Image Processing & Robot Positioning,
The Programming of a Robot Vision System
on a Low-End Microcomputer

A Thesis Presented to
The Faculty of the College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirement for the Degree
Master of Science

by
Hong Chee Fung, B.S.
June, 1990
I would like to express my appreciation to my research advisor, Professor Roy A. Lawrence, for his guidance and patience, and valuable suggestions on this thesis. Appreciation is extended to my thesis committee members, Professors Joseph Essman, Kenneth Halliday, and Satyanrayana Raju, who have reviewed the thesis and suggested improvement.

I deeply thank my parents, Drs. Sho-Hoi Fung and Kim-Ying Wang. Without their support and understanding, my education in the United States and the accomplishment of this thesis might have been impossible.

And my sincere gratitude to my cousins, Dr. Casey Kin-Chee Fung and Linda Lee, for their consistent help and advice since my arrival to the United States.
Table Of Contents

1. Introduction  1
   1-1 Thesis Objective  1
   1-2 Robot Vision  5
   1-3 Background Information  6
      1-3-1 Image and Pixel  7
      1-3-2 Thresholding and Image Histogram  7
      1-3-3 Region, Object and Background  11
      1-3-4 Boundary  14
      1-3-5 Boundary Chain Code  15
      1-3-6 Object Properties  17

2. Existing Techniques  24
   2-1 Image Thresholding  24
   2-2 Edge Detection  30
   2-3 Image Segmentation  31
   2-4 Object Identification  34
   2-5 Object Position and Orientation  36
   2-6 Selected Algorithms  37
   2-7 Present Status of Machine Vision  38

3. Algorithms  43
   3-1 Threshold Determination  43
   3-2 Image Processing  47
      3-2-1 Boundary Detection  47
         3-2-1-1 Operation Frame  48
         3-2-1-2 Rough Search  49
         3-2-1-3 Fine Search  53
         3-2-1-4 Hole Boundary Search  55
         3-2-1-4 Summery of Boundary Search  57
      3-2-2 Boundary Tracking  58
         3-2-2-1 Tracking Directions  60
         3-2-2-2 Initial Tracking Direction  61
         3-2-2-3 Tracking Straight  63
         3-2-2-4 Making Turns  65
         3-2-2-5 Removing Boundary Noise  67
         3-2-2-6 Tracking on the Operation Borders  70
         3-2-2-7 Recording Tracking Directions  71
         3-2-2-8 Hole Boundary Tracking  73
         3-2-2-9 Boundary Tracking Termination  74
      3-2-3 Error Handling  75
      3-2-4 Marking Boundaries  78
      3-2-5 Recording Chain-Codes  78
      3-2-6 Computation of Object Moments  79
      3-2-7 Hole-Object Relation  80
      3-2-8 Parameter Arrangement  82
      3-2-9 Scale Factor  84
3-2-10 Determine Camera Elevation 86
3-3 Learning Objects 87
  3-3-1 Invariants for Comparison 88
  3-3-2 Objects in Comparison 89
  3-3-3 Tolerance for Comparison 90
  3-3-4 The Comparison 91
  3-3-5 Object Views 94
  3-3-6 Invariants Update 96
3-4 Object Identification 96
  3-4-1 Name Finding 97
  3-4-2 The Comparison 97
3-5 Robot Guidance 98
  3-5-1 Construction of Homogeneous Transformation Matrices 99
    3-5-1-1 Homogeneous Transformation Matrices 99
    3-5-1-2 Matrix of Object in Image-Frame Coordinates, \( C[T]_0 \) 100
    3-5-1-3 Matrix of Image Frame in Robot-Base Coordinates, \( B[T]C \) 101
    3-5-1-3 Matrix of Object in Robot-Base Coordinates, \( B[T]_0 \) 102
  3-5-2 Object Position in Robot-Base Coordinates 103
4. Experimental System and Example Operations 108
  4-1 System Description 108
  4-2 Equipment 110
    4-2-1 Digital Camera 111
    4-2-2 Frame Grabber 111
    4-2-3 Intel Microcomputer System 112
    4-2-4 Operation Programs 112
    4-2-5 Supporting Files 113
      4-2-5-1 Data Base and its Backup 113
      4-2-5-2 On-Scene Object Storage File 115
      4-2-5-3 Trace Files 117
  4-3 Example Operation 117
    4-3-1 Camera Calibration 118
      4-3-1-1 Aperture Calibration 118
      4-3-1-2 Position Calibration 122
    4-3-2 Image Processing 124
    4-3-3 System Training 128
    4-3-4 Object Identification and Positioning 133
  4-4 System Reliability 136
5. Conclusion 139
  5-1 System Evaluation 139
  5-2 Future Improvement 141
References 143

Appendix A. Operation Manual
  A-1 Running Program IP  A-2
A-1-1 Camera Calibration  A-2
A-1-2 Change Operation Parameters  A-6
A-1-3 Process On-Scene Image  A-8
A-2 Running Program TP  A-10
  A-2-1 Training  A-11
  A-2-2 Rearrange Data Base  A-16
  A-2-3 Determine Object Position  A-19

Appendix B. Subroutine List
Appendix C. Flow Charts
Appendix D. Table of Variables
Appendix E. Program Listings
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
</table>
| Figure 1.1 | The Robot Vision System  
(a) Hardware Setup  
(b) Algorithm | 2, 3 |
| Figure 1.2 | A Portion of an Image | 8 |
| Figure 1.3 | An Image with 64 Gray Levels | 9 |
| Figure 1.4 | A Binary Image | 9 |
| Figure 1.5 | A Histogram of an Image | 11 |
| Figure 1.6 | The Image of a Background, Two Objects, and a Hole | 13 |
| Figure 1.7 | Background Pixels Are Always on the Right | 15 |
| Figure 1.8 | Tracking Direction of an Object Boundary | 16 |
| Figure 1.9 | 4-Direction and 8-Direction System | 17 |
| Figure 1.10 | Object Location and Orientation | 19 |
| Figure 3.1 | Determine Threshold on an Image Histogram | 45 |
| Figure 3.2 | Grids and Nodal Points of an Image | 46 |
| Figure 3.3 | Operation Frame within the Image Frame | 49 |
| Figure 3.4 | A Left-Right, Top-Down Rough-Search Pattern | 50 |
| Figure 3.5 | Rough Search for Boundaries | 51 |
| Figure 3.6 | An Object Lies over the Left Border | 52 |
| Figure 3.7 | Fine Search for Boundaries | 54 |
| Figure 3.8 | Detecting Marked Boundaries | 55 |
| Figure 3.9 | The Hole Boundary Which Lies over Operation Border | 56 |
| Figure 3.10 | Summery of Boundary Search | 58 |
| Figure 3.11 | Tracking Object Boundary | 59 |
| Figure 3.12 | Tracking Direction at Last Tracked Boundary Pixel | 60 |
| Figure 3.13 | Determine Initial Tracking Direction | 62 |
| Figure 3.14 | Start Tracking on Left Operation-Border | 63 |
| Figure 3.15 | Tracking Straight | 64 |
| Figure 3.16 | Left Turns of Tracking Direction | 65 |
| Figure 3.17 | Making Left Turns | 66 |
| Figure 3.18 | Right Turns of Tracking Direction | 66 |
| Figure 3.19 | Tracking Runs into a Dead End | 67 |
| Figure 3.20 | Bump Noise Detection | 68 |
| Figure 3.21 | Detect and Remove Bump Noise | 69 |
| Figure 3.22 | Hole Noise Detection | 69 |
| Figure 3.23 | Detect and Remove Hole Noise | 70 |
| Figure 3.24 | An Object and its Boundary Chain-Code | 72 |
| Figure 3.25 | Object Boundary Tracking vs. Hole Boundary Tracking | 73 |
| Figure 3.26 | Boundary Tracking Termination | 74 |
| Figure 3.27 | Boundary Length-Unit | 80 |
| Figure 3.28 | Determine Hole-Object Relation | 81 |
| Figure 3.29 | Scale Factor Correction | 85 |
Figure 3.30 Determine Camera Elevation
Figure 3.31 Invariant Comparison for Object Identification
Figure 3.32 Different Views of a Bolt
Figure 3.33 An Object in Robot-Base Coordinates
Figure 3.34 Communication Channel between Microcomputer System and Robot Controller

