            [IF (NEQ type.of.distance.ch (QUOTE Chebychev))
                THEN [IF (IGREATERP power.i 1)
                         THEN (FOR J FROM 1 TO no.features.i DO (SETA difference.array J
                                                                      (EXPT (ELT difference.array J)
                                                                            power.i)))
                              (SETA distance.array i (EXPT (FOR J FROM 1 TO no.features.i
                                                              SUM (ELT difference.array J))
                                                           (FQUOTIENT 1.0 power.i)))
                         ELSE (SETA distance.array i (FOR J FROM 1 TO no.features.i SUM (ELT difference.array J]

      (* * User want Chebychev which sets the largest element in difference.array to the distance measure to
      this library lease.)
```

```
            ELSE (SETA distance.array i (APPLY (QUOTE FMAX)
                                               (FOR J FROM 1 TO no.features.i COLLECT (ELT difference.array J]

        (* * IF the user wants more statistics THEN calculate the variance for each distance measured.)

             (IF extra.flag.b
                 THEN

        (* * Calculate the mean for the distances for one library image.)

                  (SETA mean.array i (FQUOTIENT (FOR J FROM 1 TO no.features.i SUM (ELT difference.array J))
                                                no.features.i))

        (* * Calculate the variance for the distances for one library image.)

                  (SETA variance.array i (FQUOTIENT (FOR J FROM 1 TO no.features.i
                                                        SUM (SETQ deviation.r (FDIFFERENCE (ELT difference.array J)
                                                                                           (ELT mean.array i)))
                                                            (FTIMES deviation.r deviation.r))
                                                    no.features.i]
        (IF extra.flag.b
            THEN (RETURN (LIST distance.array variance.array))
          ELSE (RETURN distance.array))
)


(* CALLED BY: METHOD of FLAVOR CLASSIFIER. PURPOSE: calculates the weighted Euclidean distance from the unknown image
feature vector to each of the library image class feature vectors. REQUIRED VARIABLES: a list of lists, each containing all
 the image feature vectors for one library image class, each list must be formed by the INTERLISP argument LIST, the number
 of different library image classes, the number of elements in the feature vector, the feature vector for the unknown
image, , a Boolean flag to indicate that the weight vectors have already been calculated and the previously calculated mean
 and variance matrices if done.flag is TRUE. RETURNS: an array containing the weighted sum of the distances from each of
the elements in the unknown image feature vector to each of the library image class feature vectors, the mean and variance
matrices used in calculating the weights and the value of the flag that indicates the weight matrices have already been
calculated. If only one image in a class then returns simple Euclidean distance and no mean or variance values for that
class. CALLS: Weight.Library or Distance.Classifier, Matrix and Find.Value)

(DEFINEQ
```

[3]

```
(Weighted.Classifier
  [LAMBDA (library.list no.in.library.i feature.array no.features.i done.flag.b mean.matrix.old variance.matrix.old)
                                                    (* edited: " 1-APR-87 16:00")

        (* * KEY VARIABLES: distance.array contains the weighted distances from each library image class to the
        unknown image feature vector. -
        -
        mean.matrix is a matrix containing the mean value for each feature element for each library image
        class. Each row contains the mean feature vectors for one image class. -
        -
        variance.matrix is a matrix containing the variance for the distance for each feature element for each
        library image class. Each row contains the variance vector for one image class.
        -
        size.of.matrix.i is the larger of no.features.i and no.in.library.i and is used to form a square
        matrix. -
        -
        temp.list is a list containing the list of library images arrays for one class.
        -
        no.in.sublist.i is the number of library images in one class. -
        -
        exception.list is a list of sequence numbers for library classes that only contain one image per class.
        -
        weight.return.list is the return value from the call to Weight.Library. It contains the mean and
        variance array for one class of images and these are placed into the appropriate row in mean and
        variance.matrix. -
        -
        variance.product.r is the product of the variances for one image class vector.)


    (PROG ((distance.array (ARRAY no.in.library.i NIL NIL))
           mean.matrix variance.matrix size.of.matrix.i temp.list no.in.sublist.i exception.list weight.return.list
           variance.product.r)

          (* * IF the weights have not been calculated for the present library THEN calculate them.)


          (IF (NOT done.flag.b)
              THEN

          (* * Find the larger of these two input numbers to create a square matrix.)


                  (SETQ size.of.matrix.i (IMAX no.in.library.i no.features.i))
                  (SETQ mean.matrix (Matrix size.of.matrix.i size.of.matrix.i 0))
                  (SETQ variance.matrix (Matrix size.of.matrix.i size.of.matrix.i 0))
```

```
        (% % Loop through each library image class.)

                (FOR i FROM 1 TO no.in.library.i
                   DO (SETQ temp.list (POP library.list)))

        (% % Check to see how many library images there are in this class. IF more than 1 THEN call
        Weight.Library ELSE call Distance.Classifier.)

                      (SETQ no.in.sublist.i (COUNT temp.list))
                      (IF (NEQ no.in.sublist.i 1)
                          THEN

        (% % Call to Weight.Library to calculate the mean and variance for one class of library images.)

                               (SETQ weight.return.list (Weight.Library temp.list (COUNT temp.list)
                                                                        (ELT mean.matrix i)
                                                                        no.features.1
                                                                        (ELT variance.matrix i)))
                               (SETQ done.flag.b T)

        (% % Place the returned arrays from Weight.Library into the matrices.)

                               (SETA mean.matrix i (POP weight.return.list))
                               (SETA variance.matrix i (POP weight.return.list))
                          ELSE

        (% % ELSE only one image in this class so use a simple Euclidean distance measure and set the value of
        distance.array directly.)

                               (PUSH exception.list i)
                               (SETA distance.array i (ELT (Distance.Classifier temp.list i feature.array no.features.i
                                                                                NIL 2 NIL)
                                                           1))
          ELSE (SETQ mean.matrix mean.matrix.old)
               (SETQ variance.matrix variance.matrix.old))

        (% % Calculate the weighted distance from the feature vector to each library class, except for the
        classes that had only one image in them. These latter ones have sequence numbers contained in
        exception.list.)
```

```
        (FOR i FROM 1 TO no.in.library.i
           DO

        (* * Calculate the product of the variances for one class.)

              (SETQ variance.product.r (EXPT (FOR J FROM 1 TO no.features.i PRODUCT (Find.Value variance.matrix i J))
                                             (FQUOTIENT 2 no.features.i)))

        (* * Reset the value of variance.product.r if it is too small for numbers in the VAX)

              (IF (FEQP variance.product.r 0.000000E+00)
                  THEN (SETQ variance.product.r 9.999997E-11))
              (SETA distance.array i (FTIMES variance.product.r
                                             (FOR J FROM 1 TO no.features.i
                                                SUM (FPLUS (EXPT (FQUOTIENT (FDIFFERENCE (ELT feature.array J)
                                                                                         (Find.Value mean.matrix i J))
                                                                            (Find.Value variance.matrix i J))
                                                                 2)
                                                           no.features.i]
           UNLESS (EQMEMB i exception.list))
        (RETURN (LIST distance.array mean.matrix variance.matrix done.flag.b])
)
```



```
(* CALLED BY: Weighted.Classifier. PURPOSE: calculates a weight for each feature element in a list of library images
belonging to the same class, based on the variance of the individual feature elements. REQUIRED VARIABLES: the list of
common images; must be input using the INTERLISP function LIST, the number of images in the library class; an array to
store the mean values, the number of elements in the feature vector and an array to store the variance values. RETURNS: the
 mean and variance arrays. CALLS: none.)

(DEFINEQ

                                                                                                                    [4]
(Weight.Library
  [LAMBDA (common.lib.list no.in.list.i mean.array no.features.i variance.array)
                                                          (* edited: " 3-MAR-87 08:57")

        (* * KEY VARIABLES: temp.list contains the list of common images. Used because POP destroys the list it
        operates on. -
        -
        zero.flag.b is TRUE only if a zero element was placed into variance.array. -
        -
        temp.one.array is a temporary array containing the value of one feature vector for an image in the
        library. -
        -
```

```
        deviation.r is the difference between one array element and the mean array element.
        ~
        temp.variance.r is a temporary variable used to hold a variance value when substituting for a zero
        variance. ~
        ~
        counter.i is an array pointer used to keep track of location in variance.array.
        ~
        smallest.r is the smallest non-zero variance in variance.array.)


    (PROG ((temp.list common.lib.list)
           (zero.flag.b NIL)
           temp.one.array deviation.r temp.variance.r counter.i smallest.r)

          (* * Loop until all common library images have been processed.)


          [FOR i FROM 1 TO no.in.list.i
             DO (SETQ temp.one.array (POP temp.list))

          (* * Loop through all the image features and calculate the mean value for each feature.)


                (FOR J FROM 1 TO no.features.i DO (SETA mean.array J (FPLUS (ELT mean.array J)
                                                                            (ELT temp.one.array JJ
          (FOR i FROM 1 TO no.features.i DO (SETA mean.array i (FQUOTIENT (ELT mean.array 1)
                                                                          no.in.list.i)))


          (* * Reinitialize temp.list)


          (SETQ temp.list common.lib.list)

          (* * Loop until all common library images have been processed.)


          [FOR i FROM 1 TO no.in.list.i
             DO (SETQ temp.one.array (POP temp.list))

          (* * Loop through all the image features and calculate the sum of the deviations for each feature.)


                (FOR J FROM 1 TO no.features.i
                   DO (SETQ deviation.r (FDIFFERENCE (ELT mean.array J)
                                                     (ELT temp.one.array J)))
                      (SETA variance.array J (FPLUS (ELT variance.array J)
                                                    (FTIMES deviation.r deviation.r]
```

```
        (% * Calculate the variance for each feature.)


        (FOR i FROM 1 TO no.features.i
           DO (SETO temp.variance.r (FQUOTIENT (ELT variance.array i)
                                               no.in.list.i))
              (IF (FEQP temp.variance.r 0.000000E+00)
                  THEN (SETO zero.flag.b T))
              (SETA variance.array i temp.variance.r))

        (% * IF one or more of the variances are zero THEN need to reprocess those elements because they will
        skew the weights for the feature vector. The new value substituted for any zero variance is chosen to
        be 20 times smaller than any other feature variance. This was an arbitrary choice.)


        [IF zero.flag.b
            THEN (SETO counter.i 1)

        (% * Loop past any leading zero variances.)


                 (DO (SETO smallest.r (ELT variance.array counter.i))
                     (ADD counter.i 1) REPEATUNTIL (OR (FGREATERP smallest.r 0.000000E+00)
                                                       (EQ counter.i no.features.i)))

        (% * IF a non-zero variance was found THEN proceed ELSE all the variances are zero so arbitrarily reset
        them all to 1.000000E-02)


                 (IF (FGREATERP smallest.r 0.000000E+00)
                     THEN

        (% * Find the smallest non-zero variance.)


                          (FOR i FROM counter.i TO no.features.i
                             DO (SETO temp.variance.r (ELT variance.array i))
                                (IF (AND (FGREATERP smallest.r temp.variance.r)
                                         (FGREATERP temp.variance.r 0.000000E+00))
                                    THEN (SETO smallest.r temp.variance.r)))
                          (SETO smallest.r (FQUOTIENT smallest.r 20))
                   ELSE (SETO smallest.r 1.000000E-02))

        (% * Replace any zero variance with the value of smallest.)
```

```
                (FOR i FROM 1 TO no.features.i DO (IF (FEQP (ELT variance.array i)
                                                            0.000000E+00)
                                                      THEN (SETA variance.array i smallest.r]
        (RETURN (LIST mean.array variance.array))
)


(* CALLED BY: METHOD of FLAVOR CLASSIFIER. PURPOSE: calculates four increasingly precise measures of the quality of the
classifier scheme used to obtain the library images. These involve comparing how close each image is to the others in its
class and how far away each class is to all other classes. REQUIRED VARIABLES: list of lists, each containing the set of
image class feature vectors for that particular library class, the number of different library classes and the number of
elements in the feature vector. RETURNS: the four classifier measures: one through four. CALLS: Matrix, Find.Value,
Set.Value, Inverse.Matrix, Find.Determinant, Matrix.Multiplication, Matrix.Addition and Find.Within.Class.Scatter.)

(DEFINEQ


(Quality.Of.Classifier                                                                                          [5]
  [LAMBDA (library.list no.in.library.i no.features.i)      (* edited: '10-MAR-87 15:36')

          (* * KEY VARIABLES: mean.matrix is a matrix that will contain the mean values for each feature in each
          separate class of library images. Each row contains the mean feature vector for one class of images.
          -
          sample.mean.array is an array that contains the population mean feature vector.
          -
          between.class.matrix is the matrix that contains a measure of the between class scatter.
          This is a measure of how each class mean vector varies around the population mean vector -
          -
          extra.matrix is a matrix that is used in the various summation processes as storage for intermediate
          results. -
          -
          new.extra.matrix is another matrix that is used in the various summation processes as storage for
          intermediate results. -
          -
          difference.matrix is actually a 1-dimension matrix that contains the difference between two feature
          vectors. -
          -
          diff.matrix.transpose is the transpose of difference.matrix. -
          -
          within.class.matrix is the matrix that contains a measure of the within class scatter.
          This is a measure of how each image in a class varies with respect to the class mean vector.
          -
          temp.list is a copy of the input parameter library.list and is needed because the POP command destroys
          the variable it is acting on. -
          -
          quality.i.r is the simplest of the four measures available that can be used to measure the overall
          quality of a classifier system based on the separability of class images. It is simply the trace of the
```

```
        sum of the within and between class matrices. -
        -
        quality.2.r is the quotient of the traces for the within and between class matrices.
        -
        quality.3.r is the third measure and is the trace of the product of the inverse of within with the
        between.class.matrix. -
        -
        quality.4.r is the last measure and is the quotient of the determinant of the sum of the two with the
        determinant of within.class.matrix. -
        -
        single.list.flag.b is a Boolean that is TRUE only when at least one library image class contains only
        one image in it. Allows for only quality.1.r to be calculated. -
        -
        temp.list is a list containing all the feature arrays for the images in one class.
        -
        no.in.sub.list.i is the number of images in one class from the library. -
        -
        temp.one.array is a temporary array that takes the value of one feature vector for an image in the
        library. -
        -
        difference.r is the difference between one element of a feature vector for an image and the mean
        feature vector element for that class. -
        -
        determinant.top.r is a temporary variable used to store the numerator used to calculate quality.4.r.
        -
        determinant.bot.r is a temporary variable used to store the denominator used to calculate quality.4.r.)


    (PROG ((mean.matrix (Matrix no.in.library.i no.features.i 0))
           (sample.mean.array (ARRAY no.features.i NIL NIL))
           (between.class.matrix (Matrix no.features.i no.features.i 0))
           (extra.matrix (Matrix no.features.i no.features.i 0))
           (new.extra.matrix (Matrix no.features.i no.features.i 0))
           (difference.matrix (Matrix no.features.i 1 0))
           (diff.matrix.transpose (Matrix 1 no.features.i 0))
           (within.class.matrix (Matrix no.features.i no.features.i 0))
           (temp.lib.list library.list)
           (quality.1.r 0)
           (quality.2.r 0)
           (quality.3.r 0)
           (quality.4.r 0)
           (single.list.flag.b NIL)
           temp.list no.in.sub.list.i temp.one.array difference.r determinant.top.r determinant.bot.r)

          (* * Loop through all the separate classes in the library.)
```

```
        [FOR i FROM 1 TO no.in.library.i
           DO (SETO temp.list (POP library.list))
              (SETQ no.in.sub.list.i (COUNT temp.list))

        (* * Check to see if this library image class contains only one image. IF it does THEN only one measure
        of classifier quality is possible.)


              (IF (EQ no.in.sub.list.i 1)
                  THEN (SETO single.list.flag.b T))

        (* * Loop until all images in one class have been processed.)


              [FOR J FROM 1 TO no.in.sub.list.i
                 DO (SETO temp.one.array (POP temp.list))

        (* * Loop through all the image features in one class and calculate the mean value for each feature.)


                    (FOR J FROM i TO no.features.i DO (Set.Value mean.matrix i J (FPLUS (Find.Value mean.matrix i J)
                                                                                        (ELT temp.one.array J)

        (* * Divide to get the mean from the sum of values.)


              (FOR J FROM 1 TO no.features.i DO (Set.Value mean.matrix i J (FQUOTIENT (Find.Value mean.matrix i J)
                                                                                      no.in.sub.list.i)

        (* * Find the population mean vector for all classes of images in the library.)


        (FOR i FROM i TO no.features.i DO (SETA sample.mean.array i (FQUOTIENT (FOR J FROM 1 TO no.in.library.i
                                                                                  SUM (Find.Value mean.matrix J i))
                                                                               no.in.library.i)))

        (* * Loop through all the separate image classes in the library and find the between class scatter
        matrix.)


        (FOR i FROM 1 TO no.in.library.i
           DO (FOR J FROM 1 TO no.features.i
                 DO (SETO difference.r (FDIFFERENCE (Find.Value mean.matrix i J)
                                                    (ELT sample.mean.array J)))
                    (Set.Value difference.matrix J 1 difference.r)
                    (Set.Value diff.matrix.transpose 1 J difference.r))
              (IF (EQ i 1)
                  THEN (Matrix.Multiplication difference.matrix diff.matrix.transpose between.class.matrix no.features.i
```

```
                                        no.features.i 1)
                ELSE (Matrix.Multiplication difference.matrix diff.matrix.transpose extra.matrix no.features.i
                                            no.features.1 1)
                     (Matrix.Addition between.class.matrix extra.matrix between.class.matrix no.features.i)))

(* * Divide each term in the matrix by the total number of classes in the library.
This is a probability term and we assume each image has equal probability of occurring.)


(FOR i FROM 1 TO no.features.i DO (FOR J FROM 1 TO no.features.i
                                       DO (Set.Value between.class.matrix i J
                                                     (QUOTIENT (Find.Value between.class.matrix i J)
                                                               no.in.library.i)

(* * IF only one image per class THEN find just quality.1.r ELSE call Find.Within.Class.Scatter to find
within.class.scatter and find the other three measures of classifier quality.)


(IF single.list.flag.b
    THEN (SETQ quality.1.r (FOR i FROM 1 TO no.features.i SUM (Find.Value between.class.matrix i i)))
  ELSE (Find.Within.Class.Scatter)
       (Matrix.Addition within.class.matrix between.class.matrix extra.matrix no.features.i)
       (SETQ quality.1.r (FOR i FROM 1 TO no.features.1 SUM (Find.Value extra.matrix i i)))

(* * Find the second measure.)


       (SETQ quality.2.r (QUOTIENT (FOR i FROM 1 TO no.features.i SUM (Find.Value between.class.matrix i i))
                                   (FOR i FROM 1 TO no.features.1 SUM (Find.Value within.class.matrix i i)

(* * Make a copy of within.class.matrix because a call to Invert.Matrix destroys the input matrix and
find the third measure.)


       (SETQ extra.matrix (COPYALL within.class.matrix))
       (SETQ new.extra.matrix (Invert.Matrix extra.matrix no.features.i))
       (Matrix.Multiplication new.extra.matrix between.class.matrix extra.matrix no.features.i no.features.i
                              no.features.i)
       (SETQ quality.3.r (FOR i FROM 1 TO no.features.i SUM (Find.Value extra.matrix i i)))

(* * Find the last measure. Make a copy of within.class.matrix because a call to Find.Determinant
destroys the input matrix.)


       (Matrix.Addition between.class.matrix within.class.matrix extra.matrix no.features.i)
       (SETQ determinant.top.r (Find.Determinant extra.matrix no.features.i))
       (SETQ extra.matrix (COPYALL within.class.matrix))
       (SETQ determinant.bot.r (Find.Determinant extra.matrix no.features.i))
       (SETQ quality.4.r (QUOTIENT determinant.top.r determinant.bot.r)))
```

```
        (RETURN (LIST quality.1.r quality.2.r quality.3.r quality.4.r))
)


(* CALLED BY: Quality.Of.Classifier, PURPOSE: Calculates the within.class.matrix, REQUIRED VARIABLES: none, All variables
used in this routine are obtained directly from the calling routine, RETURNS: within.class.matrix, CALLS: Find.Value,
Set.Value, Matrix.Multiplication, Matrix.Addition.)

(DEFINEQ


(Find.Within.Class.Scatter                                                                                    [6]
  (LAMBDA NIL
                                                    (* edited: "21-FEB-87 14:05")

        (* * Loop through all the separate image classes in the library and find the within.class.matrix.)

    (FOR i FROM 1 TO no.in.library.i
       DO (SETQ temp.list (POP temp.lib.list))
          (SETQ no.in.sub.list.i (COUNT temp.list))

          (* * Loop through all the images in each class.)


          (FOR J FROM 1 TO no.in.sub.list.i
             DO (SETQ temp.one.array (POP temp.list))

          (* * Loop through all the feature vectors for each class of images in the library.)


               (FOR k FROM 1 TO no.features.i
                  DO (SETQ difference.r (FDIFFERENCE (ELT temp.one.array k)
                                                     (Find.Value mean.matrix 1 k)))
                     (Set.Value difference.matrix k 1 difference.r)
                     (Set.Value diff.matrix.transpose 1 k difference.r))
               (IF (EQ J 1)
                   THEN (Matrix.Multiplication difference.matrix diff.matrix.transpose extra.matrix no.features.i
                                               no.features.i 1)
                 ELSE (Matrix.Multiplication difference.matrix diff.matrix.transpose new.extra.matrix no.features.i
                                             no.features.i 1)
                      (Matrix.Addition extra.matrix new.extra.matrix extra.matrix no.features.i)))

          (* * Divide each partial sum by the number in the sub.list and the total number of classes in the
          library. This latter number is a probablity term and we assume all images have equal probabilty.)
```

```
        [FOR i FROM 1 TO no.features.i DO (FOR m FROM 1 TO no.features.i
                                               DO (Set.Value extra.matrix 1 m (FQUOTIENT (Find.Value extra.matrix 1 m)
                                                                                         (FTIMES no.in.library.i
                                                                                                 no.in.sub.list.i]

        (* * Sum the partial results into within.class.matrix.)

        (Matrix.Addition within.class.matrix extra.matrix within.class.matrix no.features.i])
)


(* CALLED BY: METHOD of FLAVOR CLASSIFIER. PURPOSE: Clusters a set of image feature vectors based upon a measure of the
closeness of each pair of images. REQUIRED VARIABLES: a list of image feature vectors, the number of images to be
clustered, the number of elements in each feature vector, the type of distance measurement to be used and the power for the
 Minkowski metric. RETURNS: a list of lists, each containing the distance between the two closest image feature vectors and
 the sequence number of the two closest image vectors, for that particular iteration. CALLS: Matrix, Find.Value, Set.Value
and Distance.Classifier.)

(DEFINEQ


(Clustering
  [LAMBDA (image.list no.in.list.i no.features.i type.distance.ch power.i)
                                                    (* edited: "31-MAR-87 16:17")

        (* * KEY VARIABLES: temp.list is a copy of the arrays in image.list. Each iteration of h shrinks this
        list by one since the two closest images are averaged into one array. This copy is required because POP
        destroys the list it is operating on. -
        -
        distance.matrix is a matrix containing the distances from each image to all other images in image.list.
        -
        no.loops.i is the number of times the outer loop needs to run before all the images in image.list have
        been combined into one array. -
        -
        pointer.i is a pointer into distance.matrix. -
        -
        counter.i is a temporary variable to keep track of the number of times Distance.Classifier must be
        called for the arrays remaining in image.list. -
        -
        first.array is the first image array in image.list at each iteration of h. -
        -
        distance.array is the return array from Distance.Classifier and contains the distances from first.array
        to all other images in image.list. -
        -
        min.distance.r is the minimum distance between two image arrays in the current version of image.list.
        These two arrays will be averaged into one array. -
```

```
        -
        temp.distance.r is a temporary variable for the storage of each distance for distance.matrix as it is
        compared with the present value of min.distance.r. -
        -
        which.one.list is a list of the sequence number of the two closest lease arrays.
        -
        dendogram.list is a list of the results from this function; it contains the distance between the two
        closest lease arrays (min.distance.r) and the sequence number of those two arrays (which.one.list) for
        all iterations. -
        -
        one.array is the identity of the first of the two closest arrays to be averaged together.
        The average of the two will be placed back in this array for the next loop. -
        -
        other.array is the identity of the second of the two closest arrays. After its values have been
        averaged with one.array it is eliminated from lease.list.)


    (PROG ((temp.list (COPYALL lease.list))
           (copy.lease.list (COPYALL lease.list))
           (distance.matrix (Matrix no.in.list.i no.in.list.i 0))
           (no.loops.i (SUB1 no.in.list.i))
           pointer.i counter.i first.array distance.array min.distance.r temp.distance.r which.one.list dendogram.list
           one.array other.array)

          (* * Loop until all only two lease arrays remain.)


          [FOR h FROM 1 TO no.loops.i
             DO (SETQ counter.i no.in.list.i)

          (* * Loop through the current value of copy.lease.list and find the distances between all the arrays in
          tha list.)


                [FOR i FROM 1 TO (SUB1 counter.i)
                   DO (SETQ first.array (POP copy.lease.list))

          (* * Call Distance.Classifier to find the distance between one lease array and all the others in
          lease.list.)


                      (SETQ distance.array (Distance.Classifier copy.lease.list (ADD counter.i -i)
                                                                first.array no.features.i type.distance.ch power.i NIL))

          (* * Place the distances from one lease array to all the others in their proper place in
          distance.matrix. Only use the upper triangle of this matrix, excluding the central diagonal.)
```

```
            (SETQ pointer.i 0)
            (FOR J FROM (ADDi i) TO no.in.list.i DO (Set.Value distance.matrix i J (ELT distance.array
                                                                                       (ADD pointer.i 1]

      (* * IF this is not the last iteration of h THEN find the smallest value in distance.matrix ie.
      min.distance.r.)


            (IF (NEQ h no.loops.i)
                THEN (SETQ min.distance.r 9.990000E+28)
                     [FOR J FROM i TO no.in.list.i DO (FOR k FROM (ADDi J) TO no.in.list.i
                                                           DO (SETQ temp.distance.r (Find.Value distance.matrix J k))

      (* * IF a new min.distance.r is found THEN reset min.distance.r and LIST the sequence numbers of the
      two closest arrays.)


                                                                 (IF (FLESSP temp.distance.r min.distance.r)
                                                                     THEN (SETQ min.distance.r temp.distance.r)
                                                                          (SETQ which.one.list (LIST J k]

      (* * Place the new closest variables onto the RETURN variable.)


                     (PUSH dendogram.list (LIST min.distance.r which.one.list))

      (* * Pull out the two closest lease arrays, average their values and place the new averaged values back
      into one.array.)


                     [SETQ one.array (CAR (FNTH temp.list (CAR which.one.list]
                     [SETQ other.array (CAR (FNTH temp.list (CADR which.one.list]
                     (FOR J FROM 1 TO no.features.i DO (SETA one.array J (FQUOTIENT (FPLUS (ELT one.array J)
                                                                                           (ELT other.array J))
                                                                                    2)))

      (* * Strip out other.array from lease.list.)


                     (SETQ copy.lease.list (LDIFFERENCE temp.list (LIST other.array)))

      (* * Reset key loop variables.)
```

```
                        (SETQ temp.list copy.image.list)
                        (ADD no.in.list.i -1)
                ELSE

        (* * Last iteration of h so no min.distance.r to search for, because there is only one non-zero element
        in distance.matrix.)


                        (PUSH dendogram.list (LIST (Find.Value distance.matrix 1 2)
                                                   (LIST 1 2]
        (RETURN (LIST dendogram.list])
)
(DECLARE: DONTCOPY
  (FILEMAP (NIL (6762 9787 (Classify.Decision 6772 . 9785)) (10829 14682 (Distance.Classifier 10839 . 14680)) (15861 20400 (
Weighted.Classifier 15871 . 20398)) (20928 24868 (Weight.Library 20938 . 24866)) (25594 33595 (Quality.Of.Classifier 25604
 . 33593)) (33894 35711 (Find.Within.Class.Scatter 33904 . 35709)) (36350 41540 (Clusterins 36360 . 41538)))))
STOP
```

APPENDIX II-E. File FLAVOR

```
(FILECREATED " 8-APR-87 10:30:29" EE:<STOUT>FLAVOR.;3 30595

      previous date: " 8-APR-87 10:24:25" EE:<STOUT>FLAVOR.;2)


(PRETTYCOMPRINT FLAVORCOMS)

(RPAQQ FLAVORCONS ((* PURPOSE: Function to create a new flavor. REQUIRED VARIABLES: new flavor name, a variable list that
                       will be associated with each instance and passed by inheritence and the type of flavor ie.
                       a-kind-of. The variable list can have default values associated with any or all of these variables.
                       If a default value is desired then place that variable and its associated value in a sublist within
                       the overall list. SPECIAL NOTES: The global variable flavor.environment is altered by this function.
                       This variable consists of a list of flavors in existence and the number of instances of each. The
                       global variable consisting of the flavor name and the word .variables is set to a list consisting of
                       a-kind-of and the variable list. The global variable consisting of the flavor name and the word
                       .methods is initialized to NIL also. All variables are inherited indirectly through the a-kind-of
                       parameter. The variable parent.instance is autoeotically associated with each FLAVOR. This is how
                       separated INSTANCEs of each FLAVOR can be kept seperate and have separate values for their
                       variables. The user must create FLAVOR VANILLA first. Before an instence of a flavor can be made
                       there must be at least one unique variable and one unique method associated with that flavor ie.
                       that were not inherited. RETURNS: flavor name or error message. CALLS: none)
                   (FNS DEFFLAVOR)
                   (* PURPOSE: Function to associate a method with a flavor. A method is the same thing as a function.
                      REQUIRED VARIABLES: flavor name, (it must already be in existence), name of the method, a before or
                      after demon if required, and either a LAMBDA expression containing the actual method or the name of
                      the function which has been previously defined. Alters the global variable consisting of the flavor
                      name and the suffix '.methods. RETURNS: method name or error message for non-existent flavor name.
                      CALLS: none)
                   (FNS DEFMETHOD)
                   (* PURPOSE: Function to create an instance of a flavor. REQUIRED VARIABLES: name of the flavor, new
                      name for the instance, name of the parent instance (ie. the a-kind-of FLAVOR) which must already be
                      in existence, and any initial values for flavor variables. Initial values are input in a list with
                      the variable name first followed by its value. If the user wants the new value to be evaluated
                      before placement then put the value in a list preceded by the word EVAL. The global variable
                      flavor.environment is altered by this function. RETURNS: instance name or error message for
                      non-existent flavor name or no methods message. CALLS: none. WARNING: there must be at least one
                      unique variable and method associated with a flavor before it can be created.)
                   (FNS MAKE.INSTANCE)
                   (* PURPOSE: Function to cause the execution of a method for an instance of a flavor. REQUIRED
                      VARIABLES: name of the flavor, name of the instance, name of the method and a list of variables
                      required by either the before method, if it is present, or the main method to be executed. RETURNS:
                      the value from the last function executed ie. either the main method or the after demon. CALLS: none)
                   (FNS SEND.MESSAGE)
                   (* PURPOSE: Function to obtain the value of a variable of an instance of a flavor. Will trace back to
                      an instance of VANILLA in the search for the variable. Follows the 'parent.instance variable
                      backwards. REQUIRED VARIABLES: name of the flavor, name of the instance and the name of the
                      variable. RETURNS: a list containing the value of the found.flag and either the value of the
                      variable or an error message. Only when found.flag is TRUE has a correct value been returned. CALLS:
                      none)
```

```
(FNS GET.VALUE)
(* PURPOSE: Function to set the value of a variable in an instance of a flavor. REQUIRED VARIABLES:
   name of the flavor, name of the instance, name of the variable, and the new value for the variable.
   Only allowed to change the value of a variable from an instance of the FLAVOR that created that
   variable. No inherited changes are permitted. RETURNS: value of the variable or an error message.
   CALLS: none)
(FNS PUT.VALUE)
(* PURPOSE: Function to print out the most current flavor environment, based on the value of the global
   variable flavor.environment. Lists all flavors and instances. REQUIRED VARIABLES: none. RETURNS:
   none. CALLS: none)
(FNS PRINT.FLAVORS)
(* PURPOSE: Function to print out all the methods, variables and their values for a given instance of a
   flavor. REQUIRED VARIABLES: name of the flavor and name of the instance. RETURNS: none or error
   message. CALLS: none)
(FNS PRINT.INSTANCE)
(* CALLED BY: PRINT.INSTANCE. PURPOSE: prints out the values of a variable if it is an array or matrix.
   REQUIRED VARIABLES: none. All passed directly in from calling routine. RETURNS: none. Prints out
   values on screen. CALLS: Print.Matrix.)
(FNS Print.Array)
(* RECORD which contains the actual format for a flavor and is called from MAKE.INSTANCE)
(RECORDS flavor)))
```

E-2

(* PURPOSE: Function to create a new flavor. REQUIRED VARIABLES: new flavor name, a variable list that will be associated with each instance and passed by inheritance and the type of flavor ie. a-kind-of. The variable list can have default values associated with any or all of these variables. if a default value is desired then place that variable and its associated value in a sublist within the overall list. SPECIAL NOTES: The global variable flavor.environment is altered by this function. This variable consists of a list of flavors in existence and the number of instances of each. The global variable consisting of the flavor name and the word .variables is set to a list consisting of a-kind-of and the variable list. The global variable consisting of the flavor name and the word .methods is initialized to NIL also. All variables are inherited indirectly through the a-kind-of parameter. The variable parent.instance is automatically associated with each FLAVOR. This is how separated INSTANCEs of each FLAVOR can be kept separate and have separate values for their variables. The user must create FLAVOR VANILLA first. Before an instance of a flavor can be made there must be at least one unique variable and one unique method associated with that flavor is. that were not inherited. RETURNS: flavor name or error message. CALLS: none)

(DEFINEQ

[i]

```
(DEFFLAVOR
  (LAMBDA (flavor.name variable.list ako)                    (* edited: "18-FEB-87 11:13")

          (* * KEY VARIABLES: new.variable.list is a list of variables from variable.list after their default
          values have been removed and put on the default property lists. -
          -
          temp.name is a temporary variable used when deleting a previous flavor environment.
          -
          inherit contains the name of the a-kind-of inheritance for the flavor. Its value has the inherited list
```

of variables. -
-
flavor.variables contains the flavor name with suffix variables. Used to uniquelly specify the flavor.)

```
    (PROG ((new.variable.list NIL)
           temp.name inherit flavor.variables)

          (* * Make sure the user input at least one variable for this flavor. Variables must be in list format.)

          (IF (LISTP variable.list)
              THEN

          (* * Update or initialize the value of the global variable flavor.environment.)

                   (IF (BOUNDP (QUOTE flavor.environment))
                       THEN

          (* * If an environment already exists and a new VANILLA is being made then the old environment will be
          entirely destroyed.)

                            (IF (EQUAL flavor.name (QUOTE VANILLA))
                                THEN (PRINTOUT T
                  "WARNING...FLAVOR VANILLA being created. All
previous flavors, methods and varisbles will be destroyed."
                                       T)

          (* * Destroy the old environment.)

                                    (DO (SETQ temp.name (POP flavor.environment))
                                        (POP flavor.environment)
                                        (SET (PACK* temp.name (QUOTE .variables))
                                             NIL)
                                        (SET (PACK* temp.name (QUOTE .methods))
                                             NIL)
                                        (REMPROPLIST temp.name (PROPNAMES temp.name)) WHILE (LISTP flavor.environment)))

          (* * Put the new VANILLA into the now empty flavor environment.)

                             (PUSH flavor.environment 0)
                             (PUSH flavor.environment flavor.name)
                        ELSE (SETQ flavor.environment (LIST flavor.name 0)))
```

```
        (* * Create the variables containing .variables as a suffix and .methods as a suffix.
        These two are global variables.)

                (SETQ flavor.variables (PACK* flavor.name (QUOTE .variables)))

        (* * Make sure that each FLAVOR has the variable parent.instance on its variable list.
        This is used later to trace inheritance of variables through particular INSTANCEs.)

                (PUSHNEW variable.list (QUOTE parent.instance))
                (SET (PACK* flavor.name (QUOTE .methods))
                     NIL)

        (* * PUTPROP the default values for all the variables contained in variable.list and place the variable
        names into new.variable.list. If no default value is input then set the value to NIL.)

                (MAPC variable.list (QUOTE (LAMBDA (temp)
                                                   (IF (LISTP temp)
                                                       THEN (PUTPROP (CAR temp)
                                                                     (QUOTE default)
                                                                     (CADR temp))
                                                            (PUSH new.variable.list (CAR temp))
                                                     ELSE (PUTPROP temp (QUOTE default)
                                                                   NIL)
                                                          (PUSH new.variable.list temp]

        (* * Place the name of the variables into a list prefaced with the ako value.)

