To the Graduate Council:

I am submitting herewith a thesis written by Renato Alberto Salinas entitled "A Hierarchical Modular Vision System for Object Recognition." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

R. C. González

Ralph C. Gonzalez, Major Professor

We have read this thesis
and recommend its acceptance:

Donald W. Bouldin

Michael G. Thomason

Accepted for the Council:

Lee Minkel

Vice Provost
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Renato Dalla Pozza

Date 27- AUGUST-1985

A HIERARCHICAL MODULAR VISION SYSTEM FOR OBJECT RECOGNITION

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Renato Alberto Salinas

December 1985

DEDICATION

With love to Roxana for her understanding and continuous support, and to Carlos and Joanna born in the USA.

ACKNOWLEDGMENTS

The author wishes to acknowledge his thesis advisor, Dr. Ralph C. Gonzalez, for his advice and guidance. Thanks are also extended to committee members, Dr. Michael G. Thomason and Dr. Donald W. Bouldin for review of this thesis.

The author also wishes to thank the support received from the Universidad de Santiago de Chile, and the support provided by Martin Marietta Energy Systems under Contract 7685-PAS88. Form the CESAR people that have contributed in this work, special mention should be made to Dr. Charles Weisbin and Dr. Gerard de Saussure.

ABSTRACT

Computer vision constitutes a fundamental component in robotic applications. Object recognition and the determination of object poise are necessary steps for intelligent manipulative tasks.

In this thesis, a hierarchical modular computer vision system for object recognition was developed. It combines statistical and structural techniques, and presents an approach to the problem of associating attributes with a string of pattern primitives.

Results show that the system can recognize objects in the presence of rotation, translation, size variation, sampling size and moderate amounts of noise.

TABLE OF CONTENTS

CHAPTER		PAGE
1.	INTRODUCTION	1
2.	BACKGROUND	2
3.	SYSTEM COMPONENTS	8
	1. Gray to Binary Conversion Module	8
	2. Contour Tracking and Filtering	11
	3. Component Labeling and Hole Detection	12
	4. Convex Hull Extraction	20
	5. Processing of Concavities	22
	6. Recognition	31
	7. Training	36
4.	EXPERIMENTAL RESULTS	38
5.	CONCLUSIONS AND RECOMMENDATIONS	56
LIST OF REFERENCES		58
APPENDIXES		62
	APPENDIX A. TRANSITIVE CLOSURE	63
	APPENDIX B. DATA BASE	67
	APPENDIX C. PROGRAM LISTINGS	70
VITA		192

LIST OF TABLES

TABLE		PAGE
3.1	Connectivity Table for Example	30
3.2	Computation of Levenshtein Distance	35
4.1	Effect of Sampling Size on Recognition Features	39
4.2	Effect of Size Variation on Recognition Features	40
4.3	Effect of Object Orientation on Recognition Features	41
4.4	Data Generated for Each Concavity	45
4.5	Attributes for the Reduced String	47

LIST OF FIGURES

FIGURE		PAGE
3.1	System Block Diagram	9
3.2	Algorithm COMPONENT Applied to Binary Image with One Hole	16
3.3	Algorithm COMPONENT Applied to Binary Image with Two Holes	18
3.4	Algorithm COMPONENT Applied to Binary Image with One Hole	19
3.5	Convex Hull Extraction	23
3.6	Measurements on a Concavity	25
3.7	Additive and Nonadditive Attributes	27
3.8	Closed Contour with Five Extreme Points	28
4.1	Test Case: The Character "A"	43
4.2	String After Pass I	46
4.3	String After Pass II	46
4.4	Table of Connectivity (LTABLE)	46
4.5	Semantic Information to Discriminate Between the Letter "A" and "Scotch Tape"	48
4.6	Combination Wrench	50
4.7	Tape	53
4.8	Automobile Part	55

CHAPTER 1

INTRODUCTION

Computer vision, although limited in scope when compared with human vision, can be applied to a variety of engineering tasks, including: object inspection, handling, and control (Foith et al., [1981]).

Machine vision can increase the flexibility and usefulness of robot manipulators by allowing operation in less structured environments. It is an important component in mobile robots that are being designed today to operate in hazardous or remote environments, such as nuclear reactors, undersea exploration, or space exploration. A trainable robot provided with sensory mechanisms can be adapted to a variety of tasks, and this can be ultimately translated as higher productivity (Gonzalez and Safabakhsh, [1982]).

In this study a hierarchical modular system for object recognition is presented. The system is based on both statistical and structural techniques. Representative techniques include eigenvalues, in the first category, and, string matching in the latter. Basic components in the proposed system include: the extraction of topological features (holes); the utilization of the concept of convex hull of a set of points in two dimensions, and, the associated concept of convex deficiency. Training by showing is also provided. Modularity has been stressed in the design to facilitate upgrading.

The vision system operates in two phases. First, the set of objects is presented to the machine in the "training phase." Second, unknown objects are presented to the system for recognition during the "automatic recognition phase."

The proposed vision system has been found successful in recognizing parts in the presence of rotation, translation, size variation, sampling size, and moderate amounts of noise.

In this thesis, the programs were implemented in FORTRAN 77 in a DEC VAX 11/780 computer equipped with a closed circuit television camera, a light table, a color display monitor, and a digitizer; the pictures were made using a video film recorder camera.

In Chapter 2 a review of the literature is provided. The system components are presented in Chapter 3. Chapter 4 presents experimental results. In the last chapter conclusions and recommendations for future work are treated.

CHAPTER 2

BACKGROUND

Computer vision systems for industrial applications have appeared in the literature with some regularity since the late seventies. These systems have been classified (Agin, [1980], Gevarter, [1982]) according to the following tasks: (a) inspection, (b) part handling, (c) assembly. Other classifications (Foith et al., [1981]) that put emphasis on "in-process"-control are: (1) inspection, (2) part handling and assembly, (3) control of tools, machines and processes, e.g., welding.

Inspection deals primarily with known objects and the emphasis is on deviation with respect to a standard. Here the parts are usually constrained to a specific orientation. Material handling deals with objects whose position and orientation are arbitrary. Frequently this task involves recognition of the parts. Assembly requires the closest interaction between vision and manipulation.

Computer vision processes, on the other hand, can be described (Gonzalez and Safabakhsh, [1982]) by the following areas: sensing, segmentation, description, recognition, and interpretation.

It is evident that each task as classified will differ in the emphasis that will utilize in any or all of the previously mentioned vision processes. This is not a clear cut distinction, because

sometimes the biggest differences will be in the use or nonuse of vision in the feedback loop of the manipulator.

A number of computer vision systems for part handling and recognition have been proposed. Agin [1980] and others at Stanford Research Institute developed the SRI Vision Module, a general-purpose system that performs analysis of binary images. It is based on a solid state TV camera, with 128x128 resolution, connected to a LSI-11 microcomputer. It follows a standard bottom-up approach and uses global thresholding for segmentation; blobs are examined using connectivity analysis. Some descriptors are: maximum and minimum x and y coordinates, area, and perimeter. Recognition is made using either a nearest-neighbor classifier or a binary decision tree. Training is by showing. This vision system has served as the basis for several later systems, some of them produced commercially.

Perkins, [1978] developed at General Motors a model-based system for industrial parts. It uses a VIDICON monochrome TV camera for sensing, with 256x256 resolution in 32 intensity levels. Segmentation is based in discontinuity (Hueckel Edge Operator). Edge points must be linked to form "chains" of boundary points. These chains are further reduced to descriptors based in straight lines or circular arcs. These descriptors are called "concurves." The matching process requires three phases: (a) control structure (to avoid combinatorial explosion), (b) matching of one model and the image concurve, and (c) rotation of model looking for best fit. Programs were written in PL/I and run on an IBM 370/168 computer. Restrictions are:

(1) nontextured data, (2) two-dimensional models, (3) objects at a known distance from camera.

Holland et al., [1979] developed at GM CONSIGTH I, a vision controlled robot system for parts handling. They used a solid state line array camera, and two spatially modulated light sources focused in such a way as to detect the silhouette of passing objects. The second dimension is provided by the movement of the conveyor belt. The camera interface and the robot arm are controlled by a PDP 11/34 computer. It uses same feature vector approach as the SRI Vision Module (binary processing).

Foith et al., [1981] developed in Germany a real-time vision system for industrial application. They use a hybrid system of hardware and software modules to achieve high speed. Programs are written in PLZ, a Pascal-like language. Data processor units are a ZBO SBC microprocessor augmented with a AM9511 arithmetic processor. Standard resolution is 256x320 pixels. Segmentation is via thresholding (set manually). The filtering mode is implemented within the component labeling mode. Recognition is via model driven search. During training the operator must indicate to the system which is the "dominant" blob when the object appears fragmented after segmentation. Foith's system is menu driven, where software is implemented in three layers: basic software, problem oriented software, and operator oriented software.

Braunegg and Gonzalez, [1983] developed an industrial vision system using syntactic/semantic techniques. They use a vidicon TV camera, with 256x256 resolution. Binary images are obtained by

thresholding. Object boundary is encoded using Freeman vectors of length input by the user. Recognition is via a deterministic automaton. Learning is achieved using the well-known k-tail algorithm. Class overlaps are resolved using semantic information. Programs are implemented in FORTRAN IV on a PDP-11/40 minicomputer. Some difficulties with the system are: (a) during learning phase, the sample set must contain objects in all of the rotation orientations which would yield different starting points, and (b) the system is sensitive to rotation (due to discrete nature of Freeman coding).

Kitchin, Pugh, [1981] and others, developed in England a general purpose system for the processing of binary images based on Freeman chain coding with 8-connectivity to adjacent pixels only. They did not try processing using syntactic means, but rather they obtained a set of features, similar to those of the SRI module. A "circles" model is proposed for both recognition and orientation that can provide information for the wrong way up of a component. However, it may require operator interaction to select a good radius.

Gonzalez, R. C., [1983] proposed a hierarchical, semantic guided approach for feature description and character recognition. No algorithms were provided, but some of the concepts such as the use of a hierarchy and the use of bays and lakes for recognition, have been borrowed from this approach.

Gonzalez and Safabakhsh, [1982] provide a comprehensive review of the techniques applicable for industrial computer vision. They do not provide algorithms, but they do list over seventy references.

Tsai and Fu, [1980] developed a syntactic-statistical approach to recognition of industrial parts. Imaging is with 200x128 resolution, with 64 gray levels. Segmentation is via thresholding (dark background). From the list of boundary points they generate a description based on five types of line and curve segments (primitives). Syntax analysis is made with finite automata. Programs are implemented in FORTRAN and run on a PDP 11/45 minicomputer. {Fu and his associates contributed to the development of the "attributed grammars" for pattern recognition.}

Most of the reported material describes computer vision systems based on statistical techniques that use edge (boundary) processing. This does not fully employ all the information available about the object. The purpose of this thesis is to implement a system that combines statistical and structural techniques, that uses both local and global features, edge and region-based, that is modular, thus easy to upgrade, and menu-driven for ease of operation. It should also be robust in the sense of insensitivity to orientation, sampling size, variable camera-object distance and moderate amounts of noise.

The next chapter will provide a detailed account of the modules comprising the proposed vision system.

CHAPTER 3

SYSTEM COMPONENTS

The proposed computer vision system has been structured around the following basic modules:

1. Gray to Binary Conversion (Segmentation)
2. Contour tracking and filtering
3. Component labeling and hole detection
4. Convex hull extraction
5. Processing of concavities
   a) detection and measurements
   b) reduction to a string
6. Recognition
7. Training

Necessarily, there are a few additional modules, such as the main program, that in turn executes the module for displaying menu, module for displaying the results of intermediate steps, and other auxiliary modules. All programs are implemented in FORTRAN 77 and are included in the appendixes. A block diagram of the overall system is shown in Figure 3.1

1. Gray-to-Binary Conversion Module

Segmentation is the process of separating the constituent parts in a scene (Gonzalez and Safabakhsh, [1982]). Segmentation is based on one of two principles: discontinuity or similarity. In the first

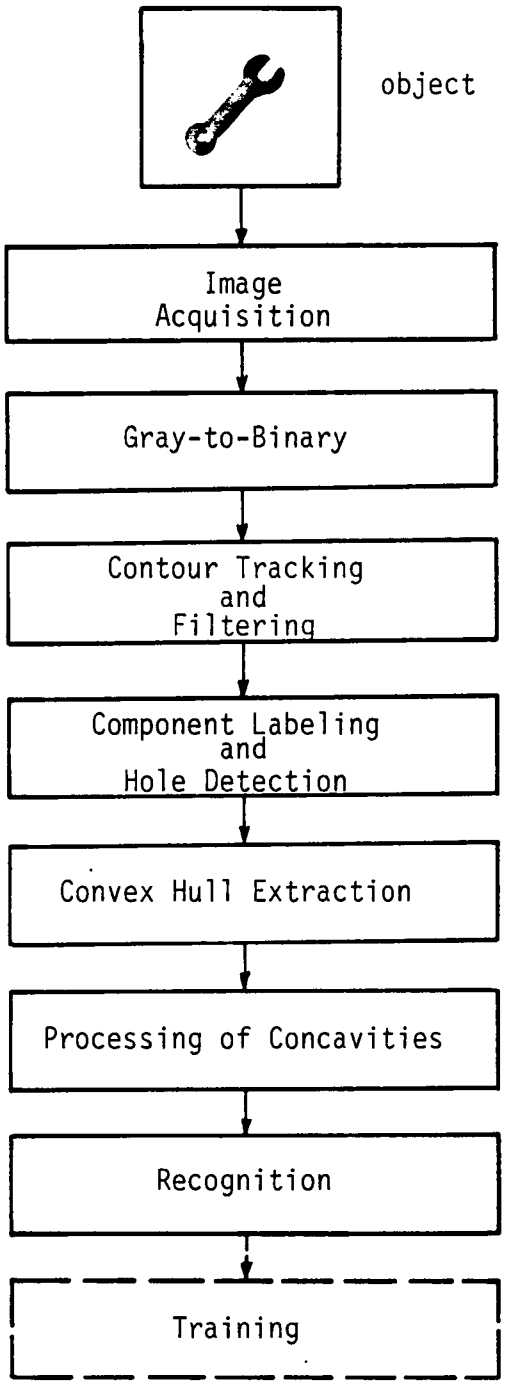


Figure 3.1. System Block Diagram.

case the principal approach is edge detection; in the second category the principal approach is thresholding.

Thresholding is most used in industrial applications because: (1) it can be implemented easily in hardware and it is fast even in software, and (b) lighting condition is usually a factor that can be manipulated.

Thresholding appears even more favorable if one considers that edge detection must be followed by a step that connects in some sense corresponding edge points.

Thresholding has been formulated as a test function of the form

$$T[x,y,g(x,y),N(x,y)] \tag{3-1}$$

which maps the grayscale into the set {object, background}; where x and y are pixel coordinates, g(x,y) is the gray level of pixel at (x,y) and N(x,y) is a local property of pixel at (x,y).

Thresholding is called global if it depends only on g(x,y), local if it depends on g(x,y) and N(x,y), dynamic if it also depends on the coordinates (x,y).

Most of the work done in this thesis deals with backlighted images (or highly contrasted scenes), threrefore global thresholding is the natural choice as the image segmentation scheme.

A dynamic (or adaptive) thresholding method developed by Contreras, [1985] was also tested in this context. The results do not differ greatly from those obtained using global thresholding; therefore the latter one was chosen.

Included in this module is the option of choosing clear objects over dark background. (Default is dark object over clear background.)

2. Contour Tracking and Filtering

After the binarization module, the resulting binary image is passed to this module, whose function is twofold: (1) those blobs deemed uninteresting are eliminated, and (2) boundary points for the object are extracted and stored for later processing.

The technique employed in this module is the following: the binary image is raster scanned until an object pixel is found; then counterclockwise contour tracking starts using 8-connectivity criterion (Rosenfeld, [1982]). This tracking stops when the initial pixel is reached again. Then, a series of tests is run on measurements extracted for the object. If any of the tests fails this blob is rejected from further consideration. To avoid encountering a blob twice, the blob is deleted from the working image.

The surviving blob is considered the object and the list of its boundary points, together with the coordinates of its bounding box, are stored for later use. The algorithm used in this stage is as follows:

Contour Tracking and Filtering Algorithm
(Contreras, [1985])

Step 1. Search for object pixel (raster scan).

Step 2. If none is found, stop.

Step 3. Follow boundary of blob counterclockwise (8-connectivity criterion).

Step 4. Test blob measurements against thresholds.
(Measurements include: height, width, and area of rectangular box containing the blob; number of boundary points, and an indication if the blob touches the frame).

Step 5. If conditions are met, store box coordinates and list of boundary points.

Step 6. Delete blob from working image; go to Step 1.

It should be mentioned that for the object, features computed in this module are: perimeter length, eigenvalues, eigenvectors, orientation angle (with respect to x-axis), and average x and y values for boundary data.

Some of these features are to be used for a robot manipulator program and others are used during the recognition stage.

3. Component Labeling and Hole Detection

The previous module produced an image containing the digitized binary version of an object. The main purpose of this module is to compute the number of holes, if any, for that object.

A. Component Labeling

Let S be the set of ones in a binary-valued digital image. We can distinguish each component of S by labeling them; specifically, we can attach a unique descriptor to each separate component of the image. Without loss of generality, we can assume that the label is a positive integer.

Some of the methods available for component labeling are (Rosenfeld, [1969]):

(1) Propagation, where the first one encountered is changed to two, and every other one connected to the first pixel is also replaced with the same label until no further change is possible.

(2) Border Following, based on the property that an "outside border" is first encountered as a transition from 0 to 1, while the border of a hole is first encountered as a transition from 1 to 0. Each time a transition is detected, the border is followed and a label is assigned to all border points. (A different label is to be assigned to border of hole points).

(3) Tracking, where the image is scanned row by row and runs of zeros and ones are recorded. A label is assigned to the ones, and for each new row a test is made for overlap with a run in previous row. If so, the same label is assigned to the run in the present row; if there is no overlap, a new label is created. Two special cases should be considered: divergences and convergences.

The third method is amenable for real time processing (Foith et al., [1981]), and was the method chosen for this work. The proposed implementation is based on the concept of transitive closure (Appendix A).

Algorithm COMPONENT (From Gonzalez, [1980])

Let f(x,y) be a binary image, x=1,...,M, y=1,...,N in which zeros correspond to the background. This algorithm assigns a unique label to each connected component of zeros of f(x,y). It is based on 4-adjacency of runs of zeros.

Step 1. Raster scan f(x,y) until a zero is encountered.

Step 2. If none is encountered, stop.

Step 3. Let r be the row where the first zero was found.

Then,

(a) Assign a different label to each run of zeros. Assume n runs and Ln being last label.

(b) Form identity matrix B(nxn).

(c) Let r = r + 1.

Step 4. If r = M + 1, go to Step 7; otherwise continue.

Step 5. For r, let K be the number of distinct runs of zeros. If K=0, go to Step 6, otherwise process each run as follows.

(a) If kth run (1<=k <=K) is not 4-adjacent to any run in previous row.

Let n = n + 1 and assign label Ln to that run.

Set B(n,n) to 1.

(b) If kth run is 4-adjacent to only one run in previous row, assign label of that run to kth run.

(c) If kth run is 4-adjacent to two or more runs in previous row, set kth run to the lowest-valued label assigned to the runs to which present run is 4-adjacent.

Let L_{min} be that label, and let $L_{min + 1}$, $L_{min + 2}$, ..., L_{max} be the labels of the other runs to which the kth run is also 4-adjacent.

Set B(min,min+1), B(min,min+2), ..., B(min,max) to 1.

Step 6. Let r = r + 1 and go to Step 4.

Step 7. If B is 1x1 go to Step 10. Else, make B symmetric and go to Step 8.

Step 8. Obtain B+ (transitive closure of B).

Step 9. Scan B+, let m=1

(a) If m = n + 1, go to Step 10, otherwise continue.

(b) (Repeat for j = 1, ..., n)

If B+(m,j) = 1 change to L_m the pixels in f(x,y) whose label is L_j. Set to zeros column j in B+.

(c) Let m = m + 1, and go to Step 9(a).

Step 10. Stop. Now all connected components of zeros in f(x,y) have a distinct label.

This algorithm is of the order O(MxN) provided that the number of assigned labels (n) is small compared with the size of the image, otherwise Step 9 corresponds to a O(nxnxMxN). Actual implementation limits the number of zero runs for a given row and therefore the number of holes that may be collinear.

In order to illustrate the operation of the algorithm COMPONENT, three examples are provided.

Example 1. Figure 3.2 (a) shows a 7x7 binary image containing one hole. Background is denoted by zeros and object pixels are denoted by ones. Figure 3.2 (b) shows the (labeled) image obtained just before entering Step 7. Figure 3.2 (c) shows the matrix B before Step 7. Figure 3.2 (d) shows matrix B after Step 7 (B is now symmetric). Figure 3.2 (e) shows matrix B after Step 8 (that is B+).

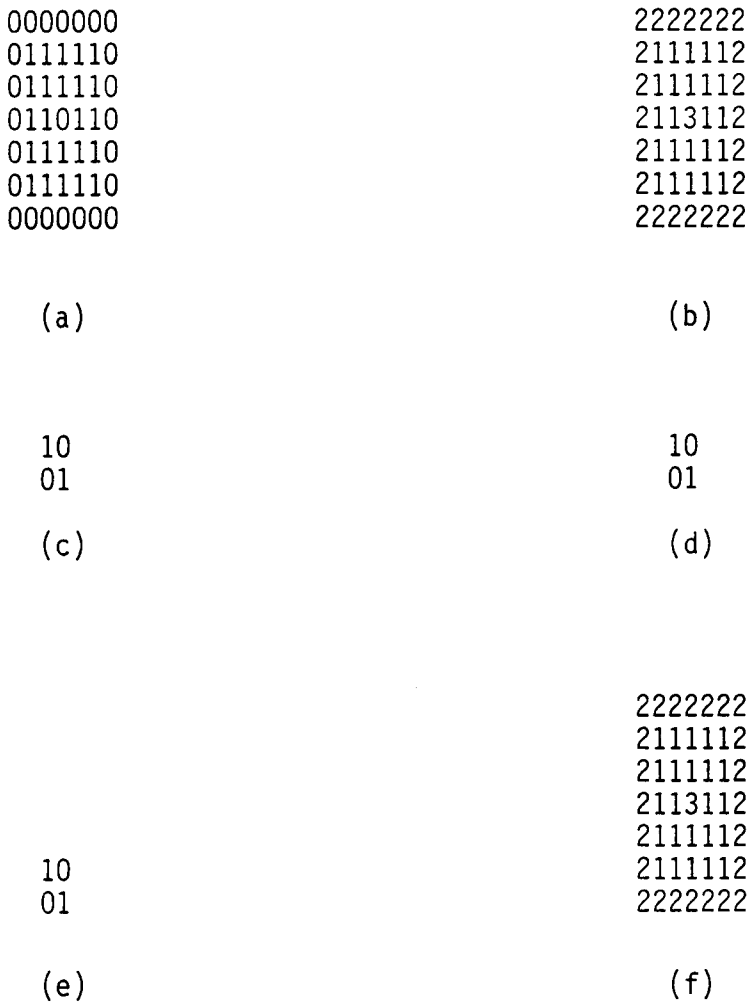


Figure 3.2 Algorithm COMPONENT applied to binary image with one hole.

(a) Initial image
(b) Labeled image (before Step 7).
(c) Matrix B
(d) B symmetric
(e) Matrix B+
(f) Final labeled image

Figure 3.2 (f) shows final labeled image. Notice that the label of the hole pixel is "3."

Example 2. Figure 3.3 (a) shows a 7x7 binary image containing two holes (more precisely, it has four connected components of background pixels, two of which will be classified as holes using algorithm HOLE). Figure 3.3 (b) and (c) shows the labeled image and the matrix B respectively, before entering Step 7. Figure 3.3 (d) shows matrix B after Step 7. Figure 3.3 (e) shows matrix B+. Figure 3.3 (f) shows the final labeled image. (Holes have labels 4 and 5).

Example 3. This final example illustrates the processing done inside Step 9 on the labeled image as processed with Steps 1 to 6. Figure 3.4 (a) shows a binary imate with one hole. Figure 3.4 (b) and (c) shows the labeled image and the matrix B just before Step 7. Figure 3.4 (d) shows the matrix B after Step 7 (B is now symmetric). Figure 3.4 (e) shows B+ after Step 8. Figure 3.4 (f) shows the resultant labeled image after Step 9.

The next processing step is to determine which connected components correspond to holes. This process is described by means of algorithm HOLE.

Algorithm HOLE

Let L be the set of labels assigned inside algorithm COMPONENT and let g(x,y) represent the labeled image.

Step 1. Scan the border of g(x,y) and let LB represent the set of different labels encountered during this scan.

Step 2. Let L_{dif} = L - LB, represent the set difference. Then, the cardinality of L_{dif} equals the number of holes in

```
0001000
0010100
0111100
0110000
0001000
0010100
0001000
```

(a)

```
2221333
2214133
2111133
2113333
2221333
2215133
2221333
```

(b)

```
1000
0100
0010
0001
```

(c)

```
1000
0100
0010
0001
```

(d)

```
1000
0100
0010
0001
```

(e)

```
2221333
2214133
2111133
2113333
2221333
2215133
2221333
```

(f)

Figure 3.3. Algorithm COMPONENT applied to binary image with two holes.

(a) Initial image
(b) Labeled image (before Step 7)
(c) Matrix B
(d) B symmetric
(e) Matrix B+
(f) Final labeled image

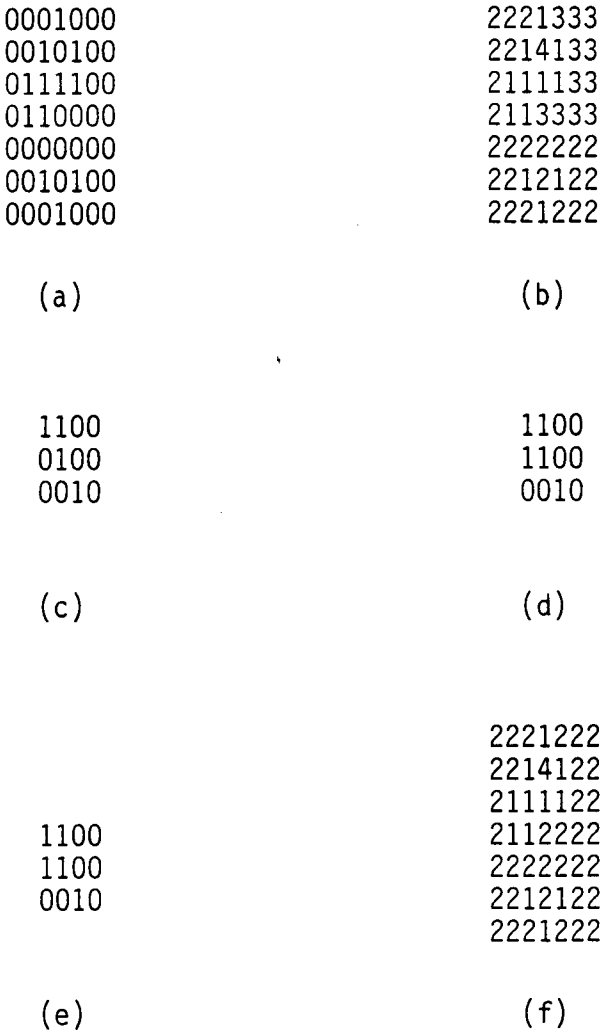


Figure 3.4 Algorithm COMPONENT applied to binary image with one hole

(a) Initial image
(b) Labeled image (before Step 7)
(c) Matrix B
(d) B symmetric
(e) Matrix B+
(f) Final labeled image

f(x,y) and set L_{dif} itself is the set of labels assigned to those holes.

Step 3. If card(L_{dif}) = 0, go to Step 4; otherwise, for every label in set L_{dif} (that is for every hole) compute: area, perimeter, and centroid.

Step 4. Stop.

This algorithm is of the order O(MxN) or size squared if the image is square. This algorithm works correctly with holes of arbitrary shape and also works with holes of unit area (one pixel). Actual implementation, however, makes no provision for the case of holes inside holes that may result after segmentation. Present scheme also assumes that only one object is in the scene, thus no extra computation is required to assign holes to blobs.

Also computed in the second module is the angular distribution for the holes. This feature and other features are to be used by the Recognition module.

4. Convex Hull Extraction

As was established in the first and second chapters, the use of "bays" represents an important feature for recognition. Early work employing this concept has been focused to character recognition (Gonzalez, [1983]), (Yamamoto, [1976]).

A subsequent problem then is how to obtain these "bays." The answer to this problem lies in the concept of the convex hull of a set of points in two dimensions.

Notation and definitions follow.:

Let $S = \{s_1, s_2, \ldots, s_n\}$ be a finite planar set of points. The convex hull of S is defined as the minimum area convex set containing the original set, and it is denoted CH(S).

The extreme points of CH(S) define CH(S) and are therefore of interest.

CD(S) = CH(S) - S, is known as the convex deficiency.

An O(nlog n) algorithm due to Graham [1972] was chosen for finding the convex hull. The correctness of this algorithm was proved by its author.

Graham's Algorithm (summarized)

Step 1. Find a point P which is in the interior of CH(S).

Step 2. Convert each point in S to polar coordinates (with origin P).

Step 3. Order this new set in terms of increasing angle.

Step 4. Delete any point with radius = 0, as well as any point having the same angle as another, but lesser radius. Let S' be the resultant set, and let n' be the number of points left. Clearly $n' \leq n$.

Step 5. For a triad of consecutive points in S':

(a) If middle point is "concave," delete it. Choose a new triad composed of the point previous to the middle one, a point previous to the former, plus the old third point.

(b) If the middle point is "convex," advance pointers by one (no deletion) and form a new triad.

After 2n' iterations of Step 5 we are left with a subset of S of all extreme points of CH(s).

It might be useful at this point to consider a more concrete (graphical) interpretation of the processing done with this module. Consider object depicted in Figure 3.5(a). The objective is to extract its "bays" as shown in Figure 3.5(e). Obviously, we are given a discrete representation as in Figure 3.5(b). We can think of each point in this discrete representation as being a "peg" inserted in a hole on a two-dimensional board. If we allow an elastic band to be wrapped around the object (Figure 3.5(c)), the set of extreme points will be given by those "pegs" that tough the band in a position where the band changes of direction. (Figure 3.5(d))

Description of the "bays" will be the purpose of the next module.

5. Processing of Concavities

After the set of extreme points has been extracted, the next logical step is the description of the concavities ("bays"). Concavities represent an invariant feature for a given object and its description should retain this quality.

Approaches for shape analysis are generally either region oriented or boundary oriented (Gonzalez and Safabakhsh, [1982]). An example in the first category is the technique of principal axis analysis; included in the second category is graph representations (strings, trees,...).

The approach followed in this module has been a heuristic one (no general theory exists for shape analysis), in the sense that

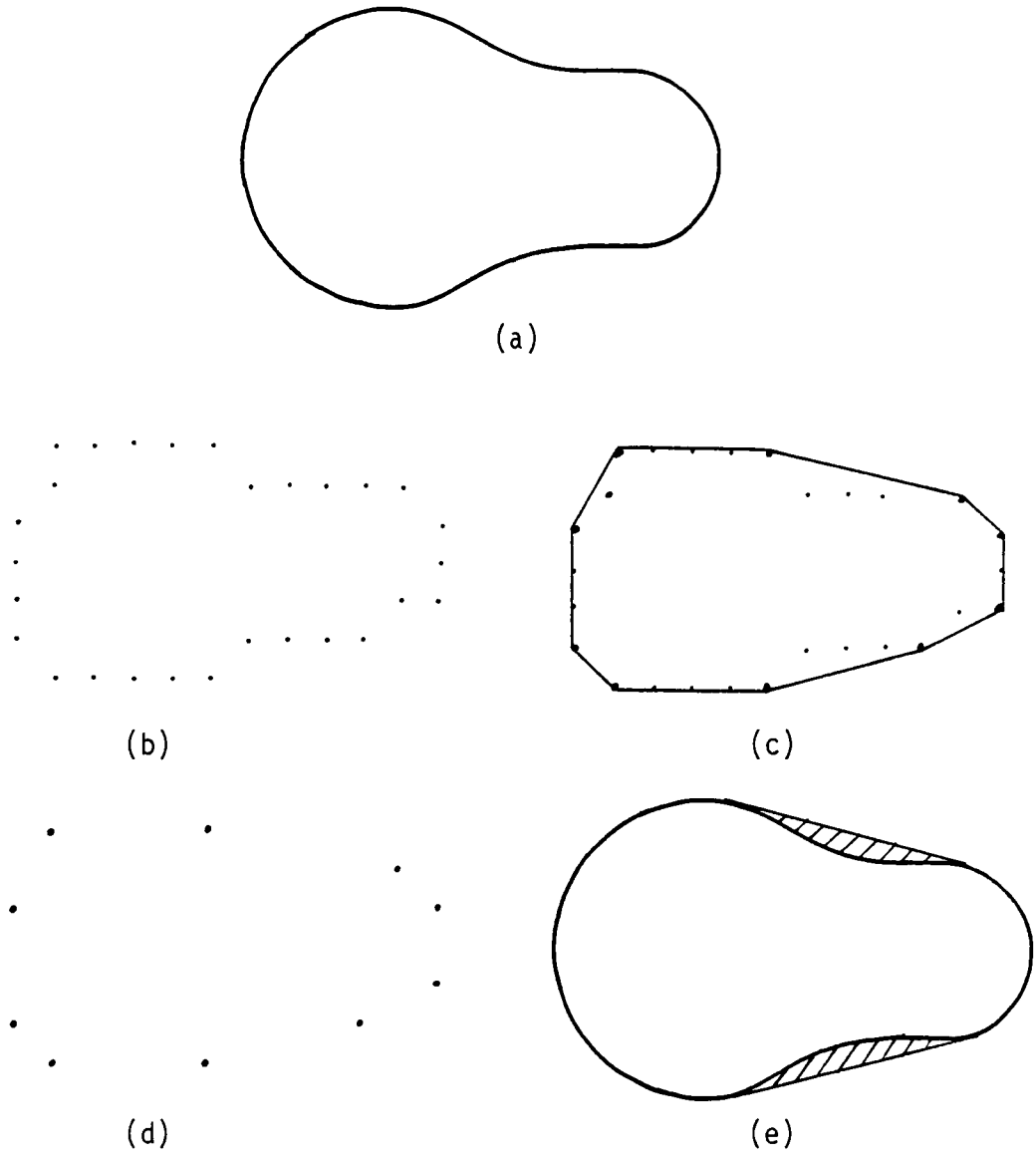


Figure 3.5. Convex Hull Extraction.

(a) An object
(b) Discrete representation
(c) Elastic band
(d) Extreme points
(e) Bays (shaded area)

concavities can be considered both a special case of a boundary and a special case of a region; therefore, their description should try to follow to some extent this dual property.

This module can be conceptually separated in two submodules: (A) Detection and measurements, and (B) Reduction to a string.

A. Detection and measurements

The set of extreme points of CH(S) obtained in a previous module defines implicitly those points that are not extreme. We shall call these points concave points. A concavity will be defined here as the set of points inside the region defined by the straight line joining two consecutive extreme points and the boundary of the object.

Let C be the centroid of CH(S), and let P_i and P_{i+1} be a pair of consecutive extreme points (not necessarily 8-connected). A concavity C_i will be considered a "real" (or proper) concavity if it satisfies the following tests:

Test 1. arclength/distance(P_i,P_{i+1}) > threshold

Test 2. area of C_i/distance(P_i,P_{i+1}) > threshold

From an implementation point of view, measurements are required prior to classifying concavities, so all measurements, including those used in Test 1 and Test 2, are computed before detection.

Figure 3.6 illustrates the meaning of these various measurements. Also included in actual implementation is a measure of symmetry (or asymmetry) as given by the third moment.