Figure 4.1 Grabbed Image
Figure 4.2 Binary Image with Average Gray Level as Threshold
Figure 4.3 Binary Images with Threshold
Figure 4.4 Objects to be Processed
Figure 4.5 Boundaries of Processed Objects
Figure 4.6 The Selected Object
Figure 4.7 Four Objects for Training
Figure 4.8 An On-Scene BOLT and a Recorded BOLT
Figure 4.9 Object Identification
Figure 4.10 RECTANGLE is Located
Chapter 1

Introduction

1-1 Thesis Objective

The objective of this thesis is to set up and program a complete robot vision system on a low-end microcomputer. This vision system should be able to search and identify arbitrary objects from different views, as well as to pinpoint the location and orientation of the recognized object in the coordinate system of a robot. Some of the existing techniques are reviewed and applied to the vision system in the areas of image thresholding, boundary tracking, geometrical property computation, and object recognition. The unique contribution of this research is to apply and modify these techniques to provide better performance in terms of simplicity, speed, and reliability.

The robot vision system utilized for this work is shown in Figure 1.1. The central piece of the system is a microcomputer system. Other components include a solid state camera, a frame grabber, a graphic monitor, and a
Figure 1.1(a) The Robot Vision System: Hardware Setup.
Figure 1.1(b) The Robot Vision System: Algorithm.
robot arm with six degrees of freedom. Detail description of the vision system is given in Chapter 4.

*Simplicity* is one of the key requirements to the robot vision system. The system will require no special illumination. The camera aperture is adjusted to accord with the strength of the lighting on the camera viewing window. A global threshold value is determined by the system to distinguish objects from their background without complex evaluation on images.

*Speed* is the other key requirement to the vision system. The system requires only one pass on a raw image to obtain all desired information about the objects within the camera viewing window. Most operations focus on object boundaries. The processing speed is maximized by minimizing the number of operations required to search for the boundaries of objects and holes.

The vision system is able to detect and remove image noise from the background and the boundaries of objects and holes. Also, the system is able to function in difficult situations, such as imperfect images.

Some *learning* abilities are introduced to the vision system. The system computes some essential plane-area
geometrical properties of the objects. Some of these properties are then modified and normalized into invariants. The system can be trained to memorize objects by storing the invariants into the computer memory as a database. The recorded invariants can be recalled and utilized to differentiate and identify objects. The recorded invariants can also be updated.

Also, the vision system can pinpoint the location and orientation of an recognized object in the robot base coordinates. The information about the determined object location and orientation is output as a command string in the format which can be understood by the robot controller. The camera is not be required to be at a fixed position. The system automatically calibrates the camera position.

1-2 Robot Vision

The purpose of robot vision is to perform robot positioning under the guidance of a machine vision. A robot can position its hand (end-effector or gripper) or even itself to a desired location with a desired orientation. By giving specified spatial coordinates (X-Y-Z) along with up to three orientation angles, a robot can be guided to the position by a set of programs stored in
its controller. *Machine vision*, also called computer vision or intelligent vision, simulates human eye-brain system to observe and understand the situation of the surrounding world. Robot vision combines the functions of robot control and machine vision. In other words, robot vision can position the robot end-effector according the visual information extracted from the surroundings of the robot.

Robot vision may be defined as the process of extracting, characterizing, and interpreting information from images of a three-dimensional world (p296, Fu 1987). The whole process may have six subprocesses: sensing, preprocessing, segmentation, description, recognition, and interpretation. Sensing yields a visual image; preprocessing reduces noises and enhances details; segmentation partitions image into objects of interest; description computes object features to differentiate objects from each other; recognition identifies objects; and interpretation gives meaning to the recognized objects.

1-3 Background Information

Some key technical terms used in the discussion of robot vision are defined below for the better understanding
of this thesis. They are image and pixel, image histogram and thresholding, region, object and background, boundary, boundary chain code, and object properties.

1-3-1 Image and Pixel

An image is a two-dimensional representation which carries visual information of a three-dimensional scene [Sanderson 1983]. It is formed by an image device, e.g., a camera, within the viewing window of the device. A digitized image appears as a matrix which consists of a number of rows and columns of pixels, as shown in Figure 1.2. A pixel is a picture element which holds some information about a unit area within the viewing window, such as brightness or color. In the case of colorless images (Figure 1.3), a pixel represents the average gray level of the unit area. In a binary image (Figure 1.4), a pixel can only be one of two gray levels, black or white.

1-3-2 Thresholding and Image Histogram

Object detection by threshold determination is the method selected for this thesis. Other techniques are discussed in Chapter 2. A threshold value is used to
Figure 1.2 A Portion of an Image. Each Number Represents the Gray Level on the Particular Pixel.
Figure 1.3 An Image with 64 Gray Levels.

Figure 1.4 A Binary Image.
discriminate the objects from their background. Image pixels can be differentiated into object pixels or background pixels by comparing their gray levels with the threshold. When the background is brighter than the objects, the differentiation rule is: if a pixel has a gray level, \( G \), greater than the threshold, \( TH \), it is a background pixel; otherwise it is an object pixel [Snyder 1985].

\[
G = \begin{cases} 
\text{background pixel} & \text{if } G > TH \\
\text{object pixel} & \text{if } G \leq TH 
\end{cases}
\]

Furthermore, the gray levels which are greater than the threshold are generally called the above-threshold gray levels, and the gray levels less-than or equal-to the threshold is called the below-threshold gray levels [Tsai 1984].