                (IF (NULL ako)
                    THEN (SET flavor.variables (LIST (QUOTE VANILLA)
                                                     new.variable.list))
                  ELSE (SET flavor.variables (LIST ako new.variable.list)))
                (RETURN flavor.name)
          ELSE (RETURN 'ERROR...NO variable list input for this flavor."])
)


(* PURPOSE: Function to associate a method with a flavor. A method is the same thing as a function. REQUIRED VARIABLES:
flavor name, (it must already be in existence), name of the method, a before or after qualifier if required, and either a
LAMBDA expression containing the actual method or the name of the function which has been previously defined. Alters the
global variable consisting of the flavor name and the suffix '.methods. RETURNS: method name or error message for
non-existent flavor name. CALLS: none)
```

```
(DEFINED

                                                                        [2]

(DEFMETHOD
  [LAMBDA (flavor.name method.name demon actual.method)     (* edited: '16-FEB-87 15:40')

          (* * KEY VARIABLE: flavor.method contains the flavor name with suffix methods. It becomes the
          equivalent of the global variable consisting of the flavor name and suffix .method.)


    (PROG (flavor.method)

          (* * First check to make sure that the flavor is already in existence. The flavor must be created
          before any methods are attached to it.)


          (IF (EQMEMB flavor.name flavor.environment)
              THEN

          (* * Check for the before or after demon. If not a demon then place the new method onto the method list
          and PUTPROP the actual function onto the appropriate property list, which consists of the method name
          and property name of methods. ELSE PUTPROP the actual function under the property list consisting of
          the method name and the appropriate demon name.)


                  (IF (NULL demon)
                      THEN (SETQ flavor.method (PACK* flavor.name (QUOTE .methods)))
                           (PUTPROP method.name (QUOTE method)
                                    actual.method)

          (* * See if this is the first method to be associated with this flavor.)


                           (IF (BOUNDP (EVAL (QUOTE flavor.method)))
                               THEN (SET flavor.method (CONS method.name (EVAL (EVAL (QUOTE flavor.method]
                             ELSE (SET flavor.method (MKLIST method.name)))
                    ELSE (PUTPROP method.name demon actual.method))
                  (RETURN method.name)
            ELSE (RETURN 'ERROR...flavor not in existance'])
)


(* PURPOSE: Function to create an instance of a flavor. REQUIRED VARIABLES: name of the flavor, new name for the instance,
name of the parent instance (ie. the a-kind-of FLAVOR) which must already be in existence, and any initial values for
flavor variables. Initial values are input in a list with the variable name first followed by its value. If the user wants
the new value to be evaluated before placement then put the value in a list preceded by the word EVAL. The global variable
flavor.environment is altered by this function. RETURNS: instance name or error message for non-existent flavor name or no
methods message. CALLS: none. WARNING: there must be at least one unique variable and method associated with a flavor
```

before it can be created.)

(DEFINEQ

[33]

```
(MAKE.INSTANCE
  [LAMBDA (flavor.name new.instance.name parent.instance.name initial.values)
                                                (* edited: " 2-MAR-87 20:57")

          (* * KEY VARIABLES: flavor.methods contains the flavor name with suffix methods.
          -
          flavor.variables contains the flavor variable with suffix variables. -
          -
          instance.variable is a list containing the methods and variables that will be associated with the
          flavor when the instance is created. -
          -
          parent contains the name of the a-kind-of slot used to trace all inheritance up to VANILLA.
          -
          additional.methods used when tracing inheritance. Contains the total list of additional methods to be
          associated with a particular instance of a flavor. -
          -
          new.list contains the list of methods for a particular flavor. -
          -
          temp.var is a temporary variable used when PUTPROPping the new FLAVOR variable values into their proper
          place. -
          -
          temp.value is a temporary variable which holds the new value for the FLAVOR variable.
          -
          remaining.variables is a list containing the name of the flavor variables that the user did not want
          set to any particular values. They will have a value of NIL placed in the property list for this
          instance of the flavor.)


      (PROG (flavor.methods flavor.variables instance.variable parent additional.methods new.list temp.var temp.value
                            remaining.variables)

          (* * Check to see if the flavor exists.)


          (IF (EQMEMB flavor.name flavor.environment)
              THEN (SETQ flavor.methods (PACK* flavor.name (QUOTE .methods)))

          (* * Make sure at least one method exists for the flavor to be created.)
```

```
                (IF (NOT (NULL (EVAL flavor.methods)))
                    THEN

        (% * Check to make sure that the parent.instance.name INSTANCE has already been created.
        To make an instance of a FLAVOR all higher FLAVORs must have been created.)


                        (IF (OR (EQUAL flavor.name (QUOTE VANILLA))
                                (BOUNDP parent.instance.name))
                            THEN

        (% * Add the new name onto the property list consisting of the flavor name and property 'names,
        and increment the instance counter contained in the global variable flavor.environment.)


                                    (LISTPUT1 flavor.environment flavor.name (ADD1 (LISTGET flavor.environment
                                                                                            flavor.name)))
                                    (ADDPROP flavor.name (QUOTE names)
                                             new.instance.name)
                                    (SETQ flavor.variables (PACK* flavor.name (QUOTE .variables)))

        (% * Get the methods and variables for the instance into one list and CREATE the instance COPYING that
        list. The flavor is created with a call to the record called flavor.)


                                    (SETQ instance.variable (CONS (EVAL flavor.methods)
                                                                  (EVAL flavor.variables)))
                                    (SET new.instance.name (CREATE flavor COPYING instance.variable))

        (% * Find out which variables the user did not specify in initial.values)


                                    (SETQ remaining.variables (DIFFERENCE (CADR (EVAL flavor.variables))
                                                                          initial.values))

        (% * PUTPROP the first values for the variables in the initial.values list onto the correct property
        list. IF the new value of the variable is a list beginning with EVAL THEN the user wants temp.value
        evaluated before PUTPROPping.)


                                    (DO (SETQ temp.var (POP initial.values))
                                        (SETQ temp.value (POP initial.values))
                                        (IF (LISTP temp.value)
                                            THEN (IF (NEQ (CAR temp.value)
                                                          (QUOTE EVAL))
                                                     THEN (PUTPROP new.instance.name temp.var temp.value)
                                                   ELSE (SETQ temp.value (CADR temp.value))
                                                        (SETQ temp.value (EVAL temp.value))
                                                        (PUTPROP new.instance.name temp.var temp.value))
```

```
                                        ELSE (PUTPROP new.instance.name temp.var temp.value))
                                    WHILE (LISTP initial.values))

        (* * PUTPROP a value of NIL on the property list of the variables the user did not specify.)


                                    (DO (PUTPROP new.instance.name (POP remaining.variables)
                                                 NIL)
                                        WHILE (LISTP remaining.variables))

        (* * Place the name of the parent instance onto the variable property list value list.)


                                    (PUTPROP new.instance.name (QUOTE parent.instance)
                                             parent.instance.name)

        (* * Trace the inheritence of this flavor up to VANILLA and add the new methods to the list called
        additional.methods.)


                                    [IF (NOT (EQUAL flavor.name (QUOTE VANILLA)))
                                        THEN (SETQ parent (CADR instance.variable))
                                             (SETQ additional.methods NIL)
                                             (DO [SETQ new.list (EVAL (PACK* parent (QUOTE .methods]
                                                 (SETQ additional.methods (APPEND new.list additional.methods))
                                                 [SETQ parent (CAR (EVAL (PACK* parent (QUOTE .variables]
                                               UNTIL (EQUAL parent (QUOTE VANILLA)))

        (* * Add the new inherited methods to the old methods previously attached to this instance of the
        flavor.)


                                             (SETQ additional.methods
                                               (APPEND [FFETCH method.list of (EVAL (CAR (GETPROP (QUOTE VANILLA)
                                                                                                  (QUOTE names]
                                                       additional.methods))
                                             (WITH flavor (EVAL new.instance.name)
                                                   (SETQ method.list (APPEND method.list additional.methods]
                                ELSE (RETURN 'ERROR...Parent instance not yet in existence.'))
                        ELSE (RETURN 'ERROR...No methods associated with this flavor.'))
                ELSE (RETURN 'ERROR...flavor name not in existance'))
)
```

```
(% PURPOSE: Function to cause the execution of a method for an instance of a flavor. REQUIRED VARIABLES: name of the
flavor, name of the instance, name of the method and a list of variables required by either the before method, if it is
present, or the main method to be executed. RETURNS: the value from the last function executed ie. either the main method
or the after demon. CALLS: none)

(DEFINEQ

                                                                                                                    [4]

(SEND.MESSAGE
  [LAMBDA (flavor.name instance.name method.name variable.list)
                                                        (% edited: '28-FEB-87 13:35')

          (% % KEY VARIABLES: before.flag is T only when a before demon method is present for this particular
          method name. -
          -
          before.demon is the name of the before demon method for a given method name, stored on a property list.
          A method does not require a before demon. -
          -
          after.demon is the name of the after demon method. -
          -
          before.return is the value returned by execution of the before demon function. -
          -
          main.return is the value returned by execution of the main method function.)


    (PROG ((before.flag NIL)
           before.demon after.demon before.return main.return)

          (% % Make sure the flavor and instance exist.)


          (IF [AND (EQMEMB flavor.name flavor.environment)
                   (EQMEMB instance.name (GETPROP flavor.name (QUOTE names]
              THEN

          (% % Make sure that the method exists for the flavor indicated.)


                   (IF (EQMEMB method.name (FFETCH method.list of (EVAL instance.name)))
                       THEN

          (% % Place instance.name onto a property list so that the before and after demons can find the correct
          variables to set and put values onto.)
```

```
                        (PUTPROP (QUOTE name)
                                 (QUOTE instance.name)
                                 instance.name)

        (* * Get the before demon and execute it if it exists. Pass in parameters from variable.list.)


                        (SETQ before.demon (GETPROP method.name (QUOTE before)))
                        (IF (NOT (NULL before.demon))
                            THEN (SETQ before.return (APPLY before.demon (MKLIST variable.list)))
                                 (SETQ before.flag T))

        (* * IF the before demon was executed then pass the parameter returned by its execution into the main
        method. ELSE pass variable.list into the main method.)


                        [IF before.flag
                            THEN

        (* * Before executing the main routine check to see that no errors were found in the before demon.)


                                 (IF (NOT (STRMEMB "ERROR" before.return))
                                     THEN (SETQ main.return (APPLY (GETPROP method.name (QUOTE method))
                                                                   (MKLIST before.return)))
                                   ELSE (RETURN "ERROR...in before demon."))
                             ELSE (SETQ main.return (APPLY (GETPROP method.name (QUOTE method))
                                                           (MKLIST variable.list]

        (* * Check to make sure there were no errors in the main routine. IF not THEN get the after demon and
        execute it. Pass into it the parameters returned by the execution of the main method.
        Return the value of the last method executed.)


                        (IF (STRMEMB "ERROR" main.return)
                            THEN (RETURN "ERROR...in
main.routine.")
                          ELSE (SETQ after.demon (GETPROP method.name (QUOTE after)))
                               (IF (NOT (NULL after.demon))
                                   THEN (RETURN (APPLY after.demon (MKLIST main.return)))
                                 ELSE (RETURN main.return)))
                    ELSE (RETURN "ERROR...Not a method for this flavor."))
            ELSE (RETURN "ERROR...Not a valid flavor or a valid instance of
a flavor."])
)
```

```
(* PURPOSE: Function to obtain the value of a variable of an instance of a flavor. Will trace back to an instance of
VANILLA in the search for the variable. Follows the 'parent.instance variable backwards. REQUIRED VARIABLES: name of the
flavor, name of the instance and the name of the variable. RETURNS: a list containing the value of the found.flag and
either the value of the variable or an error message. Only when found.flag is TRUE has a correct value been returned.
CALLS: none)

(DEFINEQ


(GET.VALUE                                                                                                    [5]
  [LAMBDA (flavor.name instance.name variable.name)          (* edited: "23-FEB-87 15:56")

          (* * KEY VARIABLE: found.flag is a boolean that is TRUE when variable.name has been found for
          instance.name or an inheritence of instance.name. -
          -
          variable.value is the value of the given variable obtained either from the default or normal property
          list. -
          -
          parent is the name of the INSTANCE of the FLAVOR above the one presently being checked.
          Obtained through the property of 'parent.instance for any INSTANCE.)                                                E-11


    (PROG ((found.flag NIL)
           variable.value parent)

          (* * First check to make sure the flavor is in existence.)


          (IF (EQMEMB flavor.name flavor.environment)
              THEN

          (* * Next make sure the instance has already been created.)


                  (IF (EQMEMB instance.name (GETPROP flavor.name (QUOTE names)))
                      THEN

          (* * Find out which FLAVOR contains the original definition of this varible. Trace back through the
          inheritance chain using the property 'parent.instance.)


                              (SETQ parent instance.name)
                              (DO
```

```
        (* * IF the variable is found on the present list being searched THEN stop ELSE trace back up one more
        FLAVOR.)


                              [IF (EQMEMB variable.name (FETCH variable.list of (EVAL parent)))
                                  THEN (SETQ found.flag T)
                                ELSE (SETQ parent (GETPROP parent (QUOTE parent.instance]
                             UNTIL (OR found.flag (NULL parent)))

        (* * IF a FLAVOR was found containing variable.name THEN find its initial value.)


                          (IF found.flag
                              THEN (SETQ variable.value (GETPROP parent variable.name)))

        (* * If the variable does not have a value get its default value from the property list.)


                                    [IF (EQ variable.value NIL)
                                        THEN (SETQ variable.value (GETPROP variable.name (QUOTE default]
                                   (RETURN (LIST found.flag variable.value))
                               ELSE (RETURN (LIST found.flag 'ERROR...No such variable in existence.')))
                      ELSE (RETURN (LIST found.flag 'ERROR...No such instance in existence.')))
             ELSE (RETURN (LIST found.flag 'ERROR...No such flavor in existence.'])
)
```



```
(* PURPOSE: Function to set the value of a variable in an instance of a flavor. REQUIRED VARIABLES: name of the flavor,
name of the instance, name of the variable, and the new value for the variable. Only allowed to change the value of a
variable from an instance of the FLAVOR that created that variable. No inherited changes are permitted. RETURNS: value of
the variable or an error message. CALLS: none)

(DEFINEQ


(PUT.VALUE                                                                                              [6]
  [LAMBDA (flavor.name instance.name variable.name new.value)
                                                          (* edited: '18-FEB-87 13:42')
    (PROG NIL

        (* * First check to make sure the flavor is in existence.)


        (IF (EQMEMB flavor.name flavor.environment)
            THEN
```

```
        (* * Next make sure the instance has already been created.)


                (IF (EQNENB instance.name (GETPROP flavor.name (QUOTE names)))
                    THEN

        (* * Make sure that this FLAVOR is the declaring one for this variable. Cannot change the value of a
        variable through an inherited FLAVOR.)


                        (IF (EQNENB variable.name (PROPNAMES instance.name))
                            THEN

        (* * Put the new value onto the appropriate property list. Does not change the default value.)


                                    (PUTPROP instance.name variable.name new.value)
                                    (RETURN new.value)
                                ELSE (RETURN "ERROR...Illegal variable or variable does not belong to this flavor ."))
                            ELSE (RETURN "ERROR...No such instance in existance."))
                    ELSE (RETURN "ERROR...No such flavor in the environment."])
)
```



```
(* PURPOSE: Function to print out the most current flavor environment: based on the value of the global variable
flavor.environment. Lists all flavors and instances. REQUIRED VARIABLES: none. RETURNS: none. CALLS: none)

(DEFINEQ

                                                                                                        [7]

(PRINT.FLAVORS
  [LAMBDA NIL                                                    (* edited: "16-FEB-87 09:06")

        (* * KEY VARIABLES: environment is a local copy of the global variable flavor.environeeent.
        -
        number is the number of instances of a particular flavor.)


    (PROG ((environment (COPY flavor.anvironeent))
           number)
          (PRINTOUT T T "The present flavor environeent for this prograe consists of
the following flavors and instances:" T T)
          (DO (PRINTOUT T "FLAVOR - " (SETQ name (POP environeent))
                      " has "
                      (SETQ number (POP environment))
                      " instance(s) in existence." T)
              (IF (IGREATERP number 0)
                  THEN (PRINTOUT T "          The list of instance names is - " (GETPROP name (QUOTE names))
```

```
                                T T))
            WHILE (LISTP environment))
        (TERPRI])
)


(* PURPOSE: Function to print out all the methods, variables and their values for a given instance of a flavor. REQUIRED
VARIABLES: name of the flavor and name of the instance. RETURNS: none or error message. CALLS: none)

(DEFINEQ

                                                                                                                [8]
(PRINT.INSTANCE
  [LAMBDA (flavor.name instance.name)                          (* edited: "23-FEB-87 16:08")

          (* * KEY VARIABLES: variable.list is a list of all the variables for an instance of a flavor.
          value is the value of the variable, stored in a property list format.)

    (PROG (variable.list value)

          (* * Check to see if the instance and flavor exist.)

          (IF [AND (EQMEMB flavor.name flavor.environment)
                   (EQMEMB instance.name (GETPROP flavor.name (QUOTE names]
              THEN (PRINTOUT T T instance.name " is an instance of FLAVOR " flavor.name ":" T T)
                   (PRINTOUT T "it is a-kind-of " (FETCH a.kind.of (EVAL instance.name))
                             T)
                   (PRINTOUT T "it has the following list of methods: " (FETCH method.list (EVAL instance.name))
                             T)
                   (PRINTOUT T "it has the following list of variables and values: " T T)

          (* * Get the list of variables and obtain each of their values or defaults.)

                  (SETQ variable.list (FETCH variable.list (EVAL instance.name)))
                  [MAPC variable.list (QUOTE (LAMBDA (temp)
                                                     (SETQ value (GETPROP instance.name temp))
                                                     [IF (EQ value NIL)
                                                         THEN (SETQ value (GETPROP temp (QUOTE default]
                                                     (IF (ARRAYP value)
                                                         THEN (Print.Array)
                                                       ELSE (PRINTOUT T temp " = " value T]
```

```
        (* * Next get the name of each set of inherited variables and print their values out.)

                (PRINTOUT T T instance.name ' inherits the following list of variables and values: ' T)
                (SETQ instance.name (GETPROP instance.name (QUOTE parent.instance)))

        (* * Loop until an INSTANCE of VANILLA is found.)

                (DO (SETQ variable.list (FETCH variable.list (EVAL instance.name)))
                    (PRINTOUT T T 'Inherited from ' instance.name ':' T T)
                    (MAPC variable.list (QUOTE (LAMBDA (temp)
                                                       (SETQ value (GETPROP instance.name temp))
                                                       (IF (ED value NIL)
                                                           THEN (SETQ value (GETPROP temp (QUOTE default)
                                                       (IF (ARRAYP value)
                                                           THEN (Print.Array)
                                                         ELSE (PRINTOUT T temp ' = ' value T)
                    (SETQ instance.name (GETPROP instance.name (QUOTE parent.instance)))
                  WHILE (NOT (NULL instance.name)))
                (TERPRI)
          ELSE (RETURN 'Error,..not an instance of a flavor'))
)
```

E-15

```
(* CALLED BY: PRINT.INSTANCE. PURPOSE: Prints out the values of a variable if it is an array or matrix. REQUIRED VARIABLES:
 none. All passed directly in from calling routine. RETURNS: none. Prints out values on screen. CALLS: Print.Matrix.)

(DEFINEQ

                                                                                                        [9]

(Print.Array
  (LAMBDA NIL                                                   (* edited: '23-FEB-87 16:22')

        (* * KEY VARIABLE: size is the size of the array or one dimension of the matrix.)


    (PROG ((size (ARRAYSIZE value)))
          (IF (ARRAYP (ELT value 1))
              THEN (PRINTOUT T temp , 'is a matrix containing the following elements: ' T)
                   (Print.Matrix value size size 1)
            ELSE (PRINTOUT T temp , 'is an array containing the following elements: ' T)
                 (FOR i FROM 1 TO size DO (PRINTOUT T (ELT value i)
                                                  ,))
                 (TERPRI))
)
```

```
(* RECORD which contains the actual format for a flavor and is called from MAKE.INSTANCE)

(DECLARE: EVAL@COMPILE

(RECORD flavor (method.list a.kind.of variable.list))
)
(DECLARE: DONTCOPY
  (FILEMAP (NIL (6876 10176 (DEFFLAVOR 6886 . 10174)) (10697 12200 (DEFMETHOD 10707 . 12198)) (12966 18367 (MAKE.INSTANCE
12976 . 18365)) (18767 21818 (SEND.MESSAGE 18777 . 21816)) (22314 24475 (GET.VALUE 22324 . 24473)) (24890 26017 (PUT.VALUE
24900 . 26015)) (26243 27062 (PRINT.FLAVORS 26253 . 27060)) (27290 29583 (PRINT.INSTANCE 27300 . 29581)) (29830 30398 (
Print.Array 29840 . 30396)))))
STOP
```

APPENDIX II-F. File IN_OUTPUT

```
(FILECREATED " 8-APR-87 10:43:41" EE:<STOUT>1N_OUTPUT.;3 14846

      previous date: " 8-APR-87 10:38:14" EE:<STOUT>1N_OUTPUT.;2)


(PRETTYCOMPRINT 1N_OUTPUTCOMS)

(RPAQQ IN_OUTPUTCOMS ((* CALLED BY: User. PURPOSE: Writes a test pattern to a data file. Test pattern contains a matrix
                         with each element a period. User can then enter an image on this pattern to be read in by
                         Read.Matrix.Pattern. The first element in this file is the maximum size of one dimension of the
                         array. REQUIREQ VARIABLES: background pixel character; then user prompted for file name. RETURNS:
                         the full VMS file name for the data file. CALLS: Write.Test.Help.)
                        (FNS Write.Test.Pattern)
                        (* CALLED BY: Write.Test.Pattern. PURPOSE: actually prints the matrix to the file. REQUIRED
                           VARIABLES: maximum size of one dimension of the test pattern and the background pixel character.
                           RETURNS: nothing. CALLS: none.)
                        (FNS Write.Test.Help)
                        (* CALLED BY: METHOD of FLAVOR VANILLA. PURPOSE: Reads in a test pattern from a file into the
                           INTERLISP environment. Places this information into a matrix. This is how the user can input a
                           new image written on a test pattern from Write.Test.Pattern. The file must be in the DATA
                           subdirectory. REQUIRED VARIABLES: image pixel; background pixel characters; size of the matrices
                           presently being processed and the file name( the user is prompted for the new name for the matrix
                           to be read in. RETURNS: the new matrix name or an error message. CALLS: Ask.Question; Matrix and
                           Set.Value.)
                        (FNS Read.Matrix.Pattern)
                        (* CALLED BY: METHOD of FLAVOR VANILLA. PURPOSE: Stores a matrix presently in the INTERLISP
                           environment into a file so that it can be retrieved much faster. Actually stored as an array of
                           arrays. All matrices; after analysis is completed should be output to a file in this fashion.
                           REQUIREQ VARIABLES: user prompted for file name and matrix name. RETURNS: list containing the
                           full file name where the matrices were stored and an updated list of matrices stored in
                           full.file.name. CALLS: Ask.Question.)
                        (FNS Store.Pattern.Matrix)
                        (* CALLED BY: METHOD of FLAVOR VANILLA. PURPOSE: Loads a list of matrices stored in a file using
                           Store.Pattern.Matrix. REQUIRED VARIABLES: the file name. RETURNS: list of matrices read into
                           INTERLISP or an error message. CALLS: Ask.Question.)
                        (FNS Load.Pattern.Matrix)
                        (* CALLED BY: METHOD of FLAVOR VANILLA. PURPOSE: Allows the user to alter image points in a matrix
                           directly. Designed to be used after the user has identified that an image on a matrix does not
                           have a closed boundary and should have one. REQUIRED VARIABLES: matrix name; image pixel
                           character; background pixel character and the size of one dimension of the matrix. RETURNS:
                           matrix with corrected image points as entered by the user. CALLS: Ask.User.)
                        (FNS Correct.Image)
                        (* CALLED BY: All I/O routines that prompt the user for input. PURPOSE: necessary because
                           occasionally the input stream has a CONTROL D placed on the front. This routine checks for this
                           condition and eliminates the CONTROL Q and returns the desired user input. REQUIRED VARIABLES:
                           user prompt. RETURNS: user response with CONTROL Q stripped off. CALLS: none.)
                        (FNS Ask.Question)))
```

```
(* CALLED BY: User, PURPOSE: writes a test pattern to a data file. Test pattern contains a matrix with each element a
period. User can then enter an image on this pattern to be read in by Read.Matrix.Pattern. The first element in this file
is the maximum size of one dimension of the array. REQUIRED VARIABLES: background pixel character; then user prompted for
file name. RETURNS: the full VMS file name for the data file. CALLS: Write.Test.Help.)

(DEFINEQ

                                                                                                                    [1]

(Write.Test.Pattern
  [LAMBDA (background.pixel.ch)                                    (* edited: '17-FEB-87 14:32')

          (* * KEY VARIABLES: full.file.name is the character string containing the full VMS path name for files
          contained in the DIRECTORY <STOUT.DATA>. This directory contains all the pattern matrix files.
          -
          max.size.i is the maximum size of one dimension of a matrix.)


    (PROG (full.file.name max.size.i)
          [SETQ max.size.i (ASKUSER 4 32 'WHAT SIZE OF TEST PATTERN DO YOU WANT? ' (QUOTE (32 16 8 4))
                                    NIL NIL (QUOTE (CONFIRMFLG NIL]
          (TERPRI)
          (TERPRI)

          (* * Create the full path name for the data file.)


          (SETQ full.file.name (PACK* (QUOTE EE:<STOUT.DATA>TEST_PATTERN)
                                      max.size.i))

          (* * Open the file for output only.)


          (OUTFILE full.file.name)

          (* * Call to write test pattern matrix to the file.)


          (Write.Test.Help max.size.i background.pixel.ch)
          (CLOSEALL)
          (RETURN full.file.name])
)
```

(* CALLED BY: Write.Test.Pattern. PURPOSE: actually prints the matrix to the file. REQUIRED VARIABLES: maximum size of one dimension of the test pattern and the background pixel character. RETURNS: nothing. CALLS: none.)

(DEFINEQ

[2]

```
(Write.Test.Help
  [LAMBDA (max.size.i background.pixel.ch)                    (* edited: '17-FEB-87 14:33')

          (* * Print max.size.i at the top of the data file.)


    (PRINT max.size.i)
    (TERPRI)

          (* * Output a matrix of background.pixel.ch to the file.)


    (FOR i FROM 1 TO max.size.i
       DO (FOR J FROM 1 TO max.size.i
             DO (PRIN1 background.pixel.ch)
                (SPACES 1))
          (TERPRI))
    (TERPRI])
)
```

(* CALLED BY: METHOD of FLAVOR VANILLA. PURPOSE: Reads in a test pattern from a file into the INTERLISP environment. Places this information into a matrix. This is how the user can input a new image written on a test pattern from Write.Test.Pattern. The file must be in the DATA subdirectory. REQUIRED VARIABLES: image pixel, background pixel characters, size of the matrices presently being processed and the file name; the user is prompted for the new name for the matrix to be read in. RETURNS: the new matrix name or an error message. CALLS: Ask.Question, Matrix and Set.Value.)

(DEFINEQ

[3]

```
(Read.Matrix.Pattern
  [LAMBDA (file.name image.pixel.ch background.pixel.ch matrix.size.i)
                                                    (* edited: ' 4-MAR-87 13:32')

          (* * KEY VARIABLES: full.file.name is the full VMS path file name. -
          -
          max.size.i is the maximum size of one dimension of a matrix. -
          -
          matrix.name is the new name for the matrix to be read in from the file. -
          -
          pixel.ch is the value of one element in the matrix being read in from the file.)
```

```
(PROG (full.file.name max.size.i matrix.name pixel.ch)

      (* * Create the path name to the DATA subdirectory.)


      (SETQ full.file.name (PACK* (DUDTE EE:<STDUT.DATA>)
                                  file.name))

      (* * Call to find the new name for the matrix.)


      (SETQ matrix.name (Ask.Question 'WHAT NAME WOULD YOU LIKE TO OIVE THIS PATTERN MATRIX?'))
      (TERPRI)

      (* * IF the file is in existence THEN read from it ELSE error message.)


      (IF (INFILEP full.file.name)
          THEN (INFILE full.file.name)
               (SETQ max.size.i (READ))

      (* * Check to make sure the new matrix to be read in has the same dimensions as the ones previously
      input.)


               (IF (EQ max.size.i matrix.size.i)
                   THEN

      (* * Set matrix.name to this new matrix being read in.)


                        (SET matrix.name (Matrix max.size.i max.size.i background.pixel.ch))
                        (FOR i FROM 1 TO max.size.i DO (FOR j FROM 1 TO max.size.i
                                                           DO (SETQ pixel.ch (READ))
                                                              (IF (EQ pixel.ch image.pixel.ch)
                                                                  THEN (Set.Value (EVAL matrix.name)
                                                                                  i j pixel.ch)
                        (CLOSEALL)
                        (RETURN matrix.name)
                   ELSE (CLOSEALL)
                        (RETURN 'ERROR...MATRIX HAS THE WRONG DIMENSIONS.'))
        ELSE (RETURN 'ERROR...FILE DOES NOT EXIST.'])
)
```

```
(* CALLED BY: METHOD of FLAVOR VANILLA. PURPOSE: Stores a matrix presently in the INTERLISP environment into a file so that
 it can be retrieved much faster. Actually stored as an array of arrays. All matrices, after analysis is completed should
be output to a file in this fashion. REQUIRED VARIABLES: user prompted for file name and matrix name. RETURNS: list
containing the full file name where the matrices were stored and an updated list of matrices stored in full.file.name.
CALLS: Ask.Question.)

(DEFINEQ
```

[4]

```
(Store.Pattern.Matrix
  (LAMBDA NIL                                          (* edited: "26-FEB-87 10:29")

          (* * KEY VARIABLES: file.name is a short VMS file name. -
          -
          full.file.name is the full VMS path file name. -
          -
          com.file is the name of the COM file where the matrix will be stored. -
          -
          matrix.name is the new name for the matrix to be read in from the file. -
          -
          answer is the user response to a question.)


    (PROG (full.file.name file.name com.file matrix.name answer)

          (* * Print out the list of files in directory <STOUT.DATA>)


          (PRINTOUT T T "THE FOLLOWING ARE THE CURRENT DATA FILES" T)
          (PRINTOUT T T (FILDIR (QUOTE EE:<STOUT.DATA>*.*))
                    T T)

          (* * Call to find which file to store the matrix in.)


          (SETQ file.name (Ask.Question "INPUT THE FILENAME FOR STORING THE MATRIX."))

          (* * Call to find the name of the matrix to be stored.)


          (SETQ matrix.name (Ask.Question "INPUT THE NAME OF THE MATRIX TO BE STORED."))
          (SETQ full.file.name (PACK* (QUOTE EE:<STOUT.DATA>)
                                      file.name))
```

```
        (* * Create the name for the COM file.)

        (SETQ com.file (PACK* file.name (QUOTE COMS)))

        (* * Turn off the COPYRIONT flas.)

        (SETQ COPYRIONTFLG NIL)

        (* * IF the COM file alreadw exists THEN add the eatrix to the list of UGLYVARS ELSE creats a naw COM
        file.)

        [IF (BOUNDP com.file)
            THEN [SETQ answer (ASKUSER 4 (QUOTE w)
                                        'DO YOU WANT TO SEE THE NAME OF THE PATTERN MATRICES ALREADY IN THIS FILE?' NIL
                                        NIL NIL (QUOTE (CONFIRMFLG NIL]
                 (TERPRI)
                 (TERPRI)

        (* * Print out the current list of UGLYVARS.)

                 [IF (EQ answer (QUOTE Y))
                     THEN (PRINTOUT T (FILECOMSLST full.file.name (QUOTE UGLYVARS]

        (* * Add the new eatrix to the list of UGLYVARS.)

                 (ADDTOCOMS com.file eatrix.name (QUOTE UGLYVARS))
          ELSE (SET com.file (LIST (LIST (QUOTE UGLYVARS)
                                         eatrix.name]

        (* * Write this inforeation to the file and turn the COPYRIONT flas back on.)

        (MAKEFILE full.film.naee)
        (SETQ COPYRIONTFLG T)
        (TERPRI)
        (RETURN (LIST full.film.naee (FILECOMSLST full.file.naee (QUOTE UGLYVARS))
)
```

```
(* CALLED BY: METHOD of FLAVOR VANILLA. PURPOSE: Loads a list of matrices stored in a file using Store.Pattern.Matrix.
REQUIRED VARIABLES: the file name. RETURNS: list of matrices read into INTERLISP or an error message. CALLS: Ask.Question.)

(DEFINEQ

                                                                                                        [5]

(Load.Pattern.Matrix
  [LAMBDA (file.name)                                          (* edited: '27-JAN-87 10:52')

          (* * KEY VARIABLES: full.file.name is the full VMS path file name. -
          -
          matrix.name is the new name for the matrix to be read in from the file. -
          -
          answer is the user response to a question.)

    (PROG (full.file.name matrix.name answer)

          (* * Create the path name to the DATA subdirectory.)

          (SETQ full.file.name (PACK* (QUOTE EE:<STOUT.DATA>)
                                      file.name))

          (* * IF the file exists THEN read the matrices in ELSE error message.)

          (IF (INFILEP full.file.name)
              THEN (LOAD full.file.name)
                   (TERPRI)
                   (RETURN (FILECOMSLST full.file.name (QUOTE UGLYVARS)))
            ELSE (RETURN 'ERROR...FILE DOES NOT EXIST.'))
)


(* CALLED BY: METHOD of FLAVOR VANILLA. PURPOSE: Allows the user to alter lease points in a matrix directly. Designed to be
 used after the user has identified that an lease on a matrix does not have a closed boundary and should have one. REQUIRED
 VARIABLES: matrix name; lease pixel character; background pixel character and the size of one dimension of the matrix.
RETURNS: matrix with corrected lease points as entered by the user. CALLS: Ask.User.)

(DEFINEQ

                                                                                                        [6]

(Correct.lease
  [LAMBDA (matrix.name lease.pixel.ch background.pixel.ch max.size.i)
                                                               (* edited: '26-FEB-87 10:31')
```

```
        (* * KEY VARIABLES! answer is the user response to a prompt. -
        -
        response is the user response to another prompt. -
        -
        value is the pixel value placed on the coordinate value desired by the user; either an image pixel or
        an background pixel.)


    (PROG (answer response value)
        (PRINTOUT T "Coordinates must be between 1 and " max.size.i T "To halt processing enter a 0.")

        (* * Loop while the user is inputting new points.)


        (DO (SETQ answer (Ask.Question "Input the row and column of the point
to be set to an image point"))

        (* * IF the user input a legal point THEN ask him if it is for an image or background pixel.)


            (IF (AND (LISTP answer)
                     (IGREATERP (CAR answer)
                                0)
                     (ILEQ (CAR answer)
                           max.size.i)
                     (IGREATERP (CADR answer)
                                0)
                     (ILEQ (CAR answer)
                           max.size.i))
                THEN (SETQ response (Ask.Question "Enter 1 for image pixel ELSE pixel becomes
a background point."))

        (* * Set value to the appropriate pixel and set that coordinate in the matrix.)


                    (IF (EQ response '1)
                        THEN (SETQ value image.pixel1,ch)
                      ELSE (SETQ value background.pixel,ch))
                    (Set.Value matrix.name (CAR answer)
                                           (CADR answer)
                                           value)))
          UNTIL (EQ answer 0))
        (RETURN matrix.name])
)
```

```
(* CALLED BY: All I/O routines that prompt the user for input. PURPOSE: necessary because occasionally the input stream has
 a CONTROL O placed on the front. This routine checks for this condition and eliminates the CONTROL O and returns the
desired user input. REQUIRED VARIABLES: user prompt. RETURNS: user response with CONTROL O stripped off. CALLS: none.)

(DEFINEQ

                                                                                                                    [7]
(Ask.Question
  [LAMBDA (question)                                          (* edited: '16-JAN-87 11:29')

          (* * KEY VARIABLE: response is the user response to the question.)


    (PROG (response)

          (* * Print out the question.)


          (PRINTOUT T T question T)
          (SETQ response (READ))

          (* * Strip off a leading CONTROL O if the read inadvertently picked it up.)