B. Reduction to a string

After the border of the object has been tracked, we are left with a string of symbols. As mentioned earlier, strings are a special

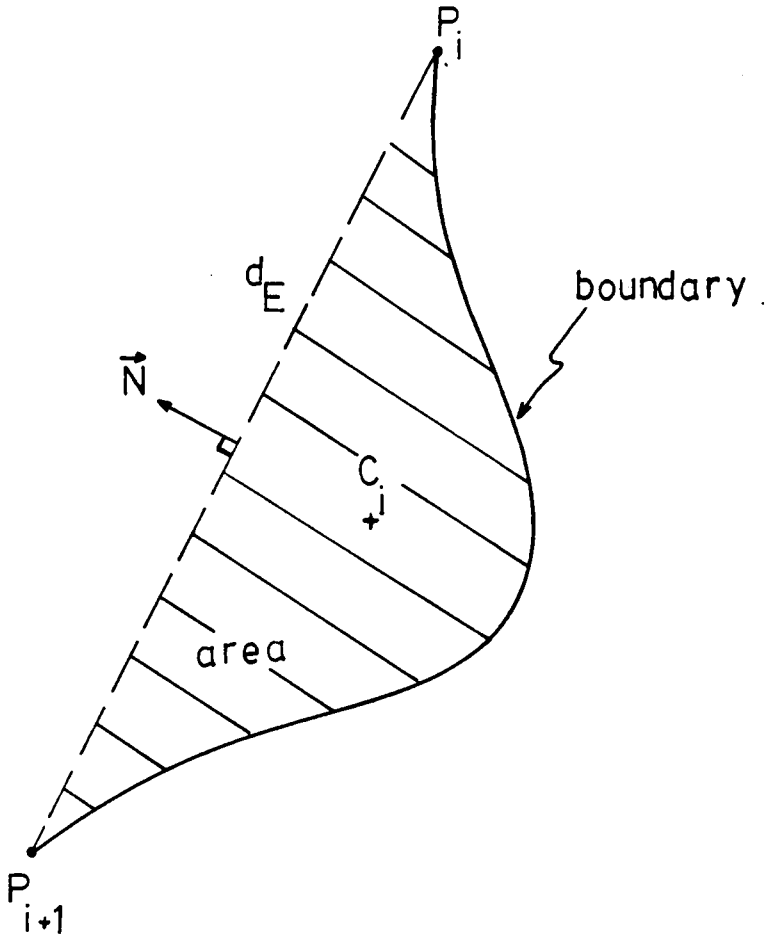


Figure 3.6. Measurements on a concavity.

P_i = ith extreme point

d_E = euclidean distance

C_i = centroid of ith concavity

d_{Nmax} = maximum distance from straight line to boundary

$\vec{N}$ = external normal to straight line P_i, P_{i+1}

case of graph representation, which is one of the boundary-oriented techniques for shape description.

It is well known, however, that string representations are sensitive to noise and sometimes sensitive to orientation of the objects (Braunegg and Gonzalez [1984]). To minimize this problem concavities have been classified using area as one of the parameters. Moreover, the resulting string can be further reduced by subsuming onto one symbol all adjacent segments considered nonconcavities. The principal reason behind this operation is to achieve a more stable string representation.

Let $S = s_1 s_2 \ldots s_n$ represent the initial string, and let $A_i = (ai_1, ai_2 \ldots ai_m)$, $i=1,\ldots,n$, represent an attribute vector associated with each symbol s_i. The output of this module will be a string $S' = s'_1\, s'_2 \ldots s'_r$, $r \leq n$, with associated attribute vector $A_j = (a_{j1},\ldots,a_{jm'})$, $j=1,\ldots,r$. It is assumed in what follows that $m'=m$.

Further information on the use of attributed grammars and semantics can be found in references (Fu, [1982]), (Knuth, [1968]), and (Thomason, [1982]).

For this work, the attributes considered are additives, with few exceptions, that is, when combining equivalent attributes the resultant attribute can be obtained by simply adding individual attributes.

Figure 3.7 shows two such additive attributes, internal angle and arclength, and one nonadditive attribute, euclidean distance.

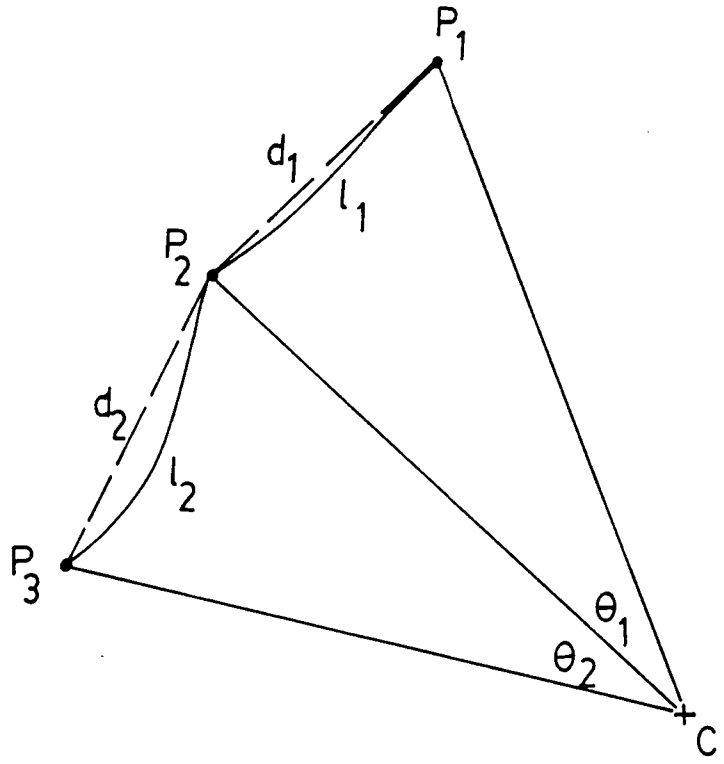


Figure 3.7. Additive and nonadditive attributes.

P_1, P_2, P_3 = Extreme points

d_1, d_2 = Euclidean distance

l_1, l_2 = Arclength

Although the Euclidean distance is not an additive attribute, for zones with smooth curvature an approximation could still be valid for small distances, but if the number of segments is large then error becomes increasingly large. Therefore, euclidean distance is preferably computed by means of an equation (a simple one though), instead of using simple additivity, as follows:

$$d = dE(P_1,P_3) = \sqrt{(x_1-x_3)^2 + (y_1-y_3)^2}$$

Proper concavities are nonconcatenables in the present scheme, thus requiring only a reordering of indices for keeping track of associated attributes.

For actual implementation, indices to original concavities are kept in a table that also shows which symbols have been connected together. In this way no recomputing is required except for nonadditive attributes.

In order to illustrate the utilization of the data structures used to code strings and attributes, a generic example will be presented. Consider the closed contour shown in Figure 3.8.

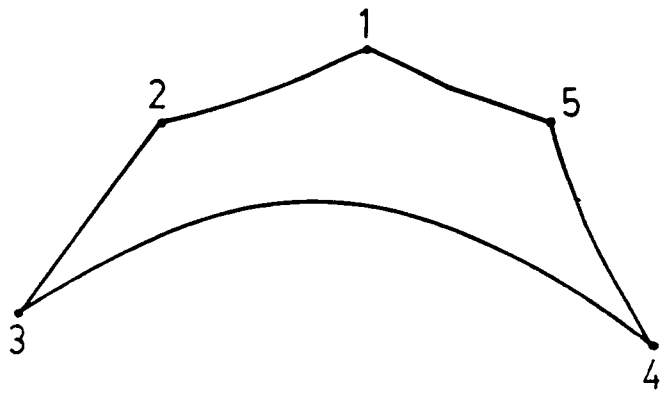


Figure 3.8. Closed contour with five extreme points.

After classification (as described in section A), a initial string is obtained containing five symbols:

$$S_I = s_1\ s_2\ s_3\ s_4\ s_5$$

Where s_i is the symbol (primitive) associated with the ith concavity (defined by extreme points i and i+1), in present scheme s_i belongs to the set {L,C}. The initial string can be described now as:

$$S_I = L\ L\ C\ L\ L$$

where,

L = Line segment

C = concavity

All adjacent line segments are allowed to merge under the criteria that curvature is small and that the concavities carry most of the information for describing the object boundary. It is also assumed that small line segments may be produced by noise and therefore should not be considered the same as concavities. The anterior criteria will allow compact string representations for the objects. The initial string is saved for analysis that might require higher resolution than the resolution obtained with the reduced strings. Further discrimination can be achieved by utilizing the attributes associated with each string.

Again, with reference to Figure 3.8, the final (reduced) string is as follows:

$$S_F = L\ C$$

after merging s_1, s_2, s_4, and s_5.

The connectivity table for this example is:

Table 3.1 Connectivity Table for Example.

j \ i	1	2	3	4	5
1	1	2	4	5	-
2	3	-	-	-	-

The columns in the table represent the input symbols (for initial string) and the rows represent the output symbols (for the final string). For instance, the entries in the first row indicate that the first symbol in the final string, that is "L," is formed by the merging of the first, second, fourth, and fifth symbols from the initial string.

Let A_i, i=1,2,...,5, represent the attribute vectors associated with the symbols of the initial string, and let B_j, j=1,2, represent the attribute vectors associated with the reduced string, then, for the case under consideration.

$$B_1 = A_1+A_2+A_4+A_5 \quad \text{(for additive components)}$$

$$B_2 = A_3$$

An example for a particular object containing actual data is presented in the next chapter.

6. Recognition

Recognition has been defined as a labeling process (Gonzalez and Safabakhsh, [1982]), by means of which the image of an unknown object is assigned a label (or class name).

Recognition can be divided in two main categories: decision-theoretic and structural. In the former approach a pattern is represented by a N-dimensional feature vector, and classification is based on decision (discriminant) functions.

Industrial vision systems are generally based on nearest-neighbor classifiers. In this method each class is represented by a prototype:

$$m_i = \frac{1}{N_i} \sum_{k=1}^{N_i} X_k \qquad , \quad i = 1,2,\ldots,M$$

where,

m_i = prototype (average) representing ith class

X_k = vector belonging to kth class

N_i = number of vectors in ith class

M = number of classes

Given an unknown pattern u, it will be classified into such class that minimizes

$$D_j = || u - m_j || \qquad , \quad j = 1,2,\ldots,M$$

Structural methods, on the other hand, are based on geometric relationships inherent in the shape of the object. String matching methods and syntactic methods are normally used for boundary data.

The use of semantic information has also been recognized as an important component in syntactic-based recognition systems. (Braunegg and Gonzalez, [1984]), (Fu, Las Vegas, [1982]).

The particular recognition module developed in this thesis is a two-level system (more levels can be added later). In the first level, a tree-based classifier uses global data to minimize processing time.

The following features are used in the first level:

1. number of holes
2. length of reduced string
3. compactness (perimeter squared/area)
4. area holes relative to area object
5. ratio of eigenvalues

Those reference-objects (Objects in Data Base) that fail in any test are rejected from further consideration. Those remaining are called viable candidates.

The algorithm used in this module is as follows:

Let M = number of classes, and let "x" be the suffix for data pertaining to the unknown object.

Step 1. i = 1; set number of viable candidates to zero

Step 2. i = i + 1, if i = M + 1, then stop; otherwise, continue

Step 3. (a) if number holes x <> number of holes reference (i), then go to Step 2; otherwise, continue

(b) if $|\#(S_x) - \#(S_i)|$ > threshold, then go to Step 2; otherwise, continue

(c) if $|PSA_x - PSA_i|$ > threshold, then go to Step 2; otherwise, continue

(d) if $|$ (area holes/area object)$_x$ - (area holes/area object)$_i|$ > threshold, then go to Step 2; otherwise, continue

(e) if $|$(ratio eigenvalues)$_x$ - (ratio eigenvalues)$_i|$ > threshold, then go to Step 2; otherwise continue

Step 4. (a) increment number of viable candidates

(b) store class name

Step 5. go to Step 2.

The second level is preceded by a decision step of the form: If there are no candidates left, then reject the unknown object. If there is only one candidate, then assign the object to the class of the candidate. Otherwise, go to Second Level.

For the second recognition level, two techniques have been proposed and implemented. First, for objects with holes, a highly probable event for industrial parts, a nearest neighbor classification scheme compares angular differences of holes against similar data stored in Data Base for the viable candidates. Second, a Levenshtein distance classifier compares the string of the unknown object with the strings associated with viable candidates. The Levenshtein distance between two strings x and y is defined as the smallest number of

transformations required to derive string y from string x. Transformations are deletion, insertion, and substitution of terminal symbols. The computation has been carried out using the technique of dynamic programming. The choice of dynamic programming is as efficient in terms of length of computation time as can be possible for string matching. Every transformation (or edit operation) above mentioned can be assigned a cost (or weight); let W_d, W_i, and W_s represent the deletion weight, insertion, and substitution weight respectively.

Before presenting the algorithm employed some notation is required:

X = a string of symbols (or primitives)

X<i> = ith symbol of string X

X<i:j> = the substring from the ith to the jth symbol of X, inclusive

X(i) = X<1:i>

D(i,j) = the edit distance of minimum cost taking the substring X(i) to substring Y(j)

Algorithm String Distance

Let $X = x_1\ x_2 \ldots x_n$, and $Y = y_1\ y_2 \ldots y_m$ represent two strings of symbols.

Let #X = length of string X (=n)

Then,

Step 1. D(0,0) = 0

Step 2. (a) for i = 1 to #X

do D(i,0) = D(i-1,0) + W_d

(b) for j = 1 to #Y

do $D(0,j) = D(0,j-1) + W_i$

Step 3. for i = 1 to #X

for j = 1 to #Y

$e_1 = D(i-1,j-1)$, if $x_i = yj$

$= D(i-1,j-1) + W_s$, otherwise

$e_2 = D(i-1,j) + W_d$

$e_3 = D(i,j-1) + W_i$

$D(i,j) = \min\{e_1,e_2,e_3\}$

Step 4. distance between X and Y is D(n,m)

This algorithm is of the order O(X x Y) and proof of correctness can be found in Wagner and Fisher, [1974].

The following example will illustrate the application of this algorithm. Let X = 2101311 and Y = 21101131 represent two strings, where every symbol belongs to the set {0,1,2,3}. Let $W_d=W_i=W_s=1$, then Table 3.2 shows the entries when algorithm is applied to the test string.

Table 3.2 Computation of Levenshtein Distance.

i \ j	0	1	2	3	4	5	6	7	8	= #Y
0	0	1	2	3	4	5	6	7	8	
1	1	0	1	2	3	4	5	6	7	
2	2	1	0	1	2	3	4	5	6	
3	3	2	1	1	1	2	3	4	5	
4	3	2	1	2	2	1	2	3	4	
5	5	4	3	2	2	2	2	2	3	
6	6	5	4	3	3	2	2	3	2	
#X = 7	7	6	5	4	4	3	2	3	3	

Therefore, the distance between these two strings is 3. Graphically, a minimum cost-matching is as follows:

```
2 1   0 1 3 1 1
| |   | | | | |
| |   | | | | |
2 1 1 0 1 1 3 1
```

where there is one insertion and two substitutions.

When rotation is a factor to consider, as it happens for object recognition, the minimum distance between two strings can occur in any rotation relative to the two strings. The simplest approach considers only the rotations of one string, the number of rotations being equal to the length of the string. More elaborate approaches can cut down the number of rotations by starting with the main concavity of the object, or similar criteria.

7. Training

Before any recognition may take place, the system must be taught the characteristics of the objects it will be trying to recognize. The technique called "training by showing" was considered most appropriate because then the computer is left with the task of extracting all pertinent features from the object.

This module must interact with the user. For example, the user has to select which object(s) should constitute the Data Base (also called "library"). Interaction is provided through a menu-driven dialogue. In this phase the vision system constructs the Data Base that will be used during the Recognition phase.

The operation of this module can be described conveniently by means of an algorithm:

Step 1. If image of object already stored, go to Step 3.

Step 2. Obtain a good image of object (on monitor).

Step 3. Select camera or file for input source.

Step 4. Select appropriate object intensity level (default is dark object over clear background).

Step 5. If object is not in Data Base, keep data files; otherwise, delete data files for the object.

Step 6. If there are more objects to add to Data Base, go to Step 1; otherwise stop.

Evidently, Step 4 has been built around the Recognition module because even if the user can clearly discriminate between two objects, there is no assurance that the Recognition module will, and the user should be informed of this event so that he or she takes appropriate action. (For instance, he or she may decide to add a new processing module or a new level to the Recognition module.)

CHAPTER 4

EXPERIMENTAL RESULTS

The performance of the computer vision system is influenced by a series of factors, starting from the image acquisition process up to the recognition module. Certain parameters are of special importance when designing computer vision systems for object recognition. They include illumination, sampling size, translation, size variation, orientation and noise. It is desirable that the system perform well under a wide range of variation within these parameters.

Illumination. This is a critical factor, because when global thresholding is being used for image segmentation, only highly contrasted scenes produce good results. Otherwise, the resulting image may be either too dark or too light. The specular reflections that new industrial parts may sometimes originate can cause problems by generating pseudo holes after segmentation.

The effect of the other parameters on the recognition features, hence on the functioning of the system, can be described conveniently by means of a table. Let F1 to F5 represent the features used in the recognition module, where,

F1 = number of holes

F2 = length of reduced string

F3 = compactness (PSA)

F4 = area holes relative to area object

F5 = ratio of eigenvalues

Sampling size. In this work square images with three different sampling sizes have been tested. These sizes are 64x64, 128x128, and 256x256, all having 256 gray levels. Bigger sizes (e.g., 512x512) can be accommodated after redimensioning of the arrays used in the programs. It is important to realize that most computer vision systems are restricted only to one sampling size, usually 256x256.

A qualitative description of the effect of the sampling size on each feature is given (all other parameters are assumed fixed) in Table 4.1.

Table 4.1 Effect of sampling size on recognition features.

Feature	Effect
F1	none
F2	some
F3	negligible
F4	negligible
F5	negligible

Therefore only the threshold associated with F2 needs to be "tuned" if other sampling sizes are to be added.

Size variation. Variation in the size of the digitized images can be due to the variable distance camera-object, the type of lenses on the TV camera, or even due to the variable size of the given class

of objects. Most systems are designed assuming fixed distance camera-object, and, for the case of visual inspection, size variation usually constitutes a rejection criterion. The present system, however, should tolerate size variations. The effect that this parameter produces on the recognition features is tabulated in Table 4.2

Table 4.2. Effect of size variation on recognition features.

Feature	Effect
F1	none
F2	some
F3	small
F4	small
F5	some

It can be seen that thresholds associated with F2 and F5 should be set carefully when allowing for size variation.

Translation. Translation means here the displacement that the object experiences with respect to a coordinate system located on the camera. Translation was the least influential factor, provided that the whole object was located in the field of view of the camera. (Partial views are rejected during the filtering stage.)

Object orientation. Unlike some recognition systems (Braunegg, [1983]) that require that the objects be presented in multiple orientations during the training phase, this system requires only one image with the object in any orientation. (It still requires one view for each stable position of the object.) This flexibility is important

for a vision system that is to be used in conjunction with a robot manipulator.

Table 4.3. Effect of object orientation on recognition features.

Feature	Effect
F1	none
F2	some
F3	small
F4	small
F5	some

Noise. All possible factors that contribute to distorting the resulting image for a given object will be considered noise. The noise that is not connected to the object (its image) is filtered out in the filtering module. A "spike" detector that works while the boundary of the object is being traced attempts to smooth the boundary by eliminating thin lines ("spikes") that might have been produced in the segmentation process. This approach is problem dependent, because it is assumed that no object (or part of it) in the Data Base is so thin as to be represented as a thin line that retraces itself when its boundary is being tracked.

Test case. In order to illustrate the operation of the system, the results generated for a particular object will be presented. The object is the symbol or character "A" and is one of the objects in the Data Base. This object will also be the subject of further discussion in the next section, which deals with class separability.

Figure 4.1 presents the various processing steps done on the original image. The explanation of every step has been detailed in the previous chapter. It should be mentioned that these operations are done only once while building the data base, and do not need to be done again when recognition is requested.

Table 4.4 shows actual data generated in the processing of the concavities. The term "zone" is used to indicate both line segments and concavities. (It is the region between two consecutive extreme points.) It must be recalled that the initial boundary has no classification whatsoever. Figure 4.2 shows the original string. This string results when every "concavity" is classified as either "line segment" or as "proper concavity." The term "proper concavity" has been used to emphasize the fact that these concavities have passed two classification tests. Figure 4.3 shows the final (reduced) string. This string is derived from the string on Figure 4.2 when all adjacent "line segments" are merged together.

Figure 4.4 depicts which of the original symbols were allowed to merge. This table serves also as a pointer to the attributes associated with the original (nonreduced) string.

Table 4.5 shows the resultant attributes for the reduced string.

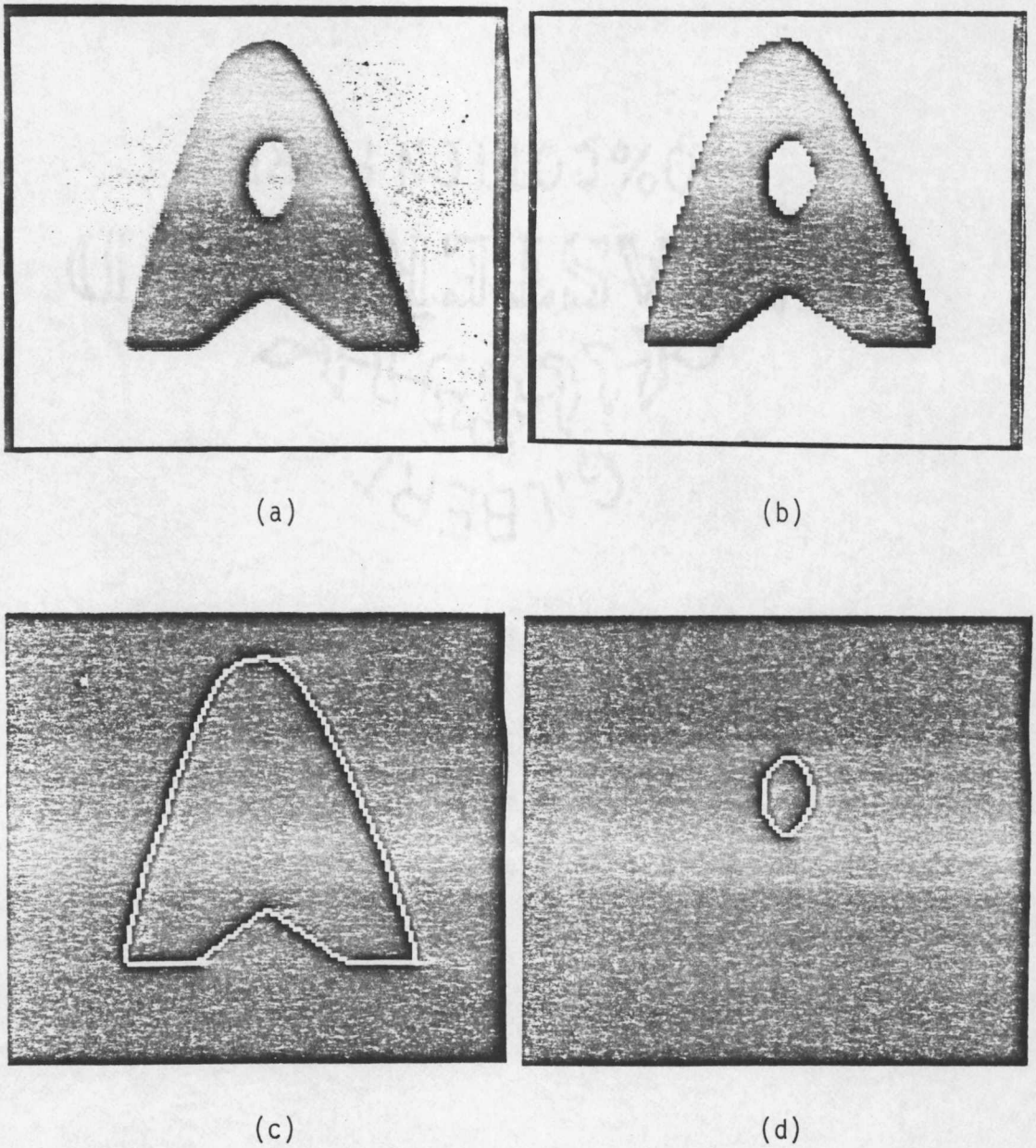
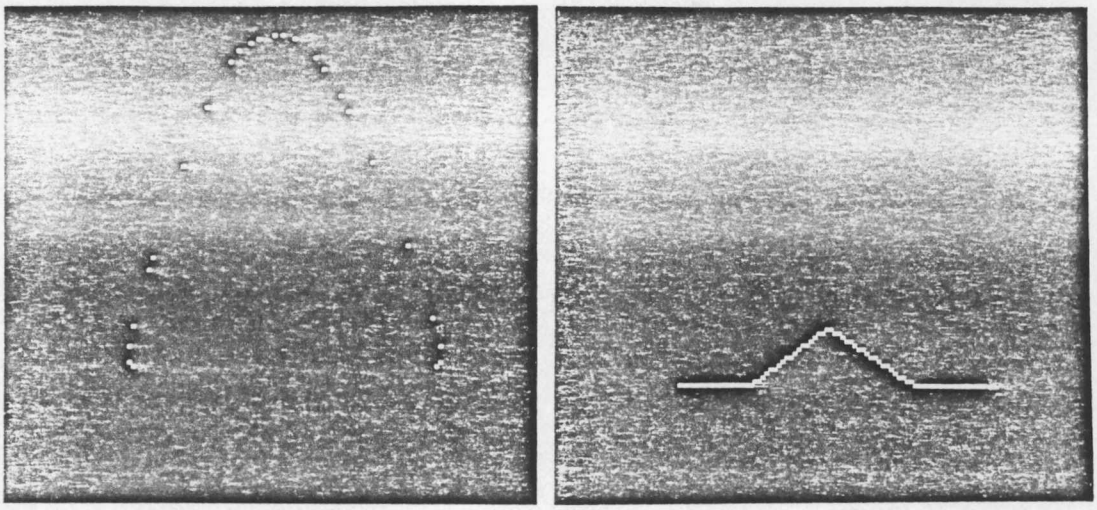


Figure 4.1. Test case: the character "A."

(a) gray level image
(b) binary image
(c) external boundary
(d) border of the hole
(e) extreme points
(f) main concavity



(e) (f)

Figure 4.1 (continued)

Table 4.4. Data generated for each concavity.

Zone	Dis	Ang	Unor	Iar	Perim	----Centroid-----	
1	5.657	225.000	0.000	0	5.656	21.000	57.000
2	5.832	239.036	-0.132	2	6.242	25.500	53.833
3	17.000	241.928	-0.323	7	18.312	35.500	48.438
4	6.708	243.435	0.000	1	7.242	46.000	42.714
5	12.083	245.556	0.325	5	13.070	54.500	38.917
6	7.616	246.801	0.310	3	8.242	63.500	34.875
7	5.385	248.199	0.269	2	5.828	69.000	32.333
8	13.928	248.962	0.071	6	15.070	78.500	28.929
9	8.544	249.444	-0.064	3	9.242	89.000	24.889
10	3.162	251.565	-0.707	1	3.414	94.500	22.750
11	2.000	270.000	0.000	0	2.000	97.000	22.000
12	78.160	356.332	0.228	391	90.834	95.544	61.000
13	3.000	90.000	0.000	0	3.000	101.500	100.000
14	31.048	104.931	0.011	14	33.312	85.000	95.548
15	7.280	105.945	-0.146	3	7.828	66.500	90.625
16	12.649	108.435	0.001	4	13.656	57.000	87.692
17	13.928	111.038	-0.071	6	15.070	44.500	83.071
18	17.889	116.565	0.000	4	19.312	30.000	76.765
19	4.243	135.000	0.000	0	4.242	20.500	71.500
20	2.236	153.435	0.000	1	2.414	18.667	69.000
21	5.000	180.000	0.000	0	5.000	18.000	65.500
22	4.123	194.036	0.626	1	4.000	18.800	61.000

L-L-L-L-L-L-L-L-L-L-L-C-L-L-L-L-L-L-L-L-L-L

Figure 4.2. String after Pass I.

where,

L = Label assigned to Line Segment
C = Label assigned to Proper Concavity

L-C

Figure 4.3. String after Pass II.

Output	Input
1	1-2-3-4-5-6-7-8-9-10-11-13-14-15-16-17-18-19-20-21-22
2	12

Figure 4.4. Table of connectivity (LTABLE)

Table 4.5. Attributes for the reduced string.

Attribute	Component	
	First	Second
Dis	78.16	78.16
Angd	277.74	82.26
Perim	202.15	90.83
Iar	63	391

where,

Dis = Euclidean Distance between two extreme points (or between initial point and last point for a combined segment)

Angd = Angle subtended from one extreme point to the following, when measured from the centroid of the object

Perim = Perimeter (arc length)

Iar = Area (an integer)

Class separability using semantic information. The following case illustrates the implementation of a specific module that helps in discriminating between a pair of objects having similar shapes. The pair of objects are the character "A" and a type of "Scotch tape."

Figure 4.5 shows these two objects. For the system, both objects have one hole and a reduced string with two symbols (with one main concavity). Since the distance camera-object is variable and

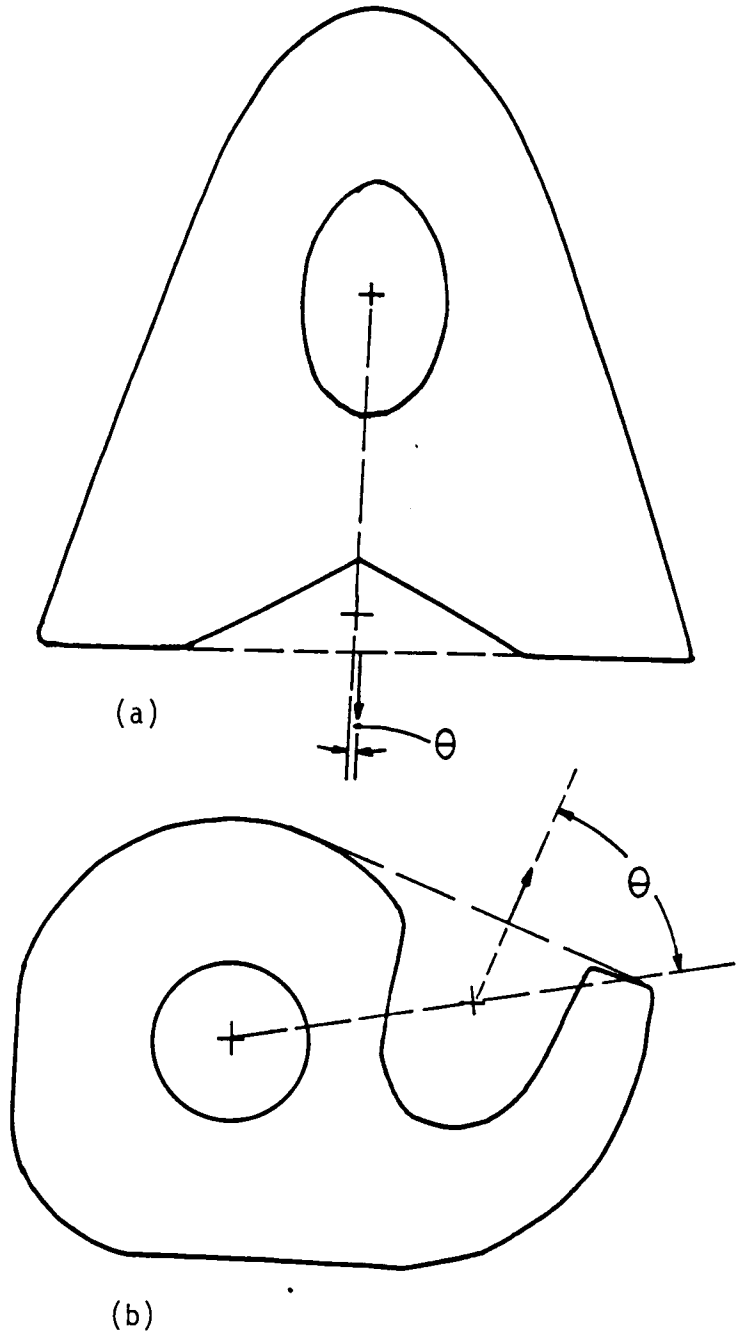


Figure 4.5. Semantic information to discriminate between the letter "A" and "Scotch tape."

(a) Letter "A"
(b) "Scotch tape"

unknown, simpler measures like area alone are doomed to failure. The proposed solution makes use of the orientation angle for the main concavity. This feature (attribute) was computed during the processing of the concavities. This angle is to be compared with the angle that the vector joining the centroid of the hole with the centroid of the concavity makes with respect to the x-axis. The minimal angle so defined will be used as the discriminant feature.

As can be seen in Figure 4.5 (a) and (b), the angle theta for the tape is much larger than the angle theta computed for the letter "A." Typical values are five degrees for the letter "A" and sixty degrees for the "tape." Therefore, a single threshold is enough to classify these two shapes.

In many engineering situations, the solution to a given problem is not necessarily unique, and this case is not an exception. Other possible solutions for this case include: (a) the use of symmetry, (b) the use of curvature, or (c) the number of boundary crossings that the line that passes through the centroid of the hole and the centroid of the concavity may encounter. All of these possible solutions require, however, extra computation when compared with the chosen approach. However, no claim is made concerning the optimality of the solution implemented.