The determination of the threshold can be based on the histogram of an image. The histogram is a chart with number of pixels (vertical axis) plotted against gray levels (horizontal axis), as shown in Figure 1.5. In most of the cases, the histogram contains two peaks and a valley between the peaks. The two peaks represent the majorities of the object pixels and background pixels, respectively. The valley represents the pixels with the immediate gray levels between the gray levels on the object pixels and
background pixels. The whole purpose of thresholding is to find a gray-level value corresponding to the valley [Snyder 1985].

![A Histogram of an Image with 64 Gray Levels.](image)

**Figure 1.5** A Histogram of an Image with 64 Gray Levels.

1-3-3 Region, Object and Background

A region is defined as an area with a relatively uniform color or gray-level value [Ballard 1982]. It is recognized if its gray level is significantly different from the gray level of its surrounding region or regions.
Objects locate on their background. The background is assumed to be the largest region having a relatively uniform gray level through out the entire image field. On two-dimensional images, the objects appear as small plane regions lying on the large background region with the different gray levels from the background (Figure 1.6). A significant gray-level difference, or contrast, between the objects and their background is necessary to distinguish the objects from their background.

An object is always surrounded by its background and is smaller than the background. Similarly, a hole is surrounded by the object and is smaller than the object. In terms of image processing, the hole can be treated as an object even though it has the above-threshold gray level. And, the host object of the hole can be treated as the background of the hole since it actually surrounds the hole. Furthermore, the pixels in the background region can be called the background pixels which have the above-threshold gray level. The pixels in the object region can be called the object pixels which have the below-threshold gray level. The pixels in the hole region can be called the hole pixels which have the above-threshold gray level. The pixels in the region of the host-object of the hole can
be called the hole-background pixels which have the below-threshold gray level.

![Diagram of a Background, Two Objects, and a Hole](image)

**Figure 1.6** The Image of a Background, Two Objects, and a Hole.

When binary thresholding is applied as mentioned in section 1-2-2, there is one *global* threshold for the entire image. In two-dimensional images, each pixel can only have a gray level either greater than the threshold or less-than-or-equal-to the threshold. Any *overlapping* objects will appear as a single region because all object pixels have the below-threshold gray level. It is impossible to distinguish the overlapping objects. Therefore,
overlapping objects are avoided in two-dimensional images with the binary thresholding.

1-3-4 Boundary

A boundary is a closed path separating one region from another. An object boundary is a separating path between the object region and the background. Similarly, a hole boundary is a separating path between the hole and the object (see Figure 1.7).

When following an object boundary in counter-clockwise direction, the object pixels are always on the left side of the boundary and background pixels are always on the right side. This argument is also valid for hole boundaries. The hole pixels are always on the left side of the hole boundary and the object pixels are always on the right side (Figure 1.7). Consequently, the hole boundary can be treated as an object boundary in terms of image processing. The only difference is the hole pixels actually have the above-threshold gray level, and the pixels on the background of the hole, i.e., the pixels on the host object, has the below-threshold gray level.
Figure 1.7 Background Pixels Are Always on the Right when Facing Counter-Clockwise on a Boundary. The Host-Object Can Be Treated as the Background of the Hole.

1-3-5 Boundary Chain Code

The boundary of an object or a hole can be recorded as a chain code [Wilf 1981]. The chain code contains sufficient information for representing the boundary and deriving the geometrical properties of the enclosed region. The chain code can be a series of locations or tracking directions of the boundary pixels. The location of a boundary pixel is given by the X-Y coordinates. The tracking direction of a boundary pixel is given by a
numerical code which represents the direction from the boundary pixel to next boundary pixel, as shown in Figure 1.8.

A tracking direction can be one of the four or eight directions, depending on which system is employed. The four direction system consists of the up, down, left and right directions. The eight direction system consists of four additional directions, the up-left, the up-right, the down-left, and the down-right. The tracking directions are labeled counter-clockwise for the convenience of generating the chain code, though they can be arranged arbitrarily. Figure 1.9 shows the arrangement of the tracking directions of a four-direction system and an eight-direction system.
1-3-6 Object Properties

The object properties include physical, spatial, and geometrical properties. The physical properties of an object, such as weight, density, and hardness, are disregarded in image processing. One exception is the light reflectivity of the object surface which plays a very important role on image thresholding. On the other hand, the spatial and geometrical properties of an object are very much of interest in positioning and identifying objects, respectively. The spatial location and the
orientation of an object are the parameters needed to position the object. These quantities are subject to variation, depending on the spatial position of the object. The geometrical invariants of an object, such as the area, perimeter, principal values and the number of holes, are the parameters needed to identify the object. These invariants do not vary because of the change of the object position. In two-dimensional case, however, an object can have several view images with different shapes, depending on the way the object is placed. Therefore, different view of an object will have different set of invariants.

The spatial and geometrical properties of an object are computed during the image processing. Some of the properties are derived directly from the construction of the boundary chain code of the object. The perimeter is the length of the chain code. The number of holes in an object is determined by finding the object boundary which surrounds the hole. Other properties are calculated by the integration either over the object region with plane surface integration, or along the object boundary with plane closed-path integration.

The location of an object is given by the coordinates of its centroid \((x_c, y_c)\) (Figure 1.10). These coordinates
are decided by the static moments, $M_x$ and $M_y$, as well as the area ($A$).

$$x_c = \frac{\Sigma M_y}{A} \quad \text{(Eq.1-1)}$$

and

$$y_c = \frac{\Sigma M_y}{A} \quad \text{(Eq.1-2)}$$

The orientation of the object can be indicated by the orientation angle ($\beta$) between the principal axes and the $x$-coordinate axis. This angle is determined from the moments of inertia, $I_{xx}$, $I_{yy}$ and product of inertia $I_{xy}$.

$$\beta = \frac{1}{2} \tan^{-1} \frac{2I_{xy}}{I_{yy} - I_{xx}} \quad \text{(Eq.1-3)}$$

where $-90^\circ \leq \beta \leq 90^\circ$.