          [IF (EQ (CAR (UNPACK response))
                  (QUOTE ))
              THEN (SETQ response (UNPACK response))
                   (SETQ response (PACK (CDR response]
          (RETURN response])
)
(DECLARE: DONTCOPY
  (FILEMAP (NIL (3855 4907 (Write.Test.Pattern 3865 . 4905)) (5134 5572 (Write.Test.Help 5144 . 5570)) (6164 7987 (
Read.Matrix.Pattern 6174 . 7985)) (8498 10804 (Store.Pattern.Matrix 8508 . 10802)) (11053 11869 (Load.Pattern.Matrix 11063
 . 11867)) (12329 13867 (Correct.Tease 12339 . 13865)) (14234 14824 (Ask.Question 14244 . 14822)))))
STOP
```

APPENDIX II-G. File MATRIX

```
(FILECREATED " 8-APR-87 10:55:08" EE:<STOUT>MATRIX.;3 24175

      previous date: " 8-APR-87 10:51:31" EE:<STOUT>MATRIX.;2)


(PRETTYCOMPRINT MATRIXCOMS)

(RPAQQ MATRIXCONS ((* PURPOSE: Function which creates a 2-dimensional matrix. REQUIRED VARIABLES: desired number of rows
                       and columns and the default value for all matrix elements. Place this command inside a SETQ to
                       obtain the matrix. RETURNS: physical address of the matrix. CALLS: none)
                   (FNS Matrix)
                   (* PURPOSE: Function that finds the value for one matrix element. REQUIRED VARIABLES: matrix name, row
                      and column numbers. RETURNS: the matrix element value. CALLS: none)
                   (FNS Find.Value)
                   (* PURPOSE: Function that finds the CDR value for one matrix element. Same arguments as the above
                      function. The two values are independent, changing one does not change the other. Each matrix is
                      initialized to have both values what the user input as the default in the Matrix function call.)
                   (FNS Find.Cdr.Value)
                   (* PURPOSE: Function that sets the value of one matrix element. REQUIRED VARIABLES: matrix name, row
                      and column number and new matrix element value. RETURNS: new value. CALLS: none)
                   (FNS Set.Value)
                   (* PURPOSE: Function used to set the CDR part of one matrix element. Uses the same arguments as the
                      above function. As above, the two values are independent of each other.)
                   (FNS Set.Cdr.Value)
                   (* PURPOSE: Function that allow the user to input an entire lease in matrix form. REQUIRED VARIABLES:
                      name of the matrix to be created and the length of one dimension of the assumed square matrix. Other
                      required variables are prompted within the function. RETURNS: the new matrix. CALLS: Matrix and
                      Set.Value.)
                   (FNS User.Input)
                   (* PURPOSE: Function prints out the input matrix on the screen. REQUIRED VARIABLES: matrix name, total
                      number of rows and columns and the number of blanks between each column. RETURNS: nothing. CALLS:
                      Find.Value)
                   (FNS Print.Matrix)
                   (* PURPOSE: Function prints out the CDR values of the input matrix. Same parameters and return as
                      Print.Matrix.)
                   (FNS Print.Cdr.Matrix)
                   (* PURPOSE: Function that adds two matrices. REQUIRED VARIABLES: name of the two matrices to be added,
                      name of the matrix that will contain the sum and the size of one dimension of the square matrices.
                      RETURNS: matrix with the sum in it. CALLS: Set.Value and Find.Value.)
                   (FNS Matrix.Addition)
                   (* PURPOSE: Function that multiplies two matrices. REQUIRED VARIABLES: name of the two matrices to be
                      multiplied, name of the matrix that will contain the product, the number of rows in the first matrix
                      to be multiplied, the number of cols in the second matrix and the number of cols in the first
                      matrix. No checking of the dimensions is done by this procedure. The user must create the product
                      matrix so that its dimensions are the number of rows in matrix one by the number of cols in matrix
                      two. RETURNS: matrix with the product in it. CALLS: Set.Value and Find.Value.)
                   (FNS Matrix.Multiplication)
                   (* ALGORITHM: from the book NUMERICAL RECIPES by W. Press, B. Flannery p. 2-37. PURPOSE: solves a
                      linear set of equations A#x=b when in matrix form. REQUIRED VARIABLES: matrix name, vector b, and
```

```
   length of one dimension of the matrix. RETURNS: solution vector x returned in b. Original matrix A
   has been destroyed. IMPORTANT NOTE: If the destruction of any of the input matrices is not desired
   then the user can use the COPYALL command immediately before calling those functions to save a copy
   of the matrix desired. This command produces a new matrix and values not just pointers to values in
   the old matrix. CALLS: Matrix.Decomp and For.Bak.Subst. Two global variables row.permutes.a and
   row.changes.i are passed into these two routines.)
(FNS Solve.Linear.Eqs)
(* ALGORITHM: from the book NUMERICAL RECIPES by W. Press, B. Flannery p. 2-38. PURPOSE: finds the
   inverse of a matrix if it exists. REQUIRED VARIABLES: matrix name and the length of one dimension of
   the square matrix. RETURNS: a matrix containing the inverse. The original matrix has been destroyed.
   CALLS: Matrix, Set.Value, Matrix.Decomp and For.Bak.Subst. Two global variables row.permutes.a and
   row.changes.i are passed into the last two routines.)
(FNS Invert.Matrix)
(* ALGORITHM: from the book NUMERICAL RECIPES by W. Press, B. Flannery p. 2-39. PURPOSE: finds the
   determinant of a matrix if it exists. REQUIRED VARIABLES: matrix name and the length of one
   dimension of the square matrix. RETURNS: the determinant. CALLS: Matrix.Decomp and Find.Value. Two
   global variables row.permutes.a and row.changes.i are passed into the first routine.)
(FNS Find.Determinant)
(* ALGORITHM: from the book NUMERICAL RECIPES by W. Press, B. Flannery p. 2-35. PURPOSE: used in
   solving linear equations. This routine replaces the input matrix with the LU decomposition of a
   rowwise permution of itself, using Crouts method with partial pivoting. Used in conJunction with
   For.Bak.Subst. REQUIRED VARIABLES: matrix name and the length of one dimension of the square matrix.
   Two global variables row.permutes.a and row.changes.i are passed in by the calling routine. RETURNS:
   decomposed matrix and the number of row permutations effect by the partial pivoting. CALLS:
   Find.Value and Set.Value)
(FNS Matrix.Decomp)
(* ALGORITHM: from the book NUMERICAL RECIPES by W. Press, B. Flannery p. 2-37. PURPOSE: solves the set
   of N linear equations A•X=B, where A is the LU decomposed matrix from Matrix.Decomp. REQUIRED
   VARIABLES: matrix name, length of one dimension of the square matrix, and the vector b. The global
   variable row.permutes.a is passed in by the calling routine. RETURNS: solution vector x contained in
   b. The decomposed matrix A is not changed by this routine. CALLS: Find.Value)
(FNS For.Bak.Subst)))


(* PURPOSE: Function which creates a 2-dimensional matrix. REQUIRED VARIABLES: desired number of rows and columns and the
default value for all matrix elements. Place this command inside a SETQ to obtain the matrix. RETURNS: physical address of
the matrix. CALLS: none)

(DEFINEQ


(Matrix                                                                                                        [1]
  [LAMBDA (no.rows.i no.cols.i default.ch)                    (* edited: " 2-JUL-86 13:33")
```

```
        (* * KEY VARIABLES: address is an array for the rows, each row is also an array of columns.
                -
        index.i is an index into the rows. -
                -
        max.i is the maximum number of rows in the matrix.)

    (PROG (address index.i max.i)

        (* * Each address set up as an array.)

        (SETQ address (ARRAY no.rows.i NIL NIL))
        (SETQ max.i no.rows.i)
        (SETQ index.i 1)

        (* * Each element in address also set up as an array. End up with an array of arrays; ie.
        a 2-d matrix.)

      loop(SETA address index.i (ARRAY no.cols.i 0 default.ch))
          (SETQ index.i (ADD1 index.i))
          (AND (ILEQ index.i max.i)
               (GO loop))
          (RETURN address])
)


(* PURPOSE: Function that finds the value for one matrix element. REQUIRED VARIABLES: matrix name, row and column numbers.
RETURNS: the matrix element value. CALLS: none)

(DEFINEQ


(Find.Value                                                                    [2]
  [LAMBDA (array.name row.i col.i)
    (ELT (ELT array.name row.i)                    (* edited: '30-JUN-86 15:00')
         col.i])
)


(* PURPOSE: Function that finds the CDR value for one matrix element. Same arguments as the above function. The two values
are independent, changing one does not change the other. Each matrix is initialized to have both values what the user input
 as the default in the Matrix function call.)
```

```
(DEFINEQ

                                                                                    [3]
(Find.Cdr.Value
  [LAMBDA (arraw.name row.i col.i)                    (* edited: '16-JUL-86 14:27')
    (ELTD (ELT arraw.name row.i)
          col.i])
)


(* PURPOSE: Function that sets the value of one matrix element. REQUIRED VARIABLES: matrix name, row and column number and
new matrix element value. RETURNS: new value. CALLS: none)

(DEFINEQ

                                                                                    [4]
(Set.Value
  [LAMBDA (arraw.name row.i col.i value.ch)           (* edited: '30-JUN-86 15:07')
    (BETA (ELT arraw.name row.i)
          col.i value.ch])
)


(* PURPOSE: Function used to set the CDR part of one matrix element. Uses the same arguments as the above function. As
above, the two values are independent of each other.)

(DEFINEQ

                                                                                    [5]
(Set.Cdr.Value
  [LAMBDA (arraw.name row.i col.i value.ch)           (* edited: '16-JUL-86 14:25')
    (SETD (ELT arraw.name row.i)
          col.i value.ch])
)


(* PURPOSE: Function that allow the user to input an entire image in matrix form. REQUIRED VARIABLES: name of the matrix to
 be created and the length of one dimension of the assumed square matrix. Other required variables are prompted within the
function. RETURNS: the new matrix. CALLS: Matrix and Set.Value.)

(DEFINEQ
```

```
(User.Input                                                                    [6]
  [LAMBDA (matrix.name length.i)                    (* edited: '19-FEB-87 08:59')

          (* * Create the matrix using matrix.name.)


    (SET matrix.name (Matrix length.i length.i 0.000000E+00))

          (* * Loop through the rows and columns.)


    (FOR i FROM 1 TO length.i DO (FOR J FROM 1 TO length.i
                                    DO (PRINTOUT T 'INPUT row' , i , 'column' , J T)

          (* * Read user input and set value of corresponding matrix element.)


                                       (SETQ user.value.ch (READ))
                                       (Set.Value (EVAL matrix.name)
                                                  i J user.value.ch])
)


(* PURPOSE: Function prints out the input matrix on the screen. REQUIRED VARIABLES: matrix name, total number of rows and
columns and the number of blanks between each column. RETURNS: nothing. CALLS: Find.Value)

(DEFINEQ


(Print.Matrix                                                                  [7]
  [LAMBDA (arraw.name max.rows.i max.cols.i no.blanks.i)    (* edited: '29-JUL-86 08:54')

          (* * Loop through the rows and columns of the matrix printing the values on the screen.)


    [FOR i FROM 1 TO max.rows.i DO ((TERPRI)
                                    (FOR J FROM 1 TO max.cols.i
                                       DO (PRIN1 (Find.Value arraw.name i J))
                                          (SPACES no.blanks.i)
    (TERPRI)
    (TERPRI))
)
```

```
(* PURPOSE: Function prints out the COR values of the input matrix. Same parameters and return as Print.Matrix.)

(DEFINEQ

                                                                                                    [8]

(Print.Cdr.Matrix
  [LAMBDA (array.name max.rows.i max.cols.i no.blanks.i)    (* edited: '19-FEB-87 08:37')

          (* * Loop through the COR of the rows and columns of the matrix printing the values on the screen.)


    [FOR i FROM 1 TO max.rows.i DO ((TERPRI)
                                    (FOR J FROM 1 TO max.cols.i
                                       DO (PRIN1 (Find.Cdr.Value array.name i J))
                                          (SPACES no.blanks.i]
    (TERPRI)
    (TERPRI])
)


(* PURPOSE: Function that adds two matrices. REQUIRED VARIABLES: name of the two matrices to be added, name of the matrix
that will contain the sum and the size of one dimension of the square matrices. RETURNS: matrix with the sum in it. CALLS:
Set.Value and Find.Value.)

(DEFINEQ

                                                                                                    [9]

(Matrix.Addition
  [LAMBDA (matrix.one matrix.two matrix.sum size.i)          (* edited: '19-FEB-87 08:41')

          (* * Loop through the rows and columns adding corresponding elements.)


    (FOR row FROM 1 TO size.i DO (FOR col FROM 1 TO size.i DO (Set.Value matrix.sum row col
                                                                        (FPLUS (Find.Value matrix.one row col)
                                                                               (Find.Value matrix.two row col])
)


(* PURPOSE: Function that multiplies two matrices. REQUIRED VARIABLES: name of the two matrices to be multiplied, name of
the matrix that will contain the product, the number of rows in the first matrix to be multiplied, the number of cols in
the second matrix and the number of cols in the first matrix. No checking of the dimensions is done by this procedure. The
user must create the product matrix so that its dimensions are the number of rows in matrix one by the number of cols in
matrix two. RETURNS: matrix with the product in it. CALLS: Set.Value and Find.Value.)
```

```
(DEFINEQ

                                                                                [10]

(Matrix.Multiplication
  [LAMBDA (matrix.one matrix.two matrix.product no.rows.1 no.cols.2 no.cols.1)
                                                    (* edited: "19-FEB-87 08:47")

          (* * Loop throush the rows and columns and sum the correct products from matrix.one and matrix.two.)


    (FOR row FROM 1 TO no.rows.1 DO (FOR col FROM 1 TO no.cols.2
                                       DO (Set.Value matrix.product row col (FOR counter FROM 1 TO no.cols.1
                                                                               SUM (FTIMES (Find.Value matrix.one row
                                                                                                       counter)
                                                                                           (Find.Value matrix.two counter
                                                                                                       col])

)


(* ALGORITHM: from the book NUMERICAL RECIPES by W. Press, B. Flannery p. 2-37. PURPOSE: solves a linear set of equations
A*x=b when in matrix form. REQUIRED VARIABLES: matrix name, vector b, and length of one dimension of the matrix. RETURNS:
solution vector x returned in b. Original matrix A has been destroyed. IMPORTANT NOTE: If the destruction of any of the
input matrices is not desired then the user can use the COPYALL command immediately before calling these functions to save
a copy of the matrix desired. This command produces a new matrix and values not just pointers to values in the old matrix.
CALLS: Matrix.Decomp and For.Bak.Subst. Two global variables row.permutes.a and row.changes.i are passed into these two
routines.)

(DEFINEQ

                                                                                [11]

(Solve.Linear.Eqs
  [LAMBDA (matrix.name sol.vector length.i)                    (* edited: "28-AUG-86 14:50")

          (* * KEY VARIABLES: row.permutes.a is an array which keeps track of row permutations.
          -
          row.changes.i: integer which keeps track of the number of row interchanges.)


    (PROG ((row.permutes.e (ARRAY length.i NIL NIL))
           row.changes.i)
          (Matrix.Decomp matrix.name length.i)                 (* call to decompose matrix)
          (For.Bak.Subst matrix.name length.i sol.vector)      (* call to solve linear eqs)
          (RETURN sol.vector])
)
```

```
(* ALGORITHM: from the book NUMERICAL RECIPES by W. Press, B. Flannery p. 2-38. PURPOSE: finds the inverse of a matrix if
it exists. REQUIRED VARIABLES: matrix name and the length of one dimension of the square matrix. RETURNS: a matrix
containing the inverse. The original matrix has been destroyed. CALLS: Matrix, Set.Value. Matrix.Decomp and For.Bak.Subst.
Two global variables row.permutes.a and row.changes.i are passed into the last two routines.)

(DEFINEQ

                                                                                                                [12]
(Invert.Matrix
  [LAMBDA (matrix.name length.i)                                    (* edited: '28-AUG-86 14:56')

          (* * KEY VARIABLES: inverted.matrix is a matrix which initially contains the identity matrix and
          eventually contains the inverse matrix. -
          -
          row.permutes.a is an array which keeps track of row permutations. -
          -
          row.changes.i is an integer which keeps track of the number of row interchanges.)


    (PROG ((inverted.matrix (Matrix length.i length.i 0.000000E+00))
           (row.permutes.a (ARRAY length.i NIL NIL))
           row.changes.i)

          (* * Loop to create identity matrix)


          (FOR i FROM 1 TO length.i DO (Set.Value inverted.matrix i i 1.0))
          (Matrix.Decomp matrix.name length.i)              (* decompose matrix once)

          (* * Loop to find the inverse of the matrix column by column)


          (FOR j FROM 1 TO length.i DO (For.Bak.Subst matrix.name length.i (ELT inverted.matrix j)))
          (RETURN inverted.matrix])
)


(* ALGORITHM: from the book NUMERICAL RECIPES by W. Press, B. Flannery p. 2-39. PURPOSE: finds the determinant of a matrix
if it exists. REQUIRED VARIABLES: matrix name and the length of one dimension of the square matrix. RETURNS: the
determinant. CALLS: Matrix.Decomp and Find.Value. Two global variables row.permutes.a and row.changes.i are passed into the
 first routine.)
```

```
(DEFINEQ

(Find.Determinant                                                    [13]
  [LAMBDA (matrix.name length.i)                    (* edited: "28-AUG-86 16:31")

          (* * KEY VARIABLES: row.permutes.a is an array which keeps track of row permutations.
          -
          row.changes.i is an integer which keeps track of the number of row interchanges.)


    (PROG ((row.permutes.a (ARRAY length.i NIL NIL))
           row.changes.i)
          (Matrix.Decomp matrix.name length.i)          (* decompose the matrix)

          (* * Since the determinant of the decomposed matrix is just the product of the diagonal elements
          multiply them together, keeping track of row interchanges)


          [FOR J FROM 1 TO length.i DO (SETQ row.changes.i (FTIMES row.changes.i (Find.Value matrix.name J J]
          (RETURN row.changes.i))
)

(* ALGORITHM: from the book NUMERICAL RECIPES by W. Press, B. Flannery p. 2-35. PURPOSE: used in solving linear equations.
This routine replaces the input matrix with the LU decomposition of a rowwise permuation of itself, using Crouts method
with partial pivoting. Used in conJunction with For.Bak.Subst. REQUIRED VARIABLES: matrix name and the length of one
dimension of the square matrix. Two global variables row.permutes.a and row.changes.i are passed in by the calling routine.
 RETURNS: decomposed matrix and the number of row permutations affect by the partial pivoting. CALLS: Find.Value and
Bat.Value)

(DEFINEQ

(Matrix.Decomp                                                       [14]
  [LAMBDA (matrix.name length.i)                    (* edited: "28-AUG-86 16:33")

          (* * KEY VARIABLES: row.scale.a is an array containing the row scaling inforamation.
          -
          summation.r is a real number containing partial row sum information. -
          -
          Pivot.value.r is a real number which is the pivot value for that particular row.
          -
          row.max.r is the maximum real value in that particular row. -
          -
          row.indicator.i is a pointer to the row being processed. -
          -
          temp.max.r is a real number containing temporary the maximum row value.)
```

```
(PROG ((row.scale.a (ARRAY length.i NIL NIL))
       summation.r pivot.value.r row.max.r row.indicator.i temp.max.r)
      (SETQ row.changes.i 1.0)                          (* no row interchanges yet)

      (* * Loop over rows to set the implicit scaling information)

      (FOR i FROM 1 TO length.i
         DO (SETQ row.max.r 0.000000E+00)

      (* * Loop over each column to find the largest element in each row)

            (FOR J FROM 1 TO length.i
               DO (SETQ temp.max.r (ABS (Find.Value matrix.name J i)))
                  (IF (FOREATERP temp.max.r row.max.r)
                      THEN (SETQ row.max.r temp.max.r)))

      (* * If all row elements are 0 have singular matrix)

            (IF (EQP row.max.r 0.000000E+00)
                THEN (PRINTOUT T "Singular Matrix" T))
            (SETA row.scale.a i (FQUOTIENT 1.0 row.max.r)))
                                                (* save the scaling information)

      (* * Loop over columns using Crouts method)

                                                (* eq 2.3-12 p.2-33)
      [FOR J FROM 1 TO length.i
         DO [IF (IOREATERP J 1)
                THEN (FOR i FROM 1 TO (SUB1 J)
                        DO (SETQ summation.r (Find.Value matrix.name J i))
                           (IF (IOREATERP i 1)
                               THEN [FOR k FROM 1 TO (SUB1 i) DO (SETQ summation.r
                                                                    (FDIFFERENCE summation.r
                                                                                 (FTIMES (Find.Value matrix.name k i)
                                                                                         (Find.Value matrix.name J k]
                                    (Set.Value matrix.name J i summation.r]
            (SETQ row.max.r 0.000000E+00)               (* initialize search for the largest pivot element)
                                                        (* eqs 2.3.12 and 2.3.13 p.2.33)
            (FOR i FROM J TO length.i
               DO (SETQ summation.r (Find.Value matrix.name J i))
                  (IF (IOREATERP J 1)
                      THEN [FOR k FROM 1 TO (SUB1 J) DO (SETQ summation.r (FDIFFERENCE summation.r
                                                                                      (FTIMES (Find.Value matrix.name
                                                                                                    k i)
```

```
                                                                                (Find.Value matrix.name
                                                                                    J k]
                            (Set.Value matrix.name J i summation.r))
                        (SETQ pivot.value.r (FTIMES (ELT row.scale.a i)
                                                    (ABS summation.r)))
                                                    (% figure of merit for the pivot)
                                                    (% is it better than the best so far?)
                        (IF (FLESSP row.max.r pivot.value.r)
                            THEN (SETQ row.indicator.i i)
                                 (SETQ row.max.r pivot.value.r)))
                                                    (% do we need to interchange rows?)
                (IF (NOT (EQP J row.indicator.i))
                    THEN (FOR k FROM 1 TO length.i
                              DO (SETQ pivot.value.r (Find.Value matrix.name k row.indicator.i))
                                 (Set.Value matrix.name k row.indicator.i (Find.Value matrix.name k J))
                                 (Set.Value matrix.name k J pivot.value.r))
                         (SETQ row.changes.i (FTIMES row.changes.i -1.0))
                                                    (% indicate a row interchange)
                                                    (% also interchange the scale factor)
                         (SETA row.scale.a row.indicator.i (ELT row.scale.a J)))
                (SETA row.permutes.a J row.indicator.i)     (% divide by the pivot element)

        (% % Avoid singular matrix problem)


                (IF (NOT (EQP J length.i))
                    THEN (IF (EQP (Find.Value matrix.name J J)
                                  0.000000E+00)
                             THEN (Set.Value matrix.name J J 1.000000E-20))
                         (SETQ pivot.value.r (FQUOTIENT 1.0 (Find.Value matrix.name J J)))
                         (FOR i FROM (ADD1 J) TO length.i DO (Set.Value matrix.name J i (FTIMES (Find.Value matrix.name J
                                                                                                    i)
                                                                                            Pivot.value.r]
        (IF (EQP (Find.Value matrix.name length.1 length.i)
                 0.000000E+00)
            THEN (Set.Value matrix.name length.i length.i 1.000000E-20])
)


(% ALGORITHM: from the book NUMERICAL RECIPES by W. Press, S. Flannery p. 2-37. PURPOSE: solves the set of N linear
equations A%x=b, where A is the LU decomposed matrix from Matrix.Decomp. REQUIRED VARIABLES: matrix name, length of one
dimension of the square matrix, and the vector b. The global variable row.permutes.a is passed in by the calling routine.
RETURNS: solution vector x contained in b. The decomposed matrix A is not changed by this routine. CALLS: Find.Value)
```

```
(DEFINEQ

(For.Bak.Subst                                                                    [15]
  [LAMBDA (matrix.name length.i input.vector)              (* edited! "28-AUG-86 16:20")

          (* * When row.index.i is set to a positive value, it will become the index of the first nonvanishing
          element of input.vector. Then do the forward substitution eq 2.3.6. Also need to unscramble the
          permutation as we go)


          (* * KEY VARIABLES! row.index.i is an integer pointer to the row being processed.
          -
          row.value.r contains row permutation information from row.permutes.a. -
          -
          summation.r! real number contains row summation information.)


    (PROG ((row.index.i 0)
           row.value.r summation.r)
          (FOR i FROM 1 TO length.i
             DO (SETQ row.value.r (ELT row.permutes.a i))
                (SETQ summation.r (ELT input.vector row.value.r))
                (SETA input.vector row.value.r (ELT input.vector i))
                (IF (NOT (EQP row.index.i 0))
                    THEN (FOR J FROM row.index.i TO (SUB1 i) DO (SETQ summation.r (DIFFERENCE summation.r
                                                                                              (TIMES (Find.Value
                                                                                                       matrix.name J i)
                                                                                                     (ELT input.vector JJ
                                                                                   (* else a nonzero element was encountered so from now on we
                                                                                   will have to do the sums in the loop above)

                  ELSEIF (NOT (EQP summation.r 0.000000E+00))
                    THEN (SETQ row.index.i i))
                (SETA input.vector i summation.r))

          (* * This loop does the backsubstitution eq. 2.3-7)


          (FOR i FROM length.i TO 1 BY -1
             DO (SETQ summation.r (ELT input.vector i))
                (IF (ILESSP i length.i)
                    THEN (FOR J FROM (ADD1 i) TO length.i DO (SETQ summation.r (DIFFERENCE summation.r
                                                                                           (TIMES (Find.Value
                                                                                                    matrix.name J i)
                                                                                                  (ELT input.vector JJ
                                                                                (* store a component of the solution vector)
                (SETA input.vector i (QUOTIENT summation.r (Find.Value matrix.name i i))
```

```
)
(DECLARE: DONTCOPY
  (FILEMAP (NIL (6524 7348 (Matrix 6534 . 7346)) (7525 7680 (Find.Value 7535 . 7678)) (7979 8140 (Find.Cdr.Value 7989 . 8138)
) (8328 8493 (Set.Value 8338 . 8491)) (8672 8841 (Set.Cdr.Value 8682 . 8839)) (9159 9751 (User.Input 9169 . 9749)) (9970
10373 (Print.Matrix 9980 . 10371)) (10491 10913 (Print.Cdr.Matrix 10501 . 10911)) (11191 11577 (Matrix.Addition 11201 .
11575)) (12159 12777 (Matrix.Multiplication 12169 . 12775)) (13529 14113 (Solve.Linear.Eqs 13539 . 14111)) (14575 15570 (
Invert.Matrix 14585 . 15568)) (15954 16736 (Find.Determinant 15964 . 16734)) (17357 21723 (Matrix.Decomp 17367 . 21721)) (
22207 24153 (For.Bak.Subst 22217 . 24151)))))
BTOP
```

APPENDIX II-H. File MOMENT

```
(FILECREATED " 8-APR-87 11:07:08" EE:<STOUT>MOMENT.;3 28504

      previous date: " 8-APR-87 11:01:20" EE:<STOUT>MOMENT.;2)


(PRETTYCOMPRINT MOMENTCONS)

(RPAQQ MOMENTCONS ((* CALLED BY: METHOD of FLAVOR INTERMEDIATE.PROCESS. PURPOSE: controls the calculation of the Zernicke
                      moments for the image whose edge points are input. These moments are invariant to rotation,
                      translation and size. REQUIRED VARIABLES: list of the edge points for the image, number of edge
                      points and a flag. When flag is 3 the user wants only second order moments; when it is 4 the user
                      wants second and third order moments; when equal to 5 the user wants second, third and fourth order
                      moments. RETURNS: array with the Zernicke moments. CALLS: Calculate.Moments, Central.Moments and
                      Get.Zernicke.Moments)
                   (FNS Calculate.Zernicke.Moments)
                   (* CALLED BY: Calculate.Zernicke.Moments. PURPOSE: calculates the proper number of moments based on the
                      the value of the flag. Uses an approximation based on Green's Theorem for the moments, where only
                      the edge points are required. It is basically a line integral approximation. REQUIRED VARIABLES:
                      list of edge points, number of edge points, an empty array to store the moments in after they are
                      calculated and the flag to indicate how many moments to calculate. RETURNS: the array with moments
                      inside. CALLS: Matrix, Find.Value and Set.Value.)
                   (FNS Calculate.Moments)
                   (* CALLED BY: Calculate.Zernicke.Moments and Calculate.Zernicke.Long. PURPOSE: normalizes and
                      centralizes the moments calculated in Calculate.Moments. REQUIRED VARIABLES: the array containing
                      the moments, the empty array for placing the normalized, centralized moments after calculations and
                      the flag to indicate how many of the moments to calculate. RETURNS: the normalized, centralized
                      moments. CALLS: none.)
                   (FNS Central.Moments)
                   (* CALLED BY: Calculate.Zernicke.Moments and Calculate.Zernicke.Long. PURPOSE: calculates the Zernicke
                      moment invariants. REQUIRED VARIABLES: the normalized, centralized moments from Central.Moments, the
                      empty array for the Zernicke moments to be stored after calculated and the flag to indicate the
                      number of invariants to be calculated. RETURNS: the array containing the Zernicke moment invariants.
                      CALLS: none.)
                   (FNS Get.Zernicke.Moments)
                   (* CALLED BY: METHOD of FLAVOR INTERMEDIATE.PROCESS. PURPOSE: controls the routines that calculate the
                      Zernicke moments for the image using all edge plus interior points. Used to compare actual moment
                      values with values using Green's Theorem to approximate all the points in the image, as found in
                      METHOD Calculate.Zernicke.Moments. REQUIRED VARIABLES: list of edge points, list of all interior
                      points, number of edge points, number of interior points and a flag indicating what order of
                      Zernicke moments to calculate. RETURNS: Zernicke moments in an array. CALLS: Calculate.Real.Moments,
                      Central.Moments and Get.Zernicke.Moments.)
                   (FNS Calculate.Zernicke.Long)
                   (* CALLED BY: Calculate.Zernicke.Long. PURPOSE: calculates moments up to the fourth order for all image
                      points. Does not use Green's Theorem for an approximation. REQUIRED VARIABLES: list of all image
                      points, total number of image points and an empty array for storing the moments after calculation.
                      RETURNS: moments for the image in an array. CALLS: Matrix, Find.Value and Set.Value.)
                   (FNS Calculate.Real.Moments)))
```

```
(* CALLED BY: METHOD of FLAVOR INTERMEDIATE.PROCESS. PURPOSE: controls the calculation of the Zernicke moments for the
image whose edge points are input. These moments are invariant to rotation, translation and size. REQUIREQ VARIABLES: list
of the edge points for the image, number of edge points and a flag. When flag is 3 the user wants only second order
moments; when it is 4 the user wants second and third order moments; when equal to 5 the user wants second, third and
fourth order moments. RETURNS: array with the Zernicke moments. CALLS: Calculate.Moments, Control.Moments and
Get.Zernicke.Moments)

(DEFINEQ
                                                                                                              [1]

(Calculate.Zernicke.Moments
  (LAMBDA (edge.list no.points.i flag.i)                    (* edited: " 2-APR-87 20:18")

          (* * KEY VARIABLES: moment.array is a 15 element array for storing the moments after they have been
          calculated. Element 1 contains m 0,0; element 2 = m 0,1; 3 = m 0,2; 4 = m 0,3; 5 = m 0,4;
          6 = m 1,0; 7 = m 1,1; 8 = m 1,2; 9 = m 1,3; 10 = m 2,0; 11 = m 2,1; 12 = m 2,2;
          13 = m 3,0; 14 = m 3,1; 15 = m 4,0. -
          -
          norm.cen.array is a 12 element array for storing the moments after they have been normalized and
          centralized. Element 1 contains mu 0,2; 2 has mu 0,3; 3 = mu 0,4; 4 = mu 1,1; 5 = mu 1,2;
          6 = mu 1,3; 7 = mu 2,0; 8 = mu 2,1; 9 = mu 2,2; 10 = mu 3,0; 11 = mu 3,1; 12 = mu 4,0.
          -
          zernicke.array is an 11 element array containing the Zernicke moment invariants.
          Elements 1 and 2 contain the second ordor inveriants ; elements 3 through 6 contain the third ordor
          moments and elements 7 through 11 contain the fourth order invariants.)


    (PROG ((moment.array (ARRAY 15 NIL NIL))
           (norm.cen.array (ARRAY 12 NIL NIL))
           zernicke.array)

          (* * Create the correct size for zernicke.array.)


          (IF (EQ flag.i 3)
              THEN (SETQ zernicke.array (ARRAY 2 NIL NIL))
            ELSEIF (EQ flag.i 4)
              THEN (SETQ zernicke.array (ARRAY 6 NIL NIL))
            ELSE (SETQ zernicke.array (ARRAY 11 NIL NIL)))

          (* * Call to calculate the moments from the list of edge points.)
```

```
        (Calculate.Moments edge.list no.points.i moment.array flag.i)

        (* * Call to normalize and centralize the previously calculated moments.)


        (Central.Moments moment.array norm.cen.array flag.i)

        (* * Call to calculate the Zernicke moment invariants from the normalized and centralized moments.)


        (Get.Zernicke.Moments norm.cen.array zernicke.array flag.i)
        (RETURN (LIST zernicke.array))
)


(* CALLED BY: Calculate.Zernicke.Moments. PURPOSE: calculates the proper number of moments based on the the value of the
flag. Uses an approximation based on Green's Theorem for the moments, where only the edge points are required. It is
basically a line integral approximation. REQUIRED VARIABLES: list of edge points, number of edge points, an empty array to
store the moments in after they are calculated and the flag to indicate how many moments to calculate. RETURNS: the array
with moments inside. CALLS: Matrix, Find.Value and Set.Value.)

(DEFINEQ
```


[2]

```
(Calculate.Moments
  [LAMBDA (edge.list no.points.i moment.array flag.i)       (* edited: "27-JAN-87 13:03")

        (* * KEY VARIABLES: counter.i is a pointer indicating which moment is being calculated presently.
        -
        coord.pointer.i is a pointer into edge.list. -
        -
        prev.point.c is the coordinate of the previous edge point that was considered. -
        -
        jump.inc.i causes the correct moments to written in the right position so that only those needed to be
        calculated will be. -
        -
        moment.matrix is a i5 by the total number of points + i matrix: each row contains a different moment,
        and each element in each row contains a the appropriate multiple of x and y terms for that edge point.
        The last element in each row contains the sum of the x and y exponents + 1.0 -
        -
        point.c is the coordinate of the present edge point being processed. -
        -
        start.i is an integer initialized to 1 at the start of each set of moment calculations.
        It represents the value of moment m 0,0. -
        -
        previous.i is an integer which represents the present value of the moment as factors of x and y are
        added on. -
        -
```

sign.i is an integer representing whether the difference in the w coordinate of the present point as compared to the previous is plus or minus)

```
(PROG ((counter.i 0)
       (coord.pointer.i 0)
       (prev.point.c (CAR (LAST edge.list)))
       (jump.inc.i (IDIFFERENCE 5 flag.i))
       (moment.matrix (Matrix 15 (ADD1 no.points.i)
                              0))
       point.c start.i previous.i sign.i)

      (* * Loop until all edge points have been considered.)


      (DO (SETQ point.c (pop edge.list))
          (SETQ sign.i (IDIFFERENCE (CADR point.c)
                                    (CADR prev.point.c)))

      (* * Set sign.i to plus or minus based on the change in w coordinates from the previous to the current
      point.)


          (IF (MINUSP sign.i)
              THEN (SETQ sign.i -1)
            ELSE (SETQ sign.i 1))
          (ADD coord.pointer.i 1)
          (SETQ counter.i 1)
          (SETQ start.i sign.i)

      (* * Loop through 3, 4 or 5 times. Moments are calculated in a true fashion, each edge point has all
      its factors of x and w calculated, starting at first order for x and zero order for w.
      After this number has been stored then another factor of w is multiplied in until the maximum has been
      reached. Then w is reset to zero order and the x order is incremented etc.)


          (FOR J FROM 1 TO flag.i
             DO (SETQ start.i (ITIMES start.i (CAR point.c)))
                (SETQ previous.i start.i))

      (* * Loop through to obtain the proper number of multiples of x and w.)


                (FOR k FROM 1 TO (IDIFFERENCE (ADD1 flag.i)
                                              j)
                   DO (Set.Value moment.matrix counter.i coord.pointer.i previous.i)
                      (SETQ previous.i (ITIMES previous.i (CADR point.c)))
                      (IF (EQ coord.pointer.i no.points.i)
                          THEN (Set.Value moment.matrix counter.i (ADD1 no.points.i)
```

```
                                                J))
                              (ADD counter.i 1))
                        (ADD counter.i Jump.inc.i))
                  (SETQ prev.point.c point.c) WHILE (LISTP edge.list))

          (* * Loop through each row of the matrix and sum up the elements and obtain the correct moment which is
          stored in moment.array.)