A Set of Samples Recognized by the System

The following three samples are representative of the kind of objects stored in the Data Base (Appendix B) and further illustrate the functioning of the system. The first sample, shown in Figure 4.6 (a) is a wrench digitized with a resolution of 64x64 pixels and 256

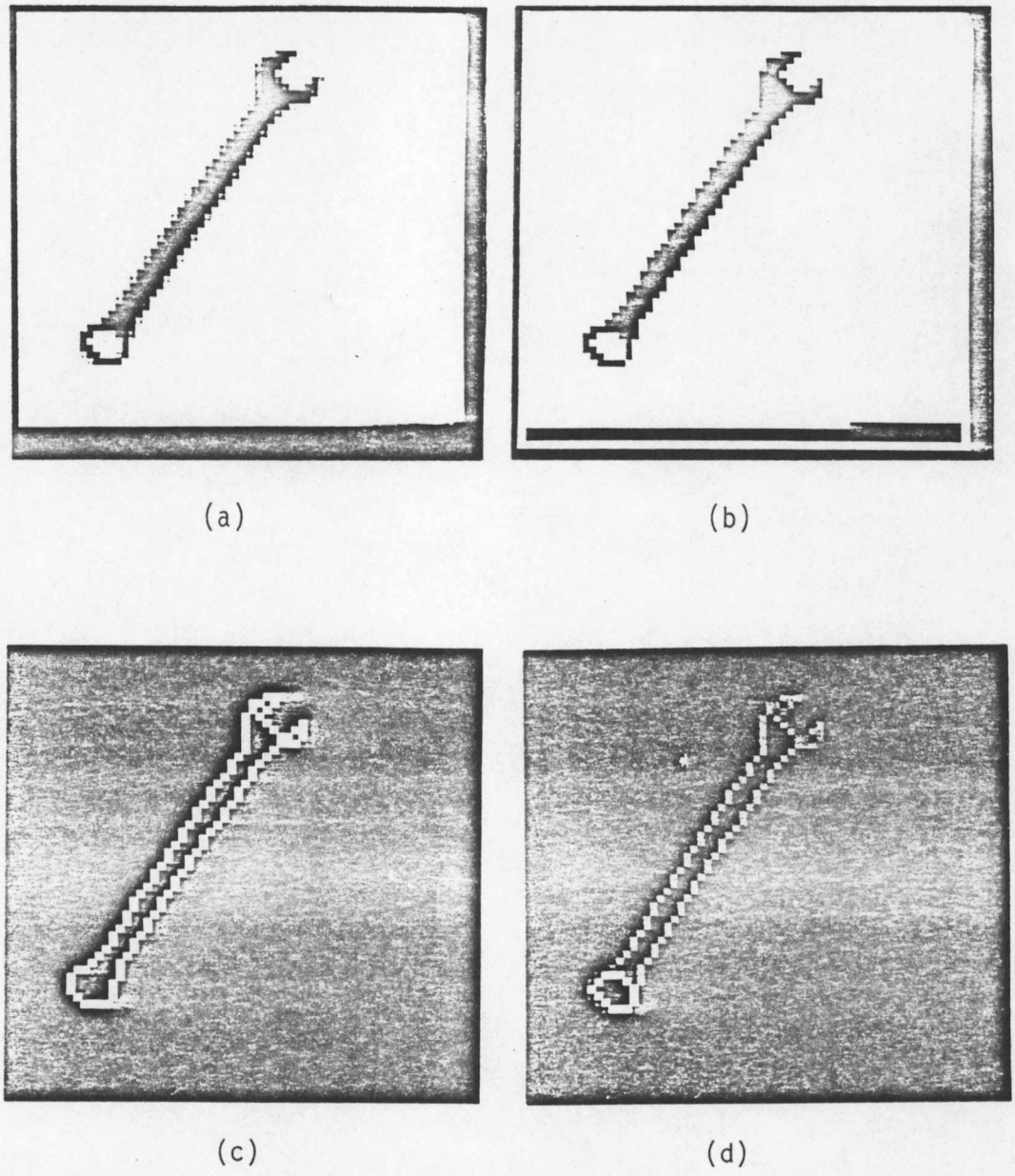
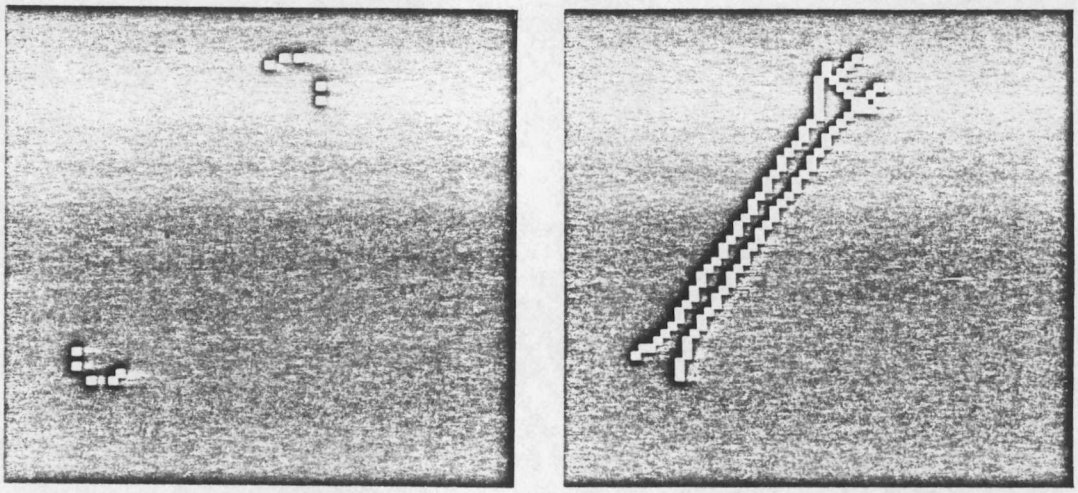


Figure 4.6. Combination Wrench.

(a) Gray level image
(b) Binary image
(c) External boundary
(d) Border of the hole
(e) Extreme points
(f) Main concavities



(e) (f)

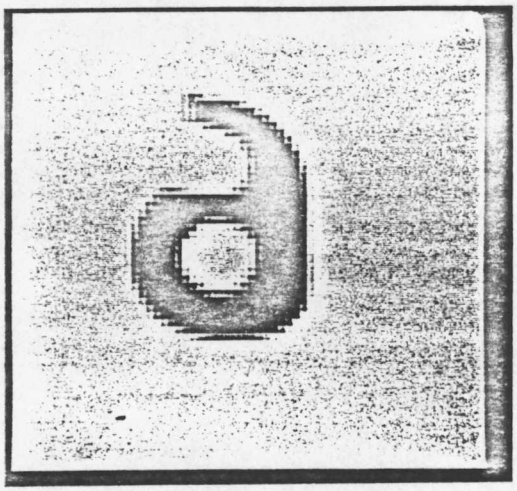
Figure 4.6 (continued)

gray levels. (The same object is stored in the Data Base with a resolution of 256x256 pixels.) Figure 4.6 (b) shows the binary image resulting after segmentation. Figure 4.6 (c) shows the external boundary. Figure 4.6 (d) shows the boundary of the hole. Figure 4.6 (e) shows the extreme points for the external boundary. Figure 4.6 (f) shows the main concavities.

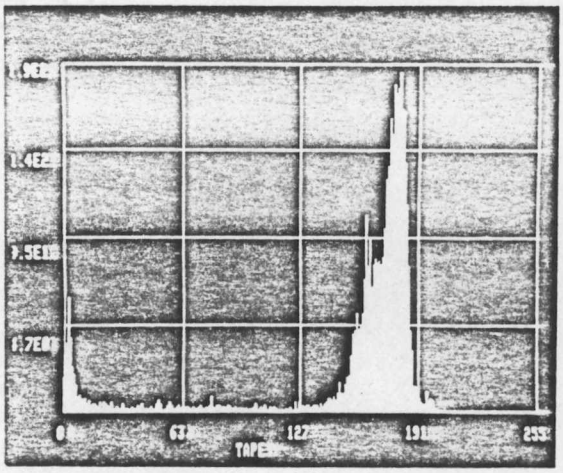
The second sample, a "Scotch" tape, is shown in Figure 4.7 (a). Figure 4.7 (b) shows the histogram for the gray level image. Figure 4.7 (c) shows the binary image. Figure 4.7 (d) shows the external boundary. Figure 4.7 (e) shows the border of the hole. Figure 4.7 (f) shows the extreme points. Figure 4.7 (g) shows the main concavity.

The third sample, an automobile part (connecting rod), is shown in Figure 4.8 (a) which contains, from top to bottom, left to right: gray level image, binary image, external boundary, and border of the holes. Figure 4.8 (b) shows: the extreme points, the main concavities (those sharing an extreme point appear to be merged, but they are considered independent in the software), external boundary and border of the holes superimposed on the binary image, and main concavities superimposed on top of the binary image.

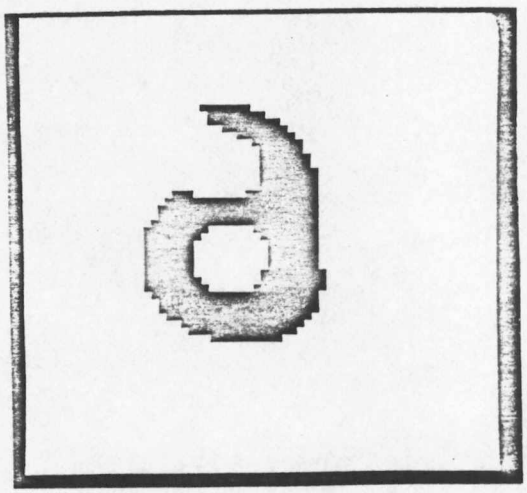
The proposed vision system has been found successful in recognizing a diversity of samples. However, there is still room for improvement, and one of the difficulties that may cause erroneous classifications is the tendency to overgeneralize, which is partially due to the many-to-one operation while reducing the initial string. This problem can be addressed by extensive use of the proposed attributes. Further recommendations on how to improve the present system are presented in the next chapter.



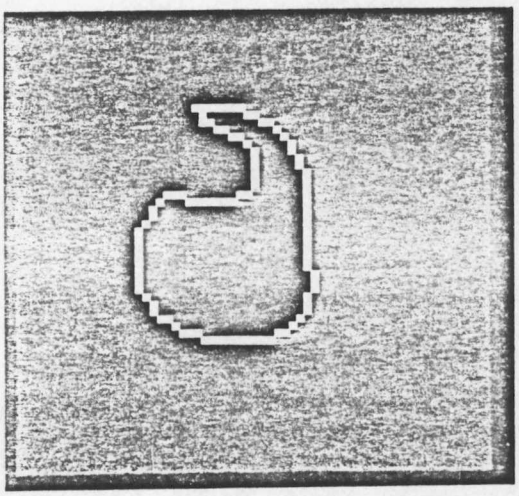
(a)



(b)



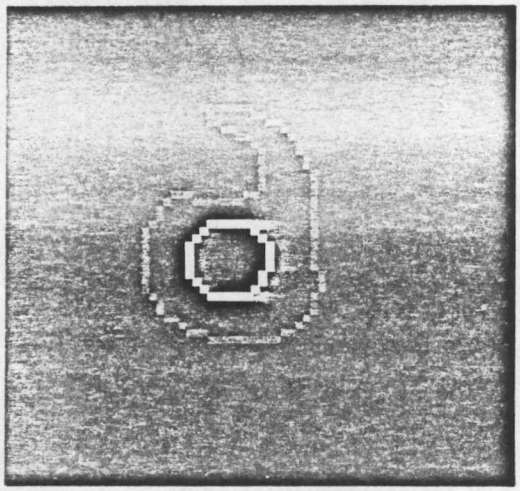
(c)



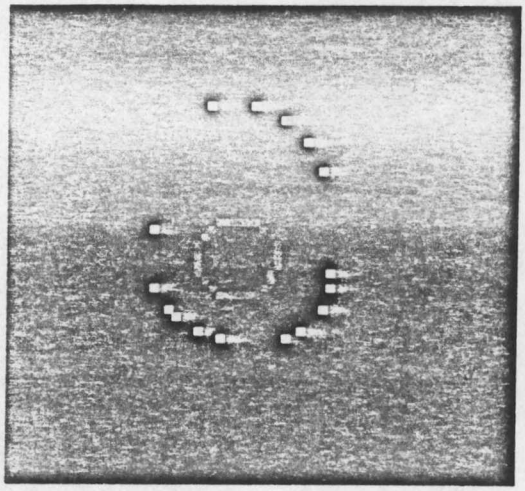
(d)

Figure 4.7. Tape.

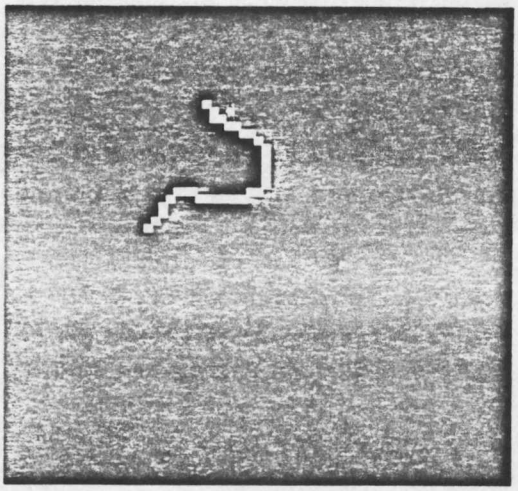
(a) gray level image
(b) histogram
(c) binary image
(d) external boundary
(e) border of the hole
(f) extreme points
(g) main concavity



(e)

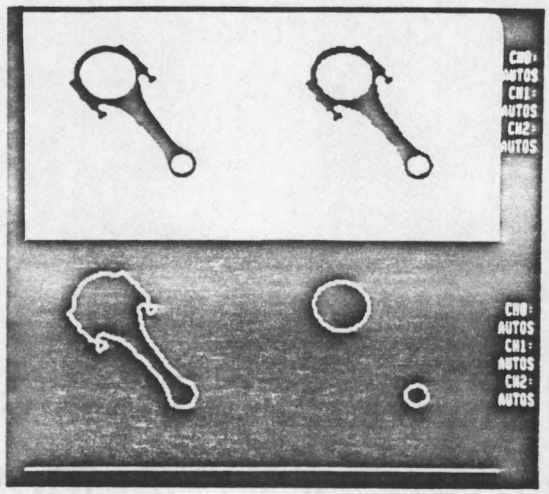


(f)

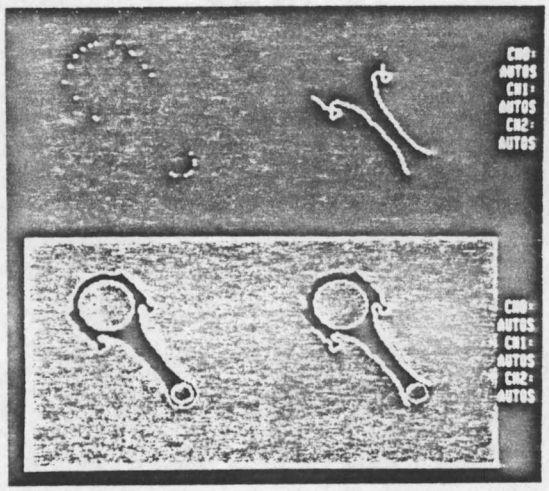


(g)

Figure 4.7 (continued)



(a)



(b)

Figure 4.8. Automobile part.

(a) Gray level image, binary image, external boundary. and border of the holes.
(b) Extreme points, main concavity, external and internal boundaries, and main concavities superimposed on top of the binary image.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

A hierarchical modular computer vision system for object recognition has been developed and implemented.

An approach was shown for combining statistical and structural methods for pattern recognition. Also, it was demonstrated that "bays" and "lakes" (concavities and holes, respectively) play an important role as shape descriptors in object recognition systems.

The general application of the system was demonstrated by its ability to handle a variety of objects. The actual Data Base ranges from automobile parts to characters.

Future research can improve this system even further. For segmentation, the method followed (global thresholding) works well for highly contrasted scenes. In less structured environments, more powerful schemes should be evaluated and developed. Simple segmentation methods sometimes produce unconnected blobs for a single object, thus forcing an increase in the complexity of the recognition stage. There exists a tradeoff between the amount of preprocessing and the complexity of later stages. Among the preprocessing operations that could be considered, "shrinks" and "expands" seem to be the most promising. Adequate combination of these two operations can help in eliminating small "holes" that might have originated during segmentation.

More research is necessary concerning both the description of the concavities and the description of the extreme points. A possible approach could combine the use of curvature and/or the use of curve fitting techniques.

The present system could also be improved by allowing concurrent processing of the module for convex hull extration and the module for component labeling and hole detection.

Still another area that could benefit from further research is the area of Data Base Management. The simple approaches developed for the present system could be replaced by relational approaches, thus allowing for logical operations and more structured handling.

LIST OF REFERENCES

LIST OF REFERENCES

Abe, K. and Sugita, N. [1982]. "Distances Between Strings of Symbols," Sixth Int. Conf. on Pattern Recognition, Munich, Germany, Oct. 19-22, pp. 172-174.

Batchelor, B. G. [1979]. "Using Concavity Trees for Shape Description," Computers and Digital Techniques, Vol. 2, No. 4, pp. 157-168.

Braunegg, D. J. and Gonzalez, R. C. [1984]. "An Approach to Industrial Computer Vision Using Syntactic/Semantic Learning Techniques," Seventh Int. Conf. on Pattern Recognition, Montreal, Canada, July 30-August 2, pp. 1366-1369.

Contreras, F. [1985]. "A Dynamic Threshold Algorithm Using Connectivity Analysis," M.Sc. Thesis, Department of Electrical Engineering, The University of Tennessee, Knoxville.

Foith, J. P., et al. [1981]. "Real-Time Processing of Binary Images for Industrial Applications," in Digital Image Processing Systems, L. Bolc and Z. Kulpa (Eds.), Berlin, Heidelberg, New York: Springer-Verlag, pp. 61-168.

Fu, K. S. [1982]. Syntactic Pattern Recognition and Applications, Prentice-Hall.

Fu, K. S. [1982]. "Attributed Grammars for Pattern Recognition - A General (Syntactic-Semantic) Approach," Proceedings of the P.R. and I.P. Confer., Las Vegas, Nevada, June 14-17, pp. 18-27.

Fu, K. S. [1983]. "A Step Towards Unification of Syntactic and Statistical Pattern Recognition," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-5, No. 2, pp. 200-205.

Gonzalez, R. C. [1980]. "Evaluation of the CHITRA Character Recognition System and Development of Feature Extraction Algorithms," NORDA Technical Note.

Gonzalez, R. C. [1983]. "Syntactic/Semantic Techniques for Feature Description and Character Recognition," NORDA Technical Note 185, January.

Gonzalez, R. C. and Safabakhsh, R. [1982]. "Computer Vision Techniques for Industrial Applications and Robot Control," Computer, Vol. 15, pp. 17-32.

Gonzalez, R. C. and Thomason, M. G. [1978]. Syntactic Pattern Recognition: An Introduction, Reading, MA: Addison-Wesley.

Gonzalez, R. C. and Wintz, P. [1977]. Digital Image Processing, Addison-Wesley, Reading, Massachusetts.

Graham, R. L. [1972]. "An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set," Information Processing Letters 1, pp. 132-133.

Kitchin, O. W. and Pugh, A. [1983]. "Processing of Binary Images," in Robot Vision, A. Pugh (Ed.), IFS (Publications) Ltd., U.K. and Springer-Verlag, Berlin, Heidelberg, New York, pp. 21-42.

Knuth, D. E. [1968]. "Semantics of Context-Free Languages," Mathematical Systems Theory, Vol. 2, No. 2, pp. 127-145.

Liu, H. and Fu, K. S. [1983]. "An Application of Syntactic Pattern Recognition to Seismic Discrimination," IEEE Transactions on Geoscience and Remote Sensing, Vol. GE-21, No. 2, pp. 125-132.

Pavlidis, T. and Ali, F. [1979]. "A Hierarchical Syntactic Shape Analyzer," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-1, No. 1, pp. 2-9.

Perkins, W. A.]1978]. "A Model-Based Vision System for Industrial Parts, IEEE Trans. Computers, Vol. C-27, pp. 126-143.

Rosenfeld, A.. [1969]. Picture Processing by Computer, Academic Press, New York, New York.

Rosenfeld, A., and Kak, A. D. [1982] Digital Picture Processing Vol. 2, Academic Press, New York, New York.

Thomason, M. G. [1982]. "Syntactic/Semantic Techniques in P. R.: A Survey," International Journal of Computer and Information Sciences, Vol. 11, No. 2, oo. 75-100.

Tsai, W. H. and Fu, K. S. [1980]. "A Syntactical-Statistical Approach to Recognition of Industrial Objects," IEEE Proc. Fifth Int. Conf. Pattern Recognition, Miami-Beach, Fl, Dec. 1-4, pp. 251-259.

Wagner, R. and Fisher, M. [1974]. "The String-to-String Correction Problem," Journal of the Association for Computer Machinery, Vol. 21, No. 1, pp. 168-174.

Warshall, S. [1972]. "A Theorem on Boolean Matrixes," Journal of the Association of Computing Machinery, pp. 11-12.

Yamamoto, K., Mori, T. and Mori, S. [1976]. "Machine Recognition of Handprinted Japanese Characters KATAKANA and Numerals - Topological Line Segment Method and Automatic Formation of Masks-," Trans. of the IECE Vol. 59-D, No. 5, pp. 414-421.

APPENDIXES

APPENDIX A

TRANSITIVE CLOSURE

This Appendix will review concepts that are relevant in the implementation of the algorithm COMPONENT.

An ordered pair (x,y) is a pair of elements with a given order associated with them.

A binary relation is a set of ordered pairs. If R represents a binary relation, and the pair (x,y) represents an element of R, we write xRy to denote that x is related to y by R.

If A and B are two sets, the cartesian product of A and B, denoted AxB, is the set containing all ordered pairs (x,y), such that x belongs to A and y belongs to B. Any subset of AxB is a binary relation referred to as a relation from A to B.

A binary relation in A is a relation from a set A to A. (It is a subset of the product AxA.) As an example, let A = $\{P_1, P_2, P_3\}$ represent a set of two-dimensional points arranged as follows:

P_1 P_2

P_3

Then, R = $\{(P_1, P_2), (P_2, P_1), (P_2, P_3), (P_3, P_2)\}$ represents the "4-connected" points in the set A.

A binary relation R over a set A is (a) reflexive if it contains (x,x) for every x in A, (b) symmetric if (x,y) in R implies (y,x), (c) transitive if the existence of (x,y) and (y,z) in R implies the existence of (x,z).

Relations can be used to examine connectivity of regions in binary images. Let S be a subset of a digital picture, then the points P_i and P_m of S are said to be connected if there exists a sequence of elements all in S, such that $P_i R P_{i+1}$, $P_{i+1} R P_{i+2}$,..., $P_{i+k} R P_m$. (Here R can be taken a 4-connectivity.)

For any point P in S, the set of P_k such that P and P_k are connected is called a connected component of S.

Definition $xR_n y$ means that there is a path of length n from x to y in R.

xR+y means that there is some path in R from x to y. R+ is sometimes called the connectivity relation for R.

If relation R does not possess the transitivity property, then the smallest relation R_1 in a set A (if it exists) that contains R and possesses that property is called the transitive closure of R.

R+ is the transitive closure of R.

For computation purposes it is useful to represent a relation in terms of a binary (or boolean) matrix:

$MR = [m_{ij}]$, where $m_{ij} = 1$ if $a_i R a_j$, otherwise 0.

An efficient algorithm for computing transitive closure is the Warshall's algorithm.

Warshall's Algorithm

Let MR = nxn binary matrix representing R

MR+ = transitive closure of R (matrix form)

Step 1. Let MR+ = MR

Step 2. For k = 1 to N

(a) For i = 1 to N

1. For j = 1 to N

MR+(i,j) = MR+(i,j) + (MR+(i,k) * MR+(k,j)