Figure 1.10 Object Location and Orientation.
The object area, static moments, moments of inertia, and product of inertia are generally called the object moments. The integrals of these moments are as follows [Tuma 1979]. By the definition in plane surface integration, the area of an object is:

\[ A = \iint_{A} dA \quad \text{(Eq.1-4)} \]

where \( dA = dx\,dy \). The Static moments of an object are:

\[ M_x = \iint_{A} y \, dA \quad \text{(Eq.1-5)} \]

and

\[ M_y = \iint_{A} x \, dA \quad \text{(Eq.1-6)} \]

The moments of inertia of an object are:

\[ I_{xx} = \iint_{A} y^2 \, dA \quad \text{(Eq.1-7)} \]

\[ I_{yy} = \iint_{A} x^2 \, dA \quad \text{(Eq.1-8)} \]
The product of inertia of an object is:

\[ I_{xy} = \iint_A xy \, dA \quad \text{(Eq. 1-9)} \]

The foregoing plane surface integrals can be reduced to plane closed-path integrals. These integrals can be determined during the boundary tracking while the chain code is being generated. The converted closed-path integrals are: [Wilf 1981]

\[
A = \frac{1}{2} \sum_{L=1}^{n} AL
\]

\[
M_x = \frac{1}{3} \sum_{L=1}^{n} AL (y_{L-1}/2D_y)
\]

\[
M_y = \frac{1}{3} \sum_{L=1}^{n} AL (x_{L+1}/2D_x)
\]

\[
I_{xx} = \frac{1}{4} \sum_{L=1}^{n} AL (y_{L-1} - yLD_y + 1/3D_y^2)
\]

\[
I_{yy} = \frac{1}{4} \sum_{L=1}^{n} AL (x_{L-1} - xLD_x + 1/3D_x^2)
\]

\[
I_{xy} = \frac{1}{4} \sum_{L=1}^{n} AL (x_{L-1} y_{L-1}/2xLD_y - 1/2yLD_x + 1/3D_xD_y)
\]

where \( D_x = x_L - x_{L-1} \); \( D_y = y_L - y_{L-1} \); \( AL = xLD_y - yLD_x \), for any \( 1 \leq L \leq n \). The integer "n" is the length of the chain code.
The above integrals subject to the coordinates system of the image frame (XOY). In order to be object invariants, these integrals must be transferred to the coordinates system (XcYc) whose origin is at the object centroid and whose axes are parallel to the principal axes of the object (see Figure 1.10). To do so, the origin of the image-frame coordinates is transformed to the centroid of the object. This results in:

\[ c_{I_{xx}} = I_{xx} - Ay_c^2 \]
\[ c_{I_{yy}} = I_{yy} - Ax_c^2 \]
\[ c_{I_{xy}} = I_{xy} - Ax_cY_c \]  

(Eq.1-11)

Then, the coordinates are rotated by the orientation angle, \( \beta \), which produces.

\[ o_{cI_{xx}} = c_{I_{xx}}\cos^2\beta + c_{I_{yy}}\sin^2\beta - c_{I_{xy}}\sin2\beta \]
\[ o_{cI_{yy}} = c_{I_{xx}}\sin^2\beta + c_{I_{yy}}\cos^2\beta + c_{I_{xy}}\sin2\beta \]  

(Eq.1-12)

\[ o_{cI_{xy}} = \frac{c_{I_{xx}} - c_{I_{yy}}}{2} \sin2\beta + c_{I_{xy}}\cos2\beta \]

As the result of the above transformation and rotation, the moments and product of inertia of the object are normalized. The values of \( o_{cI_{xx}}, o_{cI_{yy}}, o_{cI_{xy}} \), as well as the two principal values, \( I_1 \) and \( I_2 \) will then be the invariants of the object, irrespective of its location and
orientation. The \textit{principal values} are:

\[ I_{1,2} = \frac{cI_{xx} + cI_{yy}}{2} \pm \frac{1}{2} \left[ \left( (cI_{xx} - cI_{yy})^2 + 4cI_{xy}^2 \right) \right]^{1/2} \] (Eq.1-13)

The normalized moments and product of inertia are equivalent to the principal values. Therefore, only one set of the values is needed in the object identification.
Chapter 2

Existing Techniques

In this chapter, several existing algorithms will be reviewed in the areas of image thresholding, region segmentation, and object recognition. Some of the algorithms are picked to meet the design requirements of the purposed robot vision system. Also, the present status of machine vision will be sketched.

2-1 Image Thresholding

The threshold technique is usually based on the image histogram analysis. The main idea of histogram-based thresholding is to determine the valley between two peaks, each of them representing the majority of the object pixels or the background pixels (see Figure 1.5).

J. N. Kapur [Kapur 1984] introduced an entropic thresholding method, the probability distribution algorithm, to convert gray-level images to binary images.
He defined $p_1$, $p_2$, ..., $p_n$ to be the probability distribution of gray-levels from 1 to $n$. From this probability distribution, he then derived two probability distributions. One was defined for discrete values 1 to $s$ and the other for $s+1$ to $n$.

\[
A: \frac{p_1}{P_s}, \frac{p_2}{P_s}, \ldots, \frac{p_s}{P_s}
\]

\[
B: \frac{p_{s+1}}{1-P_s}, \frac{p_{s+2}}{1-P_s}, \ldots, \frac{p_n}{1-P_s}
\]

The entropies associated with each distribution are as follows:

\[
H(A) = -\sum_{i=1}^{s} \frac{p_i}{P_s} \ln \frac{p_i}{P_s}
\]

\[
= -\frac{1}{P_s} \left[ \sum_{i=1}^{s} p_i \ln p_i - P_s \ln P_s \right]
\]

\[
= \ln P_s + \frac{H_s}{P_s}
\]

\[
H(B) = -\sum_{i=s+1}^{n} \frac{p_i}{1-P_s} \ln \frac{p_i}{1-P_s}
\]

\[
= -\frac{1}{1-P_s} \left[ \sum_{i=s+1}^{n} p_i \ln p_i - (1-P_s) \ln(1-P_s) \right]
\]

\[
= \ln(1-P_s) + \frac{H_n-H_s}{1-P_s}
\]
He obtained:

\[
\Phi(s) = H(A) + H(B)
\]

\[
= \ln(1-P_s) + \frac{H_s}{P_s} + \frac{H_n-H_s}{1-P_s}
\]

The discrete value \(s\) which maximizes \(\Phi(s)\) is the threshold value.

The advantage of the algorithm is that it uses a global and objective property of the histogram. Because of its general nature, this algorithm can be used for segmentation purposes. Kapur applied the algorithm to some real-time images, such as the photographs of buildings and people. The experimental results showed that the method can automatically give a global threshold value which separates the image pixels into two groups, black and white. However, the algorithm takes a lot of computation time because some high-level calculation is involved. Furthermore, in some cases, more than one discrete value \(s\) could yield maximum \(\Phi(s)\), a choice had to be made among these discrete values by means of a suitable trade-off.