          (FOR i FROM 1 TO 15 DO (SETA moment.array i (FQUOTIENT (FOR m FROM 1 TO no.points.i
                                                                    SUM (Find.Value moment.matrix i m))
                                                                 (Find.Value moment.matrix i (ADD1 no.points.i))
)


(* CALLED BY: Calculate.Zernicke.Moments and Calculate.Zernicke.Loop. PURPOSE: normalizes and centralizes the moments
calculated in Calculate.Moments. REQUIRED VARIABLES: the array containing the moments, the empty array for placing the
normalized, centralized moments after calculations and the flag to indicate how many of the moments to calculate. RETURNS:
the normalized, centralized moments. CALLS: none.)

(DEFINEQ


(Central.Moments
  [LAMBDA (moment.array norm.con.array flag.i)              (* edited: '20-DEC-86 13:36')

          (* * KEY VARIABLES: zero.order is moment m00. -
          zero.squared, zero.two.five and zero.cubed are the squares 2.5 and cube of zero.order.
          -
          w.average is the average for the w values. -
          -
          w.ave.sq and w.ave.cub are the square and cube of w.average. -
          -
          x.average, x.ave.sq and x.ave.cub are the same things for the x values.)


    (PROG ((zero.order (ELT moment.array 1))
           w.average w.ave.sq w.ave.cub x.average x.ave.sq x.ave.cub zero.squared zero.two.five zero.cubed)
          (SETQ zero.squared (FTIMES zero.order zero.order))
          (SETQ zero.two.five (FTIMES zero.order zero.order (SQRT zero.order)))
          (SETQ zero.cubed (FTIMES zero.squared zero.order))
          (SETQ w.average (FQUOTIENT (ELT moment.array 2)
                                     zero.order))
          (SETQ w.ave.sq (FTIMES w.average w.average))
          (SETQ w.ave.cub (FTIMES w.ave.sq w.average))
          (SETQ x.average (FQUOTIENT (ELT moment.array 6)
                                     zero.order))
          (SETQ x.ave.sq (FTIMES x.average x.average))
```

```
        (SETQ x.eve.cub (FTIMES x.eve.sq x.everese))

        (* * normalized mu02 = m02-(w.ave*m01))

        (SETA norm.cen.arrew 1 (FQUQTIENT (FD1FFERENCE (ELT moment.arrew 3)
                                                       (FTIMES w.everese (ELT moment.arrew 2)))
                                          zero.squered))

        (* * normalized mu20 = m20-(x.ave * m10))

        (SETA norm.cen.arrew 7 (FQUQTIENT (FD1FFERENCE (ELT moment.arrew 10)
                                                       (FTIMES x.everese (ELT moment.arrew 6)))
                                          zero.squered))

        (* * normalized mu11 = m11-(w.ave * m10))

        (SETA norm.cen.arrew 4 (FQUQTIENT (FD1FFERENCE (ELT moment.arrew 7)
                                                       (FTIMES w.everese (ELT moment.arrew 6)))
                                          zero.squered))

        (* * User wants second and third order moments.)

        [COND
          ((IGREATERP flag.i 3)

        (* * normalized mu03 = m03-(3 * w.ave * m02)+(2 * w.eve.sq * m01))

            (SETA norm.cen.arrew 2 (FQUQTIENT (FD1FFERENCE (FPLUS (ELT moment.arrew 4)
                                                                  (FTIMES 2.0 w.eve.sq (ELT moment.arrew 2)))
                                                           (FTIMES 3.0 w.average (ELT moment.arrew 3)))
                                              zero.two.five))

        (* * normalized mu12 = m12-(2 * w.eva * m11)-(x.eve * m02)+(2 * w.ave.sq * m10))

            (SETA norm.cen.arrew 5 (FQUQTIENT [FD1FFERENCE (FPLUS (ELT moment.arrew 8)
                                                                  (FTIMES 2.0 (ELT moment.arrew 6)
                                                                          w.eve.sq))
                                                           (FPLUS (FTIMES 2.0 (ELT moment.arrew 7)
                                                                              w.everese)
                                                                  (FTIMES x.everese (ELT moment.arrew 3]
                                              zero.two.five))
```

```
(* * normalized mu21 = m21-(2 * x.ave * m11)-(w.ave * m20)+(2 * x.ave.so * m01))

    (SETA norm.cen.array 8 (FQUOTIENT (FDIFFERENCE (FPLUS (ELT moment.array 11)
                                                          (FTIMES 2.0 (ELT moment.array 2)
                                                                      x.ave.so))
                                                   (FPLUS (FTIMES (ELT moment.array 10)
                                                                  w.average)
                                                          (FTIMES 2.0 (ELT moment.array 7)
                                                                      x.average)))
                                           zero.two.five))

(* * normalized mu30 = m30-(3 * x.ave * m20)+(2 * x.ave.so * m10))

    (SETA norm.cen.array 10 (FQUOTIENT (FDIFFERENCE (FPLUS (ELT moment.array 13)
                                                           (FTIMES 2.0 (ELT moment.array 6)
                                                                       x.ave.so))
                                                    (FTIMES 3.0 (ELT moment.array 10)
                                                                x.average))
                                           zero.two.five)

(* * User wants second, third and fourth order moments.)

(COND
  ((IGREATERP flag.i 4)

(* * normalized mu04 = m04-(4 * w.ave * m03)+(6 * w.ave.so * m02)-(3 * w.ave.cub * m01))

    (SETA norm.can.array 3 (FQUOTIENT (FDIFFERENCE (FPLUS (ELT moment.array 5)
                                                          (FTIMES 6.0 (ELT moment.array 3)
                                                                      w.ave.so))
                                                   (FPLUS (FTIMES 4.0 (ELT moment.array 4)
                                                                      w.average)
                                                          (FTIMES 3.0 (ELT moment.array 2)
                                                                      w.ave.cub)))
                                           zero.cubed))

(* * normalized mu13 =
m13-(3*m12*w.ave)+(3*m11*w.ave.ao)-(x.ave*m03)+(3*m02*x.ave*w.ave)-(3*w.ave.cub*m10))

    (SETA norm.can.array 6 (FQUOTIENT (FDIFFERENCE (FPLUS (ELT moment.array 9)
                                                          (FTIMES 3.0 (ELT moment.array 7)
                                                                      w.ave.so)
                                                          (FTIMES 3.0 (ELT moment.array 3)
                                                                      x.average w.average))
```

```
                                                    (FPLUS (FTIMES 3.0 (ELT moment.array 8)
                                                                       w.average)
                                                           (FTIMES (ELT moment.array 4)
                                                                   x.average)
                                                           (FTIMES 3.0 (ELT moment.array 6)
                                                                   w.ave.cub)))
                                            zero.cubed))

    (* * nore eu22 =
    e22-(2*w.ave*m21)+(w.ave.so*e20)-(2*e12*x.ave)+(4*x.ave*w.ave*e11)-(2*x.ave*w.ave.so*e10) +(x.ave.so*
    m02)-(x.ave.so*w.ave*e01))


        (SETA nore.can.array 9 (FQUOTIENT (FDIFFERENCE (FPLUS (ELT moment.array 12)
                                                              (FTIMES (ELT moment.array 10)
                                                                      w.ave.so)
                                                              (FTIMES 4.0 (ELT moment.array 7)
                                                                      x.average w.average)
                                                              (FTIMES (ELT moment.array 3)
                                                                      x.ave.so))
                                                       (FPLUS (FTIMES 2.0 (ELT moment.array 11)
                                                                      w.average)
                                                              (FTIMES 2.0 (ELT moment.array 8)
                                                                      x.average)
                                                              (FTIMES 2.0 (ELT moment.array 6)
                                                                      x.average w.ave.so)
                                                              (FTIMES (ELT moment.array 2)
                                                                      x.ave.so w.average)))
                                            zero.cubed))

    (* * normalized eu31 =
    e31-(3*e21*x.ave)+(3*e11*x.ave.so)-(w.ave*e30)+(3*e20*x.ave*w.ave)-(3*x.ave.cub*e01))


        (SETA nore.cen.array 11 (FQUOTIENT (FDIFFERENCE (FPLUS (ELT moment.array 14)
                                                               (FTIMES 3.0 (ELT moment.array 7)
                                                                       x.ave.so)
                                                               (FTIMES 3.0 (ELT moment.array 10)
                                                                       x.average w.average))
                                                        (FPLUS (FTIMES 3.0 (ELT moment.array 11)
                                                                       x.average)
                                                               (FTIMES (ELT moment.array 13)
                                                                       w.average)
                                                               (FTIMES 3.0 (ELT moment.array 2)
                                                                       x.ave.cub)))
                                            zero.cubed))
```

```
        (* * normalized mu40 = m40-(4 * x.ave * m30)+(6 * x.ave.so * m20)-(3 * x.ave.cub * m10))


            (SETA norm.cen.array 12 (FQUOTIENT (FDIFFERENCE (FPLUS (ELT moment.array 15)
                                                                   (FTIMES 6.0 (ELT moment.array 10)
                                                                               x.ave.so))
                                                             (FPLUS (FTIMES 4.0 (ELT moment.array 13)
                                                                                x.average)
                                                                    (FTIMES 3.0 (ELT moment.array 6)
                                                                                x.ave.cub)))
                                                    zero.cubed))
)


(* CALLED BY: Calculate.Zernicke.Moments and Calculate.Zernicke.Long. PURPOSE: calculates the Zernicke moment invariants.
REQUIRED VARIABLES: the normalized, centralized moments from Central.Moments, the empty array for the Zernicke moments to
be stored after calculated and the flag to indicate the number of invariants to be calculated. RETURNS: the array
containing the Zernicke moment invariants. CALLS: none.)

(DEFINEQ


(Get.Zernicke.Moments
  (LAMBDA (norm.cen.array zernicke.array flag.i)            (* edited: '27-JAN-87 13:20')

          (* * KEY VARIABLES: PI and PI.so are Pi and Pi squared respectively. -
          intermed.1 = mu20-mu02. -
          intermed.2 = mu03-(3 * mu21). -
          intermed.3 = mu30-(3 * mu12). -
          intermed.4 = mu03 + mu21. -
          intermed.5 = mu30 + mu12. -
          intermed.4.so = intermed.4 * intermed.4. -
          intermed.5.so = intermed.5 * intermed.5. -
          extra.1 = mu40-6mu22+mu04. -
          extra.2 = mu31-mu13. -
          extra.3 = mu04-mu40. -
          extra.4 = 4*((mu31+mu13)-3mu11).)


    (PROG ((PI 3.14159)
           (PI.so 9.8696)
           (intermed.1 (FDIFFERENCE (ELT norm.cen.array 7)
                                    (ELT norm.cen.array 1)))
           intermed.2 intermed.3 intermed.4 intermed.5 intermed.4.so intermed.5.so extra.1 extra.2 extra.3 extra.4)
```

```
        (* * Element 1 = 3 * (2 * (mu20+mu02)-1) / Pi)

        (SETA zernicke.array 1 (FQUOTIENT (FTIMES 3.0 (FPLUS (FTIMES 2.0 (FPLUS (ELT norm.cen.array 7)
                                                                                 (ELT norm.cen.array 1)))
                                                              -1.0))
                                          PI))

        (* * No. 2 = 9 * ((mu20-mu02)^2 + 4*(mu11)^2) / Pi^2)

        (SETA zernicke.array 2 (FQUOTIENT (FTIMES 9.0 (FPLUS (FTIMES intermed.1 intermed.1)
                                                             (FTIMES 4.0 (ELT norm.cen.array 4)
                                                                         (ELT norm.cen.array 4)
                                          PI.so))

        (* * User wants second and third order Zernicke moment invariants.)

        (COND
          ((IGREATERP flag.1 3)
           (SETQ intermed.2 (FDIFFERENCE (ELT norm.cen.array 2)
                                         (FTIMES 3.0 (ELT norm.cen.array 8)
           (SETQ intermed.3 (FDIFFERENCE (ELT norm.cen.array 10)
                                         (FTIMES 3.0 (ELT norm.cen.array 5)
           (SETQ intermed.4 (FPLUS (ELT norm.cen.array 2)
                                   (ELT norm.cen.array 8)))
           (SETQ intermed.5 (FPLUS (ELT norm.cen.array 10)
                                   (ELT norm.cen.array 5)))
           (SETQ intermed.4.sq (FTIMES intermed.4 intermed.4))
           (SETQ intermed.5.sq (FTIMES intermed.5 intermed.5))

        (* * No. 3 = 16 * ((mu03-3*mu21)^2 + (mu30-3*mu12)^2) / Pi^2)

           (SETA zernicke.array 3 (FQUOTIENT (FTIMES 16.0 (FPLUS (FTIMES intermed.2 intermed.2)
                                                                 (FTIMES intermed.3 intermed.3)))
                                             PI.sq))

        (* * No. 4 = 144 * ((mu03+mu21)^2 + (mu30+mu12)^2) / Pi^2)

           (SETA zernicke.array 4 (FQUOTIENT (FTIMES 144.0 (FPLUS intermed.4.sq intermed.5.sq))
                                             PI.sq))
```

```
        (% * No.5 = 13824*((eu03-3eu21)*(eu03+eu21)*((eu03+eu21)^2-3*(eu30+eu12)^2))
        -(eu30-3eu12)*(eu30+eu12)*((eu30+eu12)^2-3*(eu03+eu21)^2) / pi^4)


            (SETA zernicke.arrey 5 (FQUOTIENT [FTIMES 13824.0 (FDIFFERENCE (FTIMES intereed.2 intereed.4
                                                                                (FDIFFERENCE intereed.4.sa
                                                                                             (FTIMES 3.0
                                                                                                     intereed.5.sa)))
                                                                           (FTIMES intereed.3 intereed.5
                                                                                (FDIFFERENCE intereed.5.sa
                                                                                             (FTIMES 3.0
                                                                                                     intereed.4.sa]
                                               (FTIMES PI.sa P1.sa)))

        (% * No. 6 = 864 * ((eu02-a20) ((eu03+eu21)^2-(eu30+eu12)^2) + 4eu1I* (eu03+eu21)*(eu30+euI2)) / pi^3)


            (SETA zernicke.arrey 6 (FQUOTIENT (FTIMES 864.0 (FPLUS (FTIMES (FDIFFERENCE (ELT nore.cen.arrey I)
                                                                                         (ELT nore.cen.arrey 7))
                                                                           (FDIFFERENCE intereed.4.sa intereed.5.sa))
                                                                    (FTIMES 4.0 (ELT nore.cen.arrey 4)
                                                                                intereed.4 intereed.5)))
                                               (FTIMES PI.sa P1]

        (% * User wants second, third and fourth order Zernicke moment invariants.)


        (COND
          ((IGREATERP flas.i 4)
            [SETQ extra.I (FDIFFERENCE (FPLUS (ELT nore.cen.arrey I2)
                                              (ELT nore.cen.arrey 3))
                                       (FTIMES 6.0 (ELT nore.cen.arrey 9]
            (SETQ extra.2 (FDIFFERENCE (ELT nore.cen.arrey 11)
                                       (ELT nore.cen.arrey 6)))

        (% * No.7 = 25*((eu40-6eu22+eu04)^2 + 16*((eu31-eu13)^2)) / pi^2)


            (SETA zernicke.arrey 7 (FQUOTIENT (FTIMES 25.0 (FPLUS (FTIMES extra.1 extra.1)
                                                                   (FTIMES 16.0 extra.2 extra.2)))
                                              PI.sa))
            (SETQ extra.3 (FDIFFERENCE (ELT nore.cen.arrey 3)
                                       (ELT nore.cen.arrey 12)))
            [SETQ extra.4 (FDIFFERENCE (FTIMES 4.0 (FPLUS (ELT nore.cen.arrey 11)
                                                          (ELT nore.cen.arrey 6)))
                                       (FTIMES 3.0 (ELT nore.cen.arrey 4]
```

```
(% % No.8 = 25%{{{{4%(mu04-mu40)+3%(mu20-mu02))^2) + 4%(4%(mu31+mu13) -3mu11)^2)} / pi^2)


    (SETA zernicke.array 8 (FQUOTIENT (FTIMES 25.0 (FPLUS (FTIMES (FPLUS (FTIMES 4.0 extra.3)
                                                                          (FTIMES 3.0 intermed.1))
                                                                  (FPLUS (FTIMES 4.0 extra.3)
                                                                          (FTIMES 3.0 intermed.1)))
                                                           (FTIMES 4.0 extra.4 extra.4)))
                                                PI.sq))

(% % No.9 = 5%(6%(mu40+2mu22+mu04)-6%(mu20+mu02) + 1) / pi)


    (SETA zernicke.array 9 (FQUOTIENT (FTIMES 5.0 (FPLUS (FTIMES 6.0 (FPLUS (ELT norm.cen.array 12)
                                                                             (FTIMES 2.0 (ELT norm.cen.array 9))
                                                                             (ELT norm.cen.array 3)))
                                                          (FMINUS (FTIMES 6.0 (FPLUS (ELT norm.cen.array 1)
                                                                                     (ELT norm.cen.array 7]
                                                          1)))
                                                PI))

(% % No.10 = 250%{(mu40-6mu22+mu04)%{{4%(mu04-mu40)+3%(mu20-mu02))^2
-4%{4%(mu31+mu13)-3mu11)^2)-16%{4%(mu04-mu40)+3%(mu20-mu02))%{4%(mu31+ mu13)-3mu11)%(mu31-mu13))/pi^3)


    (SETA zernicke.array 10 (FQUOTIENT [FTIMES 250.0 (FDIFFERENCE (FTIMES extra.1
                                                                          (FDIFFERENCE
                                                                            (FTIMES (FPLUS (FTIMES 4.0 extra.3)
                                                                                           (FTIMES 3.0
                                                                                                   intermed.1))
                                                                                    (FPLUS (FTIMES 4.0 extra.3)
                                                                                           (FTIMES 3.0
                                                                                                   intermed.1)))
                                                                            (FTIMES 4.0 extra.4 extra.4)))
                                                                  (FTIMES 16.0 (FPLUS (FTIMES 4 extra.3)
                                                                                      (FTIMES 3.0 intermed.1))
                                                                          (FTIMES extra.4 extra.2]
                                                (FTIMES PI.sq PI)))

(% % No.11 = 30%{{{4%(mu04-m40)+3%(mu20-mu02))%(mu02-m20)} + 4mu11%{4%(mu31+mu13)-3mu11) / pi^2)


    (SETA zernicke.array 11 (FQUOTIENT [FTIMES 30.0 (FPLUS (FTIMES (FPLUS (FTIMES 4.0
                                                                             (FDIFFERENCE (ELT
                                                                                           norm.cen.array 3)
                                                                                          (ELT
                                                                                           norm.cen.array 12))
                                                                             )
                                                                    (FTIMES 3.0 intermed.1))
```

```
                                                    (FMINUS intermed.i))
                                                (FTIMES 4.0 (ELT norm.cen.array 4)
                                                    (FDIFFERENCE (FTIMES 4.0
                                                                         (FPLUS (ELT
                                                                                norm.cen.array 11)
                                                                                (ELT
                                                                                norm.cen.array 6)))
                                                                 (FTIMES 3.0
                                                                         (ELT norm.cen.array 4)
                                                                 P1.sa))
)


(* CALLED BY: METHOD of FLAVOR INTERMEDIATE.PROCESS. PURPOSE: controls the routines that calculate the Zernicke moments for
 the image using all edge plus interior points. Used to compare actual moment values with values using Green's Theorem to
approximate all the points in the image, as found in METHOD Calculate.Zernicke.Moments. REQUIRED VARIABLES: list of edge
points, list of all interior points, number of edge points, number of interior points and a flag indicating what order of
Zernicke moments to calculate. RETURNS: Zernicke moments in an array. CALLS: Calculate.Real.Moments, Central.Moments and
Get.Zernicke.Moments.)

(DEFINEQ                                                                                                          H-13

                                                                                                                  [5]

(Calculate.Zernicke.Long
  (LAMBDA (edge.list interior.pts.list no.edge.pts no.interior.pts flag.i)
                                                    (* edited: " 2-APR-87 20:21")

          (* * KEY VARIABLES: moment.array is a 15 element array for storing the moments after they have been
          calculated. Their order is the same as described in Calculate.Zernicke.Moments. -
          -
          norm.cen.array is a 12 element array for storing the normalized and centrelized moments.
          Their order is the same as described in Calculate.Zernicke.Moments. -
          -
          zernicke.array is an 11 element array for storing the Zernicke moments. Their order is the same as
          described in Get.Zernicke.Moments.)


    (PROG ((moment.array (ARRAY 15 NIL NIL))
           (norm.cen.array (ARRAY 12 NIL NIL))
           zernicke.array)

          (* * Create the correct size for zernicke.array)
```

```
          (IF (EQ flag.i 3)
              THEN (SETQ zernicke.array (ARRAY 2 NIL NIL))
            ELSEIF (EQ flag.i 4)
              THEN (SETQ zernicke.array (ARRAY 6 NIL NIL))
            ELSE (SETQ zernicke.array (ARRAY 11 NIL NIL)))

          (* * Calculate all the moments for the image.)


          (CalculateR.Real.Moments (APPEND edge.list interior.pts.list)
                                   (IPLUS no.edge.pts no.interior.pts)
                                   moment.array)

          (* * Centralize the moments.)


          (Central.Moments moment.array norm.cen.array flag.i)

          (* * Calculate the Zernicke moments.)


          (Get.Zernicke.Moments norm.cen.array zernicke.array flag.i)
          (RETURN (LIST zernicke.array))
)


(* CALLED BY: Calculate.Zernicke.Lone. PURPOSE: calculates moments up to the fourth order for all image points. Does not
use Green's Theorem for an approximation. REQUIRED VARIABLES: list of all image points, total number of image points and an
 empty array for storing the moments after calculation. RETURNS: moments for the image in an array. CALLS: Matrix,
Find.Value and Set.Value.)

(DEFINEQ

                                                                                                                    [6]

(Calculate.Real.Moments
  [LAMBDA (image.pts.list no.points.i moment.array)          (* edited: "28-JAN-87 09:02")

          (* * KEY VARIABLES: counter.i is a pointer indicating which moment is being calculated presently.
          -
          coord.pointer.i is a pointer into edge.list. -
          -
          moment.matrix is a 15 by the total number of points + 1 matrix; each row contains a different moment,
          and each element in each row contains a the appropriate multiple of x and y terms for that edge point.
          The last element in each row contains the sum of the x and y exponents + 1.0. -
          -
          start.i is an integer initialized to 1 at the beginning of each set of moment calculations.
          It represents the value of moment m 0,0. -
          -
```

```
          Point.c is a coordinate of the present image point being processed. -
          -
          Previous.l is a real number representing the moments as powers of x and y are added on.)

    (PROG ((counter.i 0)
           (coord.Pointer.i 0)
           (moment.matrix (Matrix 15 no.Points.i 0))
           start.i Point.c Previous.l)

          (% * Loop until all edge Points have been considered.)


          (DO (SETQ Point.c (POP image.Pts.list))
              (ADD coord.Pointer.l 1)
              (SETQ counter.l 1)
              (SETQ start.i 1)

          (% * Moments are calculated in a tree fashion; each Point has all its factors of x and y calculated up
          to fourth order, starting at a 0,0. After this number has been stored then another factor of y is
          multiplied in until the maximum has been reached. Then y is reset to zero order and the x order is
          incremented etc.)


              [FOR J FROM 1 TO 5
                 DO (SETQ Previous.i start.i)

          (% * Loop through to obtain the Proper number of multiples of x and y.)


                    (FOR k FROM 1 TO (IDIFFERENCE 6 J)
                       DO (Set.Value moment.matrix counter.i coord.Pointer.i Previous.l)
                          (SETQ Previous.i (ITIMES Previous.i (CADR Point.c)))
                          (ADD counter.i 1))
                    (SETQ start.i (ITIMES start.i (CAR Point.c)]
             REPEATUNTIL (NULL image.Pts.list))

          (% * Loop through each row of the matrix and sum up the elements and obtain the correct moment which is
          stored in moment.array.)


          (SETA moment.array 1 no.Points.i)
          (FOR 1 FROM 2 TO 15 DO (SETA moment.array 1 (FOR m FROM 1 TO no.Points.i SUM (Find.Value moment.matrix 1 m])
)
(DECLARE: DONTCOPY
  (FILEMAP (NIL (4265 6185 (Calculate.Zernicke.Moments 4275 . 6183)) (6740 10134 (Calculate.Moments 6750 . 10132)) (10553
16840 (Centrel.Moments 10563 . 16838)) (17263 23571 (Get.Zernicke.Moments 17273 . 23569)) (24213 25765 (
Calculate.Zernicke.Lonm 24223 . 25763)) (26159 28482 (Calculate.Real.Moments 26169 . 28480)))))
STOP
```

APPENDIX II-I. File REG

(FILECREATED " 8-APR-87 11:20:54" EE:<STOUT>REQ.;3 29235

previous date: " 8-APR-87 11:13:57" EE:<STOUT>REQ.;2)

(PRETTYCOMPRINT REQCOMS)

```
(RPAQQ REQCOMS ((* CALLED BY: METHOD of FLAVOR INTERMEDIATE.PROCESS. PURPOSE: controls the calculations needed for the
                   auto-regressive pattern recognition technique. REQUIRED VARIABLES: list of edge points, center of area
                   for the image, the minimum number of radial vector lengths per 360 degrees (8, 16 or 32), the model
                   order, ie. the number of lag components to consider when solving the matrix equations and the number of
                   edge points in edge.list. RETURNS: an array containing the pattern recognition parameters from the
                   solution of the auto-regressive matrix equations. CALLS: Auto.Reg.Edge, Calculate.Sues and
                   Solve.Auto.Matrix.)
                  (FNS Tor.Auto.Regressive)
                  (* CALLED BY: Tor.Auto.Regressive. PURPOSE: calculates the radial vector lengths for the auto-regressive
                   pattern recognition technique. Technique is from a paper by S. Dubois and F. Glanz published in IEEE
                   Transactions of Pattern Analysis and Machine Intelligence Jan 1986 REQUIRED VARIABLES: list of edge
                   points, center of area for the image, the number of radial vector lengths per 360 degrees and the
                   number of edge points in the list. Requires one of three global variables slope.8, slope.16 or slope.32
                   stored at the end of this file. RETURNS: a list containing the number of radial vector lengths found
                   and a sublist containing all the radial vector lengths. CALLS: Find.Quadrant, Auto.Reg.Help, and
                   Find.Boundaries.)
                  (FNS Auto.Reg.Edge)
                  (* CALLED BY: Auto.Reg.Edge. PURPOSE: finds out which of the four quadrants the present edge points being
                   investigated is in. REQUIRED VARIABLES: none. Accesses all variables from the calling routine. RETURNS:
                   quadrant of the new point. CALLS: none.)
                  (FNS Find.Quadrant)
                  (* CALLED BY: Auto.Reg.Edge. PURPOSE: finds out the two radial length vectors that bound the slope value
                   of the present edge point being investigated. Also checks to see if the present edge point hit a slope
                   value directly. REQUIRED VARIABLES: none. Accesses all variables from the calling routine. RETURNS:
                   boundaries for the new edge point or processes the point if it hit a slope directly. CALLS:
                   Auto.Reg.Help only if the edge point hit a slope exactly.)
                  (FNS Find.Boundaries)
                  (* CALLED BY: Auto.Reg.Edge and Find.Boundaries. PURPOSE: called when the present edge point being
                   investigated has either a horizontal or vertical slope with respect to the center of area or has
                   crossed the slope boundaries of a previous edge point. Processes the radial length vector for that
                   point. REQUIRED VARIABLES: flag, which has four values indicating whether a boundary slope was hit
                   exactly or not. Accesses all other variables from the calling routine. RETURNS: updated auto.reg.list.
                   CALLS: none.)
                  (FNS Auto.Reg.Help)
                  (* CALLED BY: Tor.Auto.Regressive. PURPOSE: calculates the individual lag sues to be used to fore the
                   auto-regressive matrix. REQUIRED VARIABLES: list of radial length vectors, total number of elements in
                   the raw list and the model order. RETURNS: sue.array containing the required lag suesation teres and
                   raw.array containing the autoregressive radial distances. CALLS: none.)
                  (FNS Calculate.Sues)
                  (* CALLED BY: Tor.Auto.Regressive. PURPOSE: sets up the auto-regressive matrix tere by tere and solvos it.
                   REQUIRED VARIABLES: model.ordor, array containing tho lag sues, the total.number.i of elesents in the
                   sue.array and the array containing the radial distanco lengths. RETURNS: auto.array containing tho
```

```
            solution to the matrix equations and the alpha-beta term which is a type of shape signal-to-noise
            parameter. CALLS: Solve.Linear.Eqs, Set.Value and Matrix.)
          (FNS Solve.Auto.Matrix)
          (* Slope values for the three different divisions of 360 degrees. Only the values for one quadrant are
             needed. Basicelly a look-up table. One value is called from Auto.Res.Edge.)
          (VARS slope.8 slope.16 slope.32)))


(* CALLED BY: METHOD of FLAVOR INTERMEDIATE.PROCESS. PURPOSE: controls the calculations needed for the auto-regressive
pattern recognition technique. REQUIRED VARIABLES: list of edge points, center of area for the image, the minimum number of
 radial vector lengths per 360 degrees (8, 16 or 32), the model order, ie, the number of lag components to consider when
solving the matrix equations and the number of edge points in edge.list. RETURNS: an array containing the pattern
recognition parameters from the solution of the auto-regressive matrix equations. CALLS: Auto.Res.Edge, Calculate.Sums and
Solve.Auto.Matrix.)

(DEFINEQ

                                                                                                        [1]

(Tor.Auto.Regressive
  [LAMBDA (edge.list center.of.area.c no.vectors.i model.order.i no.edge.pts)
                                                          (* edited: "10-MAR-87 14:39")

        (* * KEY VARIABLES: regress.list is the list of radial lengths from the center of area to the edge
        points closest to one of the radial vectors. The CAR of this list contains the total number of lengths
        in the list. -
        -
        sum.array is an array containing the summation terms for the required lag components needed to solve
        the matrix equation. The lag numbers increase as the elements in the array do. The first term is zero
        lag. -
        -
        auto.array is the final result of this pattern recognition technique. It contains model.order.i plus
        one parameters which can be used to classify objects. The model.order.i parameters represent the lag
        constants and the last term is a combination of the alpha and beta terms. -
        -
        return.list is a list containing two arrays that represents the return variables from a call to
        Calculate.Sums. -
        -
        raw.array is an array containing the radial lengths contained in regress.list)


    (PROG (regress.list sum.array auto.array return.list raw.array)

        (* * Call to obtain the radial lengths.)
```

```
        (SETQ regress.list (Auto.Reg.Edge edge.list center.of.area.c no.vectors.i no.edge.pts))

        (* * Call to find the sucession terms required to set up the matrix to be solved.)

        (SETQ return.list (Calculate.Sues (CADR regress.list)
                                          (CAR regress.list)
                                          model.order.i))

        (* * Set up the two arrays returned from the call to Calculate.Sues.)

        (SETQ sue.array (CAR return.list))
        (SETQ rav.array (CADR return.list))

        (* * Solve the matrix equation.)

        (SETQ auto.array (Solve.Auto.Matrix model.order.i sum.array (CAR regress.list)
                                            rav.array))
        (RETURN (LIST regress.list auto.array))
)
```

I-3

(* CALLED BY: Top.Auto.Regressive. PURPOSE: calculates the radial vector lengths for the auto-regressive pattern recognition technique. Technique is from a paper by S. Dubois and F. Glenz published in IEEE Transactions of Pattern Analysis and Machine Intelligence Jan 1986 REQUIRED VARIABLES: list of edge points, center of area for the image, the number of radial vector lengths per 360 degrees and the number of edge points in the list. Requires one of three global variables slope.8, slope.16 or slope.32 stored at the end of this file. RETURNS: a list containing the number of radial vector lengths found and a sublist containing all the radial vector lengths. CALLS: Find.Quadrant, Auto.Reg.Help, and Find.Boundaries.)

```
(DEFINEQ

(Auto.Reg.Edge                                                          [2]
  [LAMBDA (edge.list center.of.area.c no.vectors.i no.edge.pts)
                                                  (* edited: " 9-FEB-87 13:31")

        (* * KEY VARIABLES: slope.array is an array containing the magnitude of the radius vectors for the
        first quadrant. Because a sequential boundary boundary follower is used to obtain an ordered set of
        radius vector-boundary intersection lengths, need not determine the actual quadrant in which a
        particular radius vector lies. -
        -
        auto.reg.list is a list containing the radius vector lengths for the pixels that were boundary points
        to the radius vectors calculated for slope.array -
        -
```

order.i is an integer counter which keeps track of how many radius vector lengths are contained in auto.reg.list. Can be greater than no.vectors.i if the image is nonconvex. -

first.flag.b is a boolean that is TRUE only for the first point in edge.list. -

array.counter.i is a pointer into slope.array and is pointing to a slope that is greater than or equal to the slope of the point being processed. -

min.pointer.i is the slope.array pointer indicating the lower bound for the previous point processed.

max.pointer.i is the slope.array pointer indicating the upper bound for the previous point processed.

mid.pointer.i is the slope.array pointer indicating the slope ray that was hit exactly by the previous point processed. Normally it is equal to zero. -

denominator.r is the difference in the x coordinates of the current point being considered and the center.of.area.c ie. the difference in cols. -

numerator.r is the difference in the y coordinates of the current point being considered and the center.of.area.c ie. the difference in rows. -

radial.length.r is the length of one of the intersected radius vectors in Euclidean terms.

slope is the slope of next.pt or the string 'vertical' with respect to the center.of.area.c.

next.pt.c is the coordinate of the present edge point being investigated. -

lower.pointer.i is the slope.array pointer indicating the lower bound for the present point processed.

upper.pointer.i is the slope.array pointer indicating the upper bound for the present point processed.

mid.pointer.i is the slope.array pointer indicating the slope ray that was hit exactly by the present point processed. Normally it is equal to zero. -

temp.slope is a temporary variable used when reading in the values for slope.array.
Set equal to the global variable containing the slope values. -

size.of.array.i is the number of slope values in slope.array. -

counter.i is an integer counter used when the no.edge.pts is smaller than the desired number of vectors. -

prev.quadrant.i is the quadrant location of the previous point processed. -

quadrant.i is the quadrant location of the present point being processed.)

```
(PROG ((auto.res.list NIL)
       (order.i 1)
       (first.flag.b T)
       (array.counter.i 0)
       (min.pointer.i 0)
       (end.pointer.i 0)
       (max.pointer.i 0)
       (prev.quadrant.i 0)
       denominator.r numerator.r radial.length.r slope next.pt.c upper.pointer.i mid.pointer.i lower.pointer.i
       slope.array temp.slope size.of.array.i counter.i quadrant.i)

      (* * See if there are insufficient edge points to cover the number of vectors desired.
      IF so THEN decrease the size of no.vectors.i.)


      (IF (IGREATERP no.vectors.i no.edge.pts)
          THEN (SETQ counter.i 3)
               (DO (ADD counter.i 1) WHILE (FGREATERP no.vectors.i (EXPT 2 counter.i)))
               (SETQ no.vectors.i (EXPT 2 counter.i)))

      (* * Initialize array parameters.)


      (SETQ size.of.array.i (ADD1 (IQUOTIENT no.vectors.i 4)))
      (SETQ slope.array (ARRAY size.of.array.i NIL MIL))

      (* * Read in the slope.array. Uses a look-up table since the slope values are already stored in this
      file. Values are from 0 to vertical covering 90 degrees.)


      [SELECTQ size.of.array.i
               [3 (SETQ temp.slope slope.8)
                  (FOR i FROM 1 TO size.of.array.i DO (SETA slope.array i (POP temp.slope]
               [5 (SETQ temp.slope slope.16)
                  (FOR i FROM 1 TO size.of.array.i DO (SETA slope.array i (POP temp.slope]
               (AND (SETQ temp.slope slope.32)
                    (FOR i FROM 1 TO size.of.array.i DO (SETA slope.array i (POP temp.slope]

      (* * Loop until all edge points have been processed.)


      (DO (SETQ next.pt.c (POP edge.list))
          (SETQ denominator.r (FDIFFERENCE (CADR next.pt.c)
                                           (CADR center.of.area.c)))
          (SETQ numerator.r (FDIFFERENCE (CAR next.pt.c)
                                         (CAR center.of.area.c)))
```

```
        (* * Find the quadrant for next.pt.c)

            (Find.Quadrant)

        (* * See if the denominator.r of the slope equation equals 0. IF it is THEN the next.pt.c is in the
        same col as the center.of.area.c and need to call Auto.Res.Help because we landed right on one of the
        boundary segment vectors.)