END OF ALGORITHM

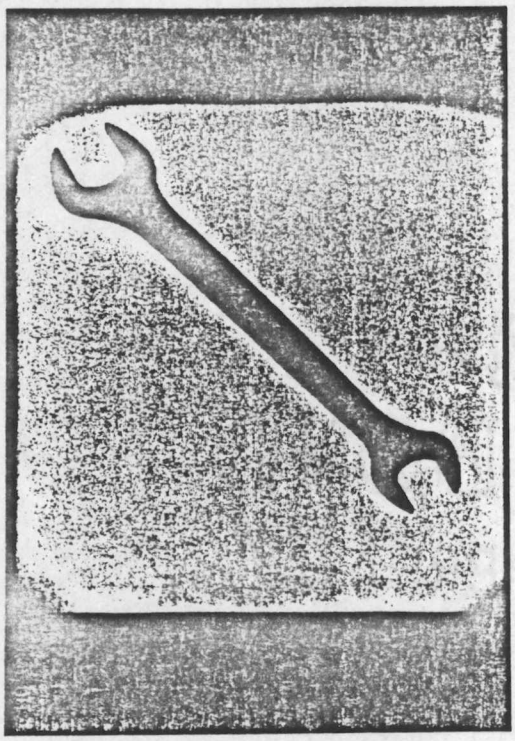
where,

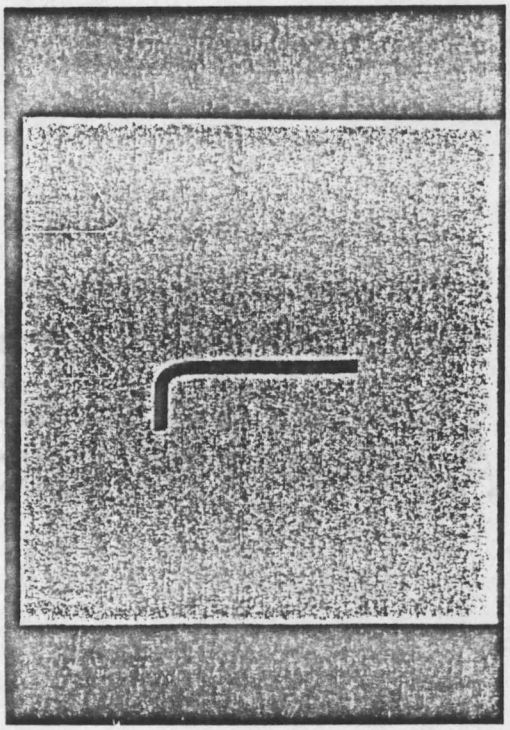
\+ = logical OR

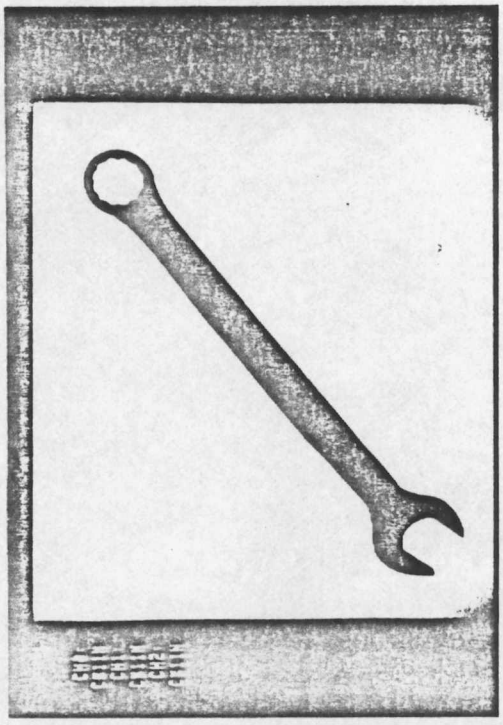
* = logical AND

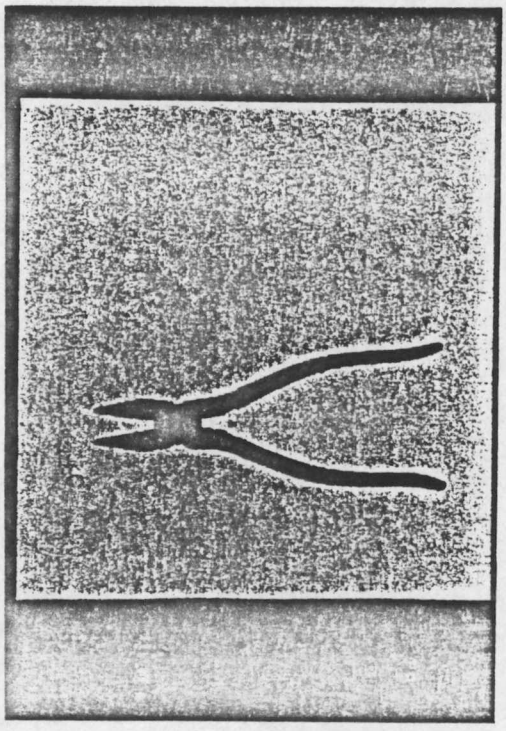
APPENDIX B

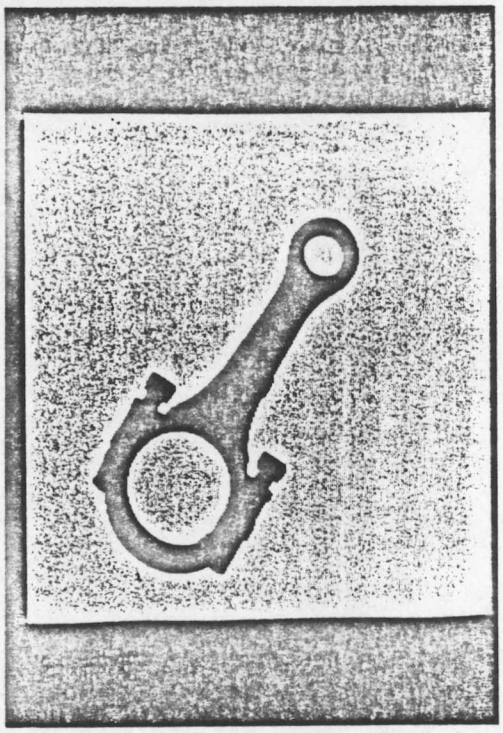
DATA BASE

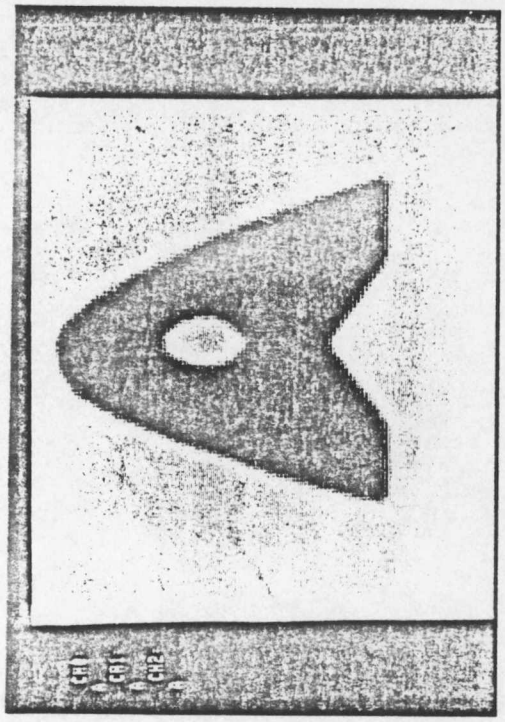


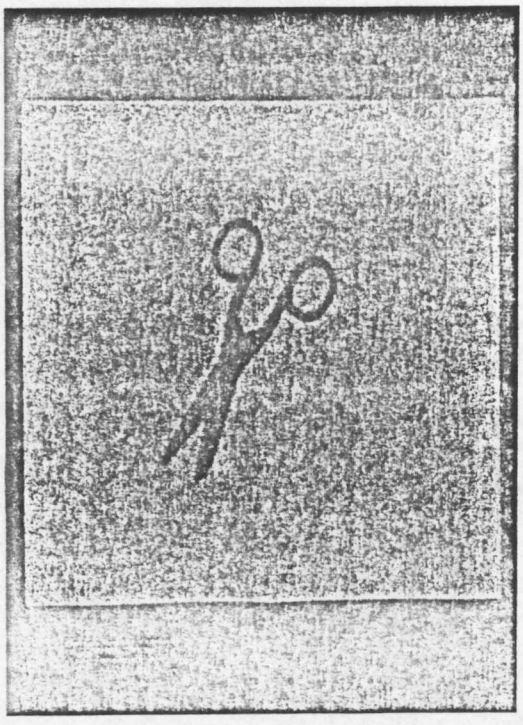


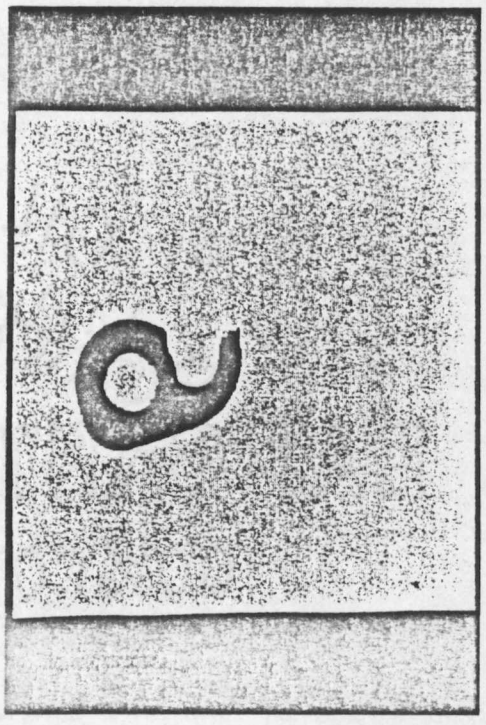












APPENDIX C

PROGRAM LISTINGS

```
C     PROGRAM : GLOBAL
C     PURPOSE : OBJECT RECOGNITION AND TRAINING
C     AUTHOR  : RENATO SALINAS
C
C     VARIABLES
C     BACK    : BACKGROUND INTENSITY LEVEL
C     OBJ     : OBJECT INTENSITY LEVEL
C     SIZE    : PICTURE SIZE ( ANY OF 64, 128 OR 256)
C     OPTION  : OPTION SELECTED BY USER
C     PBID    : PASTEBOARD IDENTIFICATION
C     DISPID  : VIRTUAL DISPLAY IDENTIFICATION
C     NCHAN   : MEMORY CHANNEL (0, 1, OR 2)
C     REPFLG  : FLAG RAISED IF NAME ALREADY ASSIGNED
C     SUCFLG  : FLAG RAISED IF OBJECT WAS RECOGNIZED
C     INITFLG : FLAG USED FOR INITIALIZATION PURPOSES
C     FNAME   : STORAGE FOR FILENAMES ( ALSO SAVNM)
C     ANSW    : STORAGE FOR USER ANSWERS
C     LINES   : ARRAY USED IN MAIN MENU

      INTEGER BACK,OBJ,SIZE
      INTEGER OPTION, PBID, DISPID
      INTEGER*2 NCHAN
      DIMENSION LAB(20)
      LOGICAL*1 REPFLG,SUCFLG,INITFLG
      CHARACTER*16 FNAME, SAVNM
      CHARACTER*1  ANSW
      CHARACTER*40 LINES(20)

C     INITIALIZE DEANZA
      CALL IP5INI
      CALL SETUP
      MEMSEL=7
      CALL ZEROM(MEMSEL)        ! CLEAR SCREEN

C     DISPLAY TITLE
      CALL CREAPB(PBID,LINES)

      INITFLG = .FALSE.

C     DISPLAY MAIN MENU
10    CALL MENU(OPTION,PBID,DISPID,LINES,INITFLG)
C
1     FORMAT(A16)
2     FORMAT(A1)
12    SUCFLG=.FALSE.
      TYPE *,' MAKE SURE THAT YOU HAVE AN IMAGE ".PIC" '
      TYPE *,' OR YOU HAVE A CAMERA SETUP AVAILABLE'
      TYPE *,' BEFORE PROCEEDING ... '
      TYPE *,' WANT TO PROCEED ? (Y/N) '
      ACCEPT 2,ANSW
      IF(ANSW.NE.'Y'.AND.ANSW.NE.'y') GO TO 10
```

```
C
C     WE MUST ASCERTAIN IF THE IMAGE WILL COME FROM
C     A FILE OR FROM THE CAMERA.
28    TYPE *,' please enter C (Camera)/ F (File) for source'
      ACCEPT 2,ANSW
      IF (ANSW.EQ.'F'.OR.ANSW.EQ.'f') THEN
       TYPE *,' OBJECT FILENAME [.PIC] ? '
       ACCEPT 1,FNAME
C    MAKE SURE THAT EXT.  IS ".PIC"
       KX=INDEX(FNAME,'.')
       IF(KX.LE.0) KX=INDEX(FNAME,' ')
       FNAME=FNAME(1:KX-1)//'.PIC'
C    SHOW OBJECT
       MEMSEL=7
       CALL ZEROM(MEMSEL)
       IQ=4                ! FOURTH QUADRANT
       NCHAN=1             ! GREEN
       CALL PICDISP(FNAME,1,IQ,NCHAN)
       CALL DELAYS(6.5)

       ELSE IF(ANSW.EQ.'C'.OR.ANSW.EQ.'c') THEN
        TYPE *,' OBJECT FILENAME [.PIC] ? '
        ACCEPT 1,FNAME
C    MAKE SURE THAT EXT.  IS ".PIC"
       KX=INDEX(FNAME,'.')
       IF(KX.LE.0) KX=INDEX(FNAME,' ')
       FNAME=FNAME(1:KX-1)//'.PIC'
              CALL LEECAM(FNAME,NCHAN)          ! DIGITIZE
              CALL KEEPIC(FNAME,NCHAN)          ! STORE
       ELSE
       GO TO 28
       END IF

       IF(OPTION.NE.4) GO TO 50
        ! CHECK IF NAME ALREADY ASSIGNED
        K=INDEX(FNAME,'.')
        IF(K.LE.0) K=INDEX(FNAME,' ')
        FNAME = FNAME(1:K-1)//'.PIC'
        SAVNM = FNAME ! SAVE NAME
              IF(REPFLG(FNAME)) THEN
                      TYPE *,' NAME ALREADY ASSIGNED'
                      GO TO 44
              END IF
        TYPE *,' NAME NOT USED, '
        ! CHECK IF OBJECT CAN BE RECOGNIZED
        ! IF SO, EITHER OBJECT IS IN D.B.  OR
        ! IMPROVED ALGORITHM REQUIRED.
        SUCFLG=.FALSE.
C
               CALL COLECT(FNAME,SUCFLG)
              IF(SUCFLG) THEN
```

```
                    TYPE *,' BUT, OBJECT ALREADY IN SYSTEM'
                    TYPE *,' OPERATOR CHECK !!! '
                    TYPE *,' TRAINING UNSUCCESSFUL'
*     TEMPORARY FILES ARE ELIMINATED NEXT
                    CALL DELFN(SAVNM)
                         ELSE
                    TYPE *,' AND OBJECT NOT IN SYSTEM'
*     KEEP FILES
*     ADD NAME TO D.B
                      CALL ADDFN(SAVNM)
      TYPE *,' TRAINING SUCCESFUL: NAME ADDED TO D.B. !! '
              END IF

44    WRITE(6,41)       !  GO TO 10
41    FORMAT(' continue TRAINING (Y/N) ?'$)
      CALL BELL(1)
      READ(5,2) ANSW
      IF(ANSW.EQ.'Y'.OR.ANSW.EQ.'y') GO TO 12
      GO TO 10 ! RETURN TO MAIN MENU

50    IF(OPTION.NE.5) THEN
              TYPE *,' ERROR IN OPTION'
              GO TO 10 ! RETURN TO MAIN MENU
              ELSE

       TYPE *,' entering RECOGNITION '
       K=INDEX(FNAME,'.')
       IF(K.LE.0) K=INDEX(FNAME,' ')
       FNAME = FNAME(1:K-1)//'.PIC'
       SAVNM = FNAME ! SAVE NAME
              IF(REPFLG(FNAME)) THEN
                      TYPE *,' NAME ALREADY ASSIGNED'
51    FORMAT(' continue RECOGNITION (Y/N) ?'$)
                      GO TO 55
              END IF
C    RECOGNITION WILL BE ATTEMPTED
      CALL COLECT(FNAME,SUCFLG)

      CALL DELFN(SAVNM)

55    WRITE(6,51)
      CALL BELL(1)
      READ(5,2) ANSW
      IF(ANSW.EQ.'Y'.OR.ANSW.EQ.'y') GO TO 12
      GO TO 10 ! RETURN TO MAIN MENU
C
      END IF
C
99999         STOP
      END
```

```
C    SUBPROGRAM COLECT

C     PURPOSE :  TO COLLECT IN ONE SUBROUTINE SEVERAL
C                ROUTINES USED FOR RECOGNITION AND TRAINING
C                IT DISPLAYS SELECTED STEPS DURING PROCESSING

      SUBROUTINE COLECT(FNAME,SUCFLG)
      INTEGER*2 NCHAN
      INTEGER IMAGEN(65536),IMAG(256,256)
      INTEGER BUFSAV(65536)
      INTEGER BUF2(65536)
      INTEGER IADJ(65536),IADJ2D(256,256)
      LOGICAL*1 IMCOM(65536)
      LOGICAL*1 REPFLG,SUCFLG
      INTEGER BACK,OBJ,SIZE
      INTEGER SIZESQ,WRK(512)
      INTEGER OPTION
      DIMENSION LAB(20)
      EQUIVALENCE (IMAGEN(1),IMAG(1,1))
      CHARACTER*16 FNAME,NAM
      CHARACTER*1  ANSW

      K=INDEX(FNAME,'.')
      IF(K.LE.0) K=INDEX(FNAME,' ')
      FNAME=FNAME(1:K-1)//'.PIC'

      OPEN(UNIT=1,NAME=FNAME,TYPE='OLD',ACCESS='DIRECT',
     $        READONLY,FORM='UNFORMATTED',RECORDSIZE=128)
      CALL IPICRD(1,WRK,1,1)
      CLOSE(UNIT=1)
      K=3.-(WRK(4)-76.)/4.
      SIZE=64*2**K
      TYPE *,' picture SIZE= ',SIZE
      SIZESQ=SIZE*SIZE

C     GRAY TO BINARY
      CALL GTOB(FNAME,SIZE,SIZESQ,OBJ,BACK,IADJ,IADJ2D,
     1  IMAGEN,BUFSAV,INIBLK,NBLOCK) !  ADJ.  ARRAYS

      LASTBL=NBLOCK+INIBLK-1

C     BOUNDARY FOLLOWING AND FILTERING
      CALL EDGE(FNAME,SIZE,OBJ,BACK,IMAGEN)

C     SHOW EXTERNAL BOUNDARY
      CALL DIBBORV(FNAME,'.BOR',SIZE,INIBLK,LASTBL)
      KI=INDEX(FNAME,'.')
      IF(KI.LE.0) KI=INDEX(FNAME,' ')
```

```
      NAM=FNAME(1:KI-1)//'.BRD'
      IQ=3                        ! THIRD QUADRANT
      NCHAN=0                     ! RED
      CALL PICDISP(NAM,1,IQ,NCHAN)
      CALL DELAYS(4.5)

C     PREPARE NOISE FREE IMAGE FOR NEXT MODULE
      CALL EXHI(FNAME,SIZE,OBJ,BACK,BUFSAV,BUF2,INIBLK,
     1 NBLOCK)

C     CONNECTIVITY ANALYSIS
      CALL COMP(FNAME,SIZE,OBJ,BACK,INIBLK,NBLOCK,BUF2,KARDL,
     1 LAB,IMCOM)

C     HOLE DETECTION
      CALL LABEL2(FNAME,SIZE,OBJ,BACK,KARDL,LAB,IMCOM)

C     SHOW THE BOUNDARY OF THE HOLES, IF ANY
      IF(KARDL.GT.0) THEN
              CALL DIBBORV(FNAME,'.CHL',SIZE,INIBLK,LASTBL)
              NAM=FNAME(1:KI-1)//'.BRD'
              IQ=3                ! THIRD QUADRANT
              NCHAN=2             ! BLUE
              CALL PICDISP(NAM,1,IQ,NCHAN)
      CALL DELAYS(4.5)
      END IF

C     COMPUTE CONVEX HULL AND PROCESS CONCAVITIES
      CALL MAIN4(FNAME)

C     SHOW THE EXTREME POINTS
              CALL DIBBORV(FNAME,'.HUL',SIZE,INIBLK,LASTBL)
              NAM=FNAME(1:KI-1)//'.BRD'
              IQ=3                ! THIRD QUADRANT
              NCHAN=1             ! GREEN
              CALL PICDISP(NAM,1,IQ,NCHAN)
      CALL DELAYS(6.5)

C     SHOW MAIN CONCAVITIES
              CALL DIBBORV(FNAME,'.CAV',SIZE,INIBLK,LASTBL)
              NAM=FNAME(1:KI-1)//'.BRD'
              IQ=3                ! THIRD QUADRANT
              NCHAN=1             ! GREEN
              CALL PICDISP(NAM,1,IQ,NCHAN)
      CALL DELAYS(4.5)

C     ATTEMPT RECOGNITION
      CALL RECON(FNAME,SUCFLG)

      RETURN
      END
```

```
C    SUBPROGRAM ADDFN

C     PURPOSE : ADD FILENAME TO DATA BASE
C
C     VARIABLES
C     FNAME    : WORKING STORAGE FOR FILENAMES
C     NEWNM    : FILENAME TO ADD

      SUBROUTINE ADDFN(NEWNM)
      CHARACTER*16 FNAME,NEWNM
      K=INDEX(NEWNM,'.')
      IF(K.LE.0) K=INDEX(NEWNM,' ')
      NEWNM=NEWNM(1:K-1)
C    OPEN FILE THAT CONTAIN NAMES
      OPEN(UNIT=1,NAME='MAY13.DAT',TYPE='OLD',
     $             FORM='FORMATTED')
      DO 10 WHILE (.TRUE.)
      READ(1,1,END=999) FNAME
1     FORMAT(A16)
10    END DO
999   WRITE(1,1) NEWNM
      CLOSE(UNIT=1)
      RETURN
      END
```

```
C    SUBPROGRAM REPFLG
C
C     PURPOSE :  INDICATES IF A GIVEN NAME IS ALREADY
C                IN DATA BASE

C     VARIABLES
C     FNAME    : FLAG THAT INDICATES STATUS OF FILENAME
C     NAM      : FILENAME TO TEST FOR REPETITION

      LOGICAL*1 FUNCTION REPFLG(NAM)
      CHARACTER*16 FNAME, NAM
      K=INDEX(NAM, '.')
      IF(K.LE.0) K=INDEX(NAM, ' ')
      NAM = NAM(1:K-1)
      REPFLG=.FALSE.  ! ASSUME NOT REPEATED FIRST
      OPEN(UNIT=1,NAME='MAY13.DAT',TYPE='OLD',
      $              READONLY,FORM='FORMATTED')
      DO 10 WHILE (.TRUE.)
      READ(1,1,END=999) FNAME
              IF(NAM.EQ.FNAME) THEN
                      REPFLG=.TRUE.
                      GO TO 999
              END IF
10    END DO
1     FORMAT(A16)
999   CLOSE(UNIT=1)
      RETURN
      END
```

```
C    SUBPROGRAM LEECAM

C     PURPOSE : TO DIGITIZE OBJECT UNDER OBSERVATION.

C     VARIABLES
C     FNAME    : FILENAME
C     MEMSEL   : USED TO SELECT MEMORY CHANNELS
C     NCHAN    : SPECIFIC CHANNEL
C     POWER    : CHANNEL AS REQUIRED BY DIGITIZING ROUTINE

      SUBROUTINE LEECAM(FNAME, NCHAN)
      INTEGER*2 NCHAN
      INTEGER*2 POWER
      CHARACTER*16 FNAME

      NCHAN=1
      MEMSEL=7                    ! TO CLEAR ALL CHANNELS
      CALL ZEROM(MEMSEL)
      POWER=2**NCHAN
      CALL SNAPX(POWER, 1)        ! SAMPLE OBJECT TO MEMORY

      RETURN
      END
```

```
C     SUBPROGRAM KEEPIC

C     PURPOSE :  TO STORE TO MASS MEMORY IMAGE DIGITIZED
C                BY ROUTINE LEECAM
C
C     VARIABLES
C     NCHAN    :  CHANNEL WHERE THE IMAGE IS READ FROM.
C     FNAME    :  FILENAME ASSIGNED TO IMAGE
C     TITULO   :  TITLE ASSIGNED TO IMAGE
C     SIZE     :  V, S, OR M (VERY SMALL, SMALL, MEDIUM)
C     ANSW     :  WORKING STORAGE FOR USER'S ANSWERS.

      SUBROUTINE KEEPIC(FNAME,NCHAN)
      INTEGER*2 NCHAN
      CHARACTER*16 FNAME,TITULO
      LOGICAL*1 SIZE,TITLE(8)
      CHARACTER*1 ANSW
C
10    TYPE *,' PLEASE ENTER SIZE (V,S,M):'
      ACCEPT 2,ANSW
2     FORMAT(A1)
      IF(ANSW.EQ.'V'.OR.ANSW.EQ.'v') THEN
               ANSW='V'
               CONTINUE
      ELSE IF (ANSW.EQ.'S'.OR.ANSW.EQ.'s') THEN
       ANSW='S'
       CONTINUE
      ELSE IF (ANSW.EQ.'M'.OR.ANSW.EQ.'m') THEN
       ANSW='M'
       CONTINUE
      ELSE IF (ANSW.EQ.'L'.OR.ANSW.EQ.'l') THEN
       ANSW='L'
       TYPE *,' Large Size Not Supported Yet'
       GO TO 10
      ELSE
       GO TO 10
      END IF
      SIZE=ICHAR(ANSW)
      TYPE *,' please enter TITLE (up to 8 char.):'
      ACCEPT 1,TITULO
1     FORMAT(A16)
      K=INDEX(TITULO,' ')
      K=K-1
      IF(K.GE.9) K=8
      DO I=1,K
      ANSW=TITULO(I:I)
      TITLE(I)=ICHAR(ANSW)
      END DO
```

```
CALL PICSTORE(FNAME,SIZE,TITLE,NCHAN)

TYPE *,' picture stored !!  '

RETURN
END
```

```
C    SUBPROGRAM BELL

C     PURPOSE : TO RING BELL "N" TIMES FOR PROMTING.

C     VARIABLES
C     BIP       : STORES ASCII CODE FOR "BELL"
C     N         : NUMBER OF RINGS

      SUBROUTINE BELL(N)
      LOGICAL*1 BIP
      BIP=7
      IF(N.LE.0) N=1
      DO I=1,N
      WRITE(6,1) BIP
      END DO
1     FORMAT(1X,1A1)

      RETURN
      END
```

```
C     SUBPROGRAM CREAPB

C     PURPOSE : TO DISPLAY IN TERMINAL TITLE AND
C               TO GENERATE PASTEBOARD ID
C     INPUTS  : NONE
C     OUTPUTS : PBID, LINES

C     VARIABLES
C     PBID     : PASTEBOARD IDENTIFICATION NUMBER
C     DTITLEID: ID NUMBER FOR TITLE DISPLAY
C     LINES    : CHARACTER ARRAY. STORES LINES OF TEXT
C     LINEST   : CHARACTER ARRAY. STORES LINES OF TEXT
C     STATUS   : STATUS OF LAST SMG$ PROCEDURE CALL
C     INIROW   : INITIAL ROW FOR DISPLAYING TEXT
C     LASTROW : LAST ROW FOR TEXT
C     IL       : POINTER TO ARRAY "LINEST"

C     IT USES SMG$ PROCEDURES (RUN TIME LIBRARY ROUTINES)
C     DATE     : 5-10-85
C
C        123.... column number
C     1
C     2    Pasteboard coord. system
C     3
C     .
C     .
C     row
C     number
C
      SUBROUTINE CREAPB(PBID,LINES)
      IMPLICIT INTEGER (S)
C
      INTEGER PBID,PBROW,PBCOL,DTITLEID
      INTEGER PASTEROW,PASTECOL,TLROW,TLCOL
     1 ,BRROW,BRCOL,STATUS
      CHARACTER*40 LINES(20),LINEST(40)
C
C    READ TITLE INTO ARRAY "LINEST"
      CALL RDTITLE(LINEST)
C     .
C    READ MENUTXT INTO ARRAY "LINES"
      CALL RDMENU(LINES)
C
      NCOLS=80 ! DIMENSIONS OF VIRTUAL DISPLAY
      NROWS=24
C
C
C   CREATE PASTEBOARD
C   DEFAULTS TO SYS$OUTPUT
C
      STATUS=SMG$CREATE_PASTEBOARD(%ref(PBID),,%ref(PBROW),
```

```
      1 %ref(PBCOL))
      IF (.NOT. STATUS) CALL LIB$STOP(%val(STATUS))

C     CREATE VIRTUAL DISPLAY FOR TITLE
      STATUS=SMG$CREATE_VIRTUAL_DISPLAY(%ref(NROWS),NCOLS,
      1 %ref(DTITLEID))
      IF (.NOT.STATUS) CALL LIB$STOP(%val(STATUS))
C
      TLROW=5 ! DEFINE RECTANGLE FOR TITLE
      TLCOL=9
      BRROW=10
      BRCOL=51
      STATUS=SMG$DRAW_RECTANGLE(%ref(DTITLEID),%ref(TLROW),
      1 %ref(TLCOL),%ref(BRROW),%ref(BRCOL))

      PASTEROW=1
      PASTECOL=1

C     SHOW TITLE
      ICOLN=10
      INIROW=6
      LASTROW=8

      DO LINEN = INIROW,LASTROW
      IL=LINEN-INIROW+1
      STATUS=SMG$PUT_CHARS(%ref(DTITLEID),%descr(LINEST(IL)),
      1 LINEN,ICOLN)
      IF (.NOT.STATUS) CALL LIB$STOP(%val(STATUS))
      END DO

      CALL SMG$PASTE_VIRTUAL_DISPLAY(%ref(DTITLEID),%ref(PBID),
      1 %ref(PASTEROW),%ref(PASTECOL))

      CALL DELAYS(2.0)          ! ALLOW SOME TIME TO READ

      STATUS=SMG$UNPASTE_VIRTUAL_DISPLAY(DTITLEID,PBID)

      STATUS=SMG$DELETE_VIRTUAL_DISPLAY(DTITLEID)

      RETURN
      END
```

```
C     SUBPROGRAM MENU
C
C     PURPOSE          : TO DISPLAY MAIN MENU AND TO ACCEPT
C                        USER'S OPTION.
C     DATE CREATION    : MAY 4, 1985
C     PROGRAMMER       : R. SALINAS

C     VARIABLES
C     PBID     : PASTEBOARD ID NUMBER
C     DID      : VIRTUAL DISPLAY ID NUMBER
C     OPTION   : USER SELECTED OPTION
C     LINES    : CHARACTER ARRAY. CONTAINS TEXT TO DISPLAY
C     ONESHOT  : FLAG
C
      SUBROUTINE MENU(OPTION,PBID,DID,LINES,ONESHOT)
      INTEGER OPTION,PBID,DID
      CHARACTER*40 LINES(20)
      LOGICAL*1 ONESHOT

      TYPE *,' '
5     CALL OPTIONS(OPTION,PBID,DID,LINES,ONESHOT)

C     ON OPTION CALL :
      GO TO (10,20,30,40,50,60),OPTION

10    CALL PANTAYA ! SHOW OBJECTS
      GO TO 5

20    CALL LISTNM  ! LIST FILENAMES IN D.B.
      GO TO 5

30    CALL MNTARCH ! D. B. MANAGEMENT
      GO TO 5

40    TYPE *,' TRAINING WAS SOLICITED'
      RETURN

50    TYPE *,' RECOGNITION REQUIRED'
      RETURN

60    STOP     ! EXIT REQUESTED: RETURN TO VMS
      END
```

```
C     SUBPROGRAM LISTNM

C     PURPOSE : TO LIST IN TERMINAL FILENAMES

      SUBROUTINE LISTNM
      CHARACTER*16 FNAME
      CHARACTER*1 ANSW

      ICON=0
      TYPE *,' NUMBER      FILENAME '
      TYPE *,' ------    ----------------'
      OPEN(UNIT=1,NAME='may13.DAT',TYPE='OLD',
     $        READONLY,FORM='FORMATTED')

      DO 10 WHILE (.TRUE.)
      READ(1,1,END=99) FNAME
1     FORMAT(A16)
      ICON=ICON+1
      TYPE 3,ICON,FNAME
3     FORMAT('   ',I3,6X,A16)
10    CONTINUE

99    CLOSE(UNIT=1)
      TYPE *,' PLEASE PRESS RETURN TO CONTINUE ...'
      ACCEPT 2,ANSW
2     FORMAT(A1)
      RETURN
      END
```

```
C     SUBPROGRAM OPTIONS

C     PURPOSE :  TO DISPLAY MAIN MENU AND ACCEPT USER
C                INPUT.
C
C     IT USES SMG$ PROCEDURES (RUN TIME LIBRARY ROUTINES)
C     DATE    : 5-7-85
C
C        123.... column number
C     1
C     2    Pasteboard coord. system
C     3
C     .
C     .
C     row
C     number
C
      SUBROUTINE OPTIONS(OPTION,PBID,DISPLYID,LINES,MEMFLG)
      IMPLICIT INTEGER (S)
C
      INTEGER PBID,PBROW,PBCOL,DISPLYID
      INTEGER PASTEROW,PASTECOL,TLROW,TLCOL
      1 BRROW,BRCOL,STATUS
      INTEGER DROW,DCOL
      INTEGER ESCFLG
      INTEGER OPTION
      INTEGER*2 TCODE
      CHARACTER*16 NAME
      CHARACTER*1  ANSW
      CHARACTER*40 LINES(20),LINEST(40)
      LOGICAL*1 MEMFLG
C
      NCOLS=80 ! DIMENSIONS OF VIRTUAL DISPLAY
      NROWS=24
C
C
C    PASTEBOARD WAS CREATED PREVIOUSLY
C    DEFAULTS TO SYS$OUTPUT
C
       IF (MEMFLG) THEN
                STATUS= SMG$REPAINT_SCREEN(PBID)
                STATUS=SMG$CREATE_VIRTUAL_KEYBOARD(KBDID)
                IF (.NOT.STATUS) CALL LIB$STOP(%val(STATUS))
                GO TO 11
       END IF
C
      STATUS=SMG$CREATE_VIRTUAL_DISPLAY(%ref(NROWS),
      1 NCOLS,%ref(DISPLYID))
```

```
      IF (.NOT.STATUS) CALL LIB$STOP(%val(STATUS))

C   CREATE VIRTUAL KEYBOARD
C   DEFAULTS TO  SYS$INPUT

      STATUS=SMG$CREATE_VIRTUAL_KEYBOARD(KBDID)
      IF (.NOT.STATUS) CALL LIB$STOP(%val(STATUS))
C
C
11    TLROW=5 ! DEFINE RECTANGLE FOR MENU
      TLCOL=9
      BRROW=17
      BRCOL=51
      STATUS=SMG$DRAW_RECTANGLE(%ref(DISPLYID),%ref(TLROW),
      1 %ref(TLCOL),%ref(BRROW),%ref(BRCOL))
C
C   PASTEROW SPECIFIES ROW IN PASTEBOARD THAT IS GOING
C   TO CONTAIN FIRST ROW OF A GIVEN VIRTUAL DISPLAY.

      PASTEROW=1
      PASTECOL=1

C   SHOW MENU
      ICOLN=10
      INIROW=6
      LASTROW=16
      DO LINEN = INIROW,LASTROW
      IL=LINEN-INIROW+1
      STATUS=SMG$PUT_CHARS(%ref(DISPLYID),%descr(LINES(IL)),
      1 LINEN,ICOLN)
      IF (.NOT.STATUS) CALL LIB$STOP(%val(STATUS))
      END DO

      CALL SMG$PASTE_VIRTUAL_DISPLAY(%ref(DISPLYID),
      1 %ref(PBID),%ref(PASTEROW),%ref(PASTECOL))
C
      NTIMES=1
      STATUS=SMG$RING_BELL(%ref(DISPLYID),%ref(NTIMES))
      IF (.NOT.STATUS) CALL LIB$STOP(%val(STATUS))
C
      PBROW=13
      PBCOL=18
22    CALL SMG$SET_PHYSICAL_CURSOR(PBID,PBROW,PBCOL)
      DROW=16
      DCOL=32
33    CALL SMG$SET_CURSOR_ABS(DISPLYID,DROW,DCOL)
C
      ESCFLG=0
      STATUS=SMG$ALLOW_ESCAPE(DISPLYID,ESCFLG)
      STATUS=SMG$READ_STRING(KBDID,%descr(ANSW),
      1,,,,,,TCODE,DISPLYID)
```

```
C   CHECK IF VALID OPTION
      OPTION=ICHAR(ANSW)
      IF(OPTION.LT.49.OR.OPTION.GT.54) GO TO 33
      OPTION=OPTION-48

      IF (TCODE.EQ.274) THEN
               DROW =DROW-1
               PBROW=PBROW-1
               ELSE IF (TCODE.EQ.275) THEN
               DROW =DROW+1
               PBROW=PBROW+1
               ELSE IF (TCODE.EQ.276) THEN
               DCOL =DCOL-1
               PBCOL=PBCOL-1
               ELSE IF (TCODE.EQ.277) THEN
               DCOL =DCOL+1
               PBCOL=PBCOL+1
               ELSE IF (TCODE.NE.13) THEN
C   ALLOW UP TO 10 READ PER SECOND FROM KBD
               CALL DELAYS(0.1)
               GO TO 22
               ELSE
               GO TO 44
      END IF
44    CONTINUE
C
      NTIMES=2
      CALL SMG$RING_BELL(%ref(DISPLYID),%ref(NTIMES))
      CALL SMG$SAVE_PHYSICAL_SCREEN(%ref(PBID),%ref(DISPLYID))

      IF(IVAL.LT.32) GO TO 5

      TYPE 2,' character was: ',ANSW
5     CONTINUE
2     FORMAT(A1)
C
      STATUS=SMG$DELETE_VIRTUAL_KEYBOARD(KBDID)
      IF (.NOT.STATUS) CALL LIB$STOP(%val(STATUS))
      CONTINUE

      MEMFLG=.TRUE.

      RETURN
      END
```

```
C     SUBPROGRAM RDTITLE

C     PURPOSE : READ LINES OF TEXT FROM FILE
C               INTO CHARACTER ARRAY
C
C     VARIABLES
C     LINES   : CHARACTER ARRAY
C     LINETXT : CHARACTER VARIABLE FOR INTERMEDIATE
C               STEP.

      SUBROUTINE RDTITLE(LINES)
      CHARACTER*40 LINETXT,LINES(20)

      OPEN(UNIT=1,NAME='TITULO1.TXT',TYPE='OLD',
     1     READONLY,FORM='FORMATTED')
      I=0
      DO WHILE (.TRUE.)
      READ(1,3,END=99) LINETXT
3     FORMAT(A40)
      I=I+1
      LINES(I)=LINETXT
      END DO

99    CLOSE(UNIT=1)

      RETURN
      END
```

```
C     SUBPROGRAM RDMENU

C     PURPOSE : READ FILE CONTAINING MAIN MENU
C               INTO CHARACTER ARRAY.
C     DATE    : MAY 10, 1985

      SUBROUTINE RDMENU(LINES)
      CHARACTER *40 LINETXT,LINES(20)

      OPEN(UNIT=1,NAME='MENU.TXT',TYPE='OLD',
     1     READONLY,FORM='FORMATTED')
      I=0
      DO WHILE (.TRUE.)
      READ(1,3,END=99) LINETXT
      I=I+1
      LINES(I)=LINETXT
3     FORMAT(A40)
      END DO
99    CLOSE(UNIT=1)
      RETURN
      END
```

```
C     SUBPROGRAM MNTARCH

C     PURPOSE : DATA BASE MANAGER
C
C     VARIABLES
C     FNAME    : STORAGE FOR ONE FILENAME
C     NEWFN    : NEW FILENAME (FOR RENAMING)
C     OLDFN    : OLD FILENAME (FOR RENAMING)
C     CMD      : COMMAND
C     NAMARR   : CHARACTER ARRAY FOR STORING
C                FILENAMES.

      SUBROUTINE MNTARCH
      INTEGER OPTION
      CHARACTER*16 FNAME, OLDFN, NEWFN, NAM
      CHARACTER*16 CMD*64, NAMARR(100)
      LOGICAL*1 FLG

10    TYPE *,'     ** DBM OPTIONS ** '
      TYPE *,' 1 - LIST OF FILENAMES IN DB'
      TYPE *,' 2 - ADD NEW NAME (FOR TESTING PURPOSES) '
      TYPE *,' 3 - ELIMINATE NAME'
      TYPE *,' 4 - RENAME FILE'
      TYPE *,' 5 - QUIT SUBPROGRAM'
      TYPE *,' '

20    ACCEPT *,IOPER
      IF((IOPER.LT.1).OR.(IOPER.GT.5)) GO TO 20
      GO TO (1,2,3,4,5),IOPER
C
1     OPEN(UNIT=1,NAME='MAY13.DAT',TYPE='OLD',
      $        FORM='FORMATTED')
      TYPE *,'  NUMBER        FILENAME '
      TYPE *,'  ------    ----------------'
      ICON =0
11    READ(1,9,END=111) FNAME
      ICON=ICON+1
      TYPE 12,ICON,FNAME
12    FORMAT('   ',I3,7X,A16)
      GO TO 11
111   TYPE *,' '
      CLOSE(UNIT=1)
      GO TO 10
C
2     TYPE *,' FILENAME TO ADD'
      ACCEPT 9,NAM
9     FORMAT(A16)
      OPEN(UNIT=1,NAME='MAY13.DAT',TYPE='OLD',
      $        FORM='FORMATTED')
55    READ(1,9,END=22) FNAME
      IF(FNAME.EQ.NAM) THEN
```

```
              TYPE *,' NAME ALREADY ASSIGNED'
              CLOSE(UNIT=1)
              GO TO 10
              ELSE
              GO TO 55
      END IF
22    WRITE(1,9) NAM
      CLOSE(UNIT=1)
      GO TO 10
C
3     TYPE *,' FILENAME TO DELETE'
      ACCEPT 9, NAM
      FLG=.FALSE.
      I=1
      OPEN(UNIT=1,NAME='MAY13.DAT',TYPE='OLD',
      $        FORM='FORMATTED')
      DO 33 WHILE (.TRUE.)
      READ(1,9,END=333) NAMARR(I)
      I=I+1
              IF(NAM.EQ.NAMARR(I-1)) THEN
              IPOS=I-1
              FLG=.TRUE.
C             CALL COMPRESS    ! FORMERLY USED TO
                               ! SUPRESS BLANKS
              CALL DELFN(NAM) ! DELETE ASSOC. FILES
              END IF
33    END DO
333   CLOSE(UNIT=1)
      IF(FLG) THEN
      OPEN(UNIT=1,NAME='MAY13.DAT',TYPE='OLD',
      $        FORM='FORMATTED')
      DO J=1,IPOS-1
      WRITE(1,9) NAMARR(J)
      END DO
      DO J =IPOS+1,I-1
      WRITE(1,9) NAMARR(J)
      END DO
      TYPE *,' FILENAME ',NAM,' DELETED !! '
      CLOSE(UNIT=1)
      ELSE
      TYPE *, ' UNEXISTENT FILE'
      END IF
      GO TO 10


4     TYPE *,' FILENAME TO RENAME ?'
      ACCEPT 9,OLDFN
      TYPE *,' NEW FILENAME ?'
      ACCEPT 9,NEWFN
      IF(OLDFN.EQ.NEWFN) GO TO 10 ! WE ARE DONE
      FLG=.FALSE.
```

```
      OPEN(UNIT=1,NAME='MAY13.DAT',TYPE='OLD',
     $        FORM='FORMATTED')
      DO 44 WHILE (.TRUE.)
      READ(1,9,END=440) FNAME
      IF(OLDFN.EQ.FNAME) FLG=.TRUE.
      IF(NEWFN.EQ.FNAME) THEN
              TYPE *,' (NEW) FILENAME ALREADY EXISTS'
              TYPE *,' RENAME UNSUCCESSFUL'
              CLOSE(UNIT=1)
      END IF
44    CONTINUE
440   CLOSE(UNIT=1)
C     CONDITIONS EXIST FOR RENAME
      IF(NOT FLG) THEN
              TYPE *,' UNEXISTENT FILENAME'
              GO TO 10
      END IF
      CALL RENM(OLDFN,NEWFN)
      TYPE *,' SUCCESSFUL RENAME'
      GO TO 10


C     EXIT PROGRAM
5     RETURN
      END
```

```
C     SUBPROGRAM COMPRESS

C     PURPOSE : TO ELIMINATE BLANK LINES IN FILE

      SUBROUTINE COMPRESS
      CHARACTER*16 FNAME, NAMARR(100)

4     OPEN(UNIT=1,NAME='MAY13.DAT',TYPE='OLD',
      $        FORM='FORMATTED')
      I = 0 ! COUNTER OF NON-BLANKS
44    READ(1,9,END=444) FNAME
9     FORMAT(A16)
      IF(FNAME.EQ.' ') THEN
               GO TO 44
               ELSE
               I=I+1
               NAMARR(I)=FNAME