Wen-Hsiang Tsai [Tsai 1984] suggests a moment-preserving thresholding process. In an image \(f\) with \(n\) pixels whose gray value at pixel \((x,y)\) is denoted by \(f(x,y)\), Tsai defined the \(i\)th moment \(m_i\) of \(f\) as
\[ m_i = (1/n) \sum_{x} \sum_{y} f^i(x,y), \quad i = 1, 2, 3, \ldots \]

He computed the moments in the following way:

\[
m_i = (1/n) \sum_{j} n_j \cdot Z_j^i
\]

\[= \sum_{j} p_j \cdot Z_j^i\]

where \( n_j \) is the total number of the pixel in \( f \) with \( j \)th gray level which value is \( Z_j \), and \( p_j = n_j/n \). In addition, \( m_0 \) is defined as 1. For bilevel thresholding, the first four moments from the original image \( f \) are:

\[ m_0 = 1 \]

\[ m_1 = \sum_{j} p_j \cdot Z_j^1 \]

\[ m_2 = \sum_{j} p_j \cdot Z_j^2 \]

\[ m_3 = \sum_{j} p_j \cdot Z_j^3 \]

The image \( f \) is converted to a binary image \( g \) which has two representative gray values, \( z_0 \) and \( z_1 \). The \( z_0 \) will replace all below-threshold gray values in \( f \) and \( z_1 \) will replace all above-threshold gray values. The first four moments of the binary image \( g \) are:
\[ m'_{0} = 1 \]

\[ m'_{1} = \sum_{j=0}^{1} p_j z_j^1 \]

\[ m'_{2} = \sum_{j=0}^{1} p_j z_j^2 \]

\[ m'_{3} = \sum_{j=0}^{1} p_j z_j^3 \]

where \( p_0 \) and \( p_1 \) denote the fractions of the below-threshold pixels and the above-threshold pixels in \( f \), respectively.

Preserving the first four moments in \( g \) means the following equalities:

\[ m'_{i} = m_{i}, \quad i = 0, 1, 2, 3. \]

They are equivalent to

\[ p_0 z_0^0 + p_1 z_1^0 = m_0 \]
\[ p_0 z_0^1 + p_1 z_1^1 = m_1 \]
\[ p_0 z_0^2 + p_1 z_1^2 = m_2 \]
\[ p_0 z_0^3 + p_1 z_1^3 = m_3 \]

Solving the above equations will give \( p_0 \) and \( p_1 \), as well as \( z_0 \) and \( z_1 \). Then the threshold \( t \) was chosen when

\[ p_0 = (1/n) \sum_{z_j \leq t} n_j \]

Tsai's approach gave very good results to the sample images of the size 80x60. Like Kapur's algorithm, this method
values are computed deterministically in such a way that the moments of an input picture is preserved in the output picture. In addition, Tsai's method can provide a representative gray value for each group of pixels. This algorithm, however, is also quite time consuming since some complicated calculation is included.

Choosing a threshold by gray-level average may be the simplest approach but still effective [Snyder 1985]. The threshold \((TH)\) is peaked by adding some increment \((SH)\) to the average gray level \((AV)\) of the image, \(TH = AV + SH\). This algorithm requires no complicated calculation. It therefore is the fastest. In addition, it provides an option of sampling some of the pixels in an image instead of taking all pixels into consideration. The increment, \(SH\), is an experimental parameters. It varies with the characteristic of the vision system and the illumination condition over the scene. The major drawback of this method is that the average gray-level value heavily depends on the amount of objects in the camera viewing window.
2-2 Edge Detection

Another pre-processing technique which is not based on image histogram is edge detection. The edge detectors look for local edges which are small areas in the image where the local gray levels are changing rapidly. Some well-known edge detectors are Gradient, Sobel, Laplacian, and Kirsch [Ballard 1982]. Gradient and Laplacian are the most common, and historically earliest, edge operators. They compute the gradient magnitude \( s(x) \) and direction \( \phi(x) \) from the image function \( f(x) \). These edge detectors map the local edges by sensing rapid changes in the local gray levels, where the gradient magnitudes are greater than a threshold value. The detected edges can then be utilized for region segmentation to construct object boundaries.

Edge detection can recognize all kinds of edges, as long as there is significant gray-level change. Because of this, the edge detector can detect not only the edge between objects and background, but the edges between overlapping objects as well. However, edge detectors can be sensitive or insensitive to the gray level-change, depending on the threshold setting. Some of the edge detectors are sensitive to the gray-value variations in the images. As a result, the detectors can discover most edges and pick up a lot of noise as well. By contrast, other edge detectors
are insensitive to the gray-value variations. They may avoid noise but lose some edges. In addition, these edge detection operators usually require more than one examination or operation on each pixel in the image. For example, a 3 by 3 mask Laplacian operator requires operation on each image pixel at least nine times. And, the edge detection operation itself requires at least one pass over the image to detect edges before applying other image processing procedures, such as edge linking or boundary detection. Therefore, edge detection operation is relatively slow when comparing with the global thresholding techniques.

2-3 Image Segmentation

Region growing and boundary tracking are two of the most basic segmentation techniques. The region growing labels individual pixel of an image and groups the same labeled pixels together into a region. The boundary tracking focuses its interests on the boundaries of the regions.

Wesley E. Snyder [Snyder 1985] explained the algorithm of region growing. He utilized a label memory corresponding to every pixel in an image. A region starts
growing with an unlabeled object pixel. Before the object pixel is labeled, its nearest neighboring pixels are examined. If any of the neighboring pixels is an object pixel and unlabeled, it will be put onto an stack. The stack is a numerical array in computer memory. The stack temporarily stores the pixels, i.e., the pixel locations and gray levels. The last pixel to be put onto the stack will be the first pixel to be taken out. A new object pixel is chosen by popping the stack. The region growing continues until the stack is empty. The same operation is restarted for a new object region and repeated for the entire image. This algorithm is simple, straightforward and is easy to program. The calculation of the geometric properties of the object regions is based on area integration which can be achieved during the region labeling. However, some important object properties cannot be obtained directly from region growing. For example, the perimeter of an object has to be estimated by some additional work, such as detecting and connecting the boundary pixels. Determining the number of holes in an object also needs extra work to find out the relationship among the regions. Additionally, the region-growing algorithm has to consider all the background pixels and object pixels in the image. It, therefore, consumes more time than the boundary-tracking algorithm described below.
Another popular algorithm for image segmentation is *boundary tracking*. This algorithm focuses only on the boundaries of the objects or holes. The algorithm generates a chain code for each boundary. Joel M. Wilf [Wilf 1981] suggested an algorithm which requires three passes over the entire image. In the first pass, he smoothed the image by removing isolated pixels, which were considered as noise. In the second pass, he used a Sobel-type three by three mask to detect edge pixels by calculating the gray-level difference between each pixel and its neighboring pixels. In the third pass, he again used a three by three mask to chain-up the edge pixels by mask pattern. This three-passes operation obviously took a lot of computer time. Besides the chain-coding algorithm, Wilf introduced the closed-path integration to calculate the geometrical properties of the object region (see Eq. 1-10). Of these, the closed-path integration is the most useful contribution to the boundary tracking algorithm.