            [IF (FEQP denominator.r 0.000000E+00)
                THEN (SETQ mid.Pointer.i size.of.array.i)
                     (SETQ lower.Pointer.i (SUB1 size.of.array.i))
                     (SETQ upper.Pointer.i (SUB1 size.of.array.i))
                     (Auto.Res.Help "denominator")

        (* * See if the numerator.r of the slope equals 0.000000E+00 IF it does THEN process next.pt.c as in
        above case.)

              ELSEIF (FEQP numerator.r 0.000000E+00)
                 THEN (SETQ mid.Pointer.i 1)
                      (SETQ lower.Pointer.i 2)
                      (SETQ upper.Pointer.i 2)
                      (Auto.Res.Help "numerator")

        (* * ELSE calculate the slope for next.pt.c. because it did not hit a vertical or horizontal slope
        exactly.)

              ELSE (SETQ slope (ABS (FQUOTIENT numerator.r denominator.r)))
                   (SETQ mid.Pointer.i 0)

        (* * Call to find the slope.array bounds for next.pt.c or to see if it is exactly on one of the
        slope.array segments, excluding the vertical and horizontal cases.)

                   (Find.Boundaries)

        (* * Enter this block only if this is not the first point in edge.list OR a slope raw was not hit
        exactly by next.pt.c.)

                   (IF (AND (EQ mid.Pointer.i 0)
                            (NOT first.flag.b))
                       THEN
```

```
(* * The previous point was between two slopes so see if a boundary segment was crossed by the present
point. IF one was find out which one and call Auto.Req.Help to process the point.)


                    (IF (OR (AND (NEQ min.pointer.i lower.pointer.i)
                                 (NEQ max.pointer.i upper.pointer.i)
                                 (EQ med.pointer.i 0))
                            (AND (NEQ quadrant.i prev.quadrant.i)
                                 (EQ max.pointer.i upper.pointer.i)
                                 (EQ min.pointer.i lower.pointer.i)))
                        THEN (Auto.Req.Help 'normal'))

(* * ELSEIF the previous point hit a slope array segment exactly, but not a vertical or horizontal
intersection THEN see if the present point has crossed either of the boundaries.
IF so THEN call Auto.Req.Help to process the point.)


                  ELSEIF (AND (NEQ min.pointer.i max.pointer.i)
                              (NEQ med.pointer.i 0))
                    THEN (IF (OR (AND (NEQ lower.pointer.i min.pointer.i)
                                      (NEQ lower.pointer.i med.pointer.i))
                                 (NEQ quadrant.i prev.quadrant.i))
                             THEN (Auto.Req.Help 'normal'))

(* * ELSEIF previous point hit a vertical or horizontal slope exactly so check if next.pt.c has crossed
a boundary.)


                  ELSEIF (EQ min.pointer.i max.pointer.i)
                    THEN

(* * Previous boundary was vertical.)


                       (IF (IGREATERP med.pointer.i max.pointer.i)
                           THEN (IF (NEQ lower.pointer.i min.pointer.i)
                                    THEN (Auto.Req.Help 'noraal')))

(* * Previous boundary was horizontal.)


                         ELSEIF (NEQ upper.pointer.i min.pointer.i)
                           THEN (Auto.Req.Help 'normal'])
```

```
          (% * Reset the bounds before getting the next point in edge.list.)

              (SETQ min.pointer.i lower.pointer.i)
              (SETQ max.pointer.i upper.pointer.i)
              (SETQ med.pointer.i mid.pointer.i)
              (SETQ prev.quadrant.i quadrant.i)
              (SETQ first.flag.b NIL) WHILE (LISTP edge.list))
          (RETURN (LIST (SUB1 order.i)
                        auto.req.list))
)


(% CALLED BY: Auto.Req.Edge. PURPOSE: finds out which of the four quadrants the present edge points being investigated is
in. REQUIRED VARIABLES: none. Accesses all variables from the calling routine. RETURNS: quadrant of the new point. CALLS:
none.)

(DEFINEQ

                                                                                                              [3]
(Find.Quadrant                                                                                                I-8
  [LAMBDA NIL                                          (% edited: " 6-FEB-87 16:01")

          (% * Find out which quadrant next.pt.c is in.)


    (IF (OR (FLESSP numerator.r 0)
            (FEQP numerator.r 0))
        THEN (IF (FGREATERP denominator.r 0)
                 THEN (SETQ quadrant.i 1)
               ELSEIF (FLESSP numerator.r 0)
                 THEN (SETQ quadrant.i 2)
               ELSE (SETQ quadrant.i 3))
      ELSEIF (OR (FGREATERP denominator.r 0)
                 (FEQP denominator.r 0))
        THEN (SETQ quadrant.i 4)
      ELSE (SETQ quadrant.i 3))
)


(% CALLED BY: Auto.Req.Edge. PURPOSE: finds out the two radial length vectors that bound the slope value of the present
edge point being investigated. Also checks to see if the present edge point hit a slope value directly. REQUIRED VARIABLES:
none. Accesses all variables from the calling routine. RETURNS: boundaries for the new edge point or processes the point if
 it hit a slope directly. CALLS: Auto.Req.Help only if the edge point hit a slope exactly.)
```

```
(QEFINEQ

                                                                                [4]
(Find.Bounderies
  (LAMBDA HIL                                          (* edited: " 8-FEB-87 12:31")
    (PROQ HIL
          (SETQ arrey.counter.i 1)

          (* * Find the value of arrey.counter.i that points to a slope that is greeter then or equal to the
          slope of next.pt.c. IF hit slope rey exactly THEN call Auto.Res.Help to process the point.)


          (QQ (IF (FEQP slope (ELT slope.arrey arrey.counter.i))
                  THEN (SETQ lower.pointer.i (SUB1 arrey.counter.i))
                       (SETQ mid.pointer.i arrey.counter.i)
                       (SETQ upper.pointer.i (ADD1 arrey.counter.i))
                       (Auto.Res.Help "boundary")
              ELSE (ADD arrey.counter.i 1))
             WHILE (AND (ILESSP arrey.counter.i size.of.arrey.i)
                        (EQ mid.pointer.i 0)
                        (OR (FGREATERP slope (ELT slope.arrey arrey.counter.i))
                            (FEQP slope (ELT slope.arrey arrey.counter.i]

          (* * Calculate the upper end lower slope.arrey bounds for next.pt.c if it did not hit a slope.arrey
          segemtn exactly.)


          (IF (EQ mid.pointer.i 0)
              THEN (SETQ lower.pointer.i (SUB1 arrey.counter.i))
                   (SETQ upper.pointer.i arrey.counter.i))
)


(* CALLED BY: Auto.Res.Edge and Find.Boundaries. PURPOSE: called when the present edge point being investigated hes either
a horizontal or vertical slope with respect to the center of area or has crossed the slope boundaries of a previous edge
point. Processes the radial length vector for thet point. REQUIRED VARIABLES: flag, which has four values indicating
whether a boundary slope was hit exactly or not. Accesses all other variables from the calling routine. RETURNS: updated
auto.res.list. CALLS: none.)

(DEFINEQ

                                                                                [5]
(Auto.Res.Help
  (LAMBDA (flag)                                       (* edited: " 9-FEB-87 10:47")
    (PROQ NIL
```

```
        (* * IF landed on a vertical boundary segment vector and the previous quadrant was different, THEN
        process the new intersection radius vector.)

        (IF (AND (EQUAL flag 'denominator')
                 (OR (NEQ quadrant.i prev.quadrant.i)
                     (NEQ mid.pointer.i mod.pointer.i)))
            THEN (PUSH auto.reg.list (ABS numerator.r))
                 (ADD order.i 1)

        (* * ELSEIF numerator.r equals zero process the new intersection radius vector if the quadrant was
        different.)


          ELSEIF (AND (EQUAL flag 'numerator')
                      (OR (NEQ quadrant.i prev.quadrant.i)
                          (NEQ mid.pointer.i mod.pointer.i)))
            THEN (PUSH auto.reg.list (ABS denominator.r))
                 (ADD order.i 1)

        (* * ELSEIF next.pt.c hit a slope boundary segment exactly.)


          ELSEIF (AND (EQUAL flag 'boundary')
                      (OR (NEQ quadrant.i prev.quadrant.i)
                          (NEQ mid.pointer.i mod.pointer.i)))
            THEN (SETQ radial.length.r (SQRT (FPLUS (FTIMES numerator.r numerator.r)
                                                    (FTIMES denominator.r denominator.r]
                 (PUSH auto.reg.list radial.length.r)
                 (ADD order.i 1)

        (* * ELSEIF a normal boundary segment was crossed so process the new intersection radius vector.)


          ELSEIF (EQUAL flag 'normal')
            THEN (SETQ radial.length.r (SQRT (FPLUS (FTIMES numerator.r numerator.r)
                                                    (FTIMES denominator.r denominator.r]
                 (PUSH auto.reg.list radial.length.r)
                 (ADD order.i 1])
)


(* CALLED BY: Top.Auto.Regressive. PURPOSE: calculates the individual lag sums to be used to form the auto-regressive
matrix. REQUIRED VARIABLES: list of radial length vectors, total number of elements in the raw list and the model order.
RETURNS: sum.array containing the required lag summation terms and raw.array containing the autoregressive radial
distances. CALLS: none.)
```

```
(DEFINEQ

(Calculate.Sums                                                              [6]
  (LAMBDA (raw.list total.number.i modal.order.i)          (* edited: '10-MAR-87 15101')

          (* * KEY VARIABLES: raw.array is an array into which are put the radial lengths from raw.list.
          -
          array.counter.i is a pointer into raw.array -
          -
          las.number.i is an integer which keeps track of how far two las components are apart while sums are
          being calculated. -
          -
          basin.element.i is a pointer into raw.array that tells this function which term is to be multiplied in.
          element (basin.element.i) is multiplied by element (basin.element.i plus las.number.i).
          -
          sum.r is the real number summation of one of the las component terms. -
          -
          loop.counter.i is a counter that keeps track of how many summation terms must be calculated to obtain
          the proper modal.order.i. -
          -
          sum.array is the final result of this function and contains the summation terms for the required number
          of las components.)


    (PROG ((raw.array (ARRAY total.number.i 0 NIL 0))
           (array.counter.i 0)
           (las.number.i 0)
           (basin.element.i 0)
           (sum.r 0.000000E+00)
           loop.counter.i sum.array)

          (* * Because of symmetry only half of the summation terms need to be calculated.
          IF there is an odd number of terms THEN one extra terms must be calculated.)


          (IF (ODDP total.number.i)
              THEN (SETQ loop.counter.i (IQUOTIENT (ADD1 total.number.i)
                                                   2))
                   (SETQ sum.array (ARRAY (ADD1 loop.counter.i)
                                          0 NIL 0))
            ELSE (SETQ loop.counter.i (IQUOTIENT total.number.i 2))
                 (SETQ sum.array (ARRAY (IPLUS loop.counter.i 2)
                                        0 NIL 0)))
```

```
(* * See how many summation terms (model.order.i) the user is interested in keeping and calculate just
that number of terms.)


(IF (IGREATERP loop.counter.i (ADD1 model.order.i))
    THEN (SETQ loop.counter.i (ADD1 model.order.i)))

(* * MAPping function that places each element in ray.list into an array in order.)


(MAPCAR ray.list (QUOTE (LAMBDA (temp)
                                (SETA ray.array array.counter.i temp)
                                (ADD array.counter.i 1)

(* * Loop to obtain the proper number of summation terms. The inner loop insures that all elements in
the original ray.list are included in each summation. First each element is multiplied by itself then
all are summed. The second time through each element is multiplied by an element adjacent to it
(les.number.i equals one). The third time les.number.i is two etc. Place each sum in the proper place
in sum.array)


(FOR x FROM 1 TO loop.counter.i
   DO (FOR y FROM 1 TO total.number.i
         DO (SETQ sum.r (FPLUS sum.r (FTIMES (ELT ray.array begin.element.i)
                                             (ELT ray.array (IMOD (IPLUS begin.element.i las.nueber.i)
                                                                  total.number.i)

            (ADD begin.element.i 1))
      (SETA sum.array les.number.i sum.r)
      (ADD las.nueber.i 1)
      (SETQ begin.element.i 0)
      (SETQ sum.r 0.000000E+00))

(* * Because of the additional symmetry of the middle term in an even series, only half of the
summation terms are required, since the other half has the identical sue.)


(IF (AND (EVENP total.number.i)
         (IOEQ model.order.i (IQUOTIENT total.number.i 2)))
    THEN (FOR z FROM 1 TO (IQUOTIENT total.number.i 2)
            DO (SETQ sum.r (FPLUS sum.r (FTIMES (ELT ray.array begin.element.i)
                                                (ELT ray.array (IMOD (IPLUS begin.element.i las.number.i)
                                                                     total.number.i)

               (ADD begin.element.i 1))
         (SETA sum.array las.number.i (FTIMES sum.r 2))
         (IF (IGREATERP model.order.i (IQUOTIENT total.number.i 2))
             THEN (SETA sum.array (ADD1 las.number.i)
                        (ELT sum.array (SUB1 las.number.i)
  ELSEIF (AND (ODDP total.number.i)
              (IGREATERP model.order.i (IQUOTIENT total.number.i 2)))
```

```
            THEN (SETA sum.array las.number.i (ELT sue.array (SUB1 las.number.i]
          (RETURN (LIST sum.array rav.array])
)


(* CALLED BY: Top.Auto.Regressive. PURPOSE: sets up the auto-regressive matrix term by term and solves it. REQUIRED
VARIABLES: model.order, array containing the las sums, the total.number.i of elements in the sum.array and the array
containing the radial distance lengths. RETURNS: auto.array containing the solution to the matrix equations and the
alpha-beta term which is a type of shape signal-to-noise parameter. CALLS: Solve.Linear.Eqs, Set.Value and Matrix.)

(DEFINEQ
```

[73]

```
(Solve.Auto.Matrix
  (LAMBDA (model.order.i sue.array total.number.i rav.array)
                                                  (* edited: '25-FEB-87 09:23')

          (* * KEY VARIABLES: beta is one of the shape signal-to-noise parameters from the basic equation being
          solved. It is an error terms and is minimized by the solution of the matrix equation.
          -
          alpha is the other shape signal-to-noise parameter and is representative of the overall shape.
          -
          temp.r is a temporary real number used in the calculation of beta. -
          -
          auto.matrix is the matrix containing the proper order of las summation teres and is the basic element
          in the matrix equation. -
          -
          row.flag.i is a pointer to the current row being processed in matrix.matrix. -
          -
          col.flag.i is a pointer to the current colum being processed in matrix.matrix.
          -
          counter.i is a pointer into sue.arraw. -
          -
          auto.array contains the vector portion of the matrix equation to be solved and after the call to
          Solve.Linear.Eqs it contains the solution vector.)


    (PROG ((beta 0.000000E+00)
           alpha temp.r auto.matrix row.flag.i col.flag.i counter.i auto.array)

          (* * Set up the auto-regressive matrix and initialize elements in it to the sum of all the individual
          radial lengths. This is done because the entire last column in the matrix contains this number except
          for the bottom row.)
```

```
[SETQ auto.matrix (Matrix (ADDi model.order.i)
                          (ADDI model.order.i)
                          (SORT (ELT sum.array 0]

(* * Loop to obtain the proper number of lag components in auto.matrix. The outer loop controls the row
and the inner one controls the column of the matrix. As one moves across a row the lag components
increase by one. As one moves down a column the lag components also increase. Lots of duplication in
matrix terms.)


(SETQ row.flag.i 0)
(SETQ col.flag.i 0)
(FOR i FROM 1 TO model.order.i
   DO (SETQ counter.i 0)
      (ADD row.flag.i 1)
      (ADD col.flag.i 1)
      (FOR J FROM row.flag.i TO model.order.i
         DO (Set.Value auto.matrix row.flag.i J (ELT sum.array counter.i)))
            (ADD counter.i i))
      (SETQ counter.i 0)
      (FOR J FROM row.flag.i TO model.order.i
         DO (Set.Value auto.matrix J col.flag.i (ELT sum.array counter.i)))
            (ADD counter.i i)))

(* * Set the element in the last row and column to total.number.i.)


(Set.Value auto.matrix (ADDi model.order.i)
           (ADDI model.order.i)
           total.number.i)

(* * Set up the vector to be passed into Solve.Linear.Eqs. It also contains lag components.)


[SETQ auto.array (ARRAY (ADDi model.order.i)
                        0
                        (SORT (ELT sum.array 0]

(* * Calculate the rest of the terms for auto.array.)


[FOR i FROM 1 TO model.order.i DO (SETA auto.array i (ELT sum.array (1ADD i total.number.i]

(* * Call to solve the matrix equations. The solution vector is returned in auto.array.)
```

```
        (Solve.Linear.Eqs auto.matrix auto.array (ADD1 model.order.i))

        (* * Calculate the alpha parameter.)


        (SETQ alpha (ELT auto.array (ADD1 model.order.i)))

        (* * Calculate the beta parameter.)


        [FOR i FROM 0 TO (SUB1 total.number.1)
           DO [SETQ temp.r (FDIFFERENCE (FDIFFERENCE (ELT raw.array 1)
                                                     alpha)
                                        (FOR J FROM 1 TO model.order.i SUM (FTIMES (ELT auto.array J)
                                                                                   (ELT raw.array
                                                                                        (IMOD (IDIFFERENCE i J)
                                                                                              total.number.1]
                (SETQ beta (FPLUS beta (FTIMES temp.r temp.r]
        (SETQ beta (FQUOTIENT beta total.number.i))

        (* * Place the alpha-beta parameter back into auto.array.)


        (SETA auto.array (ADD1 model.order.1)
              (FQUOTIENT alpha (SQRT beta)))
        (RETURN auto.array])
)


(* Slope values for the three different divisions of 360 degrees. Only the values for one quadrant are needed. Basically a
look-up table. One value is called from Auto.Res.Edge.)


(RPAQQ slope.8 (0.000000E+00 1.0 "vertical"))

(RPAQQ slope.16 (0 0.414214 1.0 2.41421 "vertical"))

(RPAQQ slope.32 (0 0.198912 0.414214 0.668179 1.0 1.49661 2.41421 5.02734 "vertical"))
(DECLARE: DONTCOPY
  (FILEMAP (NIL (4787 6773 (Top.Auto.Regressive 4797 . 6771)) (7508 15731 (Auto.Res.Edge 7518 . 15729)) (15989 16523 (
Find.Quadrant 15999 . 16521)) (16989 18120 (Find.Boundaries 16999 . 18118)) (18640 20194 (Auto.Res.Help 18650 . 20192)) (
20580 24521 (Calculate.Sums 20590 . 24519)) (24994 28840 (Solve.Auto.Matrix 25004 . 28838)))))
STOP
```

APPENDIX II-J. File SET_UP_SYSTEM

```
EE:<STOUT>SET_UP_SYSTEM.;3

(FILECREATED " 8-APR-87 12:13:57" EE:<STOUT>SET_UP_SYSTEM.;3 72305

     previous date: " 8-APR-87 11:58:26" EE:<STOUT>SET_UP_SYSTEM.;2)


(PRETTYCOMPRINT SET_UP_SYSTEMCOMS)

(RPAQQ SET_UP_SYSTEMCOMS ((* CALLED BY: Top level routine, Skeleton. PURPOSE: Contains all the FLAVOR instructions
                             required to set up the correct environment. All the FLAVORS and their METHODS are defined,
                             but no INSTANCES are created. Each FLAVOR and METHOD will be discussed in more detail below.
                             The basic METHOD consists of a before demon which obtains the needed parameters, a main
                             METHOD which does the actual work and an after demon which places the return parameters from
                             the main METHOD into the FLAVOR environment. This is a very long function. REQUIRED
                             VARIABLES: varies. RETURNS: varies. CALLS: FLAVOR routines: DEFFLAVOR, DEFMETHOD,
                             MAKE.INSTANCE, SEND.MESSAGE, PUT.VALUE, GET.VALUE and PRINT.INSTANCE. Also calls Befors.Help
                             and After.Help and most of the lower level INTERLISP routines.)
                           (FMS Set.Up.Flavors)
                           (* CALLED BY: Set.Up.Flavors. PURPOSE: pulls the required variables off of the variable list of
                              the calling FLAVOR and returns them so that the main METHOD can pass the correct parameters.
                              REQUIRED VARIABLES: the calling FLAVOR name and a list of required FLAVOR variables. RETURNS:
                              a list containing the values for the input required variables or an error message. CALLS:
                              GET.VALUE.)
                           (FMS Before.Help)
                           (* CALLED BY: Set.Up.Flavors. PURPOSE: takes the list of variables returned from the main METHOD
                              and places them into the proper FLAVOR variable slots. REQUIRED VARIABLES: the calling FLAVOR
                              name and a list containing the names of the variables. The variable values are returned in a
                              list contained in the variable names main.return from SEND.MESSAGE. RETURNS: the list of
                              variables returned from the main METHOD. CALLS: PUT.VALUE.)
                           (FMS After.Help)))


(* CALLED BY: Top level routine, Skeleton. PURPOSE: Contains all the FLAVOR instructions required to set up the correct
environment. All the FLAVORS and their METHODS are defined, but no INSTANCES are created. Each FLAVOR and METHOD will be
discussed in more detail below. The basic METHOD consists of a before demon which obtains the needed parameters, a main
METHOD which does the actual work and an after demon which places the return parameters from the main METHOD into the
FLAVOR environment. This is a very long function. REQUIRED VARIABLES: varies. RETURNS: varies. CALLS: FLAVOR routines:
DEFFLAVOR, DEFMETHOD, MAKE.INSTANCE, SEND.MESSAGE, PUT.VALUE, GET.VALUE and PRINT.INSTANCE. Also calls Before.Help and
After.Help and most of the lower level INTERLISP routines.)

(DEFINEQ


(Set.Up.Flavors
  [LAMBDA NIL                                                                                          [1]
    (PROG NIL
                                        (* edited: " 5-APR-87 13:04")
```

J-1

(* * Define the top FLAVOR VANILLA and its METHODS. All other FLAVORS inherit VANILLA's variables and METHODS. -

FLAVOR VARIABLES: quality.flag is a boolean flag that when TRUE indicates that the user wants to calculate measures of classifier quality. -

max.size is an integer representing the size of one dimension of all the matrices that will be input.

matrix.instances.list is a list of the names of all the instances of FLAVOR MATRIX. The first one on the list is the one presently being analyzed. -

matrix.counter is an integer counter that is one greater than the number of instances of FLAVOR MATRIX.

list.of.matrices is a list of the actual names of the matrices that have been loaded into the environment. -

library.file.name is the user input name of the file containing the library lease feature vectors.

library.counter is an integer counter that is one greater than the number of instances of FLAVOR LIBRARY. -

intermediate.process.counter is an integer counter that is one greater than the number of instances of FLAVOR INTERMEDIATE.PROCESS. -

lease.pixel is the character value of an lease pixel in the matrices loaded into the environment.

dribble.file.name is the name of the file that contains a printout of the present session.

which.IM.process is the name of the intermediate process the user choose. -

which.classifier is the name of the classifier the user choose. -

current.matrix is the name of the current matrix being analyzed. -

current.intermediate.process is the name of the current instance of FLAVOR INTERMEDIATE.PROCESS.

current.library is the name of the current instance of FLAVOR LIBRARY. -

cluster.flag is a boolean that when TRUE indicates that the user wants to cluster the leases contained in the input matrices. -

background.pixel is the character value of a background pixel in the matrices loaded into the environment. -

adaptive.flag is a boolean flag that when TRUE indicates that the user wants to make the classifier adaptive.)

```
(DEFFLAVOR (QUOTE VANILLA)
           (QUOTE ((quality.flag NIL)
                   (max.size 32)
                   matrix.instances.list
                   (matrix.counter 1)
                   list.of.matrices
                   (library.file.name NIL)
                   (library.counter 1)
                   (intermediate.process.counter 1)
                   (image.pixel x)
                   dribble.file.name
                   (which.IM.process Calculate.Zernicke.Moments)
                   (which.classifier Weighted.Classifier)
                   current.matrix current.intermediate.process current.library (cluster.flag NIL)
                   (background.pixel 1.)
                   (adaptive.flag NIL)
```

(* * Get.User.Variables prompts the user to input system parameters. No before demon is needed.)

```
(DEFMETHOD (QUOTE VANILLA)
           (QUOTE Get.User.Variables)
           NIL
           (QUOTE Get.User.Variables))
```

(* * KEY VARIABLES! copy.main.return is an exact copy of variable main.return. -
current.IM.instance is the name of the current instance of FLAVOR INTERMEDIATE.PROCESS.)

```
(DEFMETHOD (QUOTE VANILLA)
           (QUOTE Get.User.Variables)
           (QUOTE after)
           (QUOTE (LAMBDA NIL (PROG [(copy.main.return (COPY main.return))
                                     (current.IM.instance (CADR (DET.VALUE (QUOTE VANILLA)
                                                                           (QUOTE VANILLA)
                                                                           (QUOTE current.intermediate.process]
```

(* * IF main.return did not signal an error THEN process all the variables.)

```
                                    (IF (NOT (STRINGP main.return))
                                        THEN (SETQ main.return (CAR copy.main.return))
```

(* * Process the VANILLA variables.)

J-3

```
                                        (After.Help (QUOTE VANILLA)
                                                    (LIST (QUOTE cluster.flag)
                                                          (QUOTE list.of.matrices)
                                                          (QUOTE library.file.name)
                                                          (QUOTE which.IM.process)
                                                          (QUOTE which.classifier)
                                                          (QUOTE adaptive.flag)
                                                          (QUOTE quality.flag)))
                                        (SETQ copy.main.return (CADR copy.main.return))

(* * Process the INTERMEDIATE.PROCESS variables.)

                                        (PUT.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                                   current.IM.instance
                                                   (QUOTE model.order)
                                                   (POP copy.main.return))
                                        (PUT.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                                   current.IM.instance
                                                   (QUOTE no.slopes)
                                                   (POP copy.main.return))
                                        (PUT.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                                   current.IM.instance
                                                   (QUOTE type.flag)
                                                   (POP copy.main.return))
                                        (PUT.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                                   current.IM.instance
                                                   (QUOTE distance.power)
                                                   (POP copy.main.return))
                                        (PUT.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                                   current.IM.instance
                                                   (QUOTE no.moments)
                                                   (POP copy.main.return)))
                                (RETURN copy.main.return)

(* * Make.Lower.Instance creates an instance of a subordinate FLAVOR and keeps track of the current
instances. REQUIRED VARIABLES: name of the parent FLAVOR, instance name of the parent FLAVOR, the name
of the counter for the new FLAVOR, new FLAVOR type, whether this is a name or an actual matrix, flag
indicating whether the new instance should be pushed onto a list of instances and any variable values
to be associated with the instance creation. -
-
KEY VARIABLES: new.instance is the name of the instance for the new FLAVOR. -
sequence.number is the number of the new instance for that type of FLAVOR. -
variable.name is the name of a variable which contains a list of instances of a subordinate FLAVOR.
temp.list contains the list of instances of the above variable.)
```

```
[DEFMETHOD (QUOTE VANILLA)
           (QUOTE Make.Lower.Instance)
           NIL
           (QUOTE (LAMBDA (parent.flavor parent.name counter.name flavor.name new.current rush.flag
                           variable.values)
                          (PROG (new.instance sequence.number variable.name temp.list)
                                (* Get the new instance number and increment it.)
                                (SETQ sequence.number (CADR (GET.VALUE parent.flavor parent.name counter.name)))
                                (PUT.VALUE parent.flavor parent.name counter.name (ADD1 sequence.number))

(* * Create a new name by placing sequence number at the end of the FLAVOR type.)

                                (SETQ new.instance (PACK* flavor.name (QUOTE %.)
                                                          sequence.number))

(* * IF one of these two FLAVORS is to created THEN append the parent FLAVOR's name on also.)

                                (IF (OR (EQUAL flavor.name (QUOTE IMAGE))
                                        (EQUAL flavor.name (QUOTE CLASSIFIER)))
                                    THEN (SETQ new.instance (PACK* new.instance (QUOTE %.)
                                                                   parent.name)))

(* * Create the instance of the new FLAVOR.)

                                (MAKE.INSTANCE flavor.name new.instance parent.name variable.values)

(* * Check which variable to make current, ie. new.current or the EVAL of it) used for a new instance
of MATRIX.)

                                (IF (NULL new.current)
                                    THEN (PUT.VALUE parent.flavor parent.name (PACK* (QUOTE current.)
                                                                                     (L-CASE flavor.name))
                                                    new.instance)
                                  ELSE (PUT.VALUE parent.flavor parent.name (PACK* (QUOTE current.)
                                                                                   (L-CASE flavor.name))
                                                  (EVAL new.current)))

(* * Entered only for those FLAVORS whose parent is keeping track of all subordinate instances.)
```

```
                                    (IF push.flag
                                        THEN (SETQ variable.name (PACK* (L-CASE flavor.name)
                                                                         (QUOTE .instances.list)))
                                             (SETQ temp.list (CAOR (GET.VALUE parent.flavor parent.name
                                                                              variable.name)))
                                             (PUT.VALUE parent.flavor parent.name variable.name
                                                        (PUSH temp.list new.instance]

(* * Read.Matrix.Pattern reads in a matrix that has not been previously formatted into the INTERLISP
environment. REQUIRED VARIABLE: the name of the file.)


[DEFMETHOD (QUOTE VANILLA)
           (QUOTE Read.Matrix.Pattern)
           (QUOTE before)
           (QUOTE (LAMBDA (file.name)
                          (PROG (before.return)
                                [SETQ before.return (Before.Help (QUOTE VANILLA)
                                                                 (LIST (QUOTE lease.pixel)
                                                                       (QUOTE background.pixel)
                                                                       (QUOTE max.size]
                                (PUSH before.return file.name)
                                (RETURN before.return]
(DEFMETHOD (QUOTE VANILLA)
           (QUOTE Read.Matrix.Pattern)
           NIL
           (QUOTE Read.Matrix.Pattern))
[DEFMETHOD (QUOTE VANILLA)
           (QUOTE Read.Matrix.Pattern)
           (QUOTE after)
           (QUOTE (LAMBDA NIL (After.Help (QUOTE VANILLA)
                                          (LIST (QUOTE new.input.matrix]

(* * Store.Pattern.Matrix stores a matrix from INTERLISP to a file in a more efficient manner.)


(DEFMETHOD (QUOTE VANILLA)
           (QUOTE Store.Pattern.Matrix)
           NIL
           (QUOTE Store.Pattern.Matrix))

(* * Load.Pattern.Matrix loads a matrix from a file into INTERLISP. The matrix has already been
formatted.)
```

```
(DEFMETHOD (QUOTE VANILLA)
           (QUOTE Load.Pattern.Matrix)
           NIL
           (QUOTE Load.Pattern.Matrix))

(* * Correct.Image can be called to allow the user to change a matrix after it has already been
loaded.)


(DEFMETHOD (QUOTE VANILLA)
           (QUOTE Correct.Image)
           (QUOTE before)
           (QUOTE (LAMBDA NIL (Before.Help (QUOTE VANILLA)
                                           (LIST (QUOTE current.matrix)
                                                 (QUOTE image.Pixel)
                                                 (QUOTE background.Pixel)
                                                 (QUOTE max.size]
(DEFMETHOD (QUOTE VANILLA)
           (QUOTE Correct.Image)
           NIL
           (QUOTE Correct.Image))

(* * Define FLAVOR GENERAL.INFORMATION and its METHOD. Its parent FLAVOR is VANILLA.
This is basically a do-nothing FLAVOR that can be used to store information of use to the user.
-
FLAVOR VARIABLES: flavor.functions is a list of the routines in the FLAVOR package.
-
global.vars is a list of the key global variables. -
-
matrix.functions is a list of the functions in the Matrix package. -
-
other.functions is a list of miscellaneous functions.)


(DEFFLAVOR (QUOTE GENERAL.INFORMATION)
           (QUOTE ((flavor.functions (DEFFLAVOR DEFMETHOD GET.VALUE MAKE.INSTANCE PRINT.FLAVORS PRINT.INSTANCE
                                                PUT.VALUE SEND.MESSAGE))
                   (global.vars (flavor.environment slope.8 slope.16 slope.32))
                   (matrix.functions (Find.Cdr.Value Find.Determinant Find.Value Invert.Matrix Make.Counter
                                                     Matrix Matrix.Addition Matrix.Multiplication
                                                     Print.Cdr.Matrix Print.Matrix Set.Value Set.Cdr.Value
                                                     Solve.Linear.Eqs User.Input))
                   (other.functions (Ask.Question Make.Counter]

(* * Print.Information just prints out the variables and their values contained in this FLAVOR.)
```

```
[DEFMETHOD (QUOTE GENERAL.INFORMATION)
           (QUOTE Print.Information)
           MIL
           (QUOTE (LAMBDA MIL (PROG [(instance.name (GETPROP (QUOTE name)
                                                             (QUOTE instance.name]
                                    (PRINT.INSTANCE (QUOTE GENERAL.INFORMATION)
                                                    instance.name]

(* * Define the FLAVOR LIBRARY and its METHODS. Its parent FLAVOR is VANILLA. -
FLAVOR VARIABLES: no.in.library is the number of different lease classes contained in the library.
-
library.name.list is a list containing the name of each lease class in the library.
-
library.vector.list is a list of lists, each containing the feature vectors for one class of images.
-
library.vector.name.list is a list of lists, each containing the corresponding array names for the
above feature vectors.)


(DEFFLAVOR (QUOTE LIBRARY)
           (QUOTE (no.in.library library.name.list library.vector.list library.vector.name.list)))

(* * Create.Library.From.File is used to set up the correct library environment, based on a previously
loaded library file. REQUIRED VARIABLES: a global variable initialized when the library file was
loaded, containing the name of the leases in the library, another global variable containing the number
of leases in the library and the name of the current instance of FLAVOR INTERMEDIATE.PROCESS.)


[DEFMETHOD (QUOTE LIBRARY)
           (QUOTE Create.Library.From.File)
           (QUOTE before)
           (QUOTE (LAMBDA (library.name.list no.in.library current.IN.instance)
                          (LIST library.name.list no.in.library current.IM.instance
                                (CADR (GET.VALUE (QUOTE VANILLA)
                                                 (QUOTE VANILLA)
                                                 (QUOTE which.IN.process]

(* * KEY VARIABLES: instance.name is the name of the calling FLAVOR LIBRARY. -
library.name.list, no.in.library and current.IN.instance have the same values as discussed in the
before demon. -
which.feature.vector is the lower case name of the type of intermediate process the user requested.
model.order and no.slopes are parameters for the auto-regressive feature extraction technique.
library.name is the name of one library lease class. -
array.name is a name for an array containing a feature vector. -
library.vector.list and library.vector.name.list were discussed in the before demon.
one.vector.list and one.vector.name.list are temporary variables used to store arrays and their names
while constructing the library lists. -
copy.library.name.list is a copy of library.name.list. -
```

```
no.in.sublist is the number of leases contained in one library lease class.)

[DEFMETHOD (QUOTE LIBRARY)
           (QUOTE Create.Library.From.File)
           NIL
           (QUOTE (LAMBDA NIL (PROG ((instance.name (GETPROP (QUOTE name)
                                                             (QUOTE instance.name)))
                                     (library.name.list (POP before.return))
                                     (no.in.library (POP before.return))
                                     (current.LM.instance (POP before.return))
                                     (which.feature.vector (L-CASE (POP before.return)))
                                     model.order no.slopes library.name array.name library.vector.list
                                     library.vector.name.list one.vector.list one.vector.name.list
                                     copy.library.name.list no.in.sublist)
                                    (SETQ copy.library.name.list (COPY library.name.list))

(* * IF the user choose the auto-regressive technique THEN append the model.order and no.slopes onto
the the name of the feature vector.)

                                    (IF (EQUAL which.feature.vector (QUOTE top.auto.regressive))
                                        THEN [SETQ model.order (CADR (GET.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                                                                current.LM.instance
                                                                                (QUOTE model.order]
                                             [SETQ no.slopes (CADR (GET.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                                                              current.LM.instance
                                                                              (QUOTE no.slopes]
                                             (SETQ which.feature.vector (PACK* which.feature.vector
                                                                               (QUOTE %.)
                                                                               model.order
                                                                               (QUOTE %.)
                                                                               no.slopes)))

(* * Loop until all the leases have been processed.)

                                    (FOR i FROM 1 TO no.in.library
                                       DO (SETQ library.name (POP library.name.list))
                                          (SETQ no.in.sublist (GETPROP library.name (QUOTE no.in.sublist))))

(* * Loop until all the leases in one library lease class have been processed.)