               GO TO 44
      END IF
444   CLOSE(UNIT=1)
      OPEN(UNIT=1,NAME='MAY13.DAT',TYPE='OLD',
      $        FORM='FORMATTED')
      DO 400 J = 1, I
400   WRITE(1,9) NAMARR(J)
      CLOSE(UNIT=1)

      RETURN
      END
```

```
C     SUBPROGRAM RENM

C     PURPOSE : TO RENAME A FILE

      SUBROUTINE RENM(OLDFN,NEWFN)
      CHARACTER*16 FNAME,OLDFN,NEWFN,  CMD*64
      CHARACTER*4 EXTEN(4)
      LOGICAL*1 FLG

      DATA EXTEN/'.DT2','.DT3','.DT4','.PER'/
C     THESE ARE MANDATORY FILES FOR RENAMING.

C     CHANGE NAME IN DB FIRST
      OPEN(UNIT=1,NAME='MAY13.DAT',TYPE='OLD',
     $        FORM='FORMATTED')
C     FLG=.TRUE.
      DO WHILE (.TRUE.)
      READ(1,1,END=99) FNAME
               IF(FNAME.EQ.OLDFN) THEN
                        BACKSPACE 1
                        WRITE(1,1) NEWFN
                        GO TO 99
               END IF
      END DO
99    CLOSE(UNIT=1)

1     FORMAT(A16)
C     NOW RENAME EXTRA FILES
      K1=INDEX(OLDFN,'.')
      IF(K1.LE.0) K1=INDEX(OLDFN,' ')
      K2=INDEX(NEWFN,'.')
      IF(K2.LE.0) K2=INDEX(NEWFN,' ')
      DO 20 I = 1,4
      OLDFN=OLDFN(1:K1-1)//EXTEN(I)
      NEWFN=NEWFN(1:K2-1)//EXTEN(I)
      CMD='RENAME '//OLDFN//' '//NEWFN

      CALL LIB$SPAWN(CMD)

20    CONTINUE

      RETURN
      END
```

```
C     SUBPROGRAM DELFN

C     PURPOSE : TO DELETE FILES FROM DISK

      SUBROUTINE DELFN(FNAME)
      CHARACTER*16 FNAME, SAVNAM,DELFILE
      CHARACTER*4 EXTEN(7)
      LOGICAL*1 EX
      INTEGER STATUS
      DATA EXTEN/'.BOX','.DT1','.DT2','.DT3','.DT4','.PER'/

      K=INDEX(FNAME,'.')
      IF(K.LE.0) K=INDEX(FNAME,' ')
1     FORMAT(' deleting ',A16)

      DELFILE=FNAME(1:K-1)//'.BOX'
C     TYPE 1,DELFILE
      STATUS=LIB$DELETE_FILE(%descr(DELFILE))
      IF(.NOT. STATUS) CALL LIB$STOP(%val(STATUS))

      DELFILE=FNAME(1:K-1)//'.DT1'
      STATUS=LIB$DELETE_FILE(%descr(DELFILE))
      IF(.NOT. STATUS) CALL LIB$STOP(%val(STATUS))

      DELFILE=FNAME(1:K-1)//'.DT2'
      STATUS=LIB$DELETE_FILE(%descr(DELFILE))
      IF(.NOT. STATUS) CALL LIB$STOP(%val(STATUS))

      DELFILE=FNAME(1:K-1)//'.DT3'
      STATUS=LIB$DELETE_FILE(%descr(DELFILE))
      IF(.NOT. STATUS) CALL LIB$STOP(%val(STATUS))

      DELFILE=FNAME(1:K-1)//'.DT4'
      STATUS=LIB$DELETE_FILE(%descr(DELFILE))
      IF(.NOT. STATUS) CALL LIB$STOP(%val(STATUS))

      DELFILE=FNAME(1:K-1)//'.PER'
      STATUS=LIB$DELETE_FILE(%descr(DELFILE))
      IF(.NOT. STATUS) CALL LIB$STOP(%val(STATUS))

      DELFILE=FNAME(1:K-1)//'.BIN'
      STATUS=LIB$DELETE_FILE(%descr(DELFILE))
      IF(.NOT. STATUS) CALL LIB$STOP(%val(STATUS))

      DELFILE=FNAME(1:K-1)//'.BOR'
      STATUS=LIB$DELETE_FILE(%descr(DELFILE))
      IF(.NOT. STATUS) CALL LIB$STOP(%val(STATUS))

      DELFILE=FNAME(1:K-1)//'.BOX'
      STATUS=LIB$DELETE_FILE(%descr(DELFILE))
      IF(.NOT. STATUS) CALL LIB$STOP(%val(STATUS))
```

```
DELFILE=FNAME(1:K-1)//'.BRD'
STATUS=LIB$DELETE_FILE(%descr(DELFILE))
IF(.NOT. STATUS) CALL LIB$STOP(%val(STATUS))

DELFILE=FNAME(1:K-1)//'.CAV'
STATUS=LIB$DELETE_FILE(%descr(DELFILE))
IF(.NOT. STATUS) CALL LIB$STOP(%val(STATUS))

DELFILE=FNAME(1:K-1)//'.HUL'
STATUS=LIB$DELETE_FILE(%descr(DELFILE))
IF(.NOT. STATUS) CALL LIB$STOP(%val(STATUS))

DELFILE=FNAME(1:K-1)//'.LTB'
STATUS=LIB$DELETE_FILE(%descr(DELFILE))
IF(.NOT. STATUS) CALL LIB$STOP(%val(STATUS))

RETURN
END
```

```
C     SUBPROGRAM PANTAYA

C     PURPOSE : SHOWS ON THE DISPLAY OBJECTS IN DATA BASE
C     DATE    : 04-22-85
C     UPDATE  : 06-02-85
C     COMMENTS:
C              IT ASSUMES THAT LINE RATE SWITCH IS IN 559,
C              OTHERWISE, YOU MAY NOT BE ABLE TO SEE A SMALL
C              AMOUNT OF LINES AT THE TOP OF THE SCREEN.
C              THE PROGRAM STILL EXECUTES CORRECTLY IN ANY
C              EVENT.

C     VARIABLES
C     IHALF   : SELECTS FULL SCREEN OR QUADRANT MODE
C     IQUAD   : SELECTS QUADRANT (1-5)
C     NCHAN   : SELECTS MEMORY PLANE
C     FNAME   : WORKING VARIABLE FOR FILENAMES
C     ANSW    : AUXILIARY VARIABLE FOR KBD INPUT

      SUBROUTINE PANTAYA
      INTEGER IHALF, IQUAD
      INTEGER*2 NCHAN
      CHARACTER*16 FNAME
      CHARACTER*1 ANSW

C     INITIAL VALUES
      IQUAD = 0
      NCHAN = 1   ! GREEN
      IHALF = 1   ! FIRST QUADRANT
      MEMSEL= 7   ! OR 2**NCHAN

      CALL IP5INI
      CALL SETUP
      CALL ZALPHA(0,2000)       ! CLEAR ALPHA. CHANNEL

C     OPEN FILE CONTAINING NAMES
      OPEN(UNIT=2,NAME='may13.DAT',FORM='FORMATTED',
      $           TYPE='OLD',READONLY)

      DO 10 WHILE (.TRUE.)
      READ(2,1,END = 999) FNAME
      K=INDEX(FNAME,' ')
      FNAME=FNAME(1:K-1)//'.PIC'
      LUN=1
1     FORMAT(A16)
2     FORMAT(A1)

      IQUAD= IQUAD + 1

      IF(IQUAD.GT.4) THEN
              IQUAD = 1
```

```
            TYPE *,' '
            TYPE *,' PLEASE PRESS RETURN TO CONTINUE !!...'
            ACCEPT 2,ANSW
      END IF

      TYPE *,' OBJECT = ',FNAME
C     SHOW IMAGE
      CALL PICDISP(FNAME,IHALF,IQUAD,NCHAN)
C     SHOW TITLE
      CALL ALFAD(IQUAD,FNAME)
10    CONTINUE

999   TYPE *,' '
      TYPE *,' PLEASE PRESS RETURN TO EXIT !!...'
      ACCEPT 2,ANSW

      CALL ZEROM(MEMSEL)
      CALL ZALPHA(0,2000)
      CLOSE(UNIT=2)

      RETURN
      END
```

```
C     SUBPROGRAM ALFAD

C     PURPOSE :  TO PUT PICTURE TITLE IN
C                CHANNEL ALPHANUMERIC.
C     COMMENTS:  THE ALPHA. CHANNEL CAN BE THOUGTH
C                OF AS A 25x80 MATRIX.

C     VARIABLES
C     ISTART   : OFFSET TO A GIVEN POSITION IN
C                ALPHA. CHANNEL
C     IROW     : ROW WHERE AN INSCRIPTION IS
C                DESIRED
C     ICOL     : FIRST COLUMN FOR A GIVEN INSCR.
C     ICOLOR   : COLOR FOR TEXT DATA
C     ICOUNT   : NUMBER OF CHARACTER IN TITLE
C     IBUF     : LOGICAL ARRAY TO STORE TITLE
C                (AS REQUIRED BY IPAL SUBROUTINE
C                 "WRTALP")

      SUBROUTINE ALFAD(IQUAD,NOMBRE)
      INTEGER IHALF,IQUAD
      INTEGER IROW(4),ICOL(4)
      INTEGER*2 NCHAN
      CHARACTER*16 NOMBRE
      LOGICAL*1 IBUF(16)
      CHARACTER*1 ANSW

      DATA IROW/0,0,13,13/
      DATA ICOL/8,40,8,40/
      ICOLOR=0
1     FORMAT(A16)

      ICOUNT =LEN(NOMBRE)
      ANSW = NOMBRE(1:1)

      DO 70 I = 1,ICOUNT
      ANSW = NOMBRE(I:I)
      IBUF(I) =ICHAR(ANSW)
C     CHECK FOR MINUSCULES
      IF(IBUF(I).GE.97.AND.IBUF(I).LE.122) IBUF(I)=IBUF(I)-32
70    CONTINUE

      ISTART = 80*IROW(IQUAD) + ICOL(IQUAD)

      CALL WRTALP(ISTART,IBUF,ICOUNT,ICOLOR)

      RETURN
      END
```

```
C     SUBPROGRAM GTOB

C     PURPOSE :  TO OBTAIN A BINARY IMAGE FROM
C                A GRAY LEVEL IMAGE.
C     AUTHOR  :  R. A. SALINAS
C     DATE    :  5-2-85
C     COMMENTS:  FOR IMAGEN UP TO MEDIUM SIZE.
C                THIS MODULE CAN BE REPLACED BY
C                A DYNAMIC THRESHOLDING MODULE
C                ( SEE CONTRERAS, F., MARCH 1985)

C     VARIABLES
C     FNAME   : FILENAME OF IMAGE TO PROCESS
C     SIZE    : PICTURE SIZE
C     SIZESQ  : SIZE SQUARED
C     BACK    : BACKGROUND INTENSITY LEVEL
C     OBJ     : OBJECT INTENSITY LEVEL
C     TH      : SINGLE THRESHOLD
C     ANSW    : STORAGE FOR USER INPUT
C     IMAGEV  : VECTOR TO CONTAIN IMAGE (WORKING)
C     BUFSAV  : VECTOR TO CONTAIN IMAGE ON OUTPUT
C     INIBLK  : INITIAL DATA BLOCK
C     NBLOCK  : NUMBER OF DATA BLOCKS

      SUBROUTINE GTOB(FNAME,SIZE,SIZESQ,OBJ,BACK,IMAGEV,
      1 IMAV,IV,BUFSAV,INIBLK,NBLOCK)
      INTEGER BLOCK,SIZE,ROW,COL, WRK(512),TH,SIZESQ
      DIMENSION IMAGEV(SIZESQ),IMAV(SIZE,SIZE)
      INTEGER BUFSAV(SIZESQ),IV(SIZESQ)
      CHARACTER*16 FNAME
      INTEGER OBJ,BACK
      CHARACTER*1 ANSW

1     FORMAT(A16)
C     ASSUME EXTENSION "PIC"
      K=INDEX(FNAME,'.')
      IF(K.LE.0) K=INDEX(FNAME,' ')
      FNAME=FNAME(1:K-1)//'.PIC'
      OPEN(UNIT=1,NAME=FNAME,TYPE='OLD',ACCESS='DIRECT',
      1            READONLY,FORM='UNFORMATTED',RECORDSIZE=128)

      CALL BELL(1)
      TYPE *,' '
3     TYPE *,' PLEASE CHOOSE 1 OR 2, WHERE :'
      TYPE *,' 1 - BLACK OBJ. OVER CLEAR BACKGROUND [DEFAULT]'
      TYPE *,' 2 - CLEAR OBJECT OVER DARK BACKGROUND'

      ACCEPT 2,ANSW
```

```
2     FORMAT(A1)

      OBJ=0              ! DEFAULT
      IOPTION=1          ! DEFAULT
      IF (ANSW.EQ.'2') THEN
              OBJ=255
              IOPTION=2
      END IF

      BACK=255-OBJ
      TH=128             ! DEFAULT
      TYPE *,'WAIT...'
      FNAME=FNAME(1:K-1)//'.BIN'

      OPEN(UNIT=3,NAME=FNAME,TYPE='NEW',ACCESS='DIRECT',
      1                       FORM='UNFORMATTED',RECORDSIZE=128)
      CALL     IPICRD(1,WRK,1,1)   ! READ HEADER INFO.
      CALL     IPICWT(3,WRK,1,1)   ! COPY HEADER
      K=3.-(WRK(4)-76.)/4.
      SIZE=64*2**K
      INIBLK=256*WRK(6)+WRK(5)
      NBLOCK=256*WRK(8)+WRK(7)
      LASTBL=NBLOCK+INIBLK-1

C     CONVERT TO BINARY
      IF(IOPTION.EQ.1) THEN
      DO 10 BLOCK=INIBLK,LASTBL
      CALL IPICRD(1,WRK,BLOCK,1)
      I=(BLOCK-INIBLK)*512+1
      L=1
      DO 10 K = I,I+511
      IF(WRK(L).GT.TH) THEN
              IMAGEV(K)=BACK
              ELSE
              IMAGEV(K)=OBJ
      ENDIF
10    L=L+1
              ELSE
      DO 20 BLOCK=INIBLK,LASTBL
      CALL IPICRD(1,WRK,BLOCK,1)
      I=(BLOCK-INIBLK)*512+1
      L=1
      DO 20 K = I,I+511
      IF(WRK(L).LT.TH) THEN
              IMAGEV(K)=BACK
              ELSE
              IMAGEV(K)=OBJ
      ENDIF
20    L=L+1
      END IF
```

```
C     ADD WHITE (BLACK) FRAME
      DO I=1,SIZE+1              !CLEAR FIRST ROW
      IMAGEV(I)=BACK
      END DO
      DO I=(SIZE-1)*SIZE,SIZE*SIZE
      IMAGEV(I)=BACK             ! CLEAR LAST ROW
      END DO
      DO I=2,SIZE-1              ! CLEAR FIRST AND LAST
      J=I*SIZE                   ! COLUMNS
      IMAGEV(J)=BACK
      IMAGEV(J+1)=BACK
      END DO

C     STORE BINARY PICTURE IN OUTPUT FILE
C
      DO 40 BLOCK=INIBLK, LASTBL
      I=(BLOCK-INIBLK)*512+1
      L=1
      DO 50 K = I,I+511
      WRK(L)=IMAGEV(K)
50    L=L+1
40    CALL IPICWT(3,WRK,BLOCK,1)
      CLOSE(UNIT=1)
      CLOSE(UNIT=3)

      DO 60 I = 1,SIZE*SIZE
      IV(I) = IMAGEV(I)
60    BUFSAV(I) = IMAGEV(I)

      RETURN
      END
```

```
C     SUBPROGRAM EDGE

C     PURPOSE :  TO OBTAIN SET OF BOUNDARY POINTS
C                FOR OBJECT AND FILTER NOISE OUT.
C                VECTOR VERSION
C     METHOD  :  THE PROGRAM SCANS THE BINARY IMAGE
C                FOR AN OBJECT PIXEL. ONCE IT IS FOUND
C                ITS CONTOUR IS TRACED UNTIL BACK TO THE
C                INITIAL PIXEL; THEN, SEVERAL REJECTION
C                CRITERIA ARE APPLIED. IF THE OBJECT PASSES
C                ALL TESTS, ITS COORDINATES ARE STORED IN
C                A FILE.
C     DATE    :  2-5-85
C     COMMENTS:  THIS MODULE IS AN EXTENDED VERSION
C                OF AN EARLIER PROGRAM BY CONTRERAS,
C                F.  (MARCH 1985)
C     ENABLED:  NON-ADJ, SIZE: UP TO 256

C   NON-ADJ:
      SUBROUTINE EDGE(FNAME,SIZE,OBJ,BACK,IV)

      INTEGER A(512),BLOCK,SIZE,ROW,COL,OBJ,BACK
C     INTEGER IMAGE(SIZE*SIZE),IV(SIZE*SIZE)
      INTEGER IMAGE(65536),IV(65536)
      INTEGER X(1000), Y(1000)
C     COMMON IMAGE ! FOR ADJ.  DO NOT USE COMMON
      COMMON IMAGE ! NON-ADJ
      LOGICAL*1 SW, FLG
      CHARACTER*16 FNAME
C
      DO 5 I = 1,SIZE*SIZE
5     IMAGE(I) = IV(I)
C
1     FORMAT(A16)
      K=INDEX(FNAME,'.')
      IF(K.LE.0) K = INDEX(FNAME,' ')
C
C     OPEN OUTPUT FILE WITH EXTENSION "BOX"
      FNAME=FNAME(1:K)//'BOX'
      OPEN(UNIT=2,NAME=FNAME,TYPE='NEW',FORM='UNFORMATTED')
C
C     OPEN OUTPUT FILE WITH EXTENSION BOR(DER)
      FNAME=FNAME(1:K)//'BOR'
      OPEN(UNIT=3,NAME=FNAME,TYPE='NEW',FORM='UNFORMATTED')
C
C     OPEN OUTPUT FILE WITH EXTENSION PER(IMETER)
      FNAME=FNAME(1:K)//'PER'
      OPEN(UNIT=4,NAME=FNAME,TYPE='NEW',FORM='UNFORMATTED')
C
      SW=.FALSE.
      FLG = .FALSE.
```

```
C
      DO 30 I = SIZE+2, SIZE*(SIZE-1)-1
C     IF(IMAGE(I))30,20,30
      IF(IMAGE(I).EQ.BACK) GO TO 30
20    CALL BORDE(I,SIZE,OBJ,SW)
30    CONTINUE
C
C     IF(SW)BACKSPACE 2
      ENDFILE 2
      CLOSE(UNIT=2)
      ENDFILE 3
      CLOSE(UNIT=3)
      ENDFILE 4
      CLOSE(UNIT=4)
C
      RETURN
      END
```

```
C     SUBPROGRAM BORDE
C     PURPOSE :  TRACE COUNTOUR OF A BLOB
C                AND DECIDE IF IT CAN BELONG TO
C                AN OBJECT
C     METHOD  :  BOUNDARY FOLLOWING IS MADE USING
C                EIGTH CONNECTIVITY.  AFTER EACH
C                BLOB IS ANALYZED, THE ONE THAT
C                MIGHT BE AN OBJECT IS SCANNED
C                AGAIN TO FIND ORIENTATION, AND
C                OTHER PARAMETERS.

      SUBROUTINE BORDE(I,SIZE,OBJ,SW)
C
C     ENABLED: NON-ADJ / SIZE: < = 256
C
      INTEGER SIZE,OBJ,BACK
C     INTEGER BUF(SIZE*SIZE),MINC(256),MAXC(256)       ! ADJ
      INTEGER BUF(65536),MINC(256),MAXC(256)           ! NONADJ
      INTEGER MINR(256),MAXR(256)
      INTEGER ROW,COL,ALTO,ANCHO,PREV,BLOCK,CENT
      INTEGER X(1000), Y(1000)
C     USADOS PARA SUAVIZAR
      LOGICAL*1 SFLG
      INTEGER STK(1000), SPIKE(500), PK, SPKCTR, TK, SP
      INTEGER OLD
C
      INTEGER PTR, R, C, VTR
      LOGICAL*1 SW
      LOGICAL*1 FLG
C     COMMON BUF       ! ADJ (DO NOT USE COMMON)
      COMMON BUF       ! ADJ ( USE COMMON)
C
C     USED FOR COMPUTING ORIENTATION
      REAL EVL1,EVL2,MX,MY,ORIEN

      BACK=255-OBJ
      K=I
      PREV=K-1
      MINROW=K/SIZE+1
      MAXROW=MINROW
      MINCOL=MOD(K,SIZE)
      MAXCOL=MINCOL
      LENGTH=1
      FLG = .FALSE.
      VFLG = .FALSE.
C
C        INIT. SMOOTH
      DO 99 TK = 1,500
      SPIKE(TK) = -1 ! A VALUE NOT POSSIBLE
99    STK(TK)=0
      SFLG = .FALSE.
```

```
      SPKCTR = 0 !  1
      TK =2
      SP =3
      OLD = 0
C
      DO 10 N=1,256
      MINC(N)=256
      MAXC(N)=1
      MINR(N)=256
10    MAXR(N)=1
C
20    DO 40 N=1, 7
      NGHB=NEXT(K,PREV,SIZE)
C     IF(BUF(NGHB))30,50,30
      IF(BUF(NGHB).EQ.OBJ) GO TO 50
30    PREV=NGHB
40    CONTINUE
      MINC(MINROW)=MINCOL !  ISOLATED PIXEL
      MAXC(MINROW)=MAXCOL
      GO TO 80
50    K=NGHB
C     TYPE *,' NGHB =', K
      ROW=K/SIZE+1
C
C     "SMOOTHING" ROUTINE
C
      IF(STK(SP-TK).NE.NGHB) GO TO 169
      NSPK=1
      GO TO 171
169   IF(STK(SP-(TK+1)).NE.NGHB)GO TO 200
      NSPK=2
      TK =TK+1
171   TK = TK+1
      IF(SFLG) GO TO 180
      SFLG = .TRUE.
      PK = SP-1
      SPKCTR = SPKCTR + 1
      SPIKE(SPKCTR) = STK(PK)
      IF(NSPK.EQ.1) GO TO 179
      SPKCTR = SPKCTR + 1
      SPIKE(SPKCTR) = STK(PK-1)
C     SPKCTR=SPKCTR +1
179   CONTINUE
180   SPKCTR = SPKCTR +1
      SPIKE(SPKCTR) = NGHB
      GO TO 210
200   IF(SFLG) GO TO 205
      STK(SP) = NGHB !BOUNDARY
      SP = SP + 1
      GO TO 210
205   TK = 2 !   ERASING PROCEDURE
```

```
      SFLG = .FALSE.
      SPKCTR = SPKCTR - 1
      SP = SP - SPKCTR + OLD
      OLD = SPKCTR
      STK(SP) = NGHB
      SP = SP + 1
210   CONTINUE
C     END OF SMOOTHING
C
57    MAXROW=MAXO(ROW,MAXROW)
      MINROW=MINO(ROW,MINROW)
      COL=MOD(K,SIZE)
      IF(COL.EQ.0)COL=SIZE
      X(LENGTH) = ROW
      Y(LENGTH) = COL

      MINR(COL)=MINO(ROW,MINR(COL))
      MAXR(COL)=MAXO(ROW,MAXR(COL))

      MINCOL=MINO(COL,MINCOL)
      MINC(ROW)=MINO(COL,MINC(ROW))
      MAXCOL=MAXO(COL,MAXCOL)
      MAXC(ROW)=MAXO(COL,MAXC(ROW))

      LENGTH=LENGTH+1
C
      IF(I.NE.K) GO TO 20
C
120   ANCHO=MAXCOL-MINCOL
      ALTO=MAXROW-MINROW
      IF((ALTO*ANCHO).LT.SIZE) GO TO 80
      IF(ALTO.LE.5) GO TO 80
C     IF(ANCHO.LT.8) GO TO 80
      IF(LENGTH.LE.20) GO TO 80
      IF(MINCOL.LE.2.OR.MINROW.LE.2) GO TO 80
      IF(MAXCOL.GE.(SIZE-1).OR.MAXROW.GE.(SIZE-1)) GO TO 80
C
C     STORE THE DATA FOR THE POSSIBLE OBJECT
70    WRITE(2)MINROW,MAXROW,MINCOL,MAXCOL, SPKCTR,
      $           (MINC(II),II=MINROW,MAXROW),
      $      (MAXC(II),II=MINROW,MAXROW),
      $           (MINR(II),II=MINCOL,MAXCOL),
      $           (MAXR(II),II=MINCOL,MAXCOL)
C     TRY TO STORE SPIKE ONLY IF THERE IS ANY
         IF(SPKCTR.NE.0) WRITE(2)(SPIKE(II),II=1,SPKCTR)
C
      IF(SPKCTR.NE.0) GO TO 800
      WRITE(3) LENGTH-1, (X(II),Y(II),II=1,LENGTH-1)
C
C     DELETE SPIKES IF ANY
      IF(SPKCTR.EQ.0) GO TO 1100
```

```
800   DO 900 II =1, SPKCTR
900   BUF(SPIKE(II)) = BACK
C

      DO 1000 II =1, SP
      X(II)= STK(II)/SIZE+1
      Y(II)= MOD(STK(II),SIZE)
      IF(Y(II).EQ.0) Y(II)=SIZE
1000  CONTINUE
C
C     SP-3 IS THE TOTAL NUMBER OF USEFUL DATA
      WRITE(3) SP-3,(X(II),Y(II),II=3,SP-1)

C     COMPUTE PERIMETER

1100  CALL BORDE2(I,SIZE,OBJ,EFFL)

C     COMPUTE ORIENTATION

      IF(SPKCTR.EQ.0) THEN
      NPTS=LENGTH-1
      CALL COVAR(NPTS,X,Y,EVL1,EVL2,ORIEN,MX,MY)
             ELSE
      NPTS=SP-3
      DO II = 1, NPTS
      X(II)=X(II+2)
      Y(II)=Y(II+2)
      END DO
      CALL COVAR(NPTS,X,Y,EVL1,EVL2,ORIEN,MX,MY)
      END IF
C
C     STORE PREVIOUS RESULTS
      WRITE(4) EFFL,EVL1,EVL2,ORIEN,MX,MY
C
80    CALL DELETE(MINROW,MAXROW,MINC,MAXC,SIZE,BACK)

      RETURN
      END
```

```
C     SUBPROGRAM NEXT
C     PURPOSE : COMPUTE POSITION OF NEXT NEIGHBOR
C               (COUNTERCLOCKWISE)
C     COMMENTS: FROM CONTRERAS, F. (MARCH 1985)
C
      FUNCTION NEXT(K,PREV,SIZE)
      INTEGER  PREV,SIZE

      KPDIF=K-PREV
      IF(KPDIF)10,10,40
10    IF(IABS(KPDIF+SIZE-1)-2)20,30,30
20    NEXT=PREV+1
      RETURN
30    NEXT=PREV-SIZE
      RETURN
40    IF(IABS(KPDIF-SIZE+1)-2)50,60,60
50    NEXT=PREV-1
      RETURN
60    NEXT=PREV+SIZE
      RETURN
      END
```

```
C     SUBPROGRAM DELETE

C     PURPOSE :  DELETE FROM WORKING ARRAY OBJECT ALREADY
C                CHECKED TO AVOID ENCOUNTERING IT TWICE.
C     METHOD  :  THE OBJECT IS ERASED BY REPLACING IT
C                WITH THE BACKGROUND VALUE.

C     VARIABLES
C     MINROW  :  MINIMUM ROW FOR BLOB
C     MAXROW  :  MAXIMUM ROW FOR BLOB
C     KI      :  PTR. TO LEFMOST "COLUMN" (GIVEN ROW)
C     KF      :  PTR. TO RIGTHMOST COL. (GIVEN ROW)

      SUBROUTINE DELETE(MINROW,MAXROW,MINC,MAXC,SIZE,BACK)
C     ENABLED: NON-ADJ. / SIZE: UP TO 256

      INTEGER A(65536),MINC(256),MAXC(256),SIZE,BLOCK,ROW
      INTEGER BACK
      COMMON A
C
      DO 20 ROW=MINROW,MAXROW
      BLOCK=(ROW-1)*SIZE
      KI=BLOCK+MINC(ROW)
      KF=BLOCK+MAXC(ROW)
      DO 10 I=KI,KF
10    A(I)=BACK
20    CONTINUE
      RETURN
      END
```

```
C     SUBPROGRAM BORDE2

C     PURPOSE : COMPUTE PERIMETER

C     VARIABLES
C     I         : POINTER TO AN OBJECT PIXEL
C     PREV      : POINTER TO A BACKGOUND PIXEL
C     EEFL      : PERIMETER
C     LEN       : NUMBER OF POINTS IN BOUNDARY
C     DIAG      : NUMBER OF DIAGONAL MOVES

      SUBROUTINE BORDE2(I,SIZE,OBJ,EFFL)
C     ENABLED: NON-ADJ.  / SIZE: UP TO 256

      INTEGER BUF(65536), SIZE, OBJ, DIAG, PREV
      COMMON BUF
      K = I
      PREV = K-1
      LEN = 0
      DIAG = 0

20    DO 40 N = 1,7
      NGHB = NEXT(K,PREV,SIZE)
C     IF(BUF(NGHB))30,50,30
      IF(BUF(NGHB).EQ.OBJ) GO TO 50
30    PREV = NGHB
40    CONTINUE

      ! ISOLATED PIXEL
50    K = IABS(K-NGHB)
      IF((K.NE.1).AND.(K.NE.SIZE)) DIAG = DIAG+1
      K = NGHB
      LEN = LEN + 1
      IF(I.NE.K) GO TO 20
C     EFFECTIVE LENGTH
      EFFL = LEN + DIAG*0.414
      RETURN
      END
```

```
C     SUBPROGRAM COVAR
C     PURPOSE : TO COMPUTE THE COVARIANCE MATRIX
C                 FOR A GIVEN SET OF PIXELS
C     AUTHOR  : R. SALINAS
C     DATE    : 30-APR-84
C     LAST REV: 18-MAY-85
C     INPUTS  : NUMBER OF POINTS AND COORDINATES FOR EACH
C     OUTPUTS : EVL1, EVL2, ORIEN, MX, MY
C
C     VARIABLES
C     C       : 2x2 COVARIANCE MATRIX
C     EVL1    : PRINCIPAL EIGENVALUE
C     EVL2    : MINOR EIGENVALUE
C     MX      : X-AVERAGE
C     MY      : Y-AVERAGE
C     ORIEN   : ORIENTATION ANGLE
C     PERIM   : NUMBER OF BOUNDARY POINTS
C     VEC1,2  : EIGENVECTORS

      SUBROUTINE COVAR(PERIM, ZX, ZY, EVL1, EVL2, ORIEN, MX, MY)
C
      REAL C(4), EVL1, EVL2, VEC1(2), VEC2(2)
      REAL MX, MY
      REAL MMT1, MMT2, MMT3
      INTEGER ZX(1000), ZY(1000)
      INTEGER PERIM
      CHARACTER*16 FNAME
C
      SX=0.
      SY=0.
      ZZT1=0.
      ZZT2=0.
      ZZT3=0.
C
      DO 10 I = 1, PERIM
      SX = SX + ZX(I)
      SY = SY + ZY(I)
C
      ZZT1 = ZZT1 + ZX(I)*ZX(I)
      ZZT2 = ZZT2 + ZX(I)*ZY(I)
      ZZT3 = ZZT3 + ZY(I)*ZY(I)
10    CONTINUE

      MX = SX/FLOAT(PERIM)
      MY = SY/FLOAT(PERIM)
C
      MMT1 = MX*MX
      MMT2 = MX*MY
      MMT3 = MY*MY
C
      C(1) = ZZT1/FLOAT(PERIM) - MMT1
```

```
      C(2) = ZZT2/FLOAT(PERIM) - MMT2
      C(3) = C(2)
      C(4) = ZZT3/FLOAT(PERIM) - MMT3
C
C
      CALL EIGVAL(C,EVL1,EVL2)

      CALL EIGVEC(C,EVL1,EVL2,VEC1,VEC2)

      CALL ANGROT(VEC1,ORIEN)
C
      RETURN
      END
```

```
C     SUBPROGRAM EIGVAL

C     PURPOSE : FIND EIGENVALUES OF A 2X2 REAL SYMMETRIC
C               MATRIX C.
C     AUTHOR  : R. SALINAS

      SUBROUTINE EIGVAL(C,EVL1,EVL2)
      IMPLICIT INTEGER (A-Z)
      REAL C(4), EVL1, EVL2, LBD, RDISCR, B1
C
      IF(C(2).NE.0.0) GO TO 20
C     C IS DIAGONAL
      IF(C(1).LT.C(4)) GO TO 10
      EVL1 = C(1)
      EVL2 = C(4)
      GO TO 888
10    EVL1 = C(4)
      EVL2 = C(1)
      GO TO 888
C
C      WE NEED TO SOLVE THE QUADRATIC EQUATION
20    RDISCR = ((C(1)-C(4))**2 + 4.*C(2)*C(2))**0.5
      B1     = C(1) + C(4)

      EVL1 = (B1 + RDISCR)/2.
      EVL2 = (B1 - RDISCR)/2.

888   CONTINUE
30    FORMAT('0EVL1 = ',F10.3,'  EVL2 = ',F10.3)

      RETURN
      END
```

```
C     SUBPROGRAM EIGVEC
C     PURPOSE:  TO COMPUTE THE EIGENVECTORS FOR A REAL
C               SYMMETRIC MATRIX C, WITH EIGENVALUES
C               EVL1 AND EVL2.

      SUBROUTINE EIGVEC(C,EVL1,EVL2,VEC1,VEC2)
      IMPLICIT INTEGER (A-Z)
      REAL C(4),EVL1,EVL2,VEC1(2),VEC2(2)

      IF(C(2).NE.0.0) GO TO 10
C     C IS DIAGONAL
      VEC1(1) = 1.0
      VEC1(2) = 0.0

      VEC2(1) = 0.0
      VEC2(2) = 1.0
      GO TO 888
C
10    IF((C(1)*C(4)).EQ.(C(2)*C(2))) GO TO 20
      VEC1(2) = 1.0 ! SET TO THIS VALUE
      VEC1(1) = -C(2)/(C(1)-EVL1)

      VEC2(2) = 1.0 ! SET
      VEC2(1) = -C(2)/(C(1) - EVL2)
      GO TO 888

20    VEC1(1) = 1.0
      VEC1(2) = C(4)/C(2)

      VEC2(1) = 1.0
      VEC2(2) = -C(1)/C(2)

888   CONTINUE
30    FORMAT('0VEC1(1)=',F8.3,' VEC1(2)=',F8.3,
     $          ' VEC2(1)=',F8.3,' VEC2(2)=',F8.3)
      RETURN
      END
```

```
C     SUBPROGRAM ANGROT
C     PURPOSE:  TO COMPUTE THE ANGLE BETWEEN TWO
C               BI-DIMENSIONAL VECTORS.

      SUBROUTINE ANGROT(A, THETA)
      IMPLICIT INTEGER (A-Z)
      REAL A(2), B(2), ADOTB, LONGA, LONGB
      REAL COSIN, THETA, PI, ACOS, RADIAN
      DATA B/1., 0./
C
      ADOTB = A(1)*B(1) + A(2)*B(2)
      LONGA = ( A(1)*A(1) + A(2)*A(2) )**0.5
      LONGB = ( B(1)*B(1) + B(2)*B(2) )**0.5

      COSIN =  ADOTB/(LONGA*LONGB)
C     FOR AN EIGENVECTOR ITS LENGTH CAN NOT BE 0.

      RADIAN = ACOS(COSIN) ! USER FUNCTION
      PI = 3.14159
      THETA = 180.*RADIAN/PI

      RETURN
      END


      FUNCTION ACOS(X)
      REAL X, ACOS, SINE
      SINE = (1.-X*X)**0.5
      ACOS = ATAN2(SINE, X)
      RETURN
      END
```

```
C     SUBPROGRAM EXHI
C     PURPOSE : PREPARE NOISE FREE IMAGE FOR
C               NEXT MODULES.
C     METHOD  : GIVEN BINARY IMAGE AND COORDINATES OF
C               BOUNDING BOX, IT CREATES AN OUTPUT
C               IMAGE CONTAINING THE OBJECT ALONE.
C     FINAL PICTURE EXHIBITION
C     ENABLED : NON-ADJ. / SIZE: UP TO 256
C
C     INPUTS  : FNAME, SIZE, BUFS
C     OUTPUTS : IV

      SUBROUTINE EXHI(FNAME, SIZE, OBJ, BACK, BUFS, IV,
     1 INIBLK, NBLOCK)
      INTEGER BLOCK, SIZE, ROW, COL, WRK(512), OBJ, BACK
      INTEGER ROWI, ROWL
      INTEGER SPIKE(500), SPKCTR
      DIMENSION MINC(256), MAXC(256)
      DIMENSION MINR(256), MAXR(256)
      INTEGER A(512)
      INTEGER BUFS(65536), IV(65536)              ! NON-ADJ
      INTEGER IMAGEN(65536), IMAG(256, 256)
      INTEGER BUFSAV(65536), IMAI(256, 256)
      EQUIVALENCE (IMAGEN(1), IMAG(1, 1))
      EQUIVALENCE (BUFSAV(1), IMAI(1, 1))
      CHARACTER*16 FNAME

      DO 4 I = 1, SIZE*SIZE
4     BUFSAV(I)=BUFS(I)
C
1     FORMAT(A16)
      K=INDEX(FNAME, '. ')
      IF(K.LE.0) K=INDEX(FNAME, ' ')

C     OPEN FN.BOX
      FNAME=FNAME(1:K-1)//'.BOX'
      OPEN(UNIT=2, NAME=FNAME, TYPE='OLD', FORM='UNFORMATTED')
      TYPE *, 'WAIT...'

      LASTBL=NBLOCK+INIBLK-1

C  CLEAR ZONE ( SET TO CURRENT BACKGND)

      DO 5 I=1, SIZE*SIZE
5     IMAGEN(I)=BACK

10    READ(2, END=100)MINROW, MAXROW, MINCOL, MAXCOL, SPKCTR,
     $              (MINC(I), I=MINROW, MAXROW),
     $         (MAXC(I), I=MINROW, MAXROW),
     $              (MINR(I), I=MINCOL, MAXCOL),
     $              (MAXR(I), I=MINCOL, MAXCOL)