Robert Cunningham [Cunningham 1981] presented a boundary tracking algorithm for segmenting binary images. He employed 2x2 windows to determine the position of the next boundary pixel when traveling around the object boundary. He used direction codes to record the tracked boundaries. This algorithm is quite efficient. However, Cunningham only applied the algorithm to binary images.
And, he did not consider the noise pixels on the boundaries.

2-4 Object Identification

An unknown object can be identified by matching the characteristics of the unknown object to those of known objects. The key issue here is to choose some object characteristics which can be easily obtained and are independent of object orientation and position. These object characteristics often fall into two major categories, contours and plane properties.

In the contour category, Keith E. Price [Price 1984] suggested matching objects by line segment length and the angle between the consecutive lines. An object boundary was transformed into a sequence of straight lines. Each line had a length and orientation. An object was identified by matching these lines after a search for right sequence of match. One of the other contour-match methods is the CAD-model-based match. It automatically constructs vision models and matching procedures from CAD models of objects and from knowledge of the environment in which the vision tasks is to be performed [Lu 1988]. This method can
execute 3-D object recognition from different directions. This algorithm needs a CAD-model for each object.

Corner matching is another popular method in object recognition. It examines the relative positions of the corners on the object boundary. This method requires that the objects have sharp corners which may not be true for all objects with arbitrary shapes, such as a circular object. In addition, corner detection also needs a great deal of calculation for an arbitrary object even it has sharp corner [Zuniga 1983].

In the plane property category, the features utilized in the object identification include the moment invariants and geometrical features. Joel M. Wilf [Wilf 1981] suggested the moment invariants as the templates to be matched. The moments of an unknown object are calculated from a region growing or a boundary tracking during the image segmentation. After several stages of transformation, these moments become the invariants for the individual object irrespective of the object location, orientation and scale. These invariants are then compared with the previously recorded invariants of known objects. The unknown object is recognized if a match is achieved.
W. E. Snyder [Snyder 1985] suggested that the geometrical features be used as the characteristics to identify objects. Some of these features, such as the object area and principal values, come from the image segmentation by either area integration in region growing or closed path integration in boundary tracking. Other object properties, such as the perimeter and number of holes, are determined by adding some extra computation.

2-5 Object Position and Orientation

The position and orientation of an object have to be determined before guiding the robot hand toward the object. The object is detected and recognized in the image-frame coordinates but the object actually locates in the robot-base coordinates. Therefore, the object position and orientation need to be transferred from the image-frame coordinates to the robot-base coordinates. The most basic and common method is the application of the homogeneous transformation matrices. One matrix is determined to represent the object position and orientation in the image-frame coordinates. Another matrix is determined to represent the image-frame position and orientation in the robot-base coordinates. The matrix representing the object
position and orientation in the robot-base coordinates is the product of the first matrix by the second matrix.

2-6 Selected Algorithms

Some of the algorithms reviewed above are selected to build the robot vision system according to the available equipments and the requirements of the thesis. Some algorithms are modified to meet the actual condition and limitation of the system. A major criterion is to minimize the average number of operations required on each pixel.

The average gray-level technique is employed for the thresholding because it is simple and effective, and it provides an option to sample regions of the image instead of taking all pixels of an image into consideration. The amount of object within the camera viewing window is controlled, but only for the threshold determination. The boundary tracking algorithm is chosen for image segmentation because it limits the operations only on the boundaries. The algorithm is applied directly to the raw image to obtain the chain codes of object and hole boundaries without previous edge detection operations. The combination of a rough search and a fine search can reduce the number of operations required to locate the boundaries.
Closed-path integration is utilized to compute the object moments since it can be proceed at the same time when the boundaries are being tracked. The object properties, including its area, perimeter, minimum and maximum principal values, and number of holes, are used in the object identification since they require no complicated calculation and can be normalized into invariants based on the basic transformation and rotation of coordinates. The homogeneous transformation matrices are employed to transfer the object position and orientation from the image-frame coordinates to the robot-base coordinates.

2-7 Present Status of Machine Vision

The development of machine vision systems in industry has made tremendous strides since the work on which this thesis is based was undertaken. A number of machine vision systems are available for the application of robot guidance and automatic inspection.

SIEMENS Grey-Scale Sensor (GSS) by Corporate Research and Technology at Siemens AG, F. R. Germany is a model-based robot vision system used under real factory conditions [Rummel 1988]. The GSS can recognize touching or overlapping workpieces under variable illumination.
conditions in 0.5 to 2 seconds. The recognition is accomplished by matching a geometrical model of a workpiece with a set of features extracted from the scene to optimize a similarity measure. These features, including corners, straight lines and circular arcs, are extracted from the gray-scale image of the scene by computing gradient, extracting lines from the gradient image, splitting the lines into segments, and parameterizing the segments into features. A workpiece is considered as recognized if all features of the model have been found or estimated and a similarity threshold is exceeded. The GSS can yield an accuracy of ±1 pixels and ±1"...±2" with an image resolution of 256 x 256 pixels. However, the GSS has difficulties in recognizing flexible parts or parts without well-defined edges.

One US-made robot vision system is UNIVISION I by UNIMATION Inc. at Danbury, Connecticut [SME 1984]. UNIVISION I is an advanced vision system for UNIMATE PUMA robots made by the same company. The system can be trained to recognize the maximum of 20 different objects with up to 60 total parts in the scene at one time. For each part, 13 distinguishing features can be generated, including area, perimeter, center of gravity, number of holes, and maximum and minimum radii. Accuracy of UNIVISION I is 0.5% of the viewing window. For best result, UNIVISION I requires high
contrast between parts and background, and the parts being separated.

Machine vision is also widely used in automatic inspection. In fact, inspection has been and will be the dominant application area for machine vision [Gorog 1989]. A typical inspection system is Cognex 2000/DS machine vision development system by Cognex Corporation at Needham, Massachusetts [Cognex 1987]. The Cognex 2000/DS can be applied to inspection, gauging, machine guidance, and character recognition in the automotive and electronics industries. The system provides high measurement repeatability by utilizing gray-level information rather than binary information and, therefore, the processed image is a true picture of the actual object with arbitrary size and complexity.

The excellent performance of machine vision systems is due to the utilization of the advanced algorithms, cameras, illumination, and especially, the vision coprocessors. The SIEMENS GSS mentioned above carries out the steps of gradient computation, line following, and the first step of line-segmentation in a dedicated hardware. The heart of the Cognex 2000/DS is the Cognex 2000/VP single-board vision processor which contains an image digitizer and a vision coprocessor. As a result, the vision tasks can be
executed very well and rapidly. Other specialized processors, such as DT1458 Auxiliary Frame Processor by Data Translation Inc. at Marlboro, Massachusetts [Data 1988] and Oculus-150 by Coreco Inc. at Longueuil, Québec, Canada [Coreco 1987], consist of an arithmetic and logical unit (ALU). These processors are able to perform Laplacian or Sobel operation and detect edges based on a convolution[1].