                                          (FOR j FROM 1 TO no.in.sublist
                                             DO (SETQ array.name (PACK* library.name (QUOTE %.)
                                                                        j
                                                                        (QUOTE %.)
                                                                        which.feature.vector))
```

J-9

```
                                                (PUSH one.vector.list (EVAL array.name))
                                                (PUSH one.vector.name.list array.name))

        (* * Place the library values onto the appropriate lists.)


                                              (SETQ library.vector.list (TCONC library.vector.list one.vector.list))
                                              (SETQ library.vector.name.list (TCONC library.vector.name.list
                                                                                    one.vector.name.list))
                                              (SETQ one.vector.list NIL)
                                              (SETQ one.vector.name.list NIL))

        (* * RETURN an error message if no library names were found.)


                                          (IF (NULL library.vector.list)
                                              THEN (RETURN "ERROR...no arrays were found in the library file.")
                                            ELSE (RETURN (LIST copy.library.name.list (CAR library.vector.list)
                                                               (CAR library.vector.name.list)
                                                               no.in.library)
        [DEFNETHOD (QUOTE LIBRARY)
                   (QUOTE Create.Library.From.File)
                   (QUOTE after)
                   (QUOTE (LAMBDA NIL (IF (NOT (STRINGP main.return))
                                          THEN (After.Help (QUOTE LIBRARY)
                                                           (LIST (QUOTE library.name.list)
                                                                 (QUOTE library.vector.list)
                                                                 (QUOTE library.vector.name.list)
                                                                 (QUOTE no.in.library)))
                                        ELSE (RETURN main.return)

        (* * Add.To.Library is used to dynamically add new feature vectors to the present library, based on
        user verification of the classification calculated by this system. REQUIRED VARIABLES: class name for
        the new image, the current instance of FLAVOR INTERMEDIATE.PROCESS and the name of the current instance
        of FLAVOR IMAGE.)


        [DEFNETHOD (QUOTE LIBRARY)
                   (QUOTE Add.To.Library)
                   (QUOTE before)
                   (QUOTE (LAMBDA (type.of.image current.IM.instance current.image.instance)
                                  (PROG (before.return)
                                        [SETQ before.return (Before.Help (QUOTE LIBRARY)
                                                                         (LIST (QUOTE no.in.library)
                                                                               (QUOTE library.name.list)
                                                                               (QUOTE library.vector.list)
                                                                               (QUOTE library.vector.name.list)
                                                                               (QUOTE which.IM.process]
                                        (PUSH before.return type.of.image)
```

```
                                (PUSH before.return current.IM.instance)

(* * Finally set the feature vector for the image just analyzed.)


                                [PUSH before.return (CADR (DET.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                                                     current.IM.instance
                                                                     (QUOTE feature.vector]
                                (RETURN before.return]

(* * KEY VARIABLES! new.feature.vector is the new image's feature vector. -
current.IM.instance, type.of.image, no.in.library and library.name.list have the same values as
discussed above. -
library.vector.list and library.vector.name.list contain the arrays and names for the images presently
in the library. -
which.feature.vector is the lower case name of the type of INTERMEDIATE.PROCESS the user requested.
model.order and no.slopes are parameters for the auto-regressive technique. -
no.remaining is the number of library images in the rest of the list beginning with the class name of
type.of.image. -
position is the sequence number of type.of.image in the library list, if it exists.
next.number is the next sequence number for a library image of a given class. -
new.array.name is the new name for the input feature vector. -
part.of.vector.list and part.of.vector.name.list are the sublists containing all the array names and
arrays for a given image class. -
new.vector.sublist and new.vector.name.sublist are the sublists containing all the array names and
arrays for a given image class after the new image has been placed into the list.)


[DEFMETHOD (QUOTE LIBRARY)
           (QUOTE Add.To.Library)
           NIL
           (QUOTE (LAMBDA NIL (PROG ((new.feature.vector (POP before.return))
                                     (current.IM.instance (POP before.return))
                                     (type.of.image (POP before.return))
                                     (no.in.library (POP before.return))
                                     (library.name.list (POP before.return))
                                     (library.vector.list (POP before.return))
                                     (library.vector.name.list (POP before.return))
                                     (which.feature.vector (L-CASE (POP before.return)))
                                     model.order no.slopes no.remaining position next.number new.array.name
                                     part.of.vector.list part.of.vector.name.list new.vector.sublist
                                     new.vector.name.sublist)

(* * IF the user choose the auto-regressive technique THEN append the model.order and no.slopes onto
the the name of the feature vector.)
```

```
                                        (IF (EQUAL which.feature.vector (QUOTE top.auto.regressive))
                                            THEN (SETQ model.order (CADR (GET.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                                                                    current.IM.instance
                                                                                    (QUOTE model.order)
                                                 (SETQ no.slopes (CADR (GET.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                                                                  current.IM.instance
                                                                                  (QUOTE no.slopes)
                                                 (SETQ which.feature.vector (PACK* which.feature.vector
                                                                                   (QUOTE %.)
                                                                                   model.order
                                                                                   (QUOTE %.)
                                                                                   no.slopes)))

(* * Find the position in the library lists where type.of.image is located, if at all.)

                                        (SETQ no.remaining (LENGTH (MEMBER type.of.image library.name.list)))
                                        (IF (IGREATERP no.remaining 0)
                                            THEN (SETQ position (IDIFFERENCE (ADD1 no.in.library)
                                                                             no.remaining))
                                                 (SETQ next.number (ADD1 (GETPROP type.of.image
                                                                                  (QUOTE no.in.sublist]

(* * Create the new name for the input feature vector array and increment the counter.)

                                        (SETQ new.array.name (PACK* type.of.image (QUOTE %.)
                                                                    next.number
                                                                    (QUOTE %.)
                                                                    which.feature.vector))
                                        (PUTPROP type.of.image (QUOTE no.in.sublist)
                                                 next.number)

(* * Add the new feature vector name and array into the correct place on the library lists.)

                                        (SETQ part.of.vector.list (FNTH library.vector.list position))
                                        (SETQ part.of.vector.name.list (FNTH library.vector.name.list
                                                                             position))
                                        (SETQ new.vector.sublist (PUSH (CAR part.of.vector.list)
                                                                       new.feature.vector))
                                        (SETQ new.vector.name.sublist
                                          (PUSH (CAR part.of.vector.name.list)
                                                new.array.name))
                                        (SETQ library.vector.list (SUBST new.vector.sublist
                                                                         (CAR part.of.vector.list)
                                                                         library.vector.list))
                                        (SETQ library.vector.name.list
```

```
                                                (SUBST new.vector.name.sublist (CAR part.of.vector.name.list)
                                                       library.vector.name.list))

        (* * ELSE input image is a new image class so create a new category for it and update the library
        lists.)



                                             ELSE (PUSH library.name.list type.of.image)
                                                  (IF (NULL no.in.library)
                                                      THEN (SETQ no.in.library 1)
                                                    ELSE (ADD no.in.library 1))
                                                  (PUSH library.vector.list (LIST new.feature.vector))
                                                  (SETQ new.array.name (PACK* type.of.image (QUOTE .1.)
                                                                              which.feature.vector))
                                                  (PUSH library.vector.name.list (LIST new.array.name))
                                                  (PUTPROP type.of.image (QUOTE no.in.sublist)
                                                           1))
                                           (RETURN (LIST library.name.list library.vector.list library.vector.name.list
                                                         no.in.library)
        [DEFMETHOD (QUOTE LIBRARY)
                   (QUOTE Add.To.Library)
                   (QUOTE after)
                   (QUOTE (LAMBDA NIL (After.Help (QUOTE LIBRARY)
                                                  (LIST (QUOTE library.name.list)
                                                        (QUOTE library.vector.list)
                                                        (QUOTE library.vector.name.list)
                                                        (QUOTE no.in.library)

        (* * Store.Library.To.File writes the current library to a file in subdirectory LIBRARY in a formatted
        fashion so that it can be reloaded for later use.)


        [DEFMETHOD (QUOTE LIBRARY)
                   (QUOTE Store.Library.To.File)
                   (QUOTE before)
                   (QUOTE (LAMBDA NIL (Before.Help (QUOTE LIBRARY)
                                                   (LIST (QUOTE library.name.list)
                                                         (QUOTE library.vector.list)
                                                         (QUOTE library.vector.name.list)
                                                         (QUOTE no.in.library)

        (* * KEY VARIABLES: library.name.list, library.vector.list, library.vector.name.list and no.in.library
        have all been described above. -
        response is the user input to a prompt. -
        full.file.name is the file name for the library in the LIBRARY subdirectory. -
        com.file is the file containing the name of the variables that will be written to the file.
        part.of.list contains a list of all the images in one library image class. -
        no.in.sublist is the number of images contained in part.of.list. -
```

```
        one.name is the name of one image class.)


        [DEFMETHOD (QUOTE LIBRARY)
                   (QUOTE Store.Library.To.File)
                   NIL
                   (QUOTE (LAMBDA NIL (PROG ((new.name.list (POP before.return))
                                             (library.vector.list (POP before.return))
                                             (library.vector.name.list (POP before.return))
                                             (new.no.in.library (POP before.return))
                                             response full.file.name com.file part.of.list part.of.vector.list
                                             no.in.sublist one.name)
                                            (SETQ library.name.list (COPY new.name.list))
                                            (SETQ no.in.library (COPY new.no.in.library))

        (* * Show the user all the present library files.)


                                            (PRINTOUT T T
                                              "The followins are the current library files in subdirectory LIBRARY:"
                                                      T)
                                            (PRINTOUT T T (FILDIR (QUOTE EE:<STOUT.LIBRARY>*.*)))
                                            (PRINTOUT T T
                                      "Input the name for the new library file to be
stored in the LIBRARY subdirectory."
                                                      T T)
                                            (SETQ response (READ))

        (* * Add the directory name to the file name.)


                                            (SETQ full.file.name (PACK* (QUOTE EE:<STOUT.LIBRARY>)
                                                                        response))
                                            (SETQ com.file (PACK* response (QUOTE COMS)))
                                            (SETQ COPYRIGHTFLG NIL)

        (* * Put the required variable names into com.file.)


                                            [SET com.file (LIST (LIST (QUOTE VARS)
                                                                      (QUOTE library.name.list)
                                                                      (QUOTE no.in.library))
                                                                (LIST (QUOTE ARRAY))
                                                                (LIST (QUOTE PROPS]
                                                          (* Put the library variables into the file.)
                                            [FOR i FROM 1 TO new.no.in.library
                                               DO (SETQ part.of.list (POP library.vector.name.list))
                                                  (SETQ part.of.vector.list (POP library.vector.list))
                                                  (SETQ one.name (POP new.name.list))
```

```
                                        (ADDTOCOMS com.file (LIST one.name (QUOTE no.in.subList))
                                                (QUOTE PROPS))
                                        (SETQ no.in.sublist (GETPROP one.name (QUOTE no.in.subList)))
                                        (FOR J FROM 1 TO no.in.sublist
                                           DO (SET (CAR part.of.list)
                                                   (POP part.of.vector.list))
                                              (ADDTOCOMS com.file (POP part.of.list)
                                                      (QUOTE ARRAY)

(* * Write the file.)


                                    (MAKEFILE full.file.name)
                                    (SETQ COPYRIGHTFLG T)

(* * Define the FLAVOR MATRIX and its METHODS. Its parent FLAVOR is VANILLA. -
FLAVOR VARIABLES: start.pt is the coordinate of the first image pixel found in the matrix that has not
previously been processed. -
-
image.instances.list is a list of the subordinate instances of FLAVOR IMAGE. -
-
image.counter is a counter for the number of image instances in the present matrix.
It is one higher than the number of image instances. -
-
current.image is the name of the current image instance being processed.)


(DEFFLAVOR (QUOTE MATRIX)
           (QUOTE (start.pt image.instances.list (image.counter 1)
                          current.image)))

(* * Find.Image searches the present matrix for an image coordinate point that has not previously been
processed.)


(DEFMETHOD (QUOTE MATRIX)
           (QUOTE Find.Image)
           (QUOTE before)
           (QUOTE (LAMBDA NIL (Before.Help (QUOTE MATRIX)
                                           (LIST (QUOTE current.matrix)
                                                 (QUOTE max.size)
                                                 (QUOTE background.pixel]
(DEFMETHOD (QUOTE MATRIX)
           (QUOTE Find.Image)
           NIL
           (QUOTE Find.Image))
(DEFMETHOD (QUOTE MATRIX)
           (QUOTE Find.Image)
           (QUOTE after)
```

```
                (QUOTE (LAMBDA NIL (After.Help (QUOTE MATRIX)
                                               (LIST (QUOTE start.pt]

(% # Define the FLAVOR IMAGE and its METHODS. Its parent FLAVOR is MATRIX. -
FLAVOR VARIABLES: which.next.min is the sequence number of the library lease class that was second
closest to the image being classified. -
-
which.min is the sequence number of the library image class that was closest to the image being
classified. -
-
sorted.array contains the numerically sorted list of edge points. -
-
row.sums contains a set of lists each containing a row number and a count of how many lease pixels were
found on that row. -
-
non.closure.flag is boolean flag that when TRUE indicates the possibility that the edge may not be
closed. -
-
no.interior.pts is the number of interior lease points for the present lease. -
-
no.edge.pts is the number of points on the edge of the image. -
-
next.min.distance is the classifier distance from the second closest lease class in the library to the
present image feature vector. -
-
min.distance is the classifier distance from the closest library lease class to the present image.
-
interior.list is a list of all the interior image points. -
-
hole.list is a list of lists each containing a hole number and a list of edge points for that hole.
-
fuzziness is the quotient of next.min.distance and min.distance. -
-
fuzzy.variance is the quotient of the second closest distance variance and the closest distance
variance. -
-
feature.vector is an array containing the feature vector for this image. -
-
freeman.code is a list containing the freeman code for the list of edge points.
-
edge.list contains a list of all the image edge coordinate points. -
-
classifier.variance.array is the variance of the distances to the library images.
-
classifier.distance.array is the array containing the distance measures from the present image to each
of the library image classes. -
-
col.sums is similar to row.sums. -
-
```

```
        center.of.area and area need no explanation.)

        (DEFFLAVOR (QUOTE IMAGE)
                   (QUOTE ((which.next.min 0)
                           (which.min 0)
                           classifier.variance.arrow sorted.arrow row.sums (non.closure.flag NIL)
                           no.interior.pts no.edge.pts next.min.distance min.distance interior.list
                           (hole.list (0 NIL)))
                           fuzziness fuzzy.variance feature.vector freeman.code edge.list classifier.distance.array
                           col.sums center.of.area area))
                   (QUOTE MATRIX))

        (* * Follow.Edge completely traces the edge of an image on a matrix.)

        (DEFMETHOD (QUOTE IMAGE)
                   (QUOTE Follow.Edge)
                   (QUOTE before)
                   (QUOTE (LAMBDA NIL (Before.Help (QUOTE IMAGE)
                                                   (LIST (QUOTE current.matrix)
                                                         (QUOTE start.pt)
                                                         (QUOTE max.size)
                                                         (QUOTE image.pixel]
        (DEFMETHOD (QUOTE IMAGE)
                   (QUOTE Follow.Edge)
                   NIL
                   (QUOTE Follow.Edge))
        (DEFMETHOD (QUOTE IMAGE)
                   (QUOTE Follow.Edge)
                   (QUOTE after)
                   (QUOTE (LAMBDA NIL (After.Help (QUOTE IMAGE)
                                                  (LIST (QUOTE edge.list)
                                                        (QUOTE no.edge.pts)
                                                        (QUOTE freeman.code)
                                                        (QUOTE non.closure.flag]

        (* * Shell.Sort.Edge numerically sorts the list of edge points. This is useful later when checking for
        interior points.)


        (DEFMETHOD (QUOTE IMAGE)
                   (QUOTE Shell.Sort.Edge)
                   (QUOTE before)
                   (QUOTE (LAMBDA NIL (Before.Help (QUOTE IMAGE)
                                                   (LIST (QUOTE edge.list)
                                                         (QUOTE no.edge.pts]
        (DEFMETHOD (QUOTE IMAGE)
                   (QUOTE Shell.Sort.Edge)
```

```
                NIL
                (QUOTE Shell.Sort.Edge))
        (DEFMETHOD (QUOTE IMAGE)
                (QUOTE Shell.Sort.Edge)
                (QUOTE after)
                (QUOTE (LAMBDA NIL (After.Help (QUOTE IMAGE)
                                               (LIST (QUOTE sorted.array]

        (* * Find.Image.Points searches the image for all interior points, calculates row and column sums and
        identifies possible hole points.)


        (DEFMETHOD (QUOTE IMAGE)
                (QUOTE Find.Image.Points)
                (QUOTE before)
                (QUOTE (LAMBDA NIL (Before.Help (QUOTE IMAGE)
                                                (LIST (QUOTE sorted.array)
                                                      (QUOTE current.matrix)
                                                      (QUOTE no.edge.pts)
                                                      (QUOTE background.pixel]
        (DEFMETHOD (QUOTE IMAGE)
                (QUOTE Find.Image.Points)
                NIL
                (QUOTE Find.Image.Points))
        (DEFMETHOD (QUOTE IMAGE)
                (QUOTE Find.Image.Points)
                (QUOTE after)
                (QUOTE (LAMBDA NIL (After.Help (QUOTE IMAGE)
                                               (LIST (QUOTE interior.list)
                                                     (QUOTE no.interior.pts)
                                                     (QUOTE col.sums)
                                                     (QUOTE row.sums)
                                                     (QUOTE hole.list]

        (* * Col.Sums finishes processing column information so that it is in the same format as row.sums.)


        (DEFMETHOD (QUOTE IMAGE)
                (QUOTE Col.Sums)
                (QUOTE before)
                (QUOTE (LAMBDA NIL (Before.Help (QUOTE IMAGE)
                                                (LIST (QUOTE col.sums]
        (DEFMETHOD (QUOTE IMAGE)
                (QUOTE Col.Sums)
                NIL
                (QUOTE Col.Sums))
        (DEFMETHOD (QUOTE IMAGE)
                (QUOTE Col.Sums)
                (QUOTE after)
```

```
                    (QUOTE (LAMBDA NIL (After.Help (QUOTE IMAGE)
                                                   (LIST (QUOTE col.sues]

          (* * Replace.Iease.Points physicallw alters the matrix containing the iease presentlw beins analvzed.
          It substitutes each iease pixel with the sequence nueber of the present iease beins analwzed.)

          [DEFMETHOD (QUOTE IMAGE)
                     (QUOTE Replace.Iease.Points)
                     (QUOTE before)
                     (QUOTE (LAMBDA NIL (Before.Help (QUOTE IMAGE)
                                                     (LIST (QUOTE current.matrix)
                                                           (QUOTE edge.list)
                                                           (QUOTE interior.list)
                                                           (QUOTE iease.counter]
          (DEFMETHOD (QUOTE IMAGE)
                     (QUOTE Replace.Iease.Points)
                     NIL
                     (QUOTE Replace.Iease.Points))

          (* * Calculate.Area calculates tha area of the present iease.)

          [DEFMETHOD (QUOTE IMAGE)
                     (QUOTE Calculate.Area)
                     (QUOTE before)
                     (QUOTE (LAMBDA NIL (Before.Help (QUOTE IMAGE)
                                                     (LIST (QUOTE no.edge.pts)
                                                           (QUOTE no.interior.pts]
          [DEFMETHOD (QUOTE IMAGE)
                     (QUOTE Calculate.Area)
                     NIL
                     (QUOTE (LAMBDA NIL (PROG (area)

          (* * The area is the sue of all edge and interior points.)


                                                (SETQ area (IPLUS (CAR before.return)
                                                                  (CADR before.return)))
                                                (RETURN (LIST area]
          [DEFMETHOD (QUOTE IMAGE)
                     (QUOTE Calculate.Area)
                     (QUOTE after)
                     (QUOTE (LAMBDA NIL (After.Help (QUOTE IMAGE)
                                                    (LIST (QUOTE area]
```

```
        (* * Calculate.Center.Area calculates the center of area for the present iease.
        REQUIRED VARIABLE: name of the present instance of FLAVOR IMAGE.)

        [DEFMETHOD (QUOTE IMAGE)
                   (QUOTE Calculate.Center.Area)
                   NIL
                   (QUOTE (LAMBDA (iease.instance)

        (* * Find the coordinates of the center of area by dividing the sum of row and column sums by the
        area.)

                                  (LIST (LIST [FQUOTIENT [CAR (SEND.MESSAGE (QUOTE IMAGE)
                                                            iease.instance
                                                            (QUOTE Calculate.Center)
                                                            (COR (GET.VALUE (QUOTE IMAGE)
                                                                            iease.instance
                                                                            (QUOTE row.sums]
                                                      (CADR (GET.VALUE (QUOTE IMAGE)
                                                                       imase.instance
                                                                       (QUOTE area]
                                              (FQUOTIENT [CAR (SEND.MESSAGE (QUOTE IMAGE)
                                                            iease.instance
                                                            (QUOTE Calculate.Canter)
                                                            (COR (GET.VALUE (QUOTE IMAGE)
                                                                            iease.instance
                                                                            (QUOTE col.sums]
                                                      (CADR (GET.VALUE (QUOTE IMAGE)
                                                                       imase.instance
                                                                       (QUOTE area]
        [DEFMETHOD (QUOTE IMAGE)
                   (QUOTE Calculate.Canter.Area)
                   (QUOTE after)
                   (QUOTE (LAMBDA NIL (After.Help (QUOTE IMAGE)
                                                  (LIST (QUOTE canter.of.area]

        (* * Calculate.Center is called twice from Calculate.Cantar.Area and sues up the product of the
        individual terms in either row or col.sues.)

        (DEFMETHOD (QUOTE IMAGE)
                   (QUOTE Calculate.Center)
                   NIL
                   (QUOTE Calculate.Center))
```

```
(* * Find.No.Of.Holes traces each potential hole point identified in Find.Lease.Points to see if it is
an actual hole. Counts the number of holes and lists all of their edge points.)

[DEFMETHOD (QUOTE IMAGE)
           (QUOTE Find.No.Of.Holes)
           (QUOTE before)
           (QUOTE (LAMBDA NIL (Before.Help (QUOTE IMAGE)
                                           (LIST (QUOTE current.matrix)
                                                 (QUOTE edge.list)
                                                 (QUOTE hole.list)
                                                 (QUOTE max.size)
                                                 (QUOTE lease.Pixel]
(DEFMETHOD (QUOTE IMAGE)
           (QUOTE Find.No.Of.Holes)
           NIL
           (QUOTE Find.No.Of.Holes))
[DEFMETHOD (QUOTE IMAGE)
           (QUOTE Find.No.Of.Holes)
           (QUOTE after)
           (QUOTE (LAMBDA NIL (After.Help (QUOTE IMAGE)
                                          (LIST (QUOTE hole.list]

(* * Print.Lease.Results outputs the results from the analysis of one lease. Also updates the library
for the adaptive classifier.)

[DEFMETHOD (QUOTE IMAGE)
           (QUOTE Print.Lease.Results)
           (QUOTE before)
           (QUOTE (LAMBDA NIL (Before.Help (QUOTE IMAGE)
                                           (LIST (QUOTE lease.counter)
                                                 (QUOTE area)
                                                 (QUOTE center.of.area)
                                                 (QUOTE hole.list)
                                                 (QUOTE current.lease)
                                                 (QUOTE current.matrix)
                                                 (QUOTE max.size)
                                                 (QUOTE which.min)
                                                 (QUOTE which.next.min)
                                                 (QUOTE fuzziness)
                                                 (QUOTE adaptive.flag)
                                                 (QUOTE current.library)
                                                 (QUOTE current.intermediate.Process]
```

```
        (* * KEY VARIABLES: most of the variables are from FLAVOR IMAGE or VANILLA and have been described
        above.)

        [DEFMETHOD (QUOTE IMAGE)
                   (QUOTE Print.Image.Results)
                   NIL
                   (QUOTE (LAMBDA NIL
                                  (PROG ((image.counter (POP before.return))
                                         (area (POP before.return))
                                         (center.of.area (POP before.return))
                                         (no.of.holes (CAAR (POP before.return)))
                                         (current.image (POP before.return))
                                         (current.matrix (POP before.return))
                                         (max.size (POP before.return))
                                         (which.bin (POP before.return))
                                         (which.next.bin (POP before.return))
                                         (fuzziness (POP before.return))
                                         (adaptive.flag (POP before.return))
                                         (current.library (POP before.return))
                                         (current.IM.process (POP before.return))
                                         library.name.list current.classifier response)

        (* * Get the list of library image class names.)


                                        [SETQ library.name.list (CADR (GET.VALUE (QUOTE LIBRARY)
                                                                                 current.library
                                                                                 (QUOTE library.name.list]

        (* * Get the type of classifier that was used.)


                                        [SETQ current.classifier (CADR (GET.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                                                                  current.IM.process
                                                                                  (QUOTE current.classifier]
                                        (PRINTOUT T T current.image
                                                  " has been analyzed.  The following matrix contains this
image."
                                                  T "which is numbered " (SUB1 image.counter)
                                                  T)

        (* * Print out the matrix containing the image.)
```

```
                              (Print.Matrix current.matrix max.size max.size 1)
                              (ASKUSER 20 (QUOTE y)
                                   "Pausing 20 seconds to allow for identification of the above image."
                                   NIL NIL NIL (QUOTE (CONFIRMFLG NIL)))

     (* * Output the key results.)

                              (PRINTOUT T T current.image " has an area = " area T)
                              (PRINTOUT T "center of area = " center.of.area T)
                              (PRINTOUT T "no of holes = " no.of.holes T)

     (* * IF the library was not empty tell the user which library image class was the closest match.)

                              (IF (IGREATERP which.min 0)
                                  THEN (PRINTOUT T T "Of the following list of library images: "
                                                 library.name.list , current.image " is most similar to "
                                                 (CAR (NTH library.name.list which.min))
                                                 T T)
                                       (IF (IGREATERP which.next.min 0)
                                           THEN (PRINTOUT T
                                          "The ratio between the second closest library image distance
("
                                                          (CAR (NTH library.name.list which.next.min))
                                                          ") and the closest image distance is " fuzziness T))
                                ELSE (PRINTOUT T T
                    "No images in the library. Answer NO to the next question
if this is an adaptive classifier."
                                              T))

     (* * IF the user wants an adaptive classifier THEN ask the user for his classification of the image.)

                              (IF (EQ adaptive.flag T)
                                  THEN (TERPRI)
                                       (PUT.VALUE (QUOTE CLASSIFIER)
                                                  current.classifier
                                                  (QUOTE done.flag)
                                                  NIL)
                                       [SETQ response (ASKUSER 20 (QUOTE y)

                                      "Was the above classification of the image correct?  (DEFAULT yes) "
                                                               NIL NIL NIL (QUOTE (CONFIRMFLG NIL]
                                       [IF (EQ response (QUOTE N))
                                           THEN (PRINTOUT T T "Input the correct name for the image." T)
                                                (SETQ response (READ))
                                         ELSE (SETQ response (CAR (NTH library.name.list which.min]
```

```
        (* * Call the METHOD to add the new image to the library.)

                                        (SEND.MESSAGE (QUOTE LIBRARY)
                                                      current.library
                                                      (QUOTE Add.To.Library)
                                                      (LIST response current.IM.process current.image)))

        (* * Allow the user to get an output of the results from this analysis.)

                                    [SETQ response
                                      (ASKUSER 20 (QUOTE 2)
                                                  (PRINTOUT T T
                   "Select one of the below to output the variable values for the current instance of the flavor mentioned." T
                                                            T T "1. None." T "2. IMAGE) includes MATRIX.  (DEFAULT)" T
                                                            "3. CLASSIFIER) includes INTERMEDIATE.PROCESS." T
                                                            T "4. LIBRARY." T "5. All 3 of the above." T T)
                                                  (QUOTE (1 2 3 4 5))
                                                  NIL NIL (QUOTE (CONFIRMFLG NIL]
                                    (COND
                                      ((EQ response 2)
                                        (PRINT.INSTANCE (QUOTE IMAGE)
                                                        current.image))
                                      ((EQ response 3)
                                        (PRINT.INSTANCE (QUOTE CLASSIFIER)
                                                        current.classifier))
                                      ((EQ response 4)
                                        (PRINT.INSTANCE (QUOTE LIBRARY)
                                                        current.library))
                                      ((EQ response 5)
                                        (PRINT.INSTANCE (QUOTE IMAGE)
                                                        current.image)
                                        (PRINT.INSTANCE (QUOTE CLASSIFIER)
                                                        current.classifier)
                                        (PRINT.INSTANCE (QUOTE LIBRARY)
                                                        current.library]

        (* * Define the FLAVOR INTERMEDIATE.PROCESS and its METHODS. Its parent FLAVOR is VANILLA.
        -
        FLAVOR VARIABLES: type.flag is a boolean flag that when TRUE indicates that the user has chosen the
        CHEBYCHEV distance classifier measurement. -
        -
        no.slopes is the number of different slopes per 360 degrees that will be used in the auto-regressive
        feature extraction technique. -
        -
        no.moments is an integer flag with values of 3, 4, or 5 and allows the user to calculate less than 11
        moments for the zernicke moments feature extraction technique. -
        -
```

```
        no.features is the number of elements in the feature vector array. -
        -
        model.order is the number used in calculating the feature vector for the auto-regressive technique.
        It represents the number of lsp components involved in the matrix equations. -
        -
        distance.power is the power of the distance measurement used when the simple Distance.Classifier is
        used. -
        -
        feature.vector is the feature vector of the present image being analyzed. A copy of this array is also
        placed in the instance of FLAVOR IMAGE. -
        -
        feature.vec.long.zero, feature.vector.zero and feature.vector.ar hold a copy of feature vector also.
        Only one of these contains a value, depending on which feature extraction technique was chosen.
        -
        current.classifier contains the name of the current instance of FLAVOR CLASSIFIER in existence.
        -
        classifier.counter is a counter for the number of instances of FLAVOR CLASSIFIER in existence.
        It is one higher than the total number present. -
        -
        auto.reg.lengths contains the radial distance lengths as output from the auto-regressive technique.
        It is useful if the user wants to plot these lengths.)


        (DEFFLAVOR (QUOTE INTERMEDIATE.PROCESS)
                   (QUOTE ((type.flag NIL)
                           (no.slopes 16)
                           (no.events 4)
                           (no.features 11)
                           (model.order 2)
                           (distance.power 2)
                           feature.vector feature.vec.long.zero feature.vector.zero feature.vector.ar
                           current.classifier (classifier.counter 1)
                           auto.reg.lengths)))

        (# # Top.Auto.Regressive controls the auto-regressive feature extraction technique.
        REQUIRED VARIABLE: name of the current instance of FLAVOR IMAGE, since IMAGE and INTERMEDIATE.PROCESS
        share only FLAVOR VANILLA. -
        -
        KEY VARIABLE: IM.instance.name is the current instance of FLAVOR INTERMEDIATE.PROCESS.)


        (DEFMETHOD (QUOTE INTERMEDIATE.PROCESS)
                   (QUOTE Top.Auto.Regressive)
                   (QUOTE before)
                   (QUOTE (LAMBDA (image.instance)
                                  (PROG ((IM.instance.name (GETPROP (QUOTE name)
                                                                    (QUOTE instance.name]
                                        (RETURN (LIST (CADR (GET.VALUE (QUOTE IMAGE)
                                                                       image.instance
```

```
                                                    (QUOTE edge.list)))
                                          (CAOR (OET.VALUE (QUOTE IMAOE)
                                                           image.instance
                                                           (QUOTE center.of.area)))
                                          (CAOR (GET.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                                           IM.instance.name
                                                           (QUOTE no.slopes)))
                                          (CAOR (OET.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                                           IM.instance.naee
                                                           (QUOTE eodel.order)))
                                          (CAOR (GET.VALUE (QUOTE IMAOE)
                                                           image.instance
                                                           (QUOTE no.edge.Pts]

(OEFMETHOO (QUOTE INTERMEDIATE.PROCESS)
           (QUOTE Top.Auto.Regressive)
           MIL
           (QUOTE Top.Auto.Regressive))

(* * More complicated than normal since many of the return variables have to be put back into FLAVOR
IMAGE. -
-
KEY VARIABLES: IM.instance.name is the current instance of FLAVOR INTERMEDIATE.PROCESS.
matrix.instance.name is the current instance of FLAVOR MATRIX. -
image.instance.name is the current instance of FLAVOR IMAOE.)

COEFMETHOO (QUOTE INTERMEDIATE.PROCESS)
           (QUOTE Top.Auto.Regressive)
           (QUOTE after)
           (QUOTE (LAMBOA MIL (PROG ((IM.instance.name (OETPROP (QUOTE naee)
                                                                (QUOTE instance.name)))
                                     [matrix.instance.name (CAR (CAOR (OET.VALUE (QUOTE VANILLA)
                                                                                 (QUOTE VAMILLA)
                                                                                 (QUOTE matrix.instances.list)
                                     image.name.instance)
                                    [SETQ image.name.instance (CAOR (OET.VALUE (QUOTE MATRIX)
                                                                               matrix.instance.name
                                                                               (QUOTE current.image]
                                    (After.Help (QUOTE INTERMEDIATE.PROCESS)
                                                (LIST (QUOTE auto.ree.lengths)
                                                      (QUOTE feature.vector.ar)))
                                    (PUT.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                               IM.instance.name
                                               (QUOTE feature.vector)
                                               (CAOR main.return))
                                    (PUT.VALUE (QUOTE IMAOE)
                                               iease.name.instance
                                               (QUOTE feature.vector)
                                               (CABR main.return))
```

```
                                        (PUT.VALUE (QUOTE INTERNEDIATE.PROCESS)
                                                   IN.instance.neee
                                                   (QUOTE no.features)
                                                   (ARRAYSIZE (CAOR eain.return]

(* * Calculate.Zernicke.Noeents controls the Zernicke eoeent feature extraction techniove, which uses
just the edge points to arproxieate the actuel eoeents. It hes the seee sort of problees es above, with
the passing of pereeters between INAGE end INTERNEDIATE.PROCESS.)


[DEFNETHOD (QUOTE INTERNEDIATE.PROCESS)
           (QUOTE Calculate.Zernicke.Noeents)
           (QUOTE before)
           (QUOTE (LANBDA (ieese.instence)
                          (PROG [(IN.instance.neee (GETPROP (QUOTE neee)
                                                            (QUOTE instance.neee]
                                (RETURN (LIST (CAOR (GET.VALUE (QUOTE INAGE)
                                                               ieese.instance
                                                               (QUOTE edge.list)))
                                              (CAOR (GET.VALUE (QUOTE INAGE)
                                                               ieese.instance
                                                               (QUOTE no.edge.pts)))
                                              (CAOR (GET.VALUE (QUOTE INTERNEDIATE.PROCESS)
                                                               IN.instance.neee
                                                               (QUOTE no.eoeents]
(DEFNETHOD (QUOTE INTERNEDIATE.PROCESS)
           (QUOTE Calculate.Zernicke.Noeents)
           NIL
           (QUOTE Calculate.Zernicke.Noeents))

(* * KEY VARIABLES! see Top.Auto.Regressive after deeon.)


[DEFNETHOD (QUOTE INTERNEDIATE.PROCESS)
           (QUOTE Calculate.Zernicke.Noeents)
           (QUOTE efter)
           (QUOTE (LAMBDA NIL (PROG ((IN.instance.neee (GETPROP (QUOTE nee)
                                                                (QUOTE instance.neee)))
                                     [eetrix.instance.neee (CAR (CADR (SET.VALUE (QUOTE VANILLA)
                                                                                 (QUOTE VANILLA)
                                                                                 (QUOTE eetrix.instences.list]
                                      ieese.neee.instance)
                                     [SETQ ieese.neee.instance (CAOR (OET.VALUE (QUOTE NATRIX)
                                                                                eetrix.instance.neee
                                                                                (QUOTE current.ieese]
                                     (After.Help (QUOTE INTERNEDIATE.PROCESS)
                                                 (LIST (QUOTE feature.vector.zern)))
                                     (PUT.VALUE (QUOTE INTERNEDIATE.PROCESS)
                                                IN.instance.neee
```

```
                                        (QUOTE feature.vector)
                                        (CAR main.return))
                                (PUT.VALUE (QUOTE IMAGE)
                                        image.name.instance
                                        (QUOTE feature.vector)
                                        (CAR main.return))
                                (PUT.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                        IM.instance.name
                                        (QUOTE no.features)
                                        (ARRAYSIZE (CAR main.return]

(* * Calculate.Zernicke.Lons controls the Zernicke moment feature extraction technique, which uses all
image points to better approximate the actual moments. Same sort of problems as above with the passing
of parameters between IMAGE and INTERMEDIATE.PROCESS.)