```

```
      IF(SPKCTR.NE.0) READ(2,END=100)(SPIKE(I),I=1,SPKCTR)
C
      IFAC=512/SIZE
      IBLOCK= (MINROW+IFAC*INIBLK-1)/IFAC
      LBLOCK= (MAXROW+IFAC*INIBLK-1)/IFAC

C   LOAD BOX ( APROX. )

      DO 30 BLOCK=IBLOCK,LBLOCK
      ISHFT=BLOCK-INIBLK
      DO 30 I=1,512
      IMAGEN(I+ISHFT*512)=BUFS(I+ISHFT*512)
30    CONTINUE

C     ERASE SPIKE(S), IF ANY .
      IF(SPKCTR.LE.0) GO TO 10
      DO 200 J =1, SPKCTR
200   IMAGEN(SPIKE(J)) = BACK !255
C
      GO TO 10

100   CONTINUE
C
C     CLEAR EXTRA ZONE DUE TO LOADING METHOD
      ROWI=IFAC*(IBLOCK-INIBLK) + 1
      ROWL=IFAC*(LBLOCK-INIBLK+1)
      IF(MINROW.GT.ROWI) THEN
               DO ROW=ROWI,MINROW-1
               IROW=(ROW-1)*SIZE
                        DO COL=2,SIZE-1
                        IMAGEN(IROW+COL)=BACK
                        END DO
               END DO
      END IF
      IF(MAXROW.LT.ROWL) THEN
               DO ROW=MAXROW+1,ROWL
               IROW=(ROW-1)*SIZE
                        DO COL=2,SIZE-1
                        IMAGEN(IROW+COL)=BACK
                        END DO
               END DO
      END IF
C     CLEAR ZONE TO THE LEFT AND RIGTH OF OBJECT
      DO ROW = MINROW,MAXROW
      IROW=(ROW-1)*SIZE
               DO COL=2,MINC(ROW)-1
               IMAGEN(IROW+COL)=BACK
               END DO
               DO COL=MAXC(ROW)+1,SIZE-1
               IMAGEN(IROW+COL)=BACK
               END DO
```

```
      END DO
C
C     CLEAR ZONES ABOVE AND BELOW OBJECT
      DO COL=MINCOL, MAXCOL

               DO ROW=MINROW, MINR(COL)-1
               IROW=(ROW-1)*SIZE
               IMAGEN(IROW+COL)=BACK
               END DO

               DO ROW=MAXR(COL)+1, MAXROW
               IROW=(ROW-1)*SIZE
               IMAGEN(IROW+COL)=BACK
               END DO
      END DO

C        THIS SECTION OF CODE SHOWS HOW TO CREATE
C     AN OUTPUT FILE (INTENTIONALLY LEFT HERE)
C     DO 40 BLOCK=INIBLK, LASTBL
C     I=(BLOCK-INIBLK)*512+1
C     L=1
C     DO 50 K=I, I+511
C     A(L)=IMAGE(K)
C     L=L+1
C50   CONTINUE
C40   CALL IPICWT(3, A, BLOCK, 1)

      DO 300 I = 1, SIZE*SIZE
300   IV(I) = IMAGEN(I)

C     CLOSE(UNIT=1)
      CLOSE(UNIT=2)
C     CLOSE(UNIT=3)

      RETURN
      END
```

```
C     SUB-PROGRAM COMPV
C     PURPOSE :  TO FIND COMPONENTS OF A BINARY
C                IMAGEN (VECTOR METHOD)
C     METHOD  :  THE IMAGE IS SCANNED UNTIL AN OBJECT
C                PIXEL IS FOUND. THEN, FOR THE FIRST ROW
C                THE DIFFERENT RUNS OF OBJECT PIXELS ARE
C                ASSIGNED A DISTINCT LABEL; FOR EACH NEW
C                ROW IF A GIVEN RUN OVERLAPS WITH ONE OR
C                MORE RUNS IN THE PREVIOUS ROW, IT IS LA-
C                BELLED WITH THE MINIMUM LABEL, OTHERWISE
C                IT IS ASSIGNED A NEW LABEL. EVERY TIME
C                SOME RUNS CONVERGE, THE PREVIOUSLY ASSIGNED
C                HIGHEST LABEL FOR THE COMPONENT SHOULD BE
C                RELABELLED. AT THE END, EVERY CONNECTED
C                COMPONENT HAS A UNIQUE LABEL.

C     AUTHOR  :  R. SALINAS
C     DATE    :  10-07-84
C     REVISED :   5-22-85
C     ENABLED :  SIZE = UP TO 256

C     INPUTS  :  FNAME, SIZE, OBJ, BACK, INIBLK, NBLOCK, BUF2
C     OUTPUTS :  COUNT, LOUT, IMCOM

C     VARIABLES
C     B, BV   :  MATRIX OF CONNECTIVITY
C     COUNT   :  NUMBER OF LABELS
C     LOUT    :  SET OF LABELS
C     IMCOM   :  LABELED IMAGE

      SUBROUTINE COMP(FNAME, SIZE, OBJ, BACK, INIBLK, NBLOCK,
     1 BUF2, COUNT, LOUT, IMCOM)
      INTEGER B(7,7), BV(49), WRK(512), BLOCK
      INTEGER RUNN, ZR(7,2)
      DIMENSION LBL(20)
      INTEGER  BACK, OBJ, PTR, ROW, COL, SIZE
      LOGICAL*1  IV(65536), IMA(256,256)
      INTEGER BUF2(65536)
      LOGICAL*1 IMCOM(65536)
      CHARACTER*16 FNAME
      EQUIVALENCE (IMA(1,1), IV(1))
      EQUIVALENCE (B(1,1),BV(1))
      INTEGER EQUAL(20), LOUT(20), OUTAB(20), COUNT

      N  = 1   ! INIT. FOR BSYM & TCLOS

1     FORMAT(A16)
      K=INDEX(FNAME,'.')
      IF(K.LE.0) K=INDEX(FNAME,' ')
      TYPE *,'WAIT...'
```

```
C     OPEN OUTPUT FILE WITH EXTENSION "DT1"
      FNAME=FNAME(1:K)//'DT1'
      OPEN(UNIT=1,NAME=FNAME,TYPE='NEW',
     $         FORM='UNFORMATTED')
C
      LASTBL = INIBLK + NBLOCK -1

      DO 101 K=1,SIZE*SIZE
101   IV(K)=BUF2(K)/128

      IF(BACK.EQ.0) THEN
              DO K=1,SIZE*SIZE
              IV(K)=1-IV(K)
              END DO
      END IF

C     NEW DEFINITION FOR BACKGND. AND OBJECT VALUES
      BACK =  1
      OBJ  =  0

      DO 20 PTR = 1, SIZE*SIZE
      IF(IV(PTR).EQ.BACK) GO TO 30
20    CONTINUE
      TYPE *,' NO BACKGROUND PIXELS IN IMAGE'
      TYPE *,' PROBABLE CAUSE: PICTURE TOO DARK'
      STOP

C     PROCESS FIRST OBJECT ROW
30    ROW = PTR/SIZE + 1
      COL = MOD(PTR,SIZE)
      IF(COL.EQ.0) COL = SIZE
      CALL ZRUNSV(IV,ROW,SIZE,ZR,LASTSE,RUNN)
      CALL LBLZRV(ZR,LASTSE,RUNN,ROW,SIZE,IV,LN,LBL)
      CALL BFORM(LN,B)
      N = LN
      ROW = ROW + 1

C     PROCESS REST OF THE ROWS
35    IF(ROW.GT.SIZE) GO TO 50
      CALL ZRUNSV(IV,ROW,SIZE,ZR,LASTSE,RUNN)
      K = RUNN
      IF(K.EQ.0) GO TO 40
      CALL FORADJV(K,ZR,N,B,SIZE,IV,ROW,LBL)
40    ROW = ROW + 1
      GO TO  35
50    IF(N.EQ.1) GO TO 80 !  STEP 5

      CALL BSYM(N,B)
      CALL TCLOS(N,B)    !  STEP 6

C     FINDING SET L (OF LABELS)
```

```
      DO 100 I = 1, N
100   EQUAL(I) = I
C     CHECK IF N < = 1
      DO 110 I = 1, N-1
      DO 120 J = I+1, N
      DO 130 IX = 1, N
      IF(B(IX,I).NE.B(IX,J)) GO TO 120
130   CONTINUE
      EQUAL(J) = EQUAL(I)
120   CONTINUE
110   CONTINUE
      COUNT = 1
      OUTAB(1) = 1
      LOUT(1)  = LBL(1)
      DO 140 PTR = 2, N
      DO 150   J = 1, COUNT
      IF(EQUAL(PTR).EQ.OUTAB(J)) GO TO 140
150   CONTINUE
      COUNT = COUNT + 1
      LOUT(COUNT) = LBL(EQUAL(PTR))
      OUTAB(COUNT) = EQUAL(PTR)
140   CONTINUE

      WRITE(1) COUNT,(LOUT(J),J=1,COUNT)

C     SCAN B+ :STEP 7
      DO 60 M = 1, N
      LM = LBL(M)
      DO 60 J = 1, N
      LJ = LBL(J)
      IF(B(M,J).NE.1) GO TO 60
C     CHANGE LABELS Lm <-- LJ
      DO 65 PTR = 1, SIZE*SIZE
      IF(IV(PTR).EQ.LJ) IV(PTR)= LM
65    CONTINUE
C     ZERO OUT COLUMN J IN B+
      DO 70 ROW = 1, N
70    B(ROW,J) = 0
60    CONTINUE

95    FORMAT(7I4)

      DO I=1,SIZE*SIZE
      IMCOM(I)=IV(I)
      END DO

80    CLOSE(UNIT=3,DISPOSE='SAVE')
      CLOSE(UNIT=2,DISPOSE='SAVE')
      RETURN
      END
```

```
C     SUBPROGRAM TCLOS
C     PURPOSE : FIND TRANSITIVE CLOSURE IN MATRIX B
C
      SUBROUTINE TCLOS(N,B)
      INTEGER B(7,7)
C     INPUT MATRIX B IS DESTROYED
C     B+ IS IN B AT OUTPUT
      DO 10 I = 1, N
      DO 20 J = 1, N
      IF(B(J,I).NE.1) GO TO 20
      DO 30 K = 1, N
      IF((B(J,K).EQ.1).OR.(B(I,K).EQ.1)) B(J,K) = 1
30    CONTINUE
20    CONTINUE
10    CONTINUE
      RETURN
      END
```

```
C     SUBPROGRAM ZRUNSV
C     PURPOSE : OBTAIN START/STOP POSITIONS OF RUNS
C     ENABLED : SIZE UP TO 256

      SUBROUTINE ZRUNSV(IV,ROW,SIZE,ZR,LASTSE,RUNN)
      INTEGER ROW, SIZE, ZR(7,2)
      INTEGER BACK, OBJ, RUNN
      INTEGER BA
      LOGICAL*1 IMA(256,256)
      LOGICAL*1 IV(65536)

      BACK = 1
      OBJ  = 0
      K = OBJ
      SE = 1
      RUNN = 0
C
      DO 5 L = 1,2
      DO 5 M = 1,7
5     ZR(M,L) = 0
C
      I = ROW
      BA=(ROW-1)*SIZE

      DO 10 J = 1, SIZE
      IF(IV(BA+J).EQ.K) GO TO 10
      K = IV(BA+J)
      IF(SE.NE.1) GO TO 40
      RUNN = RUNN + 1
      ZR(RUNN,SE) = J
      SE = 2
      GO TO 10
40    ZR(RUNN,SE) = J-1
      SE = 1
10    CONTINUE

      LASTSE = SE
C     CHECK IF LAST RUN IS PAIRED
      IF(ZR(RUNN,1).GT.ZR(RUNN,2)) ZR(RUNN,2) = SIZE

      RETURN
      END
```

```
C     SUBPROGRAM LBLZRV
C     PURPOSE : LABEL ROW OF IMAGE
C     ENABLED : SIZE UP TO 256

      SUBROUTINE LBLZRV(ZR,LASTSE,RUNN,ROW,SIZE,IV,LN,LBL)
      INTEGER ZR(7,2), RUNN, SIZE, LBL(20)
      INTEGER ODD, RMAX, R,STARZR, ENDZR, COL, ROW
      INTEGER BA
      LOGICAL*1 IMA(256,256)
      LOGICAL*1 IV(65536)

      IF(RUNN.EQ.0) RETURN
      ODD = MOD(LASTSE,2)
      RMAX = RUNN

      DO 10 R = 1, RMAX
C    VALUES OF LABELS COULD BE SET DIFFERENTLY FOR DISPLAY .
      LBL(R) = R+1
      STARZR = ZR(R,1)
      IF((R.EQ.RMAX).AND.(ODD.EQ.1)) GO TO 20
      ENDZR = ZR(R,2)
      GO TO 30
20    ENDZR = SIZE
30    BA=(ROW-1)*SIZE
      DO 40 COL = STARZR, ENDZR
40    IV(BA+COL) = LBL(R) ! LABEL
10    CONTINUE

      LN = RUNN

      RETURN
      END
```

```
C     SUBPROGRAM FORADJV
C     PURPOSE : DETERMINE ADJACENCY FOR A RUN.
C
      SUBROUTINE FORADJV(K, ZR, N, B, SIZE, IV, ROW, LBL)
C     ENABLED: SIZE UP TO 256
C
      INTEGER ZR(7,2), B(7,7), ROW, COL, PROW
      INTEGER BACK, OBJ
      INTEGER BA,SIZE
      DIMENSION LBL(20), LABROW(20)
      LOGICAL*1 FLG, IMA(256,256)
      LOGICAL*1 IV(65536)

      OBJ = 0
      PROW = ROW - 1
      BA=(PROW-1)*SIZE
      DO 10 KIN = 1, K
      DO 20 COL = ZR(KIN,1), ZR(KIN,2)
      IF(IV(BA+COL).NE.OBJ) GO TO 100
20    CONTINUE
C     ' K RUN IS NOT 4-ADJ '
      LBL(N+1) =LBL(N)+1 ! OTHER DEFN'S. ALSO POSSIBLE
      N = N + 1
      B(N,N) = 1
      CALL TAGIV(SIZE, IV, ROW, ZR(KIN,1), ZR(KIN,2), LBL(N))
      GO TO 10

C     FIND LABEL FOR FIRST NON-OBJ PIXEL
100   NOWCTR = 1 !  I.E. ABOVE PIXEL WAS NOT OBJ
      KL = IV(BA+COL)
      LABROW(1) = KL
      DO 115 ICOL = COL+1, ZR(KIN,2)
      IF(IV(BA+ICOL).EQ.OBJ) GO TO 115
      IF(IV(BA+ICOL).EQ.KL)  GO TO 115
      NOWCTR = NOWCTR + 1
      KL = IV(BA+ICOL)
      LABROW(NOWCTR) = KL
115   CONTINUE
      IF(NOWCTR.GE.2) GO TO 120
C      ' K RUN ADJ.  TO ONLY 1 RUN '
      CALL TAGIV(SIZE, IV, ROW, ZR(KIN,1), ZR(KIN,2), LABROW(1))
      GO TO 10
120   CONTINUE ! ' K RUN 4-ADJ.  TO 2+ RUNS '
C     FIND LOWEST-VALUED LABEL ...(PAGE 48, REF.:...)
      MINLAB= LABROW(1)
      DO 125 L = 2, NOWCTR
      IF(LABROW(L).LT.MINLAB) MINLAB = LABROW(L)
125   CONTINUE
      CALL TAGIV(SIZE, IV, ROW, ZR(KIN,1), ZR(KIN,2), MINLAB)
C     B(MINLAB,MINLAB) IS ALREADY SET SO, THE ABOVE
C     EXPRESSION IS STILL VALID.
      DO 145 J = 1, NOWCTR
145   B(MINLAB-1,LABROW(J)-1) = 1
```

```
C
10    CONTINUE
      RETURN
      END
```

```
C     SUBPROGRAM TAGIV
C     PURPOSE : LABEL ROW.
C
      SUBROUTINE TAGIV(SIZE, IV, ROW, ICOL, LCOL, LAB)
C     ENABLED: SIZE UP TO 256
C
      LOGICAL*1 IMA(256,256)
      LOGICAL*1 IV(65536)
      INTEGER ROW, COL
      INTEGER BA,SIZE

      BA=(ROW-1)*SIZE
      DO 10 COL = ICOL, LCOL
10    IV(BA+COL) = LAB
      RETURN
      END
```

```
C     SUBPROGRAM BFORM
C     PURPOSE : TO CREATE A DIAGONAL MATRIX B.

      SUBROUTINE BFORM(LN,B)
      INTEGER B(7,7)
      DO 10 I = 1, LN
      B(I,I) = 1
10    CONTINUE

      RETURN
      END


C     SUBPROGRAM BSYM
C     PURPOSE : TO MAKE MATRIX B SYMMETRIC.

      SUBROUTINE BSYM(N,B)
      INTEGER B(7,7)
      IF(N.EQ.1) RETURN
      DO 10 LH = 1, N-1
      DO 10 LV = LH+1, N
      B(LV,LH) = B(LH,LV)
10    CONTINUE

      RETURN
      END
```

```
C     SUBPROGRAM LABEL2
C     PURPOSE :  TO OBTAIN INFORMATION ABOUT NUMBER
C                OF HOLES, ITS ANGULAR DISTRIBUTION
C                AND CENTROIDS.
C     AUTHOR  :  R. SALINAS
C     DATE    :  10-16-84
C     UPDATED :   5-22-85
C     INPUTS  :  FNAME, SIZE, OBJ, BACK, COUNT, LBL, IMCOM
C     OUTPUTS :  COUNT, "FN.DT2"
C
      SUBROUTINE LABEL2(FNAME, SIZE, OBJ, BACK, COUNT,
     1 LBL, IMCOM)
      INTEGER WRK(512), BACK, OBJ, ROW, COL, BLOCK
      INTEGER COUNT, SETDIF(20), SIZE, CARDL
      LOGICAL*1 IV(65536), IMA(256,256)
      LOGICAL*1 IMCOM(65536)
      CHARACTER*16 FNAME
      DIMENSION LBL(20), LBOR(20)
      EQUIVALENCE (IV(1), IMA(1,1))
      INTEGER A(0:255)

C     USED FOR BORDER FOLLOWING:
      INTEGER PTR,PTRB,PREV,SX,SY
      INTEGER X(1000),Y(1000), DIAG
      LOGICAL*1 TIPO(0:50), TESTED(0:50)
      INTEGER HOLE
      REAL APERI(20), HCX(20),HCY(20)
      REAL DTET(20),RADIOCC(20)
C
      BACK = 1
      OBJ  = 0
      HOLE = 2

      DO 5 I = 1, SIZE*SIZE
5     IV(I)=IMCOM(I)

1     FORMAT(A16)
      K=INDEX(FNAME, '.')
      IF(K.LE.0) K=INDEX(FNAME, ' ')
      TYPE *, 'WAIT...'
C     SET OF LABELS
      FNAME=FNAME(1:K-1)//'.DT1'
      OPEN(UNIT=1, NAME=FNAME, TYPE='OLD',
     $        FORM='UNFORMATTED')
C
C     OPEN OUTPUT FILE WITH EXTENSION "DT2"
      FNAME=FNAME(1:K)//'DT2'
      OPEN(UNIT=9, NAME=FNAME, TYPE='NEW',
     $        FORM='UNFORMATTED')
C     OPEN OUTPUT FILE WITH EXTENSION "CHL"
      FNAME=FNAME(1:K)//'CHL'
```

```
      OPEN(UNIT=10,NAME=FNAME,TYPE='NEW',
     $        FORM='UNFORMATTED')
C
      LASTBL = INIBLK + NBLOCK - 1
C
      READ(1,END=18) COUNT,(LBL(J),J=1,COUNT)
18    CARDL = COUNT ! CARDINALITY OF SET L
      CLOSE(UNIT=1,DISPOSE='SAVE')
C
C     FIND BORDER LABELS (SET LB)
C     1-D TECHNIQUE
      ROW = 1
      L   = 0
      LBOR(1) = -1
C     SCAN FIRST ROW
      DO 20   COL = 1,SIZE
      IF(IV(COL).EQ.OBJ) GO TO 20
      COUNT = 1
25    IF(COUNT.GT.L) GO TO 30
      IF(IV(COL).EQ.LBOR(COUNT)) GO TO 20
      COUNT = COUNT + 1
      GO TO 25
30    L = L + 1
      LBOR(L) = IV(COL)
20    CONTINUE

C    SCAN LAST ROW

      DO 201 COL = (SIZE-1)*SIZE+1,SIZE*SIZE
      IF(IV(COL).EQ.OBJ) GO TO 201
      COUNT = 1
251   IF(COUNT.GT.L) GO TO 301
      IF(IV(COL).EQ.LBOR(COUNT)) GO TO 201
      COUNT = COUNT + 1
      GO TO 251
301   L = L + 1
      LBOR(L) = IV(COL)
201   CONTINUE

C    SCAN FIRST COLUMN

40    COL= 1
49    DO 50 ROW = 1, SIZE-1
      IROW=ROW*SIZE+1
      IF(IV(IROW).EQ.OBJ) GO TO 50
      COUNT = 1
55    IF(COUNT.GT.L) GO TO 60
      IF(IV(IROW).EQ.LBOR(COUNT)) GO TO 50
      COUNT = COUNT + 1
      GO TO 55
60    L = L + 1
```

```
      LBOR(L) = IV(IROW)
50    CONTINUE

C     SCAN LAST COL

      DO 501 ROW = 2, SIZE-1
      IROW=ROW*SIZE
      IF(IV(IROW).EQ.OBJ) GO TO 501
      COUNT = 1
551   IF(COUNT.GT.L) GO TO 601
      IF(IV(IROW).EQ.LBOR(COUNT)) GO TO 501
      COUNT = COUNT + 1
      GO TO 551
601   L = L + 1
      LBOR(L) = IV(IROW)
501   CONTINUE

70    NUMLB = L ! CARDINALITY OF LB

C     FIND SET DIFFERENCE L-LB
      COUNT = 0
      DO 100 I = 1, CARDL  ! L
      DO 110 J = 1, NUMLB
      IF(LBL(I).EQ.LBOR(J)) GO TO 100
110   CONTINUE
      COUNT = COUNT + 1
      SETDIF(COUNT) = LBL(I)
100   CONTINUE
C
C     FIND AREA OF THE HOLES AND OBJECT
      DO 190 I = 0, SIZE-1 ! RESET COUNTERS
190   A(I) = 0
      DO 200 I = 1, SIZE*SIZE
      A(IV(I)) = A(IV(I)) + 1
200   CONTINUE
C
      IF(COUNT.EQ.0) THEN
               WRITE(9) COUNT,A(0)
               ELSE
               WRITE(9) COUNT,A(0)
               WRITE(9) (SETDIF(J),J=1,COUNT)
               WRITE(9) (A(SETDIF(J)),J=1,COUNT)
      ENDIF

C     FIND PERIMETER AND CENTROID FOR EACH HOLE
      IF(COUNT.EQ.0) GO TO 99999
C     FILL TABLE OF TYPES
      TIPO(0) = OBJ

      DO 350 J = 1, CARDL
350   TIPO(LBL(J)) = BACK
```

```
      DO 360 J = 1, COUNT
360   TIPO(SETDIF(J)) = HOLE

C     INITIALIZE TABLE OF MARKERS
      DO 370 J = 0, LBL(CARDL)
370   TESTED(J) = .TRUE. ! TRUE = NOT TESTED YET
C
      PTRB = SIZE+2
405   DO 400 PTR = PTRB, SIZE*SIZE
C     LOOK FOR HOLE PIXEL
      IF(TIPO(IV(PTR)).EQ.HOLE) GO TO 410
400   CONTINUE
C     EXIT HERE MEANS DONE
C     ' DONE1'
      GO TO 99999

C     FOLLOW HOLE BOUNDARY
C     TESTED ALREADY?
410   IF(TESTED(IV(PTR))) GO TO 420 !NO
      PTRB = PTRB + 2               !YES
C     ALL TESTED ?
480   IS = 0
      DO 430 J = 1, COUNT
430   IS = IS + TESTED(SETDIF(J))
      IF(IS.NE.0) GO TO 405         !NO
C     ' DONE2'                      !YES
C     STORE ARRAYS
      IF(COUNT.EQ.0) GO TO 99999

      DO 500  I = 1,COUNT
      WRITE(9) APERI(SETDIF(I)),HCX(SETDIF(I)),HCY(SETDIF(I))
500   CONTINUE
      GO TO 99999
C
C     FIND FIRST OBJECT PIXEL TO THE RIGTH
420   TESTED(IV(PTR)) = .FALSE.
      COL = MOD(PTR,SIZE)
      IF(COL.EQ.0) COL=SIZE
      ROW = PTR/SIZE + 1

      L = 1
      DO 418 IC = COL, SIZE
      IF(IV(PTR+L).EQ.OBJ) GO TO 419
      L = L+1
418   CONTINUE
C     EXIT HERE MEANS ERROR
      TYPE *,' ISOLATED HOLE: ERROR !!!'
      GO TO 99999

419   K = PTR + L
      IP = K
```

```
      PREV = K-1
      ROW = K/SIZE + 1
      COL = MOD(K,SIZE)

      PTRB = K
      LEN  = 0
      DIAG = 0
      SX   = 0
      SY   = 0

415   DO 440 N = 1, 7
      NGHB = NEXT(K,PREV,SIZE)
      IF(IV(NGHB).EQ.OBJ) GO TO 450
      PREV = NGHB
440   CONTINUE

C     EXIT HERE MEANS ERROR
      TYPE *,' ISOLATED HOLE: ERROR !!!'
      TYPE *,' AT ROW   COL'
      TYPE *, K/SIZE+1, COL
      GO TO 99999

C     FOUND NEW BOUNDARY PIXEL
450   K = IABS(K-NGHB)
      IF((K.NE.1).AND.(K.NE.SIZE)) DIAG = DIAG + 1
      K = NGHB
      LEN = LEN + 1

C     FIND COORDINATES
      ROW = K/SIZE + 1
      COL = MOD(K,SIZE)
      if(COL.EQ.0) COL = SIZE
      X(LEN) = ROW
      Y(LEN) = COL
      SX = SX + ROW
      SY = SY + COL

C     BACK TO IP?
      IF(K.NE.IP) GO TO 415     ! NO

C     PRINT RESULTS             ! YES
425   PERIM = LEN + 0.414*DIAG
      XC = FLOAT(SX)/FLOAT(LEN)
      YC = FLOAT(SY)/FLOAT(LEN)
      APERI(IV(PTR)) = PERIM
      HCX(IV(PTR))=XC
      HCY(IV(PTR))=YC
C
C   STORE COORDINATES FOR HOLE BOUNDARY
C
      DO 471 J = 1, LEN
```

```
471   WRITE(10) X(J), Y(J)

c     ALL HOLES ANALYZED?
      GO TO 480
C
99999         IF(COUNT.LT.2) GO TO 1000

      CALL DIFANG(FNAME,COUNT,SETDIF,HCX,HCY,DTET,RADIOCC)
C
      IF(COUNT.EQ.2) THEN
      WRITE(9) DTET(1)
               ELSE
      WRITE(9) (DTET(I),I=1,COUNT)
      END IF

1000  CONTINUE !        CLOSE(UNIT=3,DISPOSE='SAVE')
      CLOSE(UNIT=9,DISPOSE='SAVE')
      CLOSE(UNIT=10)
      RETURN
      END
```

```
C     SUBPROGRAMA DIFANG
C     PURPOSE : COMPUTE ANGULAR DIFFERENCES BETWEEN
C               CONSECUTIVE PAIRS OF HOLES.
C     VARIABLES
C     NHOYOS  : NUMBER OF HOLES
C     TET     : ANG. THETA (CENTROID WRT X-AXIS)
C     DTET    : ANGULAR DIFFERENCE
C     RADIOCC : DISTANCE FROM CENTROID TO CENTROID
C               (OBJECT-HOLE)

      SUBROUTINE DIFANG(FNAME, NHOYOS, SETDIF, HCX, HCY,
     1 DTET, RADIOCC)
      INTEGER SETDIF(20)
      REAL HCX(20), HCY(20)
      REAL X(7), Y(7), TET(20), DTET(20)
      CHARACTER*16 FNAME
      REAL RADIOCC(20)

C     OPEN FILE CONTAINING COORDS. BOX
      K=INDEX(FNAME, '.')
      IF(K.LE.0) K=INDEX(FNAME, ' ')
      FNAME=FNAME(1:K-1)//'.BOX'
      OPEN(UNIT=2, NAME=FNAME, TYPE='OLD', FORM='UNFORMATTED')
      READ(2) MINROW, MAXROW, MINCOL, MAXCOL
      CLOSE(UNIT=2)

      PX=FLOAT(MINROW+MAXROW)/2.
      PY=FLOAT(MINCOL+MAXCOL)/2.
C
      DO 10 I = 1, NHOYOS
C     COMPUTE NEW COORDINATES
      X1 =HCX(SETDIF(i)) - PX
      Y1 =HCY(SETDIF(i)) - PY
C
      RADIOCC(I)=X1*X1+Y1*Y1
C     CALCULE ANGLE WRT. X-AXIS
      TET(I) = ATAN2D(Y1, X1)
      IF(TET(I).LT.0.0) TET(i) = 360. + TET(I)
10    CONTINUE
C     SORT IN INCREASING ORDER
      CALL SORTUP2A(NHOYOS, TET, RADIOCC)
C     COMPUTE ANGULAR DIFFERENCE
      CALL DIFAN(NHOYOS, TET, DTET)

      RETURN
      END
```

```
      SUBROUTINE SORTUP(NHOYOS, TET)
      REAL TET (20)
      DO 10 I = 1, NHOYOS - 1
      do 11 J = I+1, NHOYOS
      IF (TET(J).LT.TET(I)) THEN
              TMP     = TET(I)
              TET(I) = TET(J)
              TET(J) = TMP
      END IF
11    CONTINUE
10    CONTINUE
      RETURN
      END


      SUBROUTINE SORTUP2A(NHOYOS, TET, RADIOCC)
      REAL TET(20), RADIOCC(20)
      DO 10 I = 1, NHOYOS - 1
      do 11 J = I+1, NHOYOS
      IF(TET(J).LT.TET(I)) THEN
              TMP     = TET(I)
              TET(I) = TET(J)
              TET(J) = TMP
              TMP=RADIOCC(I) ! MAY 13
              RADIOCC(I)=RADIOCC(J)
              RADIOCC(J)=TMP
      END IF
11    CONTINUE
10    CONTINUE
      RETURN
      END


      SUBROUTINE DIFAN(NHOYOS, TET, DTET)
      REAL TET(20), DTET(20)
      DO 10 I = 2, NHOYOS
10    DTET(I-1) = TET(I) - TET(I-1)
      DTET(NHOYOS) = TET(1) + 360. - TET(NHOYOS)
      RETURN
      END
```

```
C     SUBPROGRAM MAIN4
C     PURPOSE :  READS BOUNDARY DATA FOR OBJECT
C                AND BEGINS CALCULATION OF CONVEX
C                HULL.
C     DATE    :  APR-21-85

C     VARIABLES
C     FNAME   :  FILENAME
C     X, Y    :  ARRAYS TO STORE COORDINATES
C     ND      :  NUMBER OF POINTS

      SUBROUTINE MAIN4(FNAME)
      INTEGER X(1000), Y(1000)
      CHARACTER*16 FNAME

      CALL SEQLEC(FNAME, X, Y, ND) ! READ DATA ABC

      CALL STP1(FNAME, X, Y, ND)   ! FIND CH

      RETURN
      END
```

```
C     SUBPROGRAM  SEQLEC
C     PURPOSE :  TO READ A SEQUENTIAL FILE CONTAINING
C                INTEGER COORDINATES X,Y
C     AUTHOR   :  R.  SALINAS
C     DATE     :  JUN 4,  1984
C     UPDATE   :  MAY 3,  1985

      SUBROUTINE SEQLEC(FNAME,X,Y,N)
      INTEGER X(1000), Y(1000)
      CHARACTER*16 FNAME
C
C     OPEN FILE CONTAINING BORDER DATA (EXT.  "BOR")
1     FORMAT(A16)
      K=INDEX(FNAME,'.')
      IF(K.LE.0) K=INDEX(FNAME,' ')
      FNAME=FNAME(1:K-1)//'.BOR'
      OPEN(UNIT=4,NAME=FNAME,TYPE='OLD',
     $       ACCESS='SEQUENTIAL',FORM='UNFORMATTED')
      READ(4,END=85) N,(X(I),Y(I),I=1,N)
85    CLOSE(UNIT=4,DISPOSE='SAVE')

      RETURN
      END
```

```
C     SUBPROGRAM STP1
C     PURPOSE : FINDS CONVEX HULL BY GRAHAM'S
C               METHOD.
C     COMMENTS
C     ALSO INCLUDED ARE THE COMPUTATION OF
C     FEATURES FOR THE CONCAVITIES AND THE
C     REDUCTION TO A STRING.

      SUBROUTINE STP1(FNAME, X, Y, ND)
      IMPLICIT INTEGER (A-Z)
      REAL FLOAT, AUX, MX, MY
      REAL DE, DIST, PERSGM
      REAL XCNTR, YCNTR
C     REAL R(1000), TET(1000), R1(1000), TET1(1000)
      REAL*8 R(1000), TET(1000), R1(1000), TET1(1000)
      DIMENSION X(1000), Y(1000), PTRLST(1000),
     1  A(2), B(2), C(2)
      DIMENSION XS(1000), YS(1000), NPTRLS(1000)
      INTEGER PX(9), PY(9)

C     VARIABLES USED FOR FINDING MOMENT
      REAL DN(500), T(500), W1, W2, W3, W1P, W2P, W3P
      REAL M, U3, XC, YC, FTOT, SD3, U33, SD33

C     DIMENSIONING MADE FOR UP TO 100 CONCAVITIES
      REAL DNMAX, DEEP(100)

C     VARIABLES USED FOR FINDING ORIENTATION
      REAL DIFVX, DIFVY, NX, NY, PHI, ANGUL

C     VARIABLES USED TO COLLECT DATA GENERATED BY
C     EACH CONCAVITY.
      REAL DIS, ANG, DEEP1, UNOR, PERIM, T1, T2
      REAL ANGD(100), XCN, YCN, XCNT(100), YCNT(100)
      REAL RHO(100)
      LOGICAL*1 FLG, LTABLE(50, 100)
      DIMENSION DIS(100), ANG(100), DEEP1(100), UNOR(100),
     1 PERIM(100)
      DIMENSION IAR(100), STR(100), OUTSTR(50)

C     ARRAYS USED IN RECOMBINING ATTRIBUTES.
      INTEGER IAR2(50)
      LOGICAL*1 ONESYMBOL
      REAL ANGD2(50), PERIM2(50), DIS2(50), RHO2(50)
      REAL UNOR2(50), DEEP2(50)
      CHARACTER*16 FNAME


      ONESYMBOL=.FALSE.  ! I.E. ASSUME REDUCED STRING WILL
                         ! HAVE AT LEAST TWO SYMBOLS
```

```
C     OPEN OUTPUT FILE FOR DATA (ASSUME EXT. "DT3")
C1    FORMAT(A16)
      K=INDEX(FNAME,'.')
      IF(K.LE.0) K=INDEX(FNAME,' ')
      FNAME=FNAME(1:K-1)//'.DT3'
      OPEN(UNIT=11,NAME=FNAME,TYPE='NEW',
      $             FORM='UNFORMATTED')

C     OPEN OUTPUT FILE FOR REDUCED DATA
      FNAME=FNAME(1:K)//'DT4'
      OPEN(UNIT=12,NAME=FNAME,TYPE='NEW',
      $       FORM='UNFORMATTED')

C    OPEN OUTPUT FILE FOR EXTREME POINTS
      FNAME=FNAME(1:K)//'HUL'
      OPEN(UNIT=15,NAME=FNAME,TYPE='NEW',
      $    FORM='UNFORMATTED')

C     OPEN OUTPUT FILE FOR FINAL CONCAVITIES
      FNAME=FNAME(1:K)//'CAV'
      OPEN(UNIT=17,NAME=FNAME,TYPE='NEW',
      $    FORM='UNFORMATTED')

C     OPEN OUTPUT FILE FOR LTABLE
      FNAME=FNAME(1:K)//'LTB'
      OPEN(UNIT=19,NAME=FNAME,TYPE='NEW',
      $    FORM='FORMATTED')

C     OPEN  FILE WITH MX,MY
      FNAME=FNAME(1:K)//'PER'
      OPEN(UNIT=21,NAME=FNAME,TYPE='OLD',
      $    READONLY,FORM='UNFORMATTED')

C     SAVE ARRAYS X AND Y FOR FURTHER PROCESSING
C     AFTER STEP 5.
      DO 5 I =1, ND
      XS(I) = X(I)
      YS(I) = Y(I)
5     CONTINUE

C     READ(21,END=1999) AUX,AUX,AUX,AUX,XCNTR,YCNTR

C    SEARCH FOR THREE NON-COLLINEAR POINTS
C    (STEP 1 OF GRAHAM'S ALGORITHM)
      N=ND
      UPTR=1
      BOTPTR=ND
      FLAG=0
      DO I = 1,N
      PTRLST(I)=I
      END DO
```

```
C     POINT TO FIRST THREE POINTS
      K =1
      K1=2
      K2=3
C     LOAD INTO VECTORS A, B AND C.
      A(1)=X(K)
      A(2)=Y(K)
10    B(1)=X(K1)
      B(2)=Y(K1)
      C(1)=X(K2)
      C(2)=Y(K2)
C     COLLINEAR ?
      CALL COLLIN(A, B, C, FLAG)
      IF(FLAG.EQ.0) GO TO 20  ! NO
      PTRLST(K1)=1            ! YES
C     ELIMINATE MIDDLE POINT
      CALL DEL(X, Y, N, K2, UPTR, BOTPTR)