Advanced algorithms can also improve system performance. For example, sub-pixel techniques increase the accuracy of precise measurements. Cognex 2000 is able to determine the object’s x,y position within the image with an accuracy of 1/2 to 1/10 pixel.

The combination of customized camera and illumination can simplify the image processing and, therefore, can achieve a very fast machine vision operation. The Q-Scan On-Line X-ray Inspection System by Par Microsystems Corporation at New Hartford, New York is capable of inspecting glass and metal packages of various sizes and shapes at rates up to 15 containers per second [Cambier 1986]. Q-Scan achieves its processing speed by employing a pulsed X-ray tube and a vidicon-type camera, and by using a

[1] Convolution - A technique for implementing filters and edge detection on an image by multiplying the values of a group of pixels by the values of an overlay array of values, which is called a kernel. The products for each group are summed and used to construct a new image [Schaffer 1984].
high-speed programmable coprocessor for all image analysis. The coprocessor is the ZIP3216 by Mercury Computer System Inc. at Lowell, Massachusetts. CONSIGHT vision system by General Motors uses structured lighting and a linear array camera to detect objects on a moving conveyor belt [Zuech 1987].

The major setback on promoting machine vision application is that the market-available machine vision systems are still quite expensive and usually require specialists to program and operate. Even though there is still a lot research in more advanced technology, more effort should be done on low-cost, reliable, easy to apply, and relatively simple vision systems. We should see in the near future a steady increase in the installation of technically effective and economically sound machine vision systems [Freeman 1989].
Chapter 3

Algorithms

This chapter describes in detail the algorithms selected in last chapter. Modifications of these algorithms are also indicated when they are applied.

3-1 Threshold Determination

A global threshold value for the entire image is determined in order to perform image processing with a binary approach. The image gray-level average is employed to estimate the threshold value. This algorithm is simple and effective. The threshold determination operation is needed only for calibration of the camera aperture. After the threshold value is properly set, the threshold determination is not required for every individual image in the image processing.

In the robot vision system, the input image from the digital camera contains 256x320 pixels and each pixel has
one of the 64 gray levels (0-63). The average gray level, \( AV \), of the image is calculated by summing the gray levels of all pixels and dividing the summation by the total number of the pixels (Eq. 3-1).

\[
AV = \frac{\sum_{i=1}^{n} P_i(x,y)}{n} \quad \text{(Eq. 3-1)}
\]

where \( P_i(x,y) \) is the gray level of \( ith \) pixel \( P_i \) at location \((x,y)\), and \( n \) is the total number of pixels in the image. The average serves as a reference value to determine the threshold. The threshold, \( TH \), is then derived by adding some adjustment, \( SH \), to the average (Figure 3.1).

\[
TH = AV + SH \quad \text{(Eq. 3-2)}
\]

where \( SH \) can be a negative integer, 0, or positive integer. During, and only during, the calibration operation, a binary display of the input image is presented to the user. The user will judge whether the threshold is able to distinguish the objects from their background. A fine binary image should show all the details of the objects, i.e., the sharp boundaries of the objects and holes. The shift value is adjusted until a satisfactory binary image is achieved.
In order to reduce the process time, only some of the pixels in the image are selected to determine the average. The 256x320 image is divided into a square grid (Figure 3.2). Only the pixels at the nodal points are picked to calculate the average. The "n" in Eq.3-1 will be the number of pixels picked for the calculation. The size of the grids is arbitrary. The bigger the grid is, the faster the procedure can be completed. However, the calculation may lose its accuracy with an oversized grid. The suggested grid size is from 1 to 8.
Figure 3.2 Grids and Nodal Points of an Image.

The threshold is determined during the camera calibration operation. It remains unchanged as long as it works well in the image-processing operation. However, if the illumination condition varies, the previously defined threshold may become invalid. Such a threshold will cause boundary-tracking errors or system crash during the image-processing operation. If this is the case, a re-calibration of the camera, i.e., a new threshold, is required.

The number and the size of objects inside the camera viewing window are controlled during, and only during, the camera calibration. Therefore, the number of object pixels and the number of background pixels will have a certain ratio. The average will truly reflect the illumination condition over the viewing window. The choice of the object number and size is based on experience and unchanged
for the calibrations. It is proper if the objects cover 30% to 40% of the total image frame. After the camera calibration, any number or size of objects is allowed within the camera viewing window.

3-2 Image Processing

In the image processing operation, the boundaries of objects and holes are detected, tracked, and marked. Chain-codes are generated for each object or hole boundary. The relation between a hole and its host object is defined. The location, orientation, and invariants of the objects are also computed. This image-processing operation is the center piece in the programming of the vision system.

3-2-1 Boundary Detection

Boundary detection implies finding the location of a boundary. The location of a boundary can be defined as the row-column position of the first-found boundary pixel. Detecting boundaries includes two search procedures: a rough search and a fine search. The rough search works through the entire portion of the image within the operation frame. It detects the approximate location of
the boundaries. The fine search works on a small portion of the image. It pinpoints the exact location of the boundaries.

3-2-1-1 Operation Frame

An operation frame is defined to avoid noise at the edge of the raw image. Due to the quality of the input images, there are quite a few pixels with irregular gray levels on the four edges of the image frame. These irregular pixels will mislead the boundary tracking operation if they are included in the image processing. To avoid such operation malfunction, an operation frame is defined by setting its borders 4 to 8 rows and columns inward the edges of the image frame (Figure 3.3). The boundary search will be applied only within the operation frame. And the borders of the operation frame will be monitored during the boundary tracking operation. The tracking will not cross the borders.

The operation-frame concept can be utilized to focus the image processing on any portion of the image. By varying the operation frame, the image processing can be zoomed into a most-interested area in the image, or zoomed out for a over-all view of the image.
3-2-1-2 Rough Search

The *rough-search* operation starts from the top-left corner of the operation frame, and proceeds to the right with a specified *step length*. When the search reaches the right border of the operation frame, it advances downward by one step length and restarts from the left border. The rough search ends at the bottom-right corner of the operation frame (Figure 3.4).

The step length for the rough search is an integer greater than 0. The choice of the step length depends on the minimum size of the objects. A large step length
yields a fast rough search. However, if the step length is larger than the minimum size of the object, some boundary may be overlooked. A suitable step length may be an integer from 1 to 8, with 1 being the most-detailed search.