[DEFMETHOD (QUOTE INTERMEDIATE.PROCESS)
           (QUOTE Calculate.Zernicke.Lons)
           (QUOTE before)
           (QUOTE (LAMBDA (image.instance)
                          (PROG [(IM.instance.name (GETPROP (QUOTE name)
                                                            (QUOTE instance.name]
                                (RETURN (LIST (CADR (GET.VALUE (QUOTE IMAGE)
                                                               image.instance
                                                               (QUOTE edge.list)))
                                              (CADR (GET.VALUE (QUOTE IMAGE)
                                                               image.instance
                                                               (QUOTE interior.list)))
                                              (CADR (GET.VALUE (QUOTE IMAGE)
                                                               image.instance
                                                               (QUOTE no.edge.pts)))
                                              (CADR (GET.VALUE (QUOTE IMAGE)
                                                               image.instance
                                                               (QUOTE no.interior.pts)))
                                              (CADR (GET.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                                               IM.instance.name
                                                               (QUOTE no.moments]

(DEFMETHOD (QUOTE INTERMEDIATE.PROCESS)
           (QUOTE Calculate.Zernicke.Lons)
           NIL
           (QUOTE Calculate.Zernicke.Lons))

(* * KEY VARIABLES: same as above.)
```

```
[DEFMETHOD (QUOTE INTERMEDIATE.PROCESS)
           (QUOTE Calculate.Zernicke.Long)
           (QUOTE after)
           (QUOTE (LAMBDA MIL (PROG ((IM.instance.nsee (GETPROP (QUOTE name)
                                                                (QUOTE instance.nsee)))
                                     [matrix.instance.name (CAR (CADR (GET.VALUE (QUOTE VANILLA)
                                                                                 (QUOTE VANILLA)
                                                                                 (QUOTE matrix.instances.list]
                                     image.nsae.instance)
                                    [SETQ image.nsee.instance (CADR (GET.VALUE (QUOTE MATRIX)
                                                                               matrix.instance.name
                                                                               (QUOTE current.image]
                                    (After.Help (QUOTE INTERMEDIATE.PROCESS)
                                                (LIST (QUOTE feature.vect.long.zern)))
                                    (PUT.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                               IM.instance.name
                                               (QUOTE feature.vector)
                                               (CAR asin.return))
                                    (PUT.VALUE (QUOTE IMAGE)
                                               image.nsee.instance
                                               (QUOTE feature.vector)
                                               (CAR asin.return))
                                    (PUT.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                               IM.instance.name
                                               (QUOTE no.features)
                                               (ARRAYSIZE (CAR asin.return]
```

(* * Define the FLAVOR CLASSIFIER and its METHODS. Its parent FLAVOR is INTERMEDIATE.PROCESS.

-

FLAVOR VARIABLES: variance.array is an array containing the variances of the distance measures for the current instance of IMAGE when the the simple Distance.Classifier is used. A copy of this array is also placed in FLAVOR IMAGE. -

-

variance.matrix is used when Weighted.Classifier is requested by the user. It holds the variances for each feature element in each library class. Once calculated it can be reused by later calls to Weighted.Classifier if the classifier is not adaptive. -

-

quality.1 is the simplest of the measures of classifier quality and is the trace of the sue of the within and between class scatter matrices. -

-

quality.2 is the quotient of the traces for the within and between class matrices.

-

quality.3 is the trace of the product of the inverse of the within class matrix and the between class matrix. -

-

quality.4 is the quotient of the determinant of the sue of the two matrices and the determinant of the within class matrix. -

-

In all 4 cases, in general, the bigger the absolute value of the measures the better.

```
;
;mean.matrix is the mean of each feature element for each library class and is used in
;Weighted.Classifier. It also can be reused if the classifier is not adaptive. -
;
;extra.flag is a boolean flag that when TRUE indicates that the user wants to calculate the variances
;for distance measures when Distance.Classifier is used. -
;
;done.flag is a boolean flag that when TRUE indicates that mean.matrix and variance.matrix for
;Weighted.Classifier have already been calculated. It is always reset to NIL when the adaptive
;classifier is desired. -
;
;distance.array is a copy of the distance array in IMAGE and contains the distance values from the image
;to all the library image classes. -
;
;dendogram is a list containing the results of the clustering algorithm. The list contains sets of data,
;each containing a distance between two images and the sequence number of the two images.
;This information is useful in plotting the dendogram. -
;
;cluster.array.list is a list of the feature vectors for all the images to be clustered.
;
;cluster.name.list is a list of the names of the images to be clustered; it is in the same order as
;clustar.array.list so that final results can be analyzed easier.)


(DEFFLAVOR (QUOTE CLASSIFIER)
           (QUOTE (variance.array variance.matrix quality.1 quality.2 quality.3 quality.4 mean.matrix
                                  (extra.flag NIL)
                                  (done.flag NIL)
                                  distance.array dendogram cluster.array.list cluster.name.list))
           (QUOTE INTERMEDIATE.PROCESS))

(; ; Classify.Decision is called after one of the two classifiers has calculated the distance array.
;It makes the decision on the closest library image class to the new image and calculates key
;measurement values. -
;
;KEY VARIABLES! before.help is the return variable for the before demon. -
;library.name is the name of the current instance of FLAVOR LIBRARY.)


(DEFMETHOD (QUOTE CLASSIFIER)
           (QUOTE Classify.Decision)
           (QUOTE before)
           (QUOTE (LAMBDA NIL (PROG (before.help library.name)
                                    (SETQ before.help (Before.Help (QUOTE CLASSIFIER)
                                                                   (LIST (QUOTE distance.array)
                                                                         (QUOTE extra.flag)
                                                                         (QUOTE variance.array)
                                    (SETQ library.name (CADR (GET.VALUE (QUOTE VANILLA)
                                                                        (QUOTE VANILLA)
```

```
                                                                    (QUOTE current.library)
                                        [PUSH before.help (CADR (GET.VALUE (QUOTE LIBRARY)
                                                                           library.name
                                                                           (QUOTE no.in.library)
                                        (RETURN before.help)]
(DEFMETHOD (QUOTE CLASSIFIER)
           (QUOTE Classify.Decision)
           NIL
           (QUOTE Classify.Decision))

(* * More complicated than normal since most of the results are placed into FLAVOR IMAGE.
-
KEY VARIABLES: matrix.instance.name and image.name.instance are the current instances of those two
FLAVORS.)


(DEFMETHOD (QUOTE CLASSIFIER)
           (QUOTE Classify.Decision)
           (QUOTE after)
           (QUOTE (LAMBDA NIL (PROG ([matrix.instance.name (CAR (CADR (GET.VALUE (QUOTE VANILLA)
                                                                                 (QUOTE VANILLA)
                                                                                 (QUOTE matrix.instances.list]
                                      image.name.instance)
                                     [SETQ image.name.instance (CADR (GET.VALUE (QUOTE MATRIX)
                                                                                matrix.instance.name
                                                                                (QUOTE current.image]
                                     (PUT.VALUE (QUOTE IMAGE)
                                                image.name.instance
                                                (QUOTE which.min)
                                                (POP main.return))
                                     (PUT.VALUE (QUOTE IMAGE)
                                                image.name.instance
                                                (QUOTE min.distance)
                                                (POP main.return))
                                     (PUT.VALUE (QUOTE IMAGE)
                                                image.name.instance
                                                (QUOTE which.next.min)
                                                (POP main.return))
                                     (PUT.VALUE (QUOTE IMAGE)
                                                image.name.instance
                                                (QUOTE next.min.distance)
                                                (POP main.return))
                                     (PUT.VALUE (QUOTE IMAGE)
                                                image.name.instance
                                                (QUOTE fuzziness)
                                                (POP main.return))
                                     (PUT.VALUE (QUOTE IMAGE)
                                                image.name.instance
                                                (QUOTE fuzzy.variance)
```

```
                                                    (POP main.return]

        (% * Distance.Classifier is the simplest of the two classifiers. It measures the difference between the
        new image feature vector and one feature vector from each library image classe and sues these up to
        produce a distance measure. More complicated than some other METHODS because it has to get parameters
        from FLAVOR LIBRARY. -
        -
        KEY VARIABLES: before.help is the return variable for the before demon. -
        library.name is the name of the current instance of FLAVOR LIBRARY.)


        [DEFMETHOD (QUOTE CLASSIFIER)
                   (QUOTE Distance.Classifier)
                   (QUOTE before)
                   (QUOTE (LAMBDA NIL (PROG (before.help library.name)
                                            [SETQ before.help (Before.Help (QUOTE CLASSIFIER)
                                                                           (LIST (QUOTE feature.vector)
                                                                                 (QUOTE no.features)
                                                                                 (QUOTE type.flag)
                                                                                 (QUOTE distance.power)
                                                                                 (QUOTE extra.flag]
                                            [SETQ library.name (CADR (GET.VALUE (QUOTE VANILLA)
                                                                                (QUOTE VANILLA)
                                                                                (QUOTE current.library]
                                            [PUSH before.help (CADR (GET.VALUE (QUOTE LIBRARY)
                                                                               library.name
                                                                               (QUOTE no.in.library]
                                            [PUSH before.help (CADR (GET.VALUE (QUOTE LIBRARY)
                                                                               library.name
                                                                               (QUOTE library.vector.list]
                                            (RETURN before.help]
        (DEFMETHOD (QUOTE CLASSIFIER)
                   (QUOTE Distance.Classifier)
                   NIL
                   (QUOTE Distance.Classifier))

        (% * More complicated because most of the results are placed into FLAVOR CLASSIFIER and FLAVOR IMAGE.
        Has the same two key variables as above, matrix.instance.name and image.name.instance.)


        [DEFMETHOD (QUOTE CLASSIFIER)
                   (QUOTE Distance.Classifier)
                   (QUOTE after)
                   (QUOTE (LAMBDA NIL (PROG ([matrix.instance.name (CAR (CADR (GET.VALUE (QUOTE VANILLA)
                                                                                         (QUOTE VANILLA)
                                                                                         (QUOTE matrix.instances.list]
                                                image.name.instance)
                                            [SETQ image.name.instance (CADR (GET.VALUE (QUOTE MATRIX)
                                                                                       matrix.instance.name
```

```
                                                                    (QUOTE current.iease]
                                        (After.Help (QUOTE CLASSIFIER)
                                                    (LIST (QUOTE distance.array)
                                                          (QUOTE variance.array)))
                                        (PUT.VALUE (QUOTE IMAGE)
                                                   iease.nase.instance
                                                   (QUOTE classifier.distance.array)
                                                   (CAR sain.return))
                                        (PUT.VALUE (QUOTE IMAGE)
                                                   iease.nase.instance
                                                   (QUOTE classifier.variance.array)
                                                   (CADR sain.return]

(* * Weighted.Classifier is the more coeplicated of the two classifiers. In this one each feature
eleeent in the feature vectors for one iease class is weighted accordins to how much its value varies
over all the ieases in that class. The less the variance the more that feature eleeent is weighted for
that class. Sase key variables as above.)


[DEFMETHOD (QUOTE CLASSIFIER)
           (QUOTE Weighted.Classifier)
           (QUOTE before)
           (QUOTE (LAMBDA NIL (PROG (before.help library.nase)
                                    [SETQ before.help (Before.Help (QUOTE CLASSIFIER)
                                                                   (LIST (QUOTE feature.vector)
                                                                         (QUOTE no.features)
                                                                         (QUOTE done.flas)
                                                                         (QUOTE seen.satrix)
                                                                         (QUOTE variance.satrix]
                                    [SETQ library.nase (CADR (GET.VALUE (QUOTE VANILLA)
                                                                        (QUOTE VANILLA)
                                                                        (QUOTE current.Iibrary]
                                    [PUSH before.help (CADR (GET.VALUE (QUOTE LIBRARY)
                                                                       library.nase
                                                                       (QUOTE no.in.library]
                                    [PUSH before.help (CADR (GET.VALUE (QUOTE LIBRARY)
                                                                       library.nase
                                                                       (QUOTE library.vector.list]
                                    (RETURN before.help]
(DEFMETHOD (QUOTE CLASSIFIER)
           (QUOTE Weighted.Classifier)
           NIL
           (QUOTE Weighted.Classifier))

(* * Sase coeplications as in Distance.Classifier.)
```

```
[DEFMETHOD (QUOTE CLASSIFIER)
           (QUOTE Weighted.Classifier)
           (QUOTE after)
           (QUOTE (LAMBDA NIL (PROG ([matrix.instance.name (CAR (CADR (GET.VALUE (QUOTE VANILLA)
                                                                                 (QUOTE VANILLA)
                                                                                 (QUOTE matrix.instances.list]
                                     image.name.instance)
                                    [SETQ image.name.instance (CADR (GET.VALUE (QUOTE MATRIX)
                                                                               matrix.instance.name
                                                                               (QUOTE current.image]
                                    (After.Help (QUOTE CLASSIFIER)
                                                (LIST (QUOTE distance.array)
                                                      (QUOTE mean.matrix)
                                                      (QUOTE variance.matrix)
                                                      (QUOTE done.flag)))
                                    (PUT.VALUE (QUOTE IMAGE)
                                               image.name.instance
                                               (QUOTE classifier.distance.array)
                                               (CAR main.return)]

(* * Quality.Of.Classifier calculates the four measures of classifier quality as discussed in the
FLAVOR variables of CLASSIFIER. Only quality.1 is calculated if there is only one library image per
class. Needs to communicate with FLAVOR LIBRARY.)


[DEFMETHOD (QUOTE CLASSIFIER)
           (QUOTE Quality.Of.Classifier)
           (QUOTE before)
           (QUOTE (LAMBDA NIL (PROG (before.help library.name)
                                    [SETQ before.help (Before.Help (QUOTE CLASSIFIER)
                                                                   (LIST (QUOTE no.features]
                                    [SETQ library.name (CADR (GET.VALUE (QUOTE VANILLA)
                                                                        (QUOTE VANILLA)
                                                                        (QUOTE current.library]
                                    [PUSH before.help (CADR (GET.VALUE (QUOTE LIBRARY)
                                                                       library.name
                                                                       (QUOTE no.in.library]
                                    [PUSH before.help (CADR (GET.VALUE (QUOTE LIBRARY)
                                                                       library.name
                                                                       (QUOTE library.vector.list]
                                    (RETURN before.help]
(DEFMETHOD (QUOTE CLASSIFIER)
           (QUOTE Quality.Of.Classifier)
           NIL
           (QUOTE Quality.Of.Classifier))
[DEFMETHOD (QUOTE CLASSIFIER)
           (QUOTE Quality.Of.Classifier)
           (QUOTE after)
```

```
                (QUOTE (LAMBDA NIL (After.Help (QUOTE CLASSIFIER)
                                               (LIST (QUOTE quality.1)
                                                     (QUOTE quality.2)
                                                     (QUOTE quality.3)
                                                     (QUOTE quality.4)

        (* * Print.Clusterins.Results prints out the results from the clusterins algorithm.
        This includes dendogram and the names of the images clustered, in order. REQUIRED VARIABLE: the list of
        the instance names for all the images to be clustered.)


        [DEFMETHOD (QUOTE CLASSIFIER)
                   (QUOTE Print.ClusterIns.Results)
                   (QUOTE before)
                   (QUOTE (LAMBDA (cluster.name.list)
                                  (PROG (before.return)
                                        [SETQ before.return (Before.Help (QUOTE CLASSIFIER)
                                                                         (LIST (QUOTE dendogram)
                                                                               (QUOTE list.of.matrices)
                                                                               (QUOTE max.size]
                                        (PUSH before.return cluster.name.list)
                                        (RETURN before.return]

        (* * KEY VARIABLES: cluster.name.list is the list of image instance names. -
        dendogram contains the clustering information. -
        list.of.matrices contains the list of all the matrices analyzed. -
        max.size is the size of one dimension of the matrices. -
        response is the user input to a prompt. -
        new.matrix contains the value of one matrix from list.of.matrices.)


        [DEFMETHOD (QUOTE CLASSIFIER)
                   (QUOTE Print.Clusterins.Results)
                   NIL
                   (QUOTE (LAMBDA NIL (PROG ((cluster.name.list (POP before.return))
                                             (dendogram (POP before.return))
                                             (list.of.matrices (POP before.return))
                                             (max.size (POP before.return))
                                             response new.matrix)
                                            (PRINTOUT T T
                                                   'The information from the clustering calculations is as follows:'
                                                   T)
                                            (PRINTOUT T 'The ordered list of image names is: ' T cluster.name.list T)
                                            (PRINTOUT T T 'Dendogram information is: ' T dendogram T T)
                                            (PRINTOUT T
```

```
'The dendogram information contains a series of lists each containing the
sequence number of the two images that were closest together and the
distance between the two.  These two images then have their feature
vectors averaged and the new information is placed back into
the position of the first sequence number. The second sequence number
is eliminated. The list is ordered from furthest to closest.'
                                                    T T T)
                                          [SETQ response (ASKUSER 20 (QUOTE y)

                                'Do you want to see the matrices containing the images that were clustered?  (DEFAULT yes) '
                                                                NIL NIL NIL (QUOTE (CONFIRMFLG NIL)
                                          (IF (EQ response (QUOTE Y))
                                              THEN

        (* * Loop until all the matrices have been displayed.)


                                                        (DO (SETQ new.matrix (POP list.of.matrices))
                                                            (Print.Matrix (EVAL new.matrix)
                                                                          max.size max.size 1)
                                                            (ASKUSER 20 (QUOTE y)

                                                             'Pausing 20 seconds before getting next matrix if it
exists. '
                                                                          NIL NIL NIL (QUOTE (CONFIRMFLG NIL)))
                                                      UNTIL (NULL list.of.matrices]

        (* * Clustering contains the clustering algorithm. This function calculates the distances between each
        image feature vector and all others and finds the closest distance. These two images have their feature
        vectors averaged together and one of the images is eliminated. This process continues until only one
        image is left. REQUIRED VARIABLES: a list of all the feature vectors for the image to be clustered and
        an ordered list of names for the images to be clustered.)


        [DEFMETHOD (QUOTE CLASSIFIER)
                   (QUOTE Clustering)
                   (QUOTE before)
                   (QUOTE (LAMBDA (cluster.array.list cluster.name.list)
                                  (PROG (before.return)
                                        (PUT.VALUE (QUOTE CLASSIFIER)
                                                   (GETPROP (QUOTE name)
                                                            (QUOTE instance.name))
                                                   (QUOTE cluster.name.list)
                                                   cluster.name.list)
                                        [SETQ before.return (Before.Help (QUOTE CLASSIFIER)
                                                                         (LIST (QUOTE no.features)
                                                                               (QUOTE type.flag)
                                                                               (QUOTE distance.power]
                                        (PUSH before.return (LENGTH cluster.name.list))
```

```
                                        (PUSH before.return cluster.array.list)
                                        (RETURN before.return)
        (DEFMETHOD (QUOTE CLASSIFIER)
                   (QUOTE Clustering)
                   NIL
                   (QUOTE Clustering))
        (DEFNETHOD (QUOTE CLASSIFIER)
                   (QUOTE Clustering)
                   (QUOTE after)
                   (QUOTE (LAMBDA NIL (After.Help (QUOTE CLASSIFIER)
                                                  (LIST (QUOTE dendogram])
)


(* CALLED BY: Set.Up.Flavors. PURPOSE: pulls the required variables off of the variable list of the calling FLAVOR and
returns them so that the main METHOD can pass the correct parameters. REQUIRED VARIABLES: the calling FLAVOR name and a
list of required FLAVOR variables. RETURNS: a list containing the values for the input required variables or an error
message. CALLS: GET.VALUE.)

(DEFINEQ


(Before.Help
  [LAMBDA (flavor.name req.vars.list)                    (* edited: '27-FEB-87 09:40')

          (* * KEY VARIABLES: instance.name is the name of the instance of the calling FLAVOR.
          This value is PUTPROPped on in SEND.MESSAGE. -
          -
          parameter.list contains the list of variables that will be required by the main METHOD.
          -
          good.var.flag is TRUE when the CAR of the return from GET.VALUE is TRUE, and indicates that a value was
          obtained for the input variable. -
          -
          temp.var is one variable from the input list req.vars.list.)


    (PROG ((instance.name (GETPROP (QUOTE name)
                                   (QUOTE instance.name)))
           (parameter.list NIL)
           (good.var.flag T)
           temp.var)

          (* * Loop until the list of variables is exhausted or a bad name was encountered.)
```

```
        (DO (SETQ temp.var (GET.VALUE flavor.name instance.name (POP req.vars.list)))

        (* * Check the return for a good variable.)


            (IF (NEQ (CAR temp.var)
                     T)
                THEN (SETQ good.var.flag NIL)
              ELSE

        (* * Place the new variable on the end of the list.)


                   (SETQ parameter.list (TCONC parameter.list (CADR temp.var]
          WHILE (AND good.var.flag (LISTP req.vars.list)))
        (IF good.var.flag
            THEN (RETURN (CAR parameter.list))
          ELSE (RETURN 'ERROR...in obtaining required variables.'])
)
```



```
(* CALLED BY: Set.Up.Flavors. PURPOSE: takes the list of variables returned from the main METHOD and places them into the
proper FLAVOR variable slots. REQUIRED VARIABLES: the calling FLAVOR name and a list containing the names of the variables.
 The variable values are returned in a list contained in the variable names main.return from SEND.MESSAGE. RETURNS: the
list of variables returned from the main METHOD. CALLS: PUT.VALUE.)

(DEFINEQ

                                                                                                                    [3]

(After.Help
  [LAMBDA (flavor.name new.vars.list)                                    (* edited: '27-FEB-87 11:19')

        (* * KEY VARIABLES: instance.name is the name of the instance of the calling FLAVOR.
        -
        copy.main.return is a copy of the main return.)


    (PROG ((instance.name (GETPROP (QUOTE name)
                                   (QUOTE instance.name)))
           (copy.main.return (COPY main.return)))

        (* * IF main.return did not indicate an error was encountered THEN put the variable and its value into
        the FLAVOR environment.)
```

```
        (IF (NOT (STRINGP main.return))
            THEN (DO (PUT,VALUE flavor.name instance.name (POP new.vars.list)
                             (POP copy.main.return))
                     WHILE (LISTP copy.main.return)))
        (RETURN main.return))
)
(DECLARE: DONTCOPY
  (FILEMAP (NIL (2813 69333 (Set.Up.Flavors 2823 . 69331)) (69726 71090 (Before.Help 69736 . 71088)) (71531 72283 (After.Help
 71541 . 72281)))))
STOP
```

APPENDIX II-K. File START_SYSTEM

(FILECREATED " 8-APR-87 11:48:37" EE:<STDUT>START_SYSTEM.;3 30209

previous date: " 8-APR-87 11:41:04" EE:<STDUT>START_SYSTEM.;2)

(PRETTYCOMPRINT START_SYSTEMCOMS)

```
(RPAQQ START_SYSTEMCOMS ((* Top level of the system. Executed after being loaded. PURPOSE: controls the entire pattern
                            recognition and FLAVOR processes. REQUIRED VARIABLES: none. RETURNS: none. CALLS:
                            Load.System.Files, MAKE.INSTANCE, GET.VALUE, PUT.VALUE, SEND.MESSAGE, PRINT.FLAVORS and
                            Set.Up.Flavors.)
                           (FNS Skeleton)
                           (* CALLED BY: Skeleton. PURPOSE: loads all required system files if they have not already been
                              loaded. REQUIRED VARIABLES: none. RETURNS: none. CALLS: none.)
                           (FNS Load.System.Files)
                           (* METHOD of FLAVOR VANILLA. PURPOSE: prompts the user for all required system variables.
                              REQUIRED VARIABLES: none. User prompted for all entries. Defaults are provided for all entries
                              except for file names. RETURNS: list of required system variables which are placed into the
                              FLAVOR environment by the after daemon. CALLS: Get.User.Matrices.)
                           (FNS Get.User.Variables)
                           (* CALLED BY: Get.User.Variables. PURPOSE: prompts the user for the name of the files containing
                              matrices to be analyzed. REQUIRED VARIABLES: none. User is prompted for all entries. RETURNS:
                              list of matrices to be analyzed. CALLS: Read.Matrix.Pattern and Load.Pattern.Matrix)
                           (FNS Get.User.Matrices)))
```

K-1

```
(* Top level of the system. Executed after being loaded. PURPOSE: controls the entire pattern recognition and FLAVOR
processes. REQUIRED VARIABLES: none. RETURNS: none. CALLS: Load.System.Files, MAKE.INSTANCE, GET.VALUE, PUT.VALUE,
SEND.MESSAGE, PRINT.FLAVORS and Set.Up.Flavors.)

(DEFINED
```

(1)

```
(Skeleton
  (LAMBDA NIL                                          (* edited: " 4-APR-87 21:49")

          (* * KEY VARIABLES: response is the user input to a prompt. -
          -
          return.variables is a list of variables returned from a call to METHOD Get.User.Variables.
          -
          dribble.file.name is the name of the file that contains the output from a session.
          -
          current.matrix.instance is the name of the current instance of FLAVOR MATRIX. -
          -
          cur.intermed.proc.instance is the name of the current instance of FLAVOR INTERMEDIATE.PROCESS.
          -
          current.image.instance is the name of the current instance of FLAVOR IMAGE. -
          -
```

```
        current.library.instance is the name of the current instance of FLAVOR LIBRARY.
        -
        current.classifier.instance is the name of the current instance of FLAVOR CLASSIFIER.
        -
        list.of.matrices is a list of matrices to be processed. -
        -
        quality.flag is a boolean flag that when TRUE indicates the user wants to calculate measures of
        classifier quality. -
        -
        cluster.flag is a boolean flag that when TRUE indicates the user wants to cluster all the images in the
        matrices. -
        -
        first.time is a boolean flag that when TRUE indicates that the first image is being analyzed.
        -
        adaptive.flag is a boolean flag that when TRUE indicates the user wants to use an adaptive classifier.
        -
        no.to.process is the number of matrices contained in list.of.matrices. -
        -
        cluster.array.list is used when the user wants to cluster his images and contains a list of the feature
        vectors for all the images analyzed. -
        -
        cluster.name.list is also used when clustering and contains a list of names for all the images to be
        clustered. -
        -
        done.flag is a boolean flag that when TRUE indicates that the mean and variance matrices for
        Weighted.Classifier have been calculated. It must be reset to NIL each time if the user wants an
        adaptive classifier. -
        -
        library.no contains the number of different library image classes.)

    (PROG (response return.variables dribble.file.name current.matrix.instance cur.interest.proc.instance
                current.image.instance current.library.instance current.classifier.instance list.of.matrices
                (quality.flag T)
                (cluster.flag T)
                (first.time T)
                adaptive.flag no.to.process cluster.array.list cluster.name.list done.flag (library.no 0))
          (PRINTOUT T "WELCOME TO AN OBJECT-ORIENTED PATTERN RECOGNITION SYSTEM." T T)
          [SETQ response (ASKUSER 6 (QUOTE W)
                                "Do you want a file of this session? (DEFAULT yes)  " NIL NIL NIL
                                (QUOTE (CONFIRMFLG NIL]
          (IF (EQ response (QUOTE Y))
              THEN

          (* * Create the dribble file and append the date to it.)
```

```
        [SETQ dribble.file.name (PACK* (QUOTE SESSION_FILE)
                                       (QUOTE _)
                                       (RPLSTRING (SUBSTRING (DATE)
                                                             I 6)
                                                  3
                                                  (QUOTE _)
        (DRIBBLE dribble.file.name)
        (PRINTOUT T T "The name of your output file is " dribble.file.name T))
(TERPRI)
(TERPRI)

(* * Call to see if the system files need loading.)


(Load.System.Files)
(PRINTOUT T T "Creating initial FLAVOR environment.  Please standby." T T)

(* * Call to set up the FLAVOR environment.)


(Set.Up.Flavors)

(* * Loop until the user indicates he is done.)


(DO

(* * Set these global variables to NIL. This is done so that the values of these variables can be
checked after the library file has been added. If they are still NIL then the library file was not
properly formatted.)


    (SETQ no.in.library NIL)
    (SETQ library.name.list NIL)

(* * Create instances of four of the seven flavors: VANILLA, GENERAL.INFORMATION, INTERMEDIATE.PROCESS
and CLASSIFIER. The instance of FLAVOR VANILLA must be named VANILLA. All other names are arbitrary.)

    [MAKE.INSTANCE (QUOTE VANILLA)
                   (QUOTE VANILLA)
                   NIL
                   (QUOTE (dribble.file.name (EVAL dribble.file.name]
    (MAKE.INSTANCE (QUOTE GENERAL.INFORMATION)
                   (QUOTE GENERAL.INFORMATION))
    (SEND.MESSAGE (QUOTE VANILLA)
                  (QUOTE VANILLA)
                  (QUOTE Make.Lower.Instance)
```

```
                        (LIST (QUOTE VANILLA)
                              (QUOTE VANILLA)
                              (QUOTE intermediate.process.counter)
                              (QUOTE INTERMEDIATE.PROCESS)))

        (* ; Set the value of the local variable for the current instance of FLAVOR INTERMEDIATE.PROCESS.)

            [SETQ cur.intermed.proc.instance (CADR (GET.VALUE (QUOTE VANILLA)
                                                              (QUOTE VANILLA)
                                                              (QUOTE current.intermediate.process]
            (SEND.MESSAGE (QUOTE INTERMEDIATE.PROCESS)
                          cur.intermed.proc.instance
                          (QUOTE Make.Lower.Instance)
                          (LIST (QUOTE INTERMEDIATE.PROCESS)
                                cur.intermed.proc.instance
                                (QUOTE classifier.counter)
                                (QUOTE CLASSIFIER)))

        (* ; Set the value of the local variable for the current instance of FLAVOR CLASSIFIER.)

            [SETQ current.classifier.instance (CADR (GET.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                                               cur.intermed.proc.instance
                                                               (QUOTE current.classifier]
            (PRINTOUT T T "The following FLAVOR environment has been created: " T T)

        (* ; Call to output the FLAVOR environment that was just initialized.)

            (PRINT.FLAVORS)

        (* ; Get the key user-defined system variables and place them into the FLAVOR environment.)

            (SETQ return.variables (SEND.MESSAGE (QUOTE VANILLA)
                                                 (QUOTE VANILLA)
                                                 (QUOTE Get.User.Variables)))

        (* ; IF Get.User.Variables returned matrices that can be analyzed THEN continue.)

            [IF (NOT (STRINGP return.variables))
                THEN
```

```
        (* * Create an instance of FLAVOR LIBRARY only if the user did not specify that he wanted to cluster
        his data.)


                    (IF (EQ (CADR (GET.VALUE (QUOTE VANILLA)
                                             (QUOTE VANILLA)
                                             (QUOTE cluster.flag)))
                            NIL)
                        THEN

        (* * Set the values of the key boolean flags.)


                                (SETQ cluster.flag NIL)
                                (SETQ adaptive.flag (CADR (GET.VALUE (QUOTE VANILLA)
                                                                     (QUOTE VANILLA)
                                                                     (QUOTE adaptive.flag)
                                (IF (EQ (CADR (GET.VALUE (QUOTE VANILLA)
                                                         (QUOTE VANILLA)
                                                         (QUOTE quality.flag)))
                                        NIL)
                                    THEN (SETQ quality.flag NIL))

        (* * Create an instance of FLAVOR LIBRARY.)


                                (SEND.MESSAGE (QUOTE VANILLA)
                                              (QUOTE VANILLA)
                                              (QUOTE Make.Lower.Instance)
                                              (LIST (QUOTE VANILLA)
                                                    (QUOTE VANILLA)
                                                    (QUOTE library.counter)
                                                    (QUOTE LIBRARY)))

        (* * Set the value of the local variable for the current instance of FLAVOR LIBRARY.)


                                (SETQ current.library.instance (CADR (GET.VALUE (QUOTE VANILLA)
                                                                                (QUOTE VANILLA)
                                                                                (QUOTE current.library)

        (* * IF a properly formatted library file was loaded in METHOD Get.User.Variables THEN create a library
        to compare input images to.)
```

```
                    (IF (AND (NOT (NULL (CADR (GET.VALUE (QUOTE VANILLA)
                                                         (QUOTE VANILLA)
                                                         (QUOTE library.film.name]
                             (NOT (NULL no.in.library))
                             (NOT (NULL library.name.list)))
                        THEN (SEND.MESSAGE (QUOTE LIBRARY)
                                           current.library.instance
                                           (QUOTE Create.Library.From.Film)
                                           (LIST library.name.list no.in.library cur.interpmd.proc.instance))
                             (SETQ library.no no.in.library))

(* * IF the user choose Distance.Classifier THEN set the flag that allows variances of the distances to
be calculated.)


                    (IF (EQUAL (CADR (GET.VALUE (QUOTE VANILLA)
                                                (QUOTE VANILLA)
                                                (QUOTE which.classifier)))
                               (QUOTE Distance.Classifier))
                        THEN (PUT.VALUE (QUOTE CLASSIFIER)
                                        current.classifier.instance
                                        (QUOTE extra.flag)
                                        T)))

(* * Find out how many matrices need to be analyzed.)


            [SETQ list.of.matrices (CADR (GET.VALUE (QUOTE VANILLA)
                                                    (QUOTE VANILLA)
                                                    (QUOTE list.of.matrices]
            (SETQ no.to.process (LENGTH list.of.matrices))

(* * Loop until all the matrices in list.of.matrices have been processed.)


            [FOR i FROM 1 TO no.to.process
               DO

(* * Create an instance of FLAVOR MATRIX for the next matrix in list.of.matrices and begin analyzing.)


                (SEND.MESSAGE (QUOTE VANILLA)
                              (QUOTE VANILLA)
                              (QUOTE Make.Lower.Instance)
                              (LIST (QUOTE VANILLA)
                                    (QUOTE VANILLA)
                                    (QUOTE matrix.counter)
                                    (QUOTE MATRIX)
                                    (FOR list.of.matrices)
```

```
                                        T))

        (* * Set the value of the local variable for the current instance of FLAVOR MATRIX.)

                        [SETQ current.matrix.instance (CAR (CADR (GET.VALUE (QUOTE VANILLA)
                                                                            (QUOTE VANILLA)
                                                                            (QUOTE matrix.instances.list]

        (* * Look for an image point in the matrix.)

                        (SEND.MESSAGE (QUOTE MATRIX)
                                      current.matrix.instance
                                      (QUOTE Find.Image))

        (* * Loop until there are no more new images in the matrix to be analyzed.)

                        (DO

        (* * Create an instance of FLAVOR IMAGE.)

                            (SEND.MESSAGE (QUOTE MATRIX)
                                          current.matrix.instance
                                          (QUOTE Make.Lower.Instance)
                                          (LIST (QUOTE MATRIX)
                                                current.matrix.instance
                                                (QUOTE image.counter)
                                                (QUOTE IMAGE)
                                                NIL T))

        (* * Set the value of the local variable for the current instance of FLAVOR IMAGE.)

                            [SETQ current.image.instance (CADR (GET.VALUE (QUOTE MATRIX)
                                                                          current.matrix.instance
                                                                          (QUOTE current.image]
                            (PRINTOUT T T 'Analyzing ' current.image.instance '  Please standby.' T T)

        (* * Start analyzing the image.)

                            (SEND.MESSAGE (QUOTE IMAGE)
                                          current.image.instance
                                          (QUOTE Follow.Edge))
                            (IF (EQ (CADR (GET.VALUE (QUOTE IMAGE)
                                                     current.image.instance
```

```
                                        (QUOTE non.closure.flag)))
                                T)
                            THEN (PRINTOUT T T "WARNING... " current.lease.instance
                                   " may not have a closed boundary."
                                   T T))

        (* ; Check to make sure the lease has more than 1 pixel in it.)

                            (IF (ILESSP (CADR (GET.VALUE (QUOTE IMAGE)
                                                         current.lease.instance
                                                         (QUOTE no.edge.pts)))
                                        2)
                                THEN (PRINTOUT T T current.lease.instance " has only 1 lease pixel.  " T
                                               "No more analysis is possible."
                                               T T)
                                     (SEND.MESSAGE (QUOTE IMAGE)
                                                   current.lease.instance
                                                   (QUOTE Replace.Lease.Points))
                              ELSE (SEND.MESSAGE (QUOTE IMAGE)
                                                 current.lease.instance
                                                 (QUOTE Shell.Sort.Edge))
                                   (SEND.MESSAGE (QUOTE IMAGE)
                                                 current.lease.instance
                                                 (QUOTE Find.Lease.Points))
                                   (PRINTOUT T T "Calculating the area and center of area for "
                                             current.lease.instance "  Please standby." T T)
                                   (SEND.MESSAGE (QUOTE IMAGE)
                                                 current.lease.instance
                                                 (QUOTE Calculate.Area))
                                   (SEND.MESSAGE (QUOTE IMAGE)
                                                 current.lease.instance
                                                 (QUOTE Col.Sums))
                                   (SEND.MESSAGE (QUOTE IMAGE)
                                                 current.lease.instance
                                                 (QUOTE Calculate.Center.Area)
                                                 current.lease.instance)

        (* ; IF potential holes exist inside the lease THEN find out how many.)