C     POINT TO NEXT TWO POINTS
      K1=K1+1
      K2=K2+1
      IF(K2.LE.BOTPTR) GO TO 10
      TYPE 30
30    FORMAT('0ONLY TWO POINTS LEFT FROM GIVEN DATA'/
     1 ' THIS IS CONSIDERED AN ERROR !!!')
C     NOTE THAT IN ANOTHER APPLICATION THIS COULD BE
C     A VALID SITUATION.
      STOP

20    XCNTR = (A(1)+B(1)+C(1))/3.
      YCNTR = (A(2)+B(2)+C(2))/3.
      PX=XCNTR
      PY=YCNTR

      CALL STP2(X, Y, PX, PY, N, UPTR, PTRLST, R, TET)

      CALL STP3(PTRLST, R, TET, N)

      CALL STP4(PTRLST, R, TET, N, R1, TET1, N1)

      CALL STP5(PTRLST, R1, TET1, N1, NPTRLS, N)
```

```
C     FIND COORDINATES OF EXTREME POINTS
C     AND CENTROID FOR THEM.
      CX = 0
      CY = 0
      DO 100 I = 1, N
      PTRI = NPTRLS(I)
      X(PTRI) = XS(PTRI)
      CX = CX + X(PTRI)
      Y(PTRI) = YS(PTRI)
      CY = CY + Y(PTRI)
100   CONTINUE

      WRITE(15) N,(X(NPTRLS(II)),Y(NPTRLS(II)),II=1,N)

      CX = CX/N
      CY = CY/N

C     TO HANDLE CIRCULARITY OF DATA
      NPTRLS(N+1) = NPTRLS(1)

      FLG = 0  ! FLAG USED FOR LAST SEGMENT
```

```
C     PROCESSING OF CONCAVITIES FOLLOWS.
C     FIND POINTS ON BOUNDARY BETWEEN 2 EXTREME POINTS

      DO 120 I = 1, N  !  N = NUMBER OF EXTR. PTS.

      UPTR   = NPTRLS(I)
      BOTPTR = NPTRLS(I+1)
      J1 = UPTR   ! TO SIMPLIFY NOTATION
      J2 = BOTPTR !
      NSP = IABS(J1-J2)
      IF(J2.LT.J1) NSP = ND + J2 - J1
      DIAG = 0

C     COMPUTE EUCLIDEAN DISTANCE BETWEEN CONSECUTIVE
C     EXTREME POINTS
      DIST = DE(XS(J1),YS(J1),XS(J2),YS(J2))
      DIS(I) = DIST

C     FIND ORIENTATION FOR EACH DEFFICIENCY
      CALL ORIEN(XS(UPTR),YS(UPTR), XS(BOTPTR),YS(BOTPTR),
     $        CX,CY,W1,W2,W3,ANGUL)
      ANG(I) = ANGUL

C     FIND BOUNDARY FOR ACTUAL DEFFICIENCY
      STOR  = 1 ! COUNTER RELATIVE TO UPTR
      IF(I.LT.N) GO TO 110
      FLG = 1
      J2 = ND  ! PREV. N
110   DO 115 J = J1, J2
      KK =  J - J1 + STOR
      X(KK) = XS(J)
      Y(KK) = YS(J)
115   CONTINUE
      IF(FLG.NE.1) GO TO 118  ! EXIT LOOP
      FLG = 0  ! RESET FLAG
      J1  = 1
      J2  = BOTPTR
      STOR = KK + 1
      GO TO 110

C
C $$$$        FIND MOMENTS FOR THE DEFFICIENCY
C
118   M      = 0.
      DNMAX = 0.

      CALL TMEDIO(KK,W1,W2,W3,X,Y,T,DN,M,FTOT,DNMAX)
      DEEP1(I) = DNMAX
```

```
C     COMPUTE THIRD MOMENT
      JJ = KK-2
      U33 = U3(DN, T, JJ, M, FTOT)

C     COMPUTE STANDARD DEV.
      SD33 = SD3(DN, T, JJ, M, FTOT)

C     MOMENT NORMALIZATION
      IF(SD33. EQ. 0. ) GO TO 211
              U33 = U33/SD33
      GO TO 209
211   IF(U33. NE. 0. ) GO TO 213
      U33 = 0.
209   CONTINUE
      UNOR(I) = U33

C $$$          FIND AREA OF DEFFICIENCY
C
      IAREA = IADEF(X, Y, KK)
      IAR(I) = IAREA

C     FIND PERIMETER FOR SEGMENT
      DO 200 II = J1, J2-1
      IF(IABS(XS(II)-XS(II+1)). EQ. IABS(YS(II)-YS(II+1)))
      1 DIAG=DIAG+1
200   CONTINUE
      PERSGM = NSP + DIAG*0. 414
      PERIM(I) = PERSGM

C $$$    FIND CENTROID FOR CONCAVITY
C
      CALL PROM(X, Y, 1, KK, XCN, YCN)
      XCNT(I) = XCN
      YCNT(I) = YCN

C   $$$  FIND NORMAL DISTANCE FROM CENTROID TO LINE P1P2.
C
      RHO(I) = ABS(W1*XCNT(I)+W2*YCNT(I)+W3)/
      1          (W1*W1+W2*W2)**. 5

120   CONTINUE

129   FORMAT(1X, I3, 1X, F8. 3, 1X, F8. 3, 1X, F8. 3, 1X, F8. 3,
      $           1X, i5, 1X, 5F8. 3)

C     FIND ANGULAR DISTRIBUTION WITH RESPECT CENTROID

      CALL ANGDIS(NPTRLS, XS, YS, CX, CY, DIS, N, ANGD)

121   FORMAT(1X, I3, 1X, F8. 3, 1X, F8. 3, 1X, F8. 3, 1X, F8. 3,
```

```
$           1X, I5, 1X, 5F8.3)

WRITE(11) N, (DIS(I), ANG(I), DEEP1(I), UNOR(I), IAR(I),
$     PERIM(I), XCNT(I), YCNT(I), RHO(I), ANGD(I), I=1, N)
```

```
C     CONNECTIVITY ANALYSIS OF THE BOUNDARY

C     *        PASS I   *

C     DURING FIRST PASS, A CLASSIFICATION IS MADE
C     FOR EACH SEGMENT AS A "LINE" OR AS A "CONCAVITY"
      CO = 3    ! PROPER CONCAVITY
      L = 4     ! FOR LINE SEGMENT (OR ARC)
      T1= 1.1   ! THRESHOLD
280   CONTINUE
      DO 300 I = 1, N
      STR(I) = L
      IF((PERIM(I)/DIS(I))-T1) 300,300,310
310   STR(I) = CO
300   CONTINUE
      T2 = 1.1 ! THRESHOLD
      DO 305 I =1, N
      IF((IAR(I)/DIS(I))-T2) 305,305,307
307   STR(I) = CO
305   CONTINUE

C     *        PASS II   *
C     IN SECOND PASS, ALL ADJACENT L'S ARE CONNECTED
      OPTR = 0
      LREP = 0
      FLG = .FALSE.
      IUPTR1=0 ! USED TO STORE INFORMATION REGARDING THE
               ! CONNECTIVITY OF LAST ZONE(S) WITH FIRST
      IBPTR1=0 ! ONES.

      DO 320 I = 1, N
      IF(STR(I).EQ.L) GO TO 330
      OPTR = OPTR + 1
      OUTSTR(OPTR) = CO
      LTABLE(OPTR,1) = I
      IF(FLG) THEN
              FLG = .FALSE.
              LREP=0
      END IF
      GO TO 320
330   IF(FLG) GO TO 335
      FLG = .TRUE.
      OPTR = OPTR + 1
      OUTSTR(OPTR) = L
335   LREP = LREP + 1
      LTABLE(OPTR,LREP)= I
320   CONTINUE

C     CHECK IF CONNECTIVITY IS POSSIBLE FOR LAST AND
C     FIRST TERMS.
      IF((OUTSTR(1).EQ.L) .AND.  (OUTSTR(OPTR).EQ.L))
```

```
      $           THEN
                OPTR = OPTR-1 ! DELETE LAST TERM
                DO 340 I =1,N
                IF(LTABLE(1,I).EQ.0) GO TO 344
340             CONTINUE

C     ONLY ONE SYMBOL IN REDUCED STRING
C     RAISE ONE SYMBOL FLAG
      ONESYMBOL=.TRUE.
      OPTR=1
      GO TO 347

344             DO 345 J=I,N
                LTABLE(1,J) = LTABLE(OPTR+1,J-(I-1))
345             CONTINUE
                IUPTR1=LTABLE(OPTR+1,1)
                IBPTR1=LTABLE(1,I-1)
           END IF

347   CONTINUE


C     STORE LTABLE
      DO 3777 I = 1, OPTR
3777  WRITE(19,*) (LTABLE(I,J),J=1,N)

      WRITE(11) (STR(I),I=1,N)
      WRITE(12) OPTR,(OUTSTR(I),I=1,OPTR)

C     COMBINE ATTRIBUTES

      DO 400 ROW = 1, OPTR
      IF(OUTSTR(ROW).EQ.L) GO TO 450

C     CONCAVITY, KEEP TRACK OF NEW POSITIONS
      ANGD2(ROW) = ANGD(LTABLE(ROW,1))
      PERIM2(ROW)= PERIM(LTABLE(ROW,1))
      IAR2(ROW)  = IAR(LTABLE(ROW,1))
      DIS2(ROW)  = DIS(LTABLE(ROW,1))
      UNOR2(ROW) = UNOR(LTABLE(ROW,1))
      DEEP2(ROW) = DEEP1(LTABLE(ROW,1))
      RHO2(ROW)  = RHO(LTABLE(ROW,1))
C     ... MORE FEATURES CAN BE ADDED HERE
      GO TO 400

C     LINE, LOOP FOR EACH FEATURE
450   DO 460 COL = 1, N
C     NEW ANGD IS SUMMATION OF ANGD
      IF(LTABLE(ROW,COL).EQ.0) GO TO 465
460   ANGD2(ROW) = ANGD2(ROW) + ANGD(LTABLE(ROW,COL))
```

```
465   DO 470 COL = 1, N
C     NEW PERIM IS SUMMATION OF PERIM
      IF(LTABLE(ROW,COL).EQ.0) GO TO 475
470   PERIM2(ROW) = PERIM2(ROW) + PERIM(LTABLE(ROW,COL))

475   DO 480 COL = 1,N
      IF(LTABLE(ROW,COL).EQ.0) GO TO 485
480   IAR2(ROW) = IAR2(ROW)+ IAR(LTABLE(ROW,COL))
485   CONTINUE
C     ... ADD DO LOOPS FOR EACH ADD. FEATURE

      IF(ONESYMBOL) THEN
              DIS2(ROW)= -1. ! (AN IMPOSSIBLE VALUE)
              GO TO 488
      END IF

      IF(ROW.EQ.1.AND.IUPTR1.NE.0) THEN
              IUPTR1=NPTRLS(IUPTR1)
              IBPTR1=NPTRLS(IBPTR1+1)
              IX1=XS(IUPTR1)
              IY1=YS(IUPTR1)
              IX2=XS(IBPTR1)
              IY2=YS(IBPTR1)
              DIS2(ROW)= DE(IX1,IY1,IX2,IY2)
                       ELSE
              IX1=XS(NPTRLS(LTABLE(ROW,1)))
              IY1=YS(NPTRLS(LTABLE(ROW,1)))
              NZ=1
                DO WHILE (LTABLE(ROW,NZ).NE.0)
                NZ=NZ+1
                END DO
              NZ=NZ-1
              IX2=XS(NPTRLS(LTABLE(ROW,NZ)+1))
              IY2=YS(NPTRLS(LTABLE(ROW,NZ)+1))
              DIS2(ROW)=DE(IX1,IY1,IX2,IY2)
      END IF

488   UNOR2(ROW)=0. ! DEFN. FOR "LINES"
      DEEP2(ROW)=0. !   "      "     "
      RHO2(ROW) =0. !   "      "     "
400   CONTINUE


      WRITE(12) (ANGD2(I),PERIM2(I),IAR2(I),
      1  DIS2(I),UNOR2(I),DEEP2(I),RHO2(I),I=1,OPTR)
```

```
C     STORE COORDINATES OF MAIN CONCAVITIES
      L=0
      DO 600 I = 1, OPTR
      IF(OUTSTR(I).EQ.4) GO TO 600
      IPTR=LTABLE(I,1)
      UPTR=NPTRLS(IPTR)
      BOTPTR=NPTRLS(IPTR+1)
      IF(UPTR.LT.BOTPTR) THEN
               DO IPTR=UPTR,BOTPTR
               L=L+1
               X(L)=XS(IPTR)
               Y(L)=YS(IPTR)
               END DO
                       ELSE
               DO IPTR=UPTR,ND
               L=L+1
               X(L)=XS(IPTR)
               Y(L)=YS(IPTR)
               END DO
               DO IPTR=1,BOTPTR
               L=L+1
               X(L)=XS(IPTR)
               Y(L)=YS(IPTR)
               END DO
      END IF
600   CONTINUE

      WRITE(17) L, (X(I),Y(I),I=1,L)

99999         CLOSE(UNIT=11)
      CLOSE(UNIT=12)
      CLOSE(UNIT=15)
      CLOSE(UNIT=17)
      CLOSE(UNIT=19)
      CLOSE(UNIT=21)

      RETURN
      END
```

```
C     SUBPROGRAM STP2
C     PURPOSE : FIND POLAR COORDINATES FOR EACH POINT
C               WITH RESPECT TO A POINT "P."
C     USES DOUBLE PRECISION

      SUBROUTINE STP2(X,Y,PX,PY,N,UPTR,PTRLST,R,TET)
      IMPLICIT INTEGER (A-Z)
      DIMENSION X(1000), Y(1000), PTRLST(1000)
      REAL*8 R(1000),TET(1000)
      REAL*8 DX, DY, PHIDP
      REAL PHI

      NUP = UPTR-1

      DO 10 K =1,N
      J = K + NUP
      DX = X(J) - PX
      DY = Y(J) - PY
      R(K) =DSQRT(DX*DX + DY*DY)
      IF((DX.EQ.0.0D00).AND.(DY.EQ.0.0D00)) THEN
               TET(K) = 0.0D00 ! DEFINED SO
              ELSE
               TET(K) = PHIDP(DX,DY)
      ENDIF
      PTRLST(K) = PTRLST(J)   ! UPDATE PTRLST
10    CONTINUE

      RETURN
      END
```

```
C     SUBPROGRAM PHIDP
C     PURPOSE:  GIVES OUT AN ANGLE BETWEEN 0 AND
C               360 (DOUBLE PRECISION).
C     DX AND DY ARE THE X AND Y PROJECTIONS OF
C     A POINT WITH RESPECT TO A CERTAIN ORIGIN.

      FUNCTION PHIDP(DX,DY)
      REAL*8  PHIDP, DX, DY

      PHIDP = DATAN2D(DY,DX)
      IF(PHIDP.LT.0.0D00) PHIDP = 360.0D00 + PHIDP

      RETURN
      END
```

```
C     SUBPROGRAM PHI
C     PURPOSE:  GIVES OUT AN ANGLE BETWEEN 0 AND
C               360 (SINGLE PRECISION).
C     DX AND DY ARE THE X AND Y PROJECTIONS OF
C     A POINT WRT.  TO A CERTAIN ORIGIN .

      FUNCTION PHI(DX,DY)
      REAL   PHI, DX, DY

      PHI=ATAN2D(DY,DX)
      IF(PHI.LT.0.0) PHI = 360.0 + PHI

      RETURN
      END
```

```
C     SUBPROGRAM STP3
C     PURPOSE : TO ORDER ARRAY "TET" IN INCREASING ORDER.
C     DATA    : DOUBLE PRECISION

      SUBROUTINE STP3(PTRLST,R,TET,N)
      IMPLICIT INTEGER (A-Z)
      REAL*8 RMIN, TMIN, R(1000), TET(1000)
      DIMENSION PTRLST(1000)

      DO 10 K = 1, N-1
      RMIN = R(K)
      TMIN = TET(K)
      PTRMIN = PTRLST(K)

      DO 20 J = K+1, N
      IF(TET(J).GE.TMIN) GO TO 20
      TMIN = TET(J)
      RMIN = R(J)
      PTRMIN = PTRLST(J)
C
      TET(J) = TET(K)
      R(J)   = R(K)
      PTRLST(J) = PTRLST(K)
C
      TET(K) =TMIN
      R(K)   =RMIN
      PTRLST(K) = PTRMIN

20    CONTINUE
10    CONTINUE

      RETURN
      END
```

```
C     SUBPROGRAM STP4
C     PURPOSE : ELIMINATE POINTS WITH SAME ANGLE
C               BUT LESSER RADIUS.
C     THIS IS STEP 4 OF GRAHAM'S ALGORITHM
C     USAGE   : DOUBLE PRECISION DATA

      SUBROUTINE STP4(PTRLST,R,TET,N,R1,TET1,N1)
      IMPLICIT INTEGER (A-Z)
      REAL*8 R(1000), TET(1000), R1(1000), TET1(1000)
      REAL*8 RMAX, TMAX, AUX
      DIMENSION PTRLST(1000)

      N1=N
      J=0
      FLG=0

      DO 10 K = 1, N
      IF(TET(K).EQ.TET(K+1)) GO TO 30
      J = J+1
      IF(FLG.EQ.1)GO TO 20
      TET1(J) = TET(K)
      R1(J) = R(K)
      PTRLST(J) = PTRLST(K)
      GO TO 10
20    FLG = 0
      TET1(J) = TMAX
      R1(J) = RMAX
      PTRLST(J) = PTRMAX
      GO TO 10
30    AUX = R(K)
      AUXPTR = PTRLST(K)
      IF(FLG.EQ.1) AUX = RMAX
      IF(AUX.GT.R(K+1)) GO TO 33
      RMAX = R(K+1)
      PTRMAX = PTRLST(K+1)
      GO TO 35
33    RMAX = AUX
      PTRMAX = AUXPTR
35    FLG = 1
      TMAX = TET(K)
10    CONTINUE

      N1 = J

      RETURN
      END
```

```
C     SUBPROGRAM STP5
C     PURPOSE : FIND EXTREME POINTS OF SET S.
C               THIS IS FIFTH STEP OF GRAHAM'S
C               ALGORITHM.
C     USAGE   : DOUBLE PRECISION DATA

      SUBROUTINE STP5(PTRLST,R1,TET1,N1,NPTRLS,N)
      INTEGER NEXT(1000), PRIOR(1000)
      INTEGER PTRLST(1000), NPTRLS(1000)
      INTEGER ELIMARR(1000)
      INTEGER PTR
      REAL*8 R1(1000), TET1(1000), PI
      REAL*8 ALFA,ALF1,BETA,GAM,TANALF
      REAL*8 NUM1, DEN1,GAM1, FAC, GAMR, A, B
      REAL*8 NUM2, DEN2,GAM2
      REAL*8  DELTA,SUMANG,DELTA12,DELTA13


      ITER = 0
      MXITER = 2*N1
      PTR = 1
      PI=4.*DATAN(1.0D00)
C     FAC = PI/180.0D00
C
      K  = 1
      K1 = 2
      K2 = 3
      DELTA = 0.01D00

C     SET UP ARRAYS NEXT AND PRIOR
      NEXT(N1) = 1
      DO 10 I = 1, N1-1
      NEXT(I) = I+1
10    CONTINUE
      PRIOR(1) = N1
      DO 20 I = 2, N1
      PRIOR(I) = I - 1
20    CONTINUE

      IELIM=0

C     FIND ANGLE A-LPHA
30    GAM1 =DABS(TET1(K1) - TET1(K))
      IF(GAM1.GE.180.0D00) GAM1=360.0D00 - GAM ! TRIANGLE
      NUM1= R1(K1)*DSIND(GAM1)
      DEN1= (R1(K)-R1(K1)*DCOSD(GAM))
      IF(DEN1.NE.0.0D00) GO TO 32
      ALF1 = 90.0D00
```

```
      GO TO 33
32    TANALF = NUM1/DEN1
      ALF1 = DATAND(TANALF)
      IF(ALF1.LT.0.0D00) ALF1 = 180.0D00 + ALF1
33    A = 180.0D00 - (ALF1 + GAM)

C     FIND ANGLE B-ETA
      GAM2 = DABS(TET1(K2) - TET1(K1))
      IF(GAM2.GE.180.0D00) GAM2= 360.0D00 - GAM2
      NUM2 = R1(K2)*DSIND(GAM2)
      DEN2 = R1(K1) - R1(K2)*DCOSD(GAM2)
      IF(DEN2.NE.0.0D00) GO TO 37
      B = 90.0D00
      GO TO 38
37    TANALF = NUM2/DEN2
      B = DATAND(TANALF)
      IF(B.LT.0.D00) B = 180.0D00 + B

C     IS POINT CONVEX ?
38    SUMANG = A + B + DELTA

      IF((A + B+ DELTA).GE.180.0D00) GO TO 40  ! NO

888   ITER = ITER + 1                          ! YES
      PTR = NEXT(PTR)
      K = PTR
      K1= NEXT(K)
      K2= NEXT(K1)
      IF(ITER.GT.MXITER) GO TO 77
      GO TO 30
C
40    ITER = ITER + 1
      NEXT(PTR) = K2   ! HERE WE 'DELETE' PT. K+1
      PRIOR(K2) = PTR
      IELIM=IELIM+1
      ELIMARR(IELIM)=K1
      K = PRIOR(PTR)
      K1 = PTR
      PTR = K
      IF(ITER.GT.MXITER) GO TO 77
      GO TO 30

77    AUXI=0

C     POINTS IN THE GIVEN SET 'S'.
 201  I = 1
      KI = K
      KL=KI
 60   NPTRLS(I) = PTRLST(KL)
      KL = NEXT(KL)
      IF(KL.EQ.KI) GO TO 100
```

```
      I = I + 1
      GO TO 60
  100 CONTINUE

C     SORT NPTRLS
      CALL GSORT(I, NPTRLS)

      N = I  ! NUMBER OF EXTREME POINTS

      RETURN
      END
```

```
C              SUBPROGRAM GSORT
C     PURPOSE : TO SORT IN INCREASING ORDER INTEGER ARRAY

C     VARIABLES
C     A         : INTEGER ARRAY ( ONE DIMENSION)
C     N         : NUMBER OF ELEMENTS OF ARRAY

      SUBROUTINE GSORT(N,A)
      IMPLICIT INTEGER (A-Z)
      DIMENSION A(1000)

      DO 200 J = 1, N-1
      DO 210 K = J+1, N
      IF(A(K).GE.A(J)) GO TO 210
      TEMP = A(J)
      A(J) = A(K)
      A(K) = TEMP
210   CONTINUE
200   CONTINUE

      RETURN
      END
```

```
C     SUBPROGRAM ANGDIS
C     PURPOSE : COMPUTE ANGLE FOR EACH PAIR OF CONSE-
C               CUTIVE EXTREME POINTS WITH RESPECT
C               CENTROID OF THE SET OF EXTREME POINTS
C     VARIABLES
C     A,B,C    : SIDES OF TRIANGLE
C     A2,B2,C2: SIDES SQUARED
C     CX,CY    : CENTROID
C     XS,YS    : SET OF BOUNDARY POINTS
C     DIS      : EUCLIDEAN DISTANCE BETWEEN PAIR OF
C                EXTREME POINTS
C     N        : NUMBER OF EXTREME POINTS
C     ANGD     : ANGULAR DISTRIBUTION

      SUBROUTINE ANGDIS(NPTRLS,XS,YS,CX,CY,DIS,N,ANGD)
      DIMENSION NPTRLS(1000)
      INTEGER XS(1000),YS(1000), CX,CY
      REAL DIS(100), ANGD(100)
      REAL A,A2,B,B1,C,C2,COGAM

      F = 180./3.14159
      C = DIS(1)
      C2= C*C
      J1= NPTRLS(1)
      J2= NPTRLS(2)
      A=  DE(XS(J2),YS(J2),CX,CY)
      A2= A*A
      B1 = DE(XS(J1),YS(J1),CX,CY)
      B2= B1*B1
      COGAM= (A2+B2-C2)/(2.*A*B1)
      ANGD(1)= ACOS(COGAM)*F
      DO 10 I = 2, N-1
      J2 = NPTRLS(I+1)
      C2 = DIS(I)*DIS(I)
      B  = A
      B2 = A*A
      A  = DE(XS(J2),YS(J2),CX,CY)
      A2 = A*A
      COGAM= (A2+B2-C2)/(2.*A*B)
      ANGD(I)= ACOS(COGAM)*F
10    CONTINUE
      B  = A
      B2 = B*B ! A*A
      A  = B1  ! COMPUTED BEFORE DO-LOOP
      A2 = A*A ! B1*B1
      C2 = DIS(N)*DIS(N)
      COGAM = (A2+B2-C2)/(2.*A*B)
      ANGD(N)= ACOS(COGAM)*F

      RETURN
      END
```

```
C     SUBPROGRAM COLLIN
C     PURPOSE :  TEST COLLINEARITY OF THREE POINTS
C                ( IN TWO DIMENSIONS)
C     VARIABLES
C     A         : FIRST POINT
C     B         : SECOND POINT
C     C         : THIRD POINT
C     M         : SLOPE

      SUBROUTINE COLLIN(A,B,C,FLAG)
      IMPLICIT INTEGER (A-Z)
      DIMENSION A(2), B(2), C(2)
      REAL M12, M13
      REAL DX12, DY12,DX13,DY13

      FLAG = 0
      DX12 = FLOAT(B(1) - A(1))
      DY12 = FLOAT(B(2) - A(2))
      DX13 = FLOAT(C(1) - A(1))
      DY13 = FLOAT(C(2) - A(2))
      IF(DX12.NE.0) GO TO 10
      IF(DX13.NE.0) RETURN
      FLAG = 1
      RETURN
10    IF(DX13.EQ.0) RETURN
      M12 = DY12/DX12
      M13 = DY13/DX13
      IF(M12.NE.M13) RETURN
      FLAG = 1
      RETURN
      END
```

```
C     SUBPROGRAM DEL
C     PURPOSE :  DELETE FROM LIST OF POINTS, MIDDLE
C                POINT IN A SUBSET OF THREE COLLINEAR
C                POINTS.

      SUBROUTINE DEL(X,Y,N,K2,UPTR,BOTPTR)
      IMPLICIT INTEGER (A-Z)
      DIMENSION X(1000), Y(1000)

      NHALF = N/2 ! APPROX.
      IF(K2.GE.NHALF) GO TO 10
      DO 20 I = UPTR, K2-2
      II = K2 - 1 ! WE NEED ONLY ONE DELETION EACH TIME
      X(II) = X(II-1)
      Y(II) = Y(II-1)
20    CONTINUE
      N = N-1
      UPTR = UPTR + 1
      RETURN
10    DO 30 I = K2, BOTPTR
      X(I-1) = X(I)
      Y(I-1) = Y(I)
30    CONTINUE
      N = N-1
      BOTPTR = BOTPTR - 1
      RETURN
      END
```

```
C     SUBPROGRAM PROM
C     PURPOSE :  COMPUTE AVERAGES FOR TWO ARRAYS

C     VARIABLES
C     X,Y       :  ARRAYS TO ANALYZE
C     UPTR      :  POINTER TO ARRAYS X AND Y
C     BOTPTR    :  POINTER TO ARRAYS X AND Y
C     XNCTR     :  X-AVERAGE
C     YCNTR     :  Y-AVERAGE

      SUBROUTINE PROM(X,Y,UPTR,BOTPTR,XCNTR,YCNTR)
      IMPLICIT INTEGER (A-Z)
      DIMENSION X(1000), Y(1000)
      REAL XCNTR, YCNTR

      SX = 0
      SY = 0
      DO 10 I = UPTR, BOTPTR
      SX = SX + X(I)
      SY = SY + Y(I)
10    CONTINUE
      XCNTR = FLOAT(SX)/FLOAT(BOTPTR-UPTR+1)
      YCNTR = FLOAT(SY)/FLOAT(BOTPTR-UPTR+1)

      RETURN
      END
```

```
C     SUBPROGRAM TMEDIO
C     PURPOSE :  COMPUTE NORMAL DISTANCES OF CONCAVE POINTS
C                FROM LINE JOINING TWO EXTREME PINTS.

C     VARIABLES
C     KK        :  NUMBER OF POINTS FOR CONCAVITY
C     DN        :  ARRAY OF NORMAL DISTANCES
C     DNMAX     :  MAXIMUM NORMAL DISTANCE
C     W1,W2,W3:  WEIGTHS FOR EQ.  OF STRAIGTH LINE

      SUBROUTINE TMEDIO(KK,W1,W2,W3,X,Y,T,DN,M,FTOT,DNMAX)
      REAL  T(500), DN(500)
      INTEGER X(1000), Y(1000)
      REAL M, DNMAX
      REAL W1,W2,W3,FTOT
      REAL W1P,W2P,W3P
      REAL XC,YC
      REAL AX1,AY1

      M      = 0.
      FTOT   = 0.
      DNMAX  = 0.

      DO 130 J = 2, KK-1
      JJ = J - 1
      UPTR = 1
      BOTPTR = KK
C       EQ.  OF LINE PERPEND.  TO GIVEN LINE
      W1P = W2
      W2P = -W1
      W3P = W1*Y(J) - W2*X(J)
C       DISTANCE TO LINE
      DN(JJ) = ABS(W1*X(J)+W2*Y(J)+W3)/(W1*W1+W2*W2)**.5

      IF(DN(JJ).GT.DNMAX) DNMAX=DN(JJ)

C       INTERSECTION OF L AND LP
C       TEST FIRST IF LINE 'L' IS VERTICAL OR HORIZ.
      IF(W1.EQ.0.) GO TO 124
      IF(W2.EQ.0.) GO TO 126
      XC = (-W1*W2*Y(J)+W2*W2*X(J)-W3*W1)/(W1*W1+W2*W2)
      YC = (W1*XC+W3)/(-W2)
      GO TO 128
124   XC = X(J)
      YC = Y(BOTPTR)
      GO TO 128
126   XC = X(BOTPTR)
      YC = Y(J)
C       DIST.  FROM INTERSECT.  TO 'ORIGIN' P1
128   AX1 = X(UPTR)-XC
      AY1 = Y(UPTR)-YC
```

```
      T(JJ)= (AX1*AX1 + AY1*AY1)**0.5
C
      M = M + T(JJ)*DN(JJ)
      FTOT = FTOT + DN(JJ)
130   CONTINUE

      IF(FTOT.NE.0.) M = M/FTOT

      RETURN
      END
```

```
C     SUBPROGRAM ORIEN
C     PURPOSE : COMPUTE ANGULAR ORIENTATION FOR A
C               CONCAVITY

C     VARIABLES
C     XU,YU    : COORDINATES OF FIRST EXTREME POINT
C     XB,YB    : COORDINATES OF NEXT EXTREME POINT

      SUBROUTINE ORIEN(XU,YU, XB,YB, CX,CY, W1,W2,W3,
     1  ANGUL)
      INTEGER XU,YU, XB,YB, CX,CY
      REAL NX, NY,PHI
      REAL W1,W2,W3,ANGUL

C       EQ. OF LINE JOINING 2 EXTR. PTS.
      W1 = YB - YU
      W2 = XU - XB
      W3 = YU*XB - YB*XU

C       COMPUTE DIFFERENCE VECTOR P(I+1)-P(I)
      DIFVX = -W2
      DIFVY =  W1
C       FIND TWO NORMALS TO DIFF. VECTOR
      N1X = -DIFVY
      N1Y =  DIFVX
C
      N2X =  DIFVY
      N2Y = -DIFVX
C
      PAX = XU + N1X
      PAY = YU + N1Y
C
      PBX = XU + N2X
      PBY = YU + N2Y
C       COMPUTE DIST. TO CENTR.
      D1 = (CX-PAX)*(CX-PAX) + (CY-PAY)*(CY-PAY)
      D2 = (CX-PBX)*(CX-PBX) + (CY-PBY)*(CY-PBY)
      IF(D1.GT.D2) GO TO 122
      NX = N2X
      NY = N2Y
      GO TO 123
122   NX = N1X
      NY = N1Y
123   CONTINUE          !  EXT. NORMAL IS: NX,NY

C       FIND ORIENT. ANGLE
      ANGUL = PHI(NX,NY)

      RETURN
      END
```

```
C     SUBPROGRAM DE
C     PURPOSE:  TO COMPUTE EUCLIDEAN DISTANCE
C               BETWEEN A PAIR OF TWO DIMENSIONAL
C               POINTS.
C     DATE    : NOV.  14,  1984

      FUNCTION DE(X1,Y1,X2,Y2)
      INTEGER X1,Y1,X2,Y2
      REAL DE

      DE = ((X1-X2)*(X1-X2) +
     $               (Y1-Y2)*(Y1-Y2))**0.5

      RETURN
      END
```

```
      FUNCTION U3(DN, T, JJ, M, SUMDN)
      REAL DN(500), T(500)
      REAL M, SUMDN
      REAL U3
      U3 = 0.
      DO 131 L = 1, JJ
      U3 = U3 + ((T(L)-M)**3)*DN(L)
131   CONTINUE
      IF(SUMDN.EQ.0.) GO TO 140
      U3 = U3/SUMDN
      RETURN
140   U3 = 0.

      RETURN
      END

      FUNCTION SD3(DN, T, N, M, SUMDN)
      REAL DN(500), T(500)
      REAL M, SUMDN
      REAL S, VAR
      REAL SD3

      S=0.
      DO 10 I = 1, N
10    S = S + T(I)*T(I)*DN(I)
      IF(SUMDN.EQ.0.) GO TO 20
      VAR = S/SUMDN - M*M
      IF(VAR.LT.0.) THEN
               SD3 = -(ABS(VAR))**1.5
               ELSE
               SD3 = VAR**1.5
      END IF
      RETURN
20    CONTINUE ! ' FTOT IS ZERO '

      RETURN
      END
```

```
C     SUBPROGRAM IADEF
C     PURPOSE : COMPUTE AREA OF A DEFFICIENCY.
C               ( AN INTEGER )

C     VARIABLES
C     KK        : NUMBER OF POINTS
C     X, Y      : COORDINATES OF EACH POINT

      FUNCTION IADEF(X,Y,KK)
      INTEGER X(1000), Y(1000)
      INTEGER AAUX, ATRG, ARECT, ATOT
      INTEGER UPTR, BOTPTR, YMIN

      UPTR = 1
      BOTPTR = KK
      ATRG = IABS((X(BOTPTR)-X(UPTR))*(Y(BOTPTR)-Y(UPTR))/2)
      YMIN = Y(UPTR)
      IF(Y(BOTPTR).LT.Y(UPTR)) YMIN = Y(BOTPTR)
      ARECT = IABS((X(BOTPTR)-X(UPTR))*YMIN)
      AAUX = 0

      DO 135 K = 1, KK-1
      AAUX = AAUX + (Y(K+1)+Y(K))*(X(K+1)-X(K))
135   CONTINUE

      ATOT = IABS(AAUX/2)
      IADEF = IABS(ATOT - ARECT - ATRG)

C     ' AREA DEFF.=', IADEF

      RETURN
      END
```

```
C     SUBPROGRAMA RECON
C     PURPOSE : ATTEMPTS RECOGNITION OF AN OBJECT
C               AGAINST A DATA BASE
C     VARIABLES
C     FNAME   : FILENAME (OBJECT)
C     SUCFLG  : SUCCESS FLAG

      SUBROUTINE RECON(FNAME,SUCFLG)
      INTEGER COUNT
      INTEGER A(0:255)
      INTEGER SETDIF(20)
      INTEGER N,IAR(100),STR(20)
      INTEGER LTABLE(50,100)
      INTEGER OPTR,OUTSTR(50),IAR2(50)
      INTEGER LDIST(20)
      INTEGER COUNTX
      INTEGER AX(0:255)
      INTEGER SETDIFX(20)
      INTEGER ROWMIN,ROWMAX,COLMIN,COLMAX
      INTEGER NX,IARX(100),STRX(20)
      INTEGER LTABLEX(50,100)
      INTEGER OPTRX,OUTSTRX(50),IAR2X(50)
      REAL PER
      REAL EVL1,EVL2
      REAL APERI(20),HCX(20),HCY(20),DTETX(20)
      REAL DIS(100),ANG(100),UNOR(100),PERIM(100)
      REAL DEEP1(100),XCNT(100),YCNT(100),RHO(100)
      REAL ANGD(100)
      REAL ANGD2(50),PERIM2(50)
      REAL DIS2(50),UNOR2(50),DEEP2(50),RHO2(50)
      REAL ERRVEC(20)
      REAL PERX
      REAL EVL1X,EVL2X,ORIEN,MX,MY
      REAL APERIX(20),HCXX(20),HCYX(20),DTET(20)
      REAL DISX(100),ANGX(100),UNORX(100),PERIMX(100)
      REAL DEEP1X(100),XCNTX(100),YCNTX(100),RHOX(100)
      REAL ANGDX(100)
      REAL ANGD2X(50),PERIM2X(50)
      REAL DIS2X(50),UNOR2X(50),DEEP2X(50),RHO2X(50)
      CHARACTER*16 FNAME
      CHARACTER*16 NCLAS(10)
      CHARACTER*1 ANSW
      LOGICAL*1 SUCFLG
C     ASSUME (UP TO) 20 CLASES AND (UP TO) 10 TESTS
      LOGICAL*1 TOK(20)
      LOGICAL*1 TCAN(20)
      CHARACTER*16 CLASE(20)
      REAL VDIF(20,10)

C     INITIALIZE ARRAYS
      DO 10 I = 1, 20
```

```
      TOK(I) = .FALSE.
10    CLASE(I) = ' '
      KO = 0 ! OK