                                   (IF (NEQ (CAAR (CADR (GET.VALUE (QUOTE IMAGE)
                                                                    current.lease.instance
                                                                    (QUOTE hole.list]
                                            0)
                                       THEN (SEND.MESSAGE (QUOTE IMAGE)
                                                          current.lease.instance
                                                          (QUOTE Find.No.Of.Holes)))
```

```
(* * Change the matrix so that the new image has its pixel values replaced with the counter number of
that image. WARNING this permanently alters the matrix.)


                                (SEND.MESSAGE (QUOTE IMAGE)
                                              current.image.instance
                                              (QUOTE Replace.Image.Points))
                                (PRINTOUT T T "Creating feature vector for " current.image.instance
                                          "  Please standby."
                                          T T)

(* * Extract a feature vector for this image.)


                                (SEND.MESSAGE (QUOTE INTERMEDIATE.PROCESS)
                                              cur.interest.proc.instance
                                              (CADR (GET.VALUE (QUOTE VANILLA)
                                                               (QUOTE VANILLA)
                                                               (QUOTE which.IM.Process)))
                                              (LIST current.image.instance))

(* * IF the user does not want to cluster and there is at least one library image class THEN apply the
current classifier to the image.)


                                (IF (EQ cluster.flag NIL)
                                    THEN (IF (IGREATERP library.no 0)
                                             THEN (PRINTOUT T T "Applying CLASSIFIER to " current.image.instance
                                                            "  Please standby."
                                                            T T)
                                                  (SEND.MESSAGE (QUOTE CLASSIFIER)
                                                                current.classifier.instance
                                                                (CADR (GET.VALUE (QUOTE VANILLA)
                                                                                 (QUOTE VANILLA)
                                                                                 (QUOTE which.classifier]

(* * Find which library class most closely approximates the present image.)


                                                  (SEND.MESSAGE (QUOTE CLASSIFIER)
                                                                current.classifier.instance
                                                                (QUOTE Classify.Decision))

(* * Calculate the four measures of classifier quality only if the user desires and they have changed
from the previously calculated ones.)
```

```
                                        (IF (AND quality.flag (OR first.time adaptive.flag))
                                            THEN (SEND.MESSAGE (QUOTE CLASSIFIER)
                                                               current.classifier.instance
                                                               (QUOTE Quality.Of.Classifier))
                                                 (SETQ first.time NIL)))

(* * Display the results for the user.)

                                    (SEND.MESSAGE (QUOTE IMAGE)
                                                  current.image.instance
                                                  (QUOTE Print.Image.Results))

(* * IF the library was previously empty check to see if it still is.)

                                    [IF (EQ library.no 0)
                                        THEN (SETQ library.no (CADR (GET.VALUE (QUOTE LIBRARY)
                                                                               current.library.instance
                                                                               (QUOTE no.in.library]
                                ELSE

(* * ELSE the user wants to cluster the images so collect all the feature vectors and image names.)

                                    [PUSH cluster.array.list (CADR (GET.VALUE (QUOTE INTERMEDIATE.PROCESS)
                                                                              cur.intermed.proc.instance
                                                                              (QUOTE feature.vector]
                                    (PUSH cluster.name.list current.image.instance)))

(* * Reset start.pt to NIL and see if there are any more new images in the matrix.)

                        (PUT.VALUE (QUOTE MATRIX)
                                   current.matrix.instance
                                   (QUOTE start.pt)
                                   NIL)
                        (SEND.MESSAGE (QUOTE MATRIX)
                                      current.matrix.instance
                                      (QUOTE Find.Image))
                    WHILE (LISTP (CADR (GET.VALUE (QUOTE MATRIX)
                                                  current.matrix.instance
                                                  (QUOTE start.pt)
```

```
        (% * IF the user wants to cluster the issues THEN execute the clustering algorithm and print out the
        results.)

                    (IF (EQ cluster.flag T)
                        THEN (SEND.MESSAGE (QUOTE CLASSIFIER)
                                           current.classifier.instance
                                           (QUOTE Clustering)
                                           (LIST cluster.array.list cluster.name.list))
                             (SEND.MESSAGE (QUOTE CLASSIFIER)
                                           current.classifier.instance
                                           (QUOTE Print.Clustering.Results)
                                           (LIST cluster.name.list))
                      ELSE

        (% * ELSE see if the user wants to store the present configuration of the library to a file.)


                          [SETQ response (ASKUSER 10 (QUOTE n)
                                                  "Do you want to store the present library in a file? (DEFAULT no)  "
                                                  NIL NIL NIL (QUOTE (CONFIRMFLG NIL]
                          (IF (EQ response (QUOTE Y))
                              THEN (SEND.MESSAGE (QUOTE LIBRARY)
                                                 current.library.instance
                                                 (QUOTE Store.Library.To.File]
            [SETQ done.flag (ASKUSER 10 (QUOTE w)
                                     "Are you completely done? (DEFAULT yes)  " NIL NIL NIL (QUOTE (CONFIRMFLG NIL]
           UNTIL (EQ done.flag (QUOTE Y)))

        (% * Turn off the session script and close all files.)


        (DRIBBLE)
        (CLOSEALL])
)


(% CALLED BY: Skeleton. PURPOSE: loads all required system files if they have not already been loaded. REQUIRED VARIABLES:
none. RETURNS: none. CALLS: none.)

(DEFINEQ

                                                                                                        [2]

(Load.System.Files
  [LAMBDA NIL                                                (% edited: " 9-MAR-87 10:38")
    (PRINTOUT T "LOADING SYSTEM FILES...Please standby." T)
    (LOAD? (QUOTE EE:<STOUT.COMPILED>IN_OUTPUT.V))
    (LOAD? (QUOTE EE:<STOUT.COMPILED>CENTER.V))
```

```
    (LOAD? (QUOTE EE:<STOUT.COMPILED>TRACE.V))
    (LOAD? (QUOTE EE:<STOUT.COMPILED>FLAVOR.V))
    (LOAD? (QUOTE EE:<STOUT.COMPILED>MOMENT.V))
    (LOAD? (QUOTE EE:<STOUT.COMPILED>REQ.V))
    (LOAD? (QUOTE EE:<STOUT.COMPILED>CLASSIFY.V))
    (LOAD? (QUOTE EE:<STOUT.COMPILED>MATRIX.V))
    (LOAD? (QUOTE EE:<STOUT.COMPILED>SET_UP_SYSTEM.V))
)


(; METHOD of FLAVOR VANILLA, PURPOSE: prompts the user for all required system variables. REQUIRED VARIABLES: none. User
prompted for all entries. Defaults are provided for all entries except for file names. RETURNS: list of required system
variables which are placed into the FLAVOR environment by the after demon. CALLS: Get.User.Matrices.)

(DEFINED

                                                                                                                    [3]

(Get.User.Variables
  CLAMBDA NIL                                          (; edited: " 5-APR-87 12143")

          (; ; KEY VARIABLES: response is the user input to a prompt. -
          -
          cluster.flag is a boolean flag that when TRUE indicates the user wants to cluster the images.
          -
          list.of.matrices is a list containing the names of the matrices to be analyzed.
          -
          library.file.name is the name of the file in subdirectory LIBRARY containing the desired library
          information. -
          -
          which.IM.process is the name of the feature extraction technique the user desires.
          -
          model.order and no.slopes are the two key parameters for the auto-regressive feature extraction
          technique. They represent the number of lar components desired and the number of rays per 360 degress.
          -
          distance.power is the power that the differerences are raised to in Distance.Classifier.
          -
          type.flag is NIL except when the user desires the Chebychev distance measure for Distance.Classifier.
          -
          which.classifier is the name of the classifier desired by the user. -
          -
          adaptive.flag is a boolean flag that is TRUE when the user wants the classifier to be adaptive.
          -
          quality.flag is a boolean flag that is TRUE when the user wants to measure the quality of the
          classifier.)
```

```
    (PROG (response cluster.flag list.of.matrices library.file.name which.IM.process model.order no.slopes no.moments
                distance.power type.flag which.classifier adaptive.flag quality.flag)
          [SETQ response (ASKUSER 20 (QUOTE (1))
                                     (PRINTOUT T T "What do you want to do with your input images?  Select 1 or 2." T T
                                                   "1. Find closest match to a set of library images. (DEFAULT)"
                                                   T "2. Cluster the images." T T)
                                     (QUOTE (1 2))
                                     NIL NIL (QUOTE (CONFIRMFLG NIL]
          (IF (EQ response 2)
              THEN (SETQ cluster.flag T))

          (* * Call to have the user input the matrices to be analyzed.)


          (SETQ list.of.matrices (Get.User.Matrices))

          (* * IF valid matrices were entered THEN continue.)


          (COND
            [(LISTP list.of.matrices)
               [SETQ response (ASKUSER 20 (QUOTE (2))
                                          (PRINTOUT T T
                                    "Which technique to you want to use to form the feature vector?
Select 1, 2 or 3."
                                                T T "1. Auto-regressive" T
                                                "2. Zernicke moments with edge points. (DEFAULT)"
                                                T "3. Zernicke moments using edge and interior points." T T)
                                    (QUOTE (1 2 3))
                                    NIL NIL (QUOTE (CONFIRMFLG NIL]
               (IF (EQ response 1)
                   THEN (SETQ which.IM.process (QUOTE Top.Auto.Regressive))
                 ELSEIF (EQ response 3)
                   THEN (SETQ which.IM.process (QUOTE Calculate.Zernicke.Long)))

          (* * One more question to answer if the user chose Top.Auto.Regressive.)


              (IF (EQ response 1)
                  THEN [SETQ response (ASKUSER 20 (QUOTE (2))
                                                  (PRINTOUT T T
                                    "What parameters do you want for the auto-regressive technique?
Select 1, 2 or 3."
                                                        T T "1. 1 lag components and 16 slopes." T
                                                        "2. 2 lag components, 16 slopes. (DEFAULT)"
                                                        T "3. 2 lag components, 32 slopes." T T)
                                                (QUOTE (1 2 3))
                                                NIL NIL (QUOTE (CONFIRMFLG NIL]
```

```
                    (IF (EQ response 1)
                        THEN (SETQ model.order 1)
                      ELSEIF (EQ response 3)
                        THEN (SETQ no.slopes 32))
                ELSE [SETQ response (ASKUSER 20 (QUOTE (2))
                                            (PRINTOUT T T T
                                        "How many moments are desired for the Zernicke technique?
Select 1, 2 or 3."
                                                    T T "1. Second order (total of 2 moments)." T
                                                    "2. Third order (total of 6 moments).  (DEFAULT)"
                                                    T "3. Fourth order (total of 11 moments)." T T)
                                            (QUOTE (1 2 3))
                                            NIL NIL (QUOTE (CONFIRMFLG NIL]
                    (IF (EQ response 1)
                        THEN (SETQ no.moments 3)
                      ELSEIF (EQ response 3)
                        THEN (SETQ no.moments 5)))

          (* * Ask these questions only if the user is not clustering the lenses.)


              [IF (EQ cluster.flag NIL)
                  THEN [SETQ response (ASKUSER 20 (QUOTE (1))
                                            (PRINTOUT T T T
                                        "Which library of lenses do you want to compare yours
with? Select 1 or 2."
                                                    T T "1. Forsetted standard library. (DEFAULT)" T
                                                    "2. One of my forsetted libraries in subdirectory LIBRARY."

"3. No library; create it dynamically as lenses are analyzed;
The adaptive classifier is the only possible classifier."
                                                    T T)
                                            (QUOTE (1 2 3))
                                            NIL NIL (QUOTE (CONFIRMFLG NIL]
                    (IF (EQ response 1)
                        THEN (PRINTOUT T T T "Loading standard library lense data...Please standby." T T)
                             (LOAD (QUOTE EE:<STOUT.LIBRARY>STANDARD_LIBRARY))
                             (SETQ library.file.name (QUOTE EE:<STOUT.LIBRARY>STANDARD_LIBRARY))
                      ELSEIF (EQ response 2)
                        THEN (PRINTOUT T T T "The following is a list of library files in subdirectory
LIBRARY:" T T)

          (* * Printout a list of files in the library subdirectory.)
```

```
                        (PRINTOUT T T (FILDIR (DUOTE EE1<STOUT.LIBRARY>*.*))
                                      T)
                        (PRINTOUT T T T "Input the name of the file containing your
library images." T T)
                        (SETQ response (READ))
                        (SETQ response (PACK* (DUOTE EE1<STOUT.LIBRARY>)
                                              response))
                        (LOAD response)
                        (SETQ library.file.name response)
                  ELSE (SETQ adaptive.flag T))

        (* * Ask these questions only if the user loaded a library file.)


              (IF (NEQ response 3)
                  THEN (SETQ response (ASKUSER 20 (DUOTE (3))
                                               (PRINTOUT T T T
                                          "What type of classifier do you want to use?  Select 1, 2 or 3."
                                                  T T "1. Simple Euclidean distance." T
                                                  "2. Weighted Euclidean distance."
"3. Adaptive weighted Euclidean distance:  (DEFAULT)
You will verify each image classification prior to updating library."
                                                  T T)
                                               (DUOTE (1 2 3))
                                               NIL NIL (DUOTE (CONFIRMFLG NIL)

        (* * Ask this question only if the user chose Distance.Classifier.)


                        (IF (EQ response 1)
                            THEN (SETQ which.classifier (DUOTE Distance.Classifier))
                                 (SETQ response
                                   (ASKUSER 20 (DUOTE (2))
                                               (PRINTOUT T T
                                     "What type of Euclidean distance measure do you want to use?  Select 1, 2,
3 or 4."
                                                  T T "1. To the first power." T
                                                  "2. Second power ie. differences squared. (DEFAULT)"
                                                  T "3. To the third power." T
                                                  "4. Chebychev ie. the largest distance is selected."
                                                  T T)
                                               (DUOTE (1 2 3 4))
                                               NIL NIL (DUOTE (CONFIRMFLG NIL)
                                    (IF (EQ response 1)
                                        THEN (SETQ distance.power 1)
                                      ELSEIF (EQ response 3)
                                        THEN (SETQ distance.power 3)
                                      ELSEIF (EQ response 4)
```

```
                                        THEN (SETQ type.flag (QUOTE Chebychev)))
                                    ELSEIF (EQ response 3)
                                      THEN (SETQ adaptive.flag T)))
                        (TERPRI)
                        (TERPRI)
                        (SETQ response (ASKUSER 6 (QUOTE y)
                                               'Do you want to calculate measures of classifier
quality? (DEFAULT yes)  ' NIL NIL NIL (QUOTE (CONFIRMFLG NIL)
                        (IF (EQ response (QUOTE Y))
                            THEN (SETQ quality.flag T))
                ELSE

          (* * The user chose clustering so the classifier must be Distance.Classifier.)


                    (SETQ which.classifier (QUOTE Distance.Classifier))
                    (SETQ response (ASKUSER 20 (QUOTE (2))
                                           (PRINTOUT T T
              'What type of Euclidean distance measure do you want to use for the clustering?
Select 1, 2, 3 or 4.'
                                                   T T '1. To the first power.' T
                                                   '2. Second power ie. differences squared. (DEFAULT)'
                                                   T '3. To the third power.' T
                                                   '4. Chebychev ie. the largest distance is selected.'
                                                   T T)
                                           (QUOTE (1 2 3 4))
                                           NIL NIL (QUOTE (CONFIRMFLG NIL)
                    (IF (EQ response 1)
                        THEN (SETQ distance.power 1)
                      ELSEIF (EQ response 3)
                        THEN (SETQ distance.power 3)
                      ELSEIF (EQ response 4)
                        THEN (SETQ type.flag (QUOTE Chebychev)
                (TERPRI)
                (RETURN (LIST (LIST cluster.flag list.of.matrices library.file.name which.LM.process which.classifier
                                    adaptive.flag quality.flag)
                              (LIST model.order no.slopes type.flag distance.power no.moments)
              (T (RETURN 'ERROR...No matrices were input.'))
)


(* CALLED BY: Get.User.Variables. PURPOSE: prompts the user for the name of the file containing matrices to be analyzed.
REQUIRED VARIABLES: none. User is prompted for all entries. RETURNS: list of matrices to be analyzed. CALLS:
Read.Matrix.Pattern and Load.Pattern.Matrix)
```

```
(DEFINEQ

                                                                                [4]
(Get.Usar.Matrices
  [LAMBDA NIL                                          (* edited: ' 4-MAR-87 15:06')

          (* * KEY VARIABLES: response is the user input to a prompt. -
          -
          list.of.matrices is a list of the matrices to be analyzed. -
          -
          new.matrix is the list of matrices returned from calls to Load.Pattern.Matrix and Read.Matrix.Pattern.)


     (PROG (response list.of.matrices new.matrix)
           (PRINTOUT T T T T 'All data files must be in the <STOUT.DATA> directory.' T T)
           (PRINTOUT T T 'The following is a list of files in <STOUT.DATA>.' T T)

           (* * Print out a list of files in the DATA subdirectory.)


           (PRINTOUT T T T (FILDIR (QUOTE EE:<STOUT.DATA>*.*))
                     T T)
           [SETQ response (ASKUSER 20 (QUOTE w)


'Do you have any files containing matrices with headers
to be analyzed? These matrices should not have been previously
formatted within the INTERLISP environment by the Store.Pattern.Matrix
method.   (DEFAULT yes)  '
                                NIL NIL NIL (QUOTE (CONFIRMFLG NIL)
           (IF (EQ response (QUOTE Y))
               THEN (PRINTOUT T T

'Input the list of file names containing the unformatted matrices.  Begin
with a left parenthesis and and with a right parenthesis only if more
than one file name is to be input.  Continue on more than one line
if necessary.'
                           T T)
                     (SETQ response (READ))
                     (SETQ response (MKLIST response))

           (* * Call to read in the matrices.)


                     (DO [SETQ new.matrix (SEND.MESSAGE (QUOTE VANILLA)
                                                        (QUOTE VANILLA)
                                                        (QUOTE Read.Matrix.Pattern)
                                                        (MKLIST (POP response)
```

```
          (* * See if any matrices were loaded.)


                        (IF (STRINGP new.matrix)
                            THEN (PRINTOUT T T new.matrix T T)
                          ELSE (PUSH list.of.matrices new.matrix))
                       WHILE (LISTP response)))
          (SETQ response (ASKUSER 20 (QUOTE n)

"Do you have any files containing matrices with images
to be analyzed? These matrices should already have been previously
formatted within the INTERLISP environment by the Store.Pattern.Matrix
method.  (DEFAULT no)  "
                                    MIL MIL MIL (QUOTE (CONFIRMFLG MIL)
          (IF (EQ response (QUOTE Y))
              THEN (PRINTOUT T T

"Input the list of file names containing the formatted matrices.  Begin
with a left parenthesis and end with a right parenthesis only if more
than one file name is to be entered. Continue on more than one line
if necessary."
                              T T)
                    (SETQ response (READ))
                    (SETQ response (MKLIST response))

          (* * Call Load.Pattern.Matrix to load matrices.)


                    (DO (SETQ new.matrix (SEND.MESSAGE (QUOTE VANILLA)
                                                       (QUOTE VANILLA)
                                                       (QUOTE Load.Pattern.Matrix)
                                                       (MKLIST (POP response)

          (* * See if any matrices were loaded.)


                        (IF (STRINGP new.matrix)
                            THEN (PRINTOUT T T new.matrix T T)
                          ELSE (SETQ list.of.matrices (APPEND list.of.matrices new.matrix)))
                       WHILE (LISTP response)))
          (RETURN (MKLIST list.of.matrices))
)
(DECLARE: DONTCOPY
  (FILENAP (MIL (1735 17575 (Skeleton 1745 . 17573)) (17739 18359 (Load.System.Files 17749 . 18357)) (18709 26876 (
Get.User.Variables 18719 . 26874)) (27159 30187 (Get.User.Matrices 27169 . 30185)))))
STOP
```

APPENDIX II-L. File TRACE

```
EE:<STOUT>TRACE.;3

(FILECREATED " 8-APR-87 11:35:55" EE:<STOUT>TRACE.;3 13779

      previous date: " 8-APR-87 11:32:23" EE:<STOUT>TRACE.;2)


(PRETTYCOMPRINT TRACECOMS)

(RPAQQ TRACECOMS ((* CALLED BY: METHOD of FLAVOR MATRIX. PURPOSE: searches a matrix to find an image point that has not
                     already been processed. REQUIRED VARIABLES: name of the matrix to be searched, the size of one
                     dimension of that matrix and the character value for the background pixel, usually a period. RETURNS:
                     the coordinate for the edge of one image on the matrix or an error message. CALLS: Find.Value.)
                  (FNS Find.Image)
                  (* CALLED BY: METHOD of FLAVOR IMAGE. PURPOSE: traces around the edge of the image whose first edge
                     point was found by Find.Image. Sets a flag, as appropriate, to indicate the possibility that the
                     image just traced may not have a closed boundary. User can then look at the image at alter it as
                     desired. REQUIRED VARIABLES: the matrix in which the image is located, the coordinates of the first
                     edge point, the size of one dimension of the matrix and the character value for an image pixel.
                     RETURNS: a list containing all the edge points for the image, the number of edge points, the freeman
                     code for the edge points and the value for the non-closure flag. CALLS: Get.Next.Dir and Next.Coord.)
                  (FNS Follow.Edge)
                  (* CALLED BY: Follow.Edge. PURPOSE: calculates the proper order of directions to search for the next
                     image point. Always searches so as to find the image of maximum area. The list of directions is
                     dependent on the previous direction in which an image point was found. REQUIRED VARIABLES: the
                     previous direction in which an image point was found. RETURNS: an ordered list of directions to be
                     searched. CALLS: none.)
                  (FNS Get.Next.Dir)
                  (* CALLED BY: Follow.Edge and Follow.Hole.Edge. PURPOSE: calculates the value of the next coordinate in
                     a given direction from the present image edge point. Makes sure that no point outside the matrix
                     boundary is searched. REQUIRED VARIABLES: direction to the next point, the coordinates of the present
                     edge point, the name of the matrix, the size of one dimension of the matrix and the character value
                     of an image pixel. RETURNS: the coordinates of the next point in a given direction or NIL if at the
                     edge of the matrix. CALLS: Find.Value.)
                  (FNS Next.Coord)
                  (* CALLED BY: METHOD of FLAVOR IMAGE. PURPOSE: displays the edge points for an image which might not be
                     closed. User can then check the image and make appropriate corrections. REQUIRED VARIABLES: matrix
                     name, the list of edge points and the size of one dimension of the matrix. RETURNS: nothing. CALLS:
                     Get.Value and Print.Matrix.)
                  (FNS Show.Edge)))


(* CALLED BY: METHOD of FLAVOR MATRIX. PURPOSE: searches a matrix to find an image point that has not already been
processed. REQUIRED VARIABLES: name of the matrix to be searched, the size of one dimension of that matrix and the
character value for the background pixel, usually a period. RETURNS: the coordinate for the edge of one image on the matrix
 or an error message. CALLS: Find.Value.)
```

```
(DEFINEQ

                                                                                [1]

(Find.Image
  [LAMBDA (matrix.name size.of.arrau.i background.pixel.ch)
                                                    (* edited: '27-FEB-87 11:24')

          (* * KEY VARIABLES: matrix.value.ch is a character representing the value of the matrix at a particular
          coordinate. -
          -
          matrix.coord.c are the coordinates of a particular location in the matrix.)


    (PROG (matrix.value.ch matrix.coord.c)

          (* * Loop through the rows with an inner loop through the cols until an image point is found.
          Start at the top left of the matrix.)


          (FOR row.i FROM 1 TO size.of.arrau.i DO (FOR col.i FROM 1 TO size.of.arrau.i
                                                     DO (SETQ matrix.coord.c (LIST row.i col.i))

          (* * Call Find.Value to see if an image point has been found. Skip over any points which have numeric
          values since this indicates that they have already been processed.)


                                                        (SETQ matrix.value.ch (Find.Value matrix.name row.i col.i))
                                                     REPEATWHILE (OR (EQ matrix.value.ch background.pixel.ch)
                                                                     (NUMBERP matrix.value.ch)))
             REPEATWHILE (OR (EQ matrix.value.ch background.pixel.ch)
                             (NUMBERP matrix.value.ch)))

          (* * Error message if no image found in the matrix.)


          (IF (OR (EQ matrix.value.ch background.pixel.ch)
                  (NUMBERP matrix.value.ch))
              THEN (RETURN 'NO NEW IMAGES FOUND IN MATRIX')
            ELSE (RETURN (LIST matrix.coord.c))
)


(* CALLED BY: METHOD of FLAVOR IMAGE. PURPOSE: traces around the edge of the image whose first edge point was found by
Find.Image. Sets a flag, as appropriate, to indicate the possibility that the image just traced may not have a closed
boundary. User can then look at the image at alter it as desired. REQUIRED VARIABLES: the matrix in which the image is
located, the coordinates of the first edge point, the size of one dimension of the matrix and the character value for an
image pixel. RETURNS: a list containing all the edge points for the image, the number of edge points, the freeman code for
the edge points and the value for the non-closure flag. CALLS: Get.Next.Dir and Next.Coord.)
```

```
(DEFINEQ

(Follow.Edge                                                                    [2]
  [LAMBDA (matrix.name begin.pt.c size.of.array.i lease.pixel.ch)
                                                  (* edited: " 4-APR-87 21:45")

          (* * KEY VARIABLES: next.dir.l is a ordered list of directions used to locate the next lease edge
          point. -
          -
          stop.flag.b is a boolean that when true indicates the next edge point has been found.
          -
          edge.pts.l is a list containing each edge point as it is found. It may contain some coordinates more
          than once. Needed to keep track of the location of the edge pointer. -
          -
          actual.edge.pts.l is a list containing the edge points for the image. Each coordinate is only appears
          once in this list. -
          -
          non.closure.b is a boolean flag that is set to true the first time an edge point is found that has been
          previously found. This indicates the possibility that the lease may not be closed.
          -
          freeman.code.l is a list of the freeman code for the edge points. It is the direction (0 to 8) from the
          present edge point to the next edge point. 0 is along +X axis; 1 at -45 deg; 2 along -Y axis; 3 at -135
          deg; 4 along -X axis; 5 at 135 deg; 6 along +Y axis; and 7 at 45 deg. -
          -
          perimeter.i is the integer number of unique edge points and is the number of coordinates in
          actual.edge.list. -
          -
          temp.c are the coordinates of the next point to be checked in the search for edge points.
          -
          temp.dir.i is an integer indicating the last direction in which an edge point was found.)

    (PROG ((next.dir.l (QUOTE (5 6 7 0 1 2)))
           (stop.flag.b NIL)
           (edge.pts.l (CONS begin.pt.c NIL))
           (actual.edge.list (CONS begin.pt.c NIL))
           (non.closure.b NIL)
           (freeman.code.l NIL)
           (perimeter.i 0)
           temp.c temp.dir.i)

          (* * Loop until the next lease point found equals the start point.)
```

```
(% * First call to set the coordinate of an adjacent point. Exit the inner DO loop when an image point
has been found and processed.)


[DO (FOR direction.i IN next.dir.l
       DO (SETQ temp.c (Next.Coord direction.i (CAR edge.pts.l)
                                   matrix.name size.of.array.i image.pixel.ch))

(% * IF the next point is an image point THEN process it.)


          (IF (LISTP temp.c)
              THEN (SETQ stop.flag.b T)
                   (SETQ freeman.code.l (TCONC freeman.code.l direction.i))
                   (SETQ temp.dir.i direction.i)
                   (PUSH edge.pts.l temp.c)

(% * See if the new edge point has previously been input into the list of edge points.
IF not THEN add it to the true edge list and increment the perimeter.i counter ELSE set the
non.closure.b flag.)


                        (IF (OR (NOT (MEMBER temp.c actual.edge.list))
                                (EQUAL temp.c begin.pt.c))
                            THEN (PUSH actual.edge.list temp.c)
                                 (ADD perimeter.i 1)
                          ELSEIF (AND (MEMBER temp.c actual.edge.list)
                                      (NOT non.closure.b))
                            THEN (SETQ non.closure.b T)))
      UNLESS (EQ stop.flag.b T))
   (SETQ stop.flag.b NIL)

(% * Call to set the next ordered list of directions.)


    (SETQ next.dir.l (Get.Next.Dir temp.dir.i)) REPEATUNTIL (OR (EQUAL (CAR edge.pts.1)
                                                                       begin.pt.c)
                                                               (EQ (LENGTH freeman.code.l)
                                                                   (1TIMES size.of.array.i size.of.array.i]

(% * IF only one pixel in the image THEN return that point ELSE strip off the duplicate starting point
and RETURN the edge list)
```

```
          (IF (ILESSP (FLENGTH actual.edge.list)
                      2)
              THEN (RETURN (LIST actual.edge.list 1 NIL T))
            ELSE (RETURN (LIBT (CDR actual.edge.list)
                               perimeter.i freeman.code.l non.closure.b3)
)
```

(% CALLED BY: Follow.Edge. PURPOSE: calculates the proper order of directions to search for the next lease point. Always searches so as to find the lease of maximum area. The list of directions is dependent on the previous direction in which an lease point was found. REQUIRED VARIABLES: the previous direction in which an lease point was found. RETURNS: an ordered list of directions to be searched. CALLS: none.)

(DEFINED

[3]

```
(Det.Next.Dir
  [LAMBDA (last.dir.i)                                    (% edited: "16-JUL-86 09:20")
    (SELECTQ last.dir.i
             (0 (QUOTE (6 7 0 1 2 3 4)))
             (1 (QUOTE (7 0 1 2 3 4 5)))
             (2 (QUOTE (0 1 2 3 4 5 6)))
             (3 (QUOTE (1 2 3 4 5 6 7)))
             (4 (QUOTE (2 3 4 5 6 7 0)))
             (5 (QUOTE (3 4 5 6 7 0 1)))
             (6 (QUOTE (4 5 6 7 0 1 2)))
             (7 (QUOTE (5 6 7 0 1 2 3)))
             NIL))
)
```


(% CALLED BY: Follow.Edge and Follow.Hole.Edge. PURPOSE: calculates the value of the next coordinate in a given direction from the present lease edge point. Makes sure that no point outside the matrix boundary is searched. REQUIRED VARIABLES: direction to the next point, the coordinates of the present edge point, the name of the matrix, the size of one dimension of the matrix and the character value of an lease pixel. RETURNS: the coordinates of the next point in a given direction or NIL if at the edge of the matrix. CALLS: Find.Value.)

(DEFINED

[4]

```
(Next.Coord
  [LAMBDA (direction.i point.c matrix.name size.of.array.i lease.pixel.ch)
                                                          (% edited: " 4-APR-87 14:14")
```

```
        (* * KEY VARIABLES! new.coordinate.c is the result of the case statement -
        -
        next.point.c is the coordinates of the next point in a given direction.)

    (PROG (new.coordinate.c next.point.c)

        (* * Case statement to find the cartesian coordinates of a point, based on a direction from a previous
        point. Checks to make sure point is not on the edge of the matrix.)

        (SETQ new.coordinate.c (SELECTQ direction.i
                                        [0 (IF (NEQ (CADR point.c)
                                                    size.of.array.i)
                                               THEN (SETQ next.point.c (LIST (CAR point.c)
                                                                             (ADD1 (CADR point.c]
                                        [1 (IF (AND (NEQ (CAR point.c)
                                                         size.of.array.i)
                                                    (NEQ (CADR point.c)
                                                         size.of.array.i))
                                               THEN (SETQ next.point.c (LIST (ADD1 (CAR point.c))
                                                                             (ADD1 (CADR point.c]
                                        [2 (IF (NEQ (CAR point.c)
                                                    size.of.array.i)
                                               THEN (SETQ next.point.c (LIST (ADD1 (CAR point.c))
                                                                             (CADR point.c]
                                        [3 (IF (AND (NEQ (CAR point.c)
                                                         size.of.array.i)
                                                    (NEQ (CADR point.c)
                                                         1))
                                               THEN (SETQ next.point.c (LIST (ADD1 (CAR point.c))
                                                                             (SUB1 (CADR point.c]
                                        [4 (IF (NEQ (CADR point.c)
                                                    1)
                                               THEN (SETQ next.point.c (LIST (CAR point.c)
                                                                             (SUB1 (CADR point.c]
                                        [5 (IF (AND (NEQ (CAR point.c)
                                                         1)
                                                    (NEQ (CADR point.c)
                                                         1))
                                               THEN (SETQ next.point.c (LIST (SUB1 (CAR point.c))
                                                                             (SUB1 (CADR point.c]
                                        [6 (IF (NEQ (CAR point.c)
                                                    1)
                                               THEN (SETQ next.point.c (LIST (SUB1 (CAR point.c))
                                                                             (CADR point.c]
                                        [7 (IF (AND (NEQ (CAR point.c)
                                                         1)
                                                    (NEQ (CADR point.c)
```

```
                                                    size.of.array.i))
                                            THEN (SETQ next.point.c (LIST (SUB1 (CAR point.c))
                                                                          (ADD1 (CADR point.c]
                                     NIL))

        (* * Call to find if the new point is an image point. IF it is THEN return the new coordinates.)


        (IF (NEQ new.coordinate.c NIL1
            THEN (IF (EQ (Find.Value matrix.name (CAR new.coordinate.c)
                                     (CADR new.coordinate.c))
                         image.Pixel.ch)
                     THEN (RETURN next.point.c])
)


(* CALLED BY: METHOD of FLAVOR IMAGE. PURPOSE: displays the edge points for an image which might not be closed. User can
then check the image and make appropriate corrections. REQUIRED VARIABLES: matrix name; the list of edge points and the
size of one dimension of the matrix. RETURNS: nothing. CALLS: Set.Value and Print.Matrix.)

(DEFINEQ


(Show.Edge
  [LAMBDA (matrix.name edge.list size.of.array.i)          (* edited: '12-FEB-87 14124')

        (* * KEY VARIABLE: temp.matrix is a matrix used to display the edge points of one image in a matrix.)


    (PROG [(temp.matrix (Matrix size.of.array.i size.of.array.i (QUOTE %.]

        (* * Loop through all the coordinates in the list of edge points and print them with an e.)


        (DO (SETQ temp.coord (POP edge.list))
            (Set.Value temp.matrix (CAR temp.coord)
                       (CADR temp.coord)
                       (QUOTE e))
          WHILE (LISTP edge.list))
        (Print.Matrix temp.matrix size.of.array.i size.of.array.i 1)
)
(DECLARE: DONTCOPY
  (FILEMAP (NIL (3245 4682 (Find.Image 3255 . 4680)) (5385 9011 (Follow.Edge 5395 . 9009)) (9434 9857 (Get.Next.Dir 9444 .
9855)) (10409 12776 (Next.Coord 10419 . 12774)) (13115 13757 (Show.Edge 13125 . 13755)))))
STOP
```

AN OBJECT-ORIENTED PATTERN RECOGNITION SYSTEM

by

LAWRENCE MYKEL STOUT

B. A., University of Wisconsin, Madison, 1978

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1987

A computer vision system consists of five components: data acquisition, low, intermediate and high-level image processing, and classification. The pattern recognition system described here is concerned with the software implementation of the low, intermediate-level and classification processes. The system was developed using INTERLISP and allows the user to input, store, analyze, extract features, and classify any 2-dimensional, binary geometric object.

The low-level processing contains the algorithms that trace the edge of an image and calculate its center of area. The intermediate-level processing contains two statistical feature extraction techniques: the autoregressive model and the Zernicke moments. The feature vectors produced by these techniques are relatively insensitive to translational, scale and rotational transformations. The Zernicke technique, using all image points, obtained feature elements that were the most invariant to these transformations.

The classifier contains two distance measuring

algorithms that identify unknown objects by their proximity to known image classes. Class membership is based on the smallest distance between the image and a class. The simple classifier uses Euclidean distance measurements to obtain a value for the distance between an unknown image and an image class.

A more complicated classification technique transforms the feature space in such a way that the distance between images within a class is reduced. An adaptive classifier using this technique, proved to be less accurate than the simple distance classifier, especially for blurred or noisy images.

When no image classes are available or desired, the user can cluster a set of unknown images. This technique creates a tree-like structure in which objects are clustered according to their degree of similarity.

The pattern recognition system is placed into an object-oriented software shell. This shell (flavor system) is an independent, general-purpose software development tool that the user can use to build and

execute complicated programs. It consists of eight INTERLISP functions that provide all the advantages of object-oriented programming. These advantages are information hiding, data abstraction, dynamic binding and inheritance. This flavor system provides an excellent way to represent, manipulate and display system knowledge.