      SUCFLG =.FALSE.  ! DEFAULT

C     THE OBJECT
      TYPE *,' '
      TYPE *,'----------------------'
      TYPE *,' FOR OBJECT:  ',FNAME
      K=INDEX(FNAME,'.')
      IF(K.LE.0) K=INDEX(FNAME,' ')

C     OBTAIN BOUNDING BOX DATA
      FNAME=FNAME(1:K-1)//'.BOX'
      OPEN(UNIT=1,NAME=FNAME,TYPE='OLD',
     $        FORM='UNFORMATTED')
      READ(1,END=11) ROWMIN,ROWMAX,COLMIN,COLMAX
11    CLOSE(UNIT=1)

      FNAME=FNAME(1:K-1)//'.PER'
      OPEN(UNIT=14,NAME=FNAME,TYPE='OLD',
     $        FORM='UNFORMATTED')

C     OPEN FILE WITH EXTENSION "DT2"
      FNAME=FNAME(1:K-1)//'.DT2'
      OPEN(UNIT=15,NAME=FNAME,TYPE='OLD',
     $        FORM='UNFORMATTED')

C     OPEN FILE WITH EXTENSION "DT3"
      FNAME=FNAME(1:K-1)//'.DT3'
      OPEN(UNIT=16,NAME=FNAME,TYPE='OLD',
     $        FORM='UNFORMATTED')

C     OPEN FILE WITH EXTENSION "DT4"
      FNAME=FNAME(1:K)//'DT4'
      OPEN(UNIT=17,NAME=FNAME,TYPE='OLD',
     $        FORM='UNFORMATTED')

C     OPEN FILE WITH EXTENSION "LTB"
      FNAME=FNAME(1:K)//'LTB'
      OPEN(UNIT=21,NAME=FNAME,TYPE='OLD',
     $        FORM='FORMATTED')


C     OPEN FILE CONTAINING NAMES IN D.B.
      OPEN(UNIT=1,NAME='MAY13.DAT',TYPE='OLD',
     $        FORM='FORMATTED')

C ** READ DATA FOR THE OBJECT
C
      READ(14) PERX,EVL1X,EVL2X,ORIEN,MX,MY
```

```
      TYPE *,' PERX ET AL.: ',PERX,EVL1X,EVL2X
      READ(15) COUNTX, AX(0)
950   TYPE *,' FOR OBJECT, NUM_HOLES & AREA',COUNTX,AX(0)
      IF(COUNTX.GE.1) GO TO 880
      TYPE *,' OBJECT WITHOUT HOLES'
             GO TO 200 ! SKIP NEXT READ'S
880   TYPE *,' OBJECT WITH HOLES'
             READ(15,END=960) (SETDIFX(I),I=1,COUNTX)
960          CONTINUE !TYPE *,' SETDIFX'
C            TYPE *,(SETDIFX(I),I=1,COUNTX)
             READ(15,END=970)(AX(SETDIFX(I)),I=1,COUNTX)
970   CONTINUE !       TYPE *,' A HOLES'

C     READ DATA OF HOLES
      DO 800 I= 1, COUNTX
      READ(15,END=980) APERIX(SETDIFX(I)),HCXX(SETDIFX(I)),
      $             HCYX(SETDIFX(I))
800   CONTINUE
980   CONTINUE !       TYPE *,' APERI-X'

      IF(COUNTX.LT.2) THEN
             TYPE *,' NO DTET FOR 0/1 HOLE'
      ELSE IF (COUNTX.EQ.2) THEN
                      READ(15,END=200) DTETX(1)
             ELSE
             READ(15,END=200) (DTETX(I),I=1,COUNTX)
      END IF

200   CONTINUE
      ! TYPE *,' *** ARCH.OBJE- DT3 ***'
      READ(16,END=100) NX,(DISX(I),ANGX(I),DEEP1X(I),
      $     UNORX(I),IARX(I),PERIMX(I),XCNTX(I),YCNTX(I),
      $     RHOX(I),ANGDX(I),I=1,NX)
      READ(16,END=100) (STRX(I),I=1,NX)

C     TYPE *,' *** ARCH. OBJE- DT4 ***'
      READ(17,END=100) OPTRX,(OUTSTRX(I),I=1,OPTRX)
      TYPE *,'OPTR-X =',OPTRX
      TYPE *,' OUTSTR-X'
      TYPE *, (OUTSTRX(I),I=1,OPTRX)
      READ(17,END=100) (ANGD2X(I),PERIM2X(I),IAR2X(I),
      $  DIS2X(I),UNOR2X(I),DEEP2X(I),RHO2X(I),I=1,OPTRX)

C     READ LTABLE FOR (UNKNOWN) OBJECT
      DO I=1,OPTRX
      READ(21,*) (LTABLEX(I,J),J=1,NX)
      END DO

C $$$         READ NAME IN D. B. ( LOOP UNTIL EOF)
C
      KC=0 ! COUNTER FOR NUMBER OF CLASSES
```

```
      KANDI=0
5     READ(1,1,END=77) FNAME
      KC=KC+1
      TYPE *,' '
      TYPE *,'** FOR NAME IN D.B.  = ',FNAME
1     FORMAT(A16)
      K=INDEX(FNAME,'.')
      IF(K.LE.0) K=INDEX(FNAME,' ')

C     OPEN FILE EXT.  "PER"
      FNAME=FNAME(1:K-1)//'.PER'
      OPEN(UNIT=7,NAME=FNAME,TYPE='OLD',
     $        FORM='UNFORMATTED')

C     OPEN FILE EXT.  "DT2"
      FNAME=FNAME(1:K-1)//'.DT2'
      OPEN(UNIT=9,NAME=FNAME,TYPE='OLD',
     $        FORM='UNFORMATTED')

C     OPEN FILE EXT.  "DT3"
      FNAME=FNAME(1:K-1)//'.DT3'
      OPEN(UNIT=11,NAME=FNAME,TYPE='OLD',
     $        FORM='UNFORMATTED')

C     OPEN FILE EXT.  "DT4"
      FNAME=FNAME(1:K)//'DT4'
      OPEN(UNIT=12,NAME=FNAME,TYPE='OLD',
     $        FORM='UNFORMATTED')

C     OPEN FILE EXT.  "LTB" (FUTURE IMPLEMENTATION)
C     FNAME=FNAME(1:K)//'LTB'
C     OPEN(UNIT=22,NAME=FNAME,TYPE='OLD',
C      $        FORM='UNFORMATTED')

C **  READ DATA FILES (FROM D.B.)

      READ(7,END=94) PER,EVL1,EVL2
94     TYPE *,  ' PERIM ET AL. = ',PER,EVL1,EVL2

      READ(9) COUNT, A(0)
95     TYPE *,' NUMBER OF HOLES & AREA ',COUNT,A(0)
      IF(COUNT.GE.1) GO TO 88
C     TYPE *,' OBJECT WITHOUT HOLES'
              GO TO 20 ! SKIP NEXT READ'S
88    CONTINUE ! TYPE *,' OBJECT WITH HOLES'
              READ(9,END=96) (SETDIF(I),I=1,COUNT)
96            CONTINUE ! TYPE *,' SETDIF'
C             TYPE *,(SETDIF(I),I=1,COUNT)
              READ(9,END=97) (A(SETDIF(I)),I=1,COUNT)
97    CONTINUE ! TYPE *,' A HOLES'
```

```
      DO 80 I= 1, COUNT
         READ(9,END=98) APERI(SETDIF(I)),HCX(SETDIF(I)),
      $             HCY(SETDIF(I))
80    CONTINUE

      IF(COUNT.EQ.1) GO TO 98
      IF(COUNT.EQ.2) THEN
              READ(9,END=98) DTET(1)
              ELSE
              READ(9,END=98) (DTET(I),I=1,COUNT)
      END IF
98    CONTINUE !        TYPE *,' APERI'

      TYPE *,' DTET REF'
      IF(COUNT.EQ.1) THEN
              TYPE *,' NO DTET FOR REF'
      ELSE IF(COUNT.EQ.2) THEN
              TYPE *,DTET(1)
              ELSE
              TYPE *, (DTET(I),I=1,COUNT)
      END IF

20    CONTINUE
      ! TYPE *,' *** ARCH. DT3 ***'
      READ(11,END=100) N,(DIS(I),ANG(I),DEEP1(I),
      $     UNOR(I),IAR(I),PERIM(I),XCNT(I),YCNT(I),
      $     RHO(I),ANGD(I),I=1,N)

      READ(11,END=100) (STR(I),I=1,N)

C     TYPE *,' *** ARCH. DT4 ***'
      READ(12,END=100) OPTR,(OUTSTR(I),I=1,OPTR)
      TYPE *,'OPTR=',OPTR
      TYPE *,' OUTSTR'
      TYPE *, (OUTSTR(I),I=1,OPTR)
      READ(12,END=100) (ANGD2(I),PERIM2(I),IAR2(I),
      $     DIS2(I),UNOR2(I),DEEP2(I),RHO2(I),I=1,OPTR)

C     READ LTABLE FOR OBJECT IN DB ( FUTURE IMPLEM. )
C     READ(22,END=100) ((LTABLE(I,J),J=1,N),I=1,OPTR)

C $$$       COMPARE "OBJECT" IN D.B. AGAINST GIVEN OBJECT

      KONT = 0
C TO1:
      TYPE *,' TO1:  ',COUNTX,COUNT
      IF(COUNTX.NE.COUNT) GO TO 66     ! REJECT THIS OBJ IN
                                       ! DB FROM LIST.
      VDIF(KC,1)=0
C TO2:
      TYPE *,' TO2: ',OPTRX,OPTR
```

```
      ITH2=3
      IF(IABS(OPTRX-OPTR).GT.iTH2) GO TO 66 ! REJECT
       VDIF(KC,2) = OPTRX-OPTR    ! ENTER DIFFER.
C TO3:
      COMPX = AX(0)/(PERX*PERX)
      COMP  = A(0)/(PER*PER)
      TYPE *,' TO3: COMPX  COMP',COMPX,COMP
      TH3=0.3 ! 30%
      ERR03=ABS(COMPX-COMP)/COMP
      TYPE *,' ERR03: 0.3',ERR03
      IF(ABS((COMPX-COMP)/COMP).GT.TH3) GO TO 66 !REJECT
       VDIF(KC,3)= COMPX-COMP

C     COMPARE RELATIVE AREA OF HOLES WRT. AREA OBJECT
C TO4:
      TH4=0.25 ! 25% DIFF.
      TYPE *,' TO4: '
      IF(COUNTX.EQ.0) GO TO 55
      STMPX=0.
      STMP=0.
      DO JJ=1,COUNTX ! AT THIS POINT, COUNTX=COUNT
      STMPX=STMPX+AX(SETDIFX(JJ))
      STMP=STMP+A(SETDIF(JJ))
      END DO
      ERR04=ABS(STMPX/AX(0)-STMP/A(0))
      TYPE *,' ERR04 : 0.25 ',ERR04
      IF(ABS(STMPX/AX(0)-STMP/A(0)).GT.TH4) GO TO 66
      VDIF(KC,4)=STMPX/AX(0)-STMP/A(0)

C TO5:
C     COMPARE EIGENVALUES
55    TH5=0.30          ! 30%
      TYPE *,' TO5: '
      TMPX=EVL2X/EVL1X
      TMP =EVL2/EVL1
      ERR05=ABS(TMPX-TMP)/TMP
      TYPE *,' ERR05 : 0.3', ERR05
      IF(ABS((TMPX-TMP)/TMP).GT.TH5) GO TO 66
      VDIF(KC,5)=ABS(TMPX-TMP)

C     AT THIS POINT WE HAVE A VIABLE CANDIDATE
      KANDI = KANDI+1
      K1=INDEX(FNAME,'.')
      FNAME=FNAME(1:K1-1)
      TCAN(KANDI) = KC
      CLASE(KANDI) = FNAME

66    GO TO 5 ! READ DATA FOR ANOTHER NAME IN D.B.

77    TYPE *,' '
```

```
C $$$          ASSIGN (UNKNOWN) OBJECT TO A CLASS

C     BECAUSE IN ANY CASE WE MAY NEED TO KNOW LOCATION
C     AND ORIENTATION FOR THE OBJECT TYPE IT NOW:

      type *,'*********************************************'
      TYPE *,' CENTROID: (',MX,',',MY,')'
      TYPE *,' ORIENTATION = ', ORIEN,' (DEGREES)'
      TYPE * ,' COORDS B.BOX ARE '
      WRITE(6,299) ROWMIN,COLMIN,ROWMAX,COLMAX
299   FORMAT(' (',I3,',',I3,')  &  (',I3,',',I3,')')
      type *,'*********************************************'

      IF(KANDI-1)300,400,500
300   TYPE *,' ** OBJECT CAN NOT BE CLASSIFIED '
      GO TO 100
400   TYPE *,' ** OBJECT ASSIGNED TO'
      TYPE 410, CLASE(1)
410   FORMAT(' CLASS = ',A16)
      SUCFLG=.TRUE.

C      CHECK IF OBJECT WAS LETTER "A" OR "TAPE".
C      THIS POST-DECISION IS MADE TO HELP IN THE
C      DISCRIMINATION BETWEEN THIS TWO SHAPES.

      IF((CLASE(1).EQ.'A').OR.(CLASE(1).EQ.'TAPE')) THEN

      CALL SEPARA(COUNTX,SETDIFX,HCXX,HCYX,
      1 XCNTX,YCNTX,ANGX,OPTRX,OUTSTRX,LTABLEX,ANSW)

              IF(ANSW.EQ.'E') THEN
                     TYPE *,' OBJECT CAN NOT BE CLASSIFIED'
              ELSE IF(ANSW.EQ.'A') THEN
                     TYPE *,' CLASS AFTER POST-PROC.: ',ANSW
                     ELSE
                     TYPE *,' CLASS AFTER POST-PROC.: TAPE'
              END IF
              GO TO 100
                              ELSE
                       GO TO 100
      END IF

500   TYPE *,' ** OBJECT AMBIGUOUS'
      TYPE *,' [ SECOND  LEVEL] '
      TYPE *,' POSSIBLES CLASSES ARE:'
      DO 510 I = 1,KANDI
510   TYPE 520, CLASE(I)
520   FORMAT('   ',A16)

C   *** SECOND LEVEL ***
C              CHECK IF OBJECT HAS HOLES
```

```
C    T11: COMPARE DTET
     IF (COUNTX.EQ.0) THEN
              TYPE *,' objx has no holes, end of process-I'
              ELSE IF (COUNTX.EQ.1) THEN
              TYPE *,' objx has 1 hole, end OF process-I'
              ELSE IF(COUNTX.EQ.2) THEN
              TYPE *,' objx has 2 holes, => compare '
              CALL CMPDTET(COUNTX,DTETX,CLASE,KANDI,NKAN,
     1        ERRVEC,NCLAS)
              ELSE
              TYPE *,' objx has',countx,' holes => compare'
              CLOSE(unit=9) ! TO AVOID CONTENTION PROBLEMS
              CALL CMPDTET(COUNTX,DTETX,CLASE,KANDI,NKAN,
     1        ERRVEC,NCLAS)
     END IF
     TH11=18.        !(APPROX. 10*SQRT(3))
     if(NKAN.EQ.0) THEN
              TYPE *,' reject object '
              GO TO 100
     END IF

     IF (ERRVEC(1).GT.TH11) THEN
              TYPE *,' CMPDTET could not discriminate'
              ELSE IF (ERRVEC(2).GT.TH11) THEN
                       TYPE *,'in T11 selected is:',NCLAS(1)
                       GO TO 100 ! EXIT
              ELSE
                       TYPE *,' at least 2 candidates left'
                       TYPE *,nclas(1),' and ',nclas(2)
                       TYPE *,' more tests are required'
              END IF


C    T12: COMPARE DISTANCES AMONG STRINGS

     CLOSE(UNIT=17)  ! TO AVOID CONTENTION PROBLEMS
     CALL DISTRNG(OPTRX,OUTSTRX,CLASE,KANDI,,LDIST,NCLAS)
     ITH13 = 3
     IF(LDIST(1).GT.ITH13) THEN
              TYPE *,' REJECT OBJECT -LEV.DIST-'
              ELSE
              TYPE *,' OBJECT ASSIGNED TO CLASS: ',NCLAS(1)
     END IF

C    IF NO DISCRIMINATION HAS BEEN ACHIEVED TRY EXTRA
C    TESTS.
C    T13: COMPARE RADIOS (PROPOSED)
C              VARIOUS SUBTESTS
C              ST11: DISTR. ANG
C              ST12: AREA
C              ST13: P2/A
```

```
C               ST14:  PERIM/DIST
C               ST15:  UNOR
C               ST16:  DEEP/RHO
C               ETC.

C     EXIT
100   CLOSE(UNIT=1)
      CLOSE(UNIT=7)
      CLOSE(UNIT=9)
      CLOSE(UNIT=11)
      CLOSE(UNIT=12)
      CLOSE(UNIT=14)
      CLOSE(UNIT=15)
      CLOSE(UNIT=16)
      CLOSE(UNIT=17)
      CLOSE(UNIT=21)
      CLOSE(UNIT=22)

      RETURN
      END
```

```
C     SUBPROGRAM SEPARA
C     SPECIAL PURPOSE ROUTINE USED TO ATTEMPT
C     A CLASSIFICATION BETWEEN LETTER "A" AND
C     A SCOTCH TAPE.
C     DATE     : MAY-30-85

      SUBROUTINE SEPARA(COUNTX,SETDIFX,HCXX,HCYX,
      1  XCNTX,YCNTX,ANGX,OPTRX,OUTSTRX,LTABLEX,ANSW)

      INTEGER LTABLEX(50,100),COUNTX,SETDIFX(20)
      REAL  HCXX(20),HCYX(20)
      INTEGER OPTRX,OUTSTRX(50)
      REAL XCNTX(100),YCNTX(100),ANGX(100)
      INTEGER LTABLE(50,100),COUNT,SETDIF(20)
C     REAL HCX(20),HCY(20)
C     REAL ANG(100)
      CHARACTER*16 FNAME,OBJX
      CHARACTER*1 ANSW

C    FIND THE LOCATION OF THE CONCAVITY
      DO I=1,OPTRX
      IF(OUTSTRX(I).EQ.3) GO TO 10
      END DO
      TYPE *,' EXPECTED CONCAVITY NOT PRESENT'
      RETURN

C  FIND THE INDEX OF CORRESPONDING ELEMENTARY CONCAV.
10    INDEX=LTABLEX(I,1)

C  GET CENTROID OF HOLE (ASSUME 1 HOLE)
      XH=HCXX(SETDIFX(1))
      YH=HCYX(SETDIFX(1))

C  GET CENTROID OF CONCAVITY
      XC=XCNTX(INDEX)
      YC=YCNTX(INDEX)

C  COMPUTE ANGLE (WRT X-AXIS OF VECTOR H-->C)
      DX=XC-XH
      DY=YC-YH
      ANGHC=PHI(DX,DY)

C  OBTAIN ORIENTATION ANGLE FOR CONCAVITY
      ANGCON=ANGX(INDEX)

C  COMPUTE NOW THE ANGULAR DIFFERENCE
      DELTET=ANGHC-ANGCON
      IF(ABS(DELTET).GE.180.) THEN
```

```
        IF(DELTET.GT.0) THEN
              DELTET=DELTET-360.0
                       ELSE
              DELTET=DELTET+360.0
        END IF
      END IF

C   COMPARE AGAINST TEMPLATES
      IF(ABS(DELTET).LT.25.0) THEN
              ANSW='A'
              ELSE IF (ABS(DELTET).GE.25.AND.
     1        ABS(DELTET).LT.90.) THEN
              ANSW='T'
              ELSE
              ANSW='E'          ! "E" STANDS FOR ERROR
      END IF

      RETURN
      END
```

```
C     SUBPROGRAM CMPDTET
C     PURPOSE :  IT COMPARES TWO REAL VECTORS
C                ALLOWING FOR ROTATION.
C     COMMENTS
C     MAIN APPLICATION OF THIS PROCEDURE IS IN
C     CHOOSING BEST MATCH FOR VECTORS REPRESENTING
C     ANGULAR DIFFERENCES, E.G., GIVEN AN OBJECT
C     WITH HOLES, ANGULAR DIFFERENCES BETWEEN PAIRS
C     OF HOLES (WITH RESPECT TO A GIVEN AXIS) CAN
C     BE COMPUTED AND COMPARED WITH REFERENCE(S).
C     IT ASSUMES OBJECT AND REFERENCES WITH
C     SAME NUMBER OF HOLES.
C     DATE     : MAY 12, 1985

C     VARIABLES
C     COUNTX   : NUMBER OF HOLES IN OBJECT
C     DTETX    : DELTA THETA FOR OBJECT
C     CLASE    : NAMES OF THE CANDIDATES
C     KANDI    : NUMBER OF CANDIDATES
C     VECTERR  : STORES SQR OF SUM OF QUADRATIC ERRORS IN
C                INCREASING ORDER.
C     REFNAMES:  STORES THE NAMES OF THE CLASSES CORRESPON-
C                DING TO VECTERR.

      SUBROUTINE CMPDTET(COUNTX,DTETX,CLASE,KANDI,
     1  J,VECTERR,REFNAMES)

      REAL DTETX(20)
      REAL DTET(20),VECTERR(20)
      INTEGER SETDIFX(20),SETDIF(20),AX(20),A(20)
      INTEGER  COUNTX,COUNT, PTRS(20)
      REAL APERIX(2),HCXX(20),HCYX(20)
      REAL APERI(2),HCX(20),HCY(20)
      CHARACTER*16 FNAME, CLASE(20), REFNAMES(10)

      TH  =10. ! (DEGREES)
      NROT=0
C     SELECT NUMBER OF ROTATIONS
      IF(COUNTX.EQ.2) .THEN
               IUP= 0
               ELSE
               IUP=COUNTX-1
      END IF

      NHOLES=COUNTX

C     READ DTET FOR CANDIDATES
C     INITIALIZES ARRAYS
      DO  I = 1, KANDI
      VECTERR(I)=-1 ! AN IMPOSSIBLE VALUE
      END DO
```

```
      DO I =1,KANDI
      PTRS(I)=I
      END DO

C     MAIN LOOP

      DO 10 I =1, KANDI
      TMP=0.
      K=INDEX(CLASE(I),'.')
      IF(K.LE.0) K=INDEX(CLASE(I),' ')
      FNAME=CLASE(I)(1:K-1)//'.DT2'

      OPEN(UNIT=1,NAME=FNAME,TYPE='OLD',
     1     READONLY,FORM='UNFORMATTED')
      READ(1,END=99)ICOUNT,IAO
      READ(1,END=99) (SETDIF(II),II=1,ICOUNT)
      READ(1,END=99) (AX(SETDIF(II)),II=1,ICOUNT)
      DO II =1, ICOUNT
      READ(1,END=99) (APERI(SETDIF(II)),HCX(SETDIF(II)),
     1        HCY(SETDIF(II)))
      END DO
      IF(ICOUNT.EQ.2) THEN
              READ(1,END=99) DTET(1)
                       ELSE
              READ(1,END=99) (DTET(II),II=1,ICOUNT)
      END IF

99    CLOSE(UNIT=1)

33    DO 15 NROT=0,IUP
      SUMERR=0.
      DO 20 J =1,NHOLES
      ERR=DTETX(J)-DTET(J)
      IF(ABS(ERR).GT.TH) THEN
              SUMERR=-1.          ! AN IMPOSSIBLE VALUE
              GO TO 21            ! EXIT INNERMOST DO LOOP
              ELSE
              SUMERR=SUMERR+ERR*ERR
      END IF
      CONTINUE
20    END DO
21    IF(SUMERR.LT.0.) THEN
              TMP=SUMERR ! 0.
      ELSE IF (VECTERR(I).LT.0.) THEN
              VECTERR(I)=SUMERR
      ELSE
              VECTERR(I)=AMIN1(VECTERR(I),SUMERR)
      END IF

      CALL ROTE1(NHOLES,DTETX)
```

```
15    END DO              ! NROT=NROT+1

      IF(TMP.LT.0.) THEN
              VECTERR(I)=-1
      END IF
      CONTINUE

10    END DO

C     NOW SORT BY INCREASING ERROR ( KEEPING TRACK
C     OF POINTERS).
      CALL SORTUP11(KANDI,VECTERR,PTRS)

C     FORGET THE NEGATIVE VALUES IN VECTERR, IF ANY

      J=0       ! COUNTER FOR CANDIDATES

      DO I=1,KANDI
      IF (VECTERR(I).LT.0.) THEN
              CONTINUE          ! 'FOR POS I, ' VERR < 0.'
              ELSE
      J=J+1
      VECTERR(J)=SQRT(VECTERR(I))
      REFNAMES(J)=CLASE(PTRS(I))
      END IF
      CONTINUE
      END DO

      RETURN
      END
```

```
C     SUBPROGRAM ROTE1
C     PURPOSE : ROTATES EVERY ELEMENT OF A REAL
C               VECTOR ONE POSITION LEFT.
C     VARIABLES
C     AR        : REAL VECTOR
C     N         : NUMBER OF COMPONENTS OF "AR"

      SUBROUTINE ROTE1(N,AR)
      REAL AR(20)
      TMP =  AR(1)
      DO I = 2, N
              AR(I-1)=AR(I)
      END DO
      AR(N)=TMP
      RETURN
      END


C     SUBPROGRAM SORTUP11
C     PURPOSE : SORTS REAL ARRAY UP WHILE
C               KEEPING TRACK OF ASSOCIATED
C               POINTERS STORED IN INTEGER
C               ARRAY

      SUBROUTINE SORTUP11(N,A1,IA2)
      INTEGER IA2(20).
      REAL A1(2)

      DO 10 I=1,N-1
              DO 20 J=I+1,N
              IF (A1(J).LT.A1(I)) THEN
                      TMP  =A1(I)
                      A1(I)=A1(J)
                      A1(J)=TMP
                      ITMP  =IA2(I)
                      IA2(I)=IA2(J)
                      IA2(J)=ITMP
              END IF
              CONTINUE
20            END DO
10    END DO
      RETURN
      END
```

```
C     SUBPROGRAMA DISTRNG
C     PURPOSE :  IT COMPARES TWO STRINGS TAKING INTO
C                ACCOUNT THE EFFECT OF ROTATION.
C     DATE    :  MAY 12, 1985

C     VARIABLES
C     CLASE   :  CLASSES OF CANDIDATES (INPUT)
C     KANDI   :  NUMBER OF CANDIDATES
C     LSTRX   :  LENGTH OF STRING X
C     STRX    :  STRING X
C     LDIST   :  STORES ERRORS BY INCREASING ORDER
C     REFNAMES:  NAMES OF THE CLASES CORRESPONDING
C                TO "LDIST."

      SUBROUTINE DISTRNG(LSTRX,STRX,CLASE,KANDI,J,LDIST,
      1 REFNAMES)
      INTEGER STRX(20),STR(20),PTRS(20)
      INTEGER D(0:19,0:19), LDIST(20)
      INTEGER E1,E2,E3
      CHARACTER*16 FNAME,CLASE(20),REFNAMES(10)
      LOGICAL*1 FLG


      NROT=0

C
C     READ STRINGS FOR CANDIDATES
C     LSTR    :  LENGTH OF A STRING
C     STR     :  STRING
C  INITIALIZE ARRAYS

      DO  I = 1, KANDI
      LDIST(I)=1000 !  AN IMPROBABLE  VALUE
      END DO

      DO I =1,KANDI
      PTRS(I)=I
      END DO

C   MAIN LOOP

      DO 10 I =1, KANDI
      K=INDEX(CLASE(I),'.')
      IF(K.LE.0) K=INDEX(CLASE(I),' ')
      FNAME=CLASE(I)(1:K-1)//'.DT4'
      OPEN(UNIT=1,NAME=FNAME,TYPE='OLD',
      1     READONLY,FORM='UNFORMATTED')
      READ(1,END=99) LSTR,(STR(II),II=1,LSTR)
99    CLOSE(UNIT=1)

C   CHOOSE AS UPPER LIMIT MIN(LSTRX,LSTR)
```

```
32    FLG=.FALSE.
      IF(LSTRX.LE.LSTR) THEN
               IUP=LSTRX-1
               FLG=.TRUE.
               ELSE
               IUP=LSTR-1
      END IF

33    DO 15 NROT=0,IUP
      ISUMERR=0
      D(0,0) =0
      DO 100 II =1, LSTRX
100   D(II,0)=D(II-1,0)+1

      DO 200 JJ=1,LSTR
200   D(0,JJ)=D(0,JJ-1)+1

      DO 300 II=1,LSTRX
      DO 300 JJ=1,LSTR
      IF(STRX(II).EQ.STR(JJ)) THEN
               E1=D(II-1,JJ-1)
               ELSE
               E1=D(II-1,JJ-1)+1
      END IF
      E2=D(II-1,JJ)+1
      E3=D(II,JJ-1)+1
      D(II,JJ)=JMINO(E1,E2,E3)
300   CONTINUE

      LDIST(I)=JMINO(LDIST(I),D(LSTRX,LSTR))

      IF(FLG) THEN
               CALL ROTE1I(LSTRX,STRX)
               ELSE
               CALL ROTE1I(LSTR,STR)
      END IF
      CONTINUE
15    END DO            ! NROT=NROT+1

      CONTINUE

10    END DO            ! END OF MAIN LOOP
C
C     NOW SORT BY INCREASING ERROR
C     ( KEEPING TRACK OF POINTERS).

      CALL SORTUP2I(KANDI,LDIST,PTRS)
C
C     J IS USED IF A THRESHOLD VALUE IS USED INSIDE
C     THIS SUBPROGRAM
C     J=0      ! COUNTER OF ACCEPTABLE CANDIDATES
```

```
      DO I=1,KANDI
C     J=J+1
      REFNAMES(I)=CLASE(PTRS(I))
      END DO

      RETURN
      END
```

```
C     SUBPROGRAM ROTE1I
C     PURPOSE : ROTATES INTEGER ARRAY ONE POSTION
C               LEFT.
      SUBROUTINE ROTE1I(N, IAR)
      INTEGER IAR(20)

      ITMP=IAR(1)
      DO I = 2, N
              IAR(I-1)=IAR(I)
      END DO
      IAR(N)=ITMP
      RETURN
      END


C     SUBPROGRAM SORTUP2I
C     PURPOSE : SORTS INTEGER ARRAY UP, WHILE
C               KEEPING TRACK OF POINTERS
C               CONTAINED IN ASSOCIATED ARRAY.

      SUBROUTINE SORTUP2I(N, IA1, IA2)
      INTEGER IA1(20), IA2(20)

      DO 10 I=1, N-1
              DO 20 J=I+1, N
              IF (IA1(J).LT. IA1(I)) THEN
                      ITMP=IA1(I)
                      IA1(I)=IA1(J)
                      IA1(J)=ITMP
              ITMP=IA2(I)
              IA2(I)=IA2(J)
              IA2(J)=ITMP
              END IF
              CONTINUE
20            END DO
10    END DO
      RETURN
      END
```

```
C     SUBPROGRAM DIBBORV
C     PURPOSE :  IT CREATES A FILE WITH THE BOUNDARY
C                OF A BINARY IMAGE ( VECTOR METHOD)
C                FOR DISPLAY.
C
C     INPUTS  :  FNAME, EXT, FILE TYPE ".PIC", FILE EXT
C                ".EXT"
C     OUTPUTS :  FILE EXT.  ".BRD"
C
C     VARIABLES
C     BLOCK   :  POINTER TO AN IMAGE BLOCK
C     BACK    :  BACKGROUND LEVEL
C     OBJ     :  OBJECT LEVEL
C
      SUBROUTINE DIBBORV(FNAME,EXT,SIZE,INIBLK,LASTBL)

      INTEGER BLOCK,SIZE,ROW,COL
      INTEGER WRK(512)
      INTEGER OBJ, BACK
      INTEGER X(1000),Y(1000)
      INTEGER IMAGE(65536),IMA(256,256)
      INTEGER IV(65536),IMA2(256,256)
      INTEGER A(512)
      INTEGER BA
      EQUIVALENCE (IMAGE(1),IMA(1,1))
      EQUIVALENCE (IV(1),IMA2(1,1))
      CHARACTER*16 FNAME,SAVFN
      CHARACTER*4 EXT
C
C     FNAME.EXT CONTAINS THE DATA TO BE PLOTTED
C
      SAVFN=FNAME       ! SAVE FNAME FOR RETURN
      OBJ= 255          !  REVERSE FOR BLACK BOUNDARY
      BACK=0            !  ON CLEAR BACKGROUND.
1     FORMAT(A16)

C     ASSUME EXTENSION "PIC"
      K=INDEX(FNAME,'.')
      IF(K.LE.0) K=INDEX(FNAME,' ')
      FNAME=FNAME(1:K-1)//'.PIC'

      OPEN(UNIT=1,NAME=FNAME,TYPE='OLD',ACCESS='DIRECT',
     1           READONLY,FORM='UNFORMATTED',RECORDSIZE=128)

C     OPEN OUT. FILE (CON EXT. BRD)
      FNAME=FNAME(1:K)//'BRD'
      OPEN(UNIT=2,NAME=FNAME,TYPE='NEW',FORM='UNFORMATTED',
     $    ACCESS='DIRECT',RECORDSIZE=128)

C     OPEN FNAME.EXT
      FNAME=FNAME(1:K-1)//EXT
```

```
      OPEN(UNIT=3, NAME=FNAME, TYPE='OLD',
     1            READONLY, FORM='UNFORMATTED')

      CALL     IPICRD(1,WRK,1,1)
      CALL IPICWT(2,WRK,1,1)

      DO 5 I=1,65536  ! INCREASE FOR "L" PICTURES
      IV(I)   =BACK
5     IMAGE(I)=BACK

      IF(EXT.EQ.'.CHL') THEN
C             READ HOLE INFO.
              I=1
              DO WHILE (.TRUE.)
              READ(3,END=30) X(I),Y(I)
              I=I+1
              END DO
30            N=I-1
                      ELSE
      READ(3) N,(X(I),Y(I),I=1,N)
      END IF

C     THEN STORE IN OUTPUT ARRAY
      DO 40 I=1,N
      BA=(X(I)-1)*SIZE
      IV(BA+Y(I))=OBJ
40    CONTINUE

100   DO 50 BLOCK=INIBLK, LASTBL
      I=(BLOCK-INIBLK)*512+1
      L=1
      DO 55 K=I,I+511
      WRK(L)=IV(K)
55    L=L+1
50    CALL IPICWT(2,WRK,BLOCK,1)
      CLOSE(UNIT=1)
      CLOSE(UNIT=2)
      CLOSE(UNIT=3)

      FNAME=SAVFN      ! RECOVER INPUT FNAME

      RETURN
      END
```

VITA

Renato Alberto Salinas was born in Temuco, Chile on April 29, 1950. He was graduated from the Liceo de Hombres No. 1 in December 1966. In March 1967 he entered the Universidad Tecnica del Estado, Temuco and transferred to the Santiago campus of the same university in March 1970, where he graduated with the degree of Ingeniero Civil Electricista in August 1974.

Since April 1974 he has been with the Electrical Engineering Department of the Universidad Tecnica del Estado, Santiago.

In 1982, he was awarded a scholarship from the Universidad de Santiago de Chile to pursue graduate studies in The University of Tennessee, Knoxville, USA, which he entered in the Fall of 1982.

The author is a member of Eta Kappa Nu and The Institute of Electrical and Electronics Engineers.

He will resume his work at the Universidad de Santiago de Chile upon completion of his graduate studies.