![Figure 3.4 A Left-Right, Top-Down Rough-Search Pattern.](image)

The propose of the rough search is to quickly estimate the approximate location of the boundary of either an object or a hole. It compares the gray-levels of a pixel and the pixel one step-length to its right. A boundary is discovered if the gray levels of the two pixels are on the different sides of the threshold value. When a pixel has
(a) Rough search for OBJECT boundary.

(b) Rough search for HOLE boundary.

Pixel examined in ROUGH search.

Fine-search region.

Figure 3.5 Rough Search for Boundaries, (a) Object Boundary; (b) Hole Boundary.
the above-threshold gray level and the pixel to its right has the below-threshold gray level, an object boundary is said to exist between the two pixels (Figure 3.5a). Vice versa, when a pixel has the below-threshold gray level and the pixel to its right has the above-threshold gray level, a hole boundary is said to exist between the two pixels (Figure 3.5b). The fine search will take over here to pinpoint the exact location of the boundary. The fine search is followed by the boundary tracking which tracks and marks the boundary. The rough search will resume upon the completion of the boundary tracking.

If an object is discovered lying over the left border of the operation frame (Figure 3.6), the fine search will

---

Figure 3.6 An Object Lies over the Left Border of the Operation Frame.
be skipped. The boundary tracking will start right after the object is detected.

3-2-1-3 Fine Search

The fine search pinpoints the exact location of a boundary after the rough search detects the existence of the boundary. The fine search operation examines the pixels one by one between the last two pixels examined in the rough search. The operation starts from the pixel at left-end of the fine-search region (see Figure 3.5) and proceeds to the right. If there is a gray-level transition across the threshold between a pair neighboring pixels, the last-examined pixel in the fine search is the exact location of the boundary (Figure 3.7). If the last-examined pixel has the below-threshold gray level, it will be the first boundary pixel of the object (Figure 3.7a). The procedure will be the same for pinpointing the exact location of a hole boundary except that the last-examined pixel, or the first boundary pixel of the hole, will have the above-threshold gray level (Figure 3.7b).

Before the boundary tracking operation begins, two pixels have to be examined to ensure that the detected boundary is a new boundary instead of a tracked boundary.
(a) Fine search for OBJECT boundary.  (b) Fine search for HOLE boundary.

Figure 3.7 Fine Search for Boundaries.  
(a) for an Object Boundary,  
(b) for a Hole Boundary.

One of the two pixels locates at the left next to the last examined pixel in the fine search, i.e., the second-last examined pixel. Another pixel is the last examined pixel itself. The boundary is a previously tracked boundary if a marker, which is placed during the previous boundary-tracking operation, is found at either of the two pixels (Figure 3.8). A marked boundary will be ignored and the rough search will resume. Otherwise, the boundary tracking will begin with the last-examined pixel in the fine search as the starting point.
Figure 3.8 Detecting Marked Boundaries.
(a) Marked Object Boundary when Searching for Object Boundary;
(b) Marked Object Boundary when Searching for Hole Boundary;
(c) Marked Hole Boundary when Searching for Hole Boundary;
(d) Marked Hole Boundary when Searching for Object Boundary;

3-2-1-4 Hole Boundary Search

A hole must be in its host object. The host object must be found before the detection of the hole. Therefore, the hole-boundary search starts when the rough search is performed inside the object region. The hole-boundary
search will continue within and only within the object region. It will stop and object-boundary search will resume when the rough search goes outside the object region (see Figure 3.10).

A hole boundary will not be detected on the borders of the operation frame. A hole may actually lie over one of the borders of the operation frame. The hole boundary, however, will be tracked as part of the boundary of its host object since the object boundary must be detected and tracked before the hole boundary (Figure 3.9). The hole boundary will, therefore, be detected as a marked object boundary and will be ignored.

---

**Figure 3.9** The Hole Boundary Which Lies over Operation Border will be Tracked as Part of the Boundary of its Host Object.
3-2-1-4 Summery of Boundary Search

Figure 1.10 summarizes the boundary search process. The boundary-search starts from top-left corner of the image. It first searches for the first object boundary. If an object boundary is found, it pinpoints the location of the boundary and the boundary is tracked. Then, the boundary-search starts looking for a hole boundary. If a boundary is found, it pinpoints the boundary location and checks if the boundary is a hole boundary or a marked object boundary. The boundary will be tracked if it is a new hole boundary, or the boundary will be ignored if it is a marked object boundary. The object boundary search resumes for other object boundaries. If a boundary is found, the boundary-search pinpoints the boundary location and checks if the boundary is a new object boundary or a marked hole boundary. The boundary will be tracked if it is a new object boundary, or the boundary will be ignored if it is a marked hole boundary. Then, the hole boundary search is restarted. The same procedures are repeated until the boundary-search reaches the bottom-right corner of the image.
3-2-2 Boundary Tracking

The meaning of boundary tracking is to travel around the object, or hole, along its boundary. The main point of the boundary tracking is to keep the tracking right on the
boundary by monitoring the pixels on both sides of the boundary. The left and right turns of the tracking will be made to keep the tracking on the boundary. Noise will be removed to smooth the boundary. The tracking will terminate when the last tracked boundary pixel meets the first tracked boundary pixel (Figure 3.11). The tracking may be interrupted if the generated chain-code grows too long or if too many turns have to be made for a single tracking step. This means that the tracking is out of control or runs into a dead end.

![Diagram](image)

**Figure 3.11 Tracking Object Boundary.**

The strategies of object boundary tracking and hole boundary tracking are the same except a few differences. The object boundary tracking will be described in detail. Then the specialty of hole boundary tracking will be stated.
3-2-2-1 Tracking Directions

The eight direction system is employed in the boundary tracking. The tracking direction to the left (WEST) has the number of 0 and the tracking direction advances counter-clockwise to 7, with 45° between directions (see Figure 1.9). Each tracking direction is figured as a vector with its origin on the last tracked boundary pixel and its direction pointing to next boundary pixel (Figure 3.12). As the tracking continues, the origin of the direction vector moves from one boundary pixel to its neighboring boundary pixel along the vector direction.

Figure 3.12 Tracking Direction at Last Tracked Boundary Pixel.
3-2-2-2 Initial Tracking Direction

The tracking starts at the first boundary pixel with an initial tracking direction. The first boundary pixel, which is called the starting-point of the boundary, is pinpointed by the fine search. The initial tracking direction is determined before the formal boundary tracking.

The initial tracking direction is determined by examining the nearest neighboring pixels of the starting-point. The pixel on direction 0 and next to the starting-point does not need to be examined because the nature of the fine search indicates that it is a background pixel; otherwise, itself should be the starting-point of the boundary. The pixels next to the starting-point in other directions are examined, beginning from direction 1 and advancing counter-clockwise. When the first object pixel is found in one of the directions, this direction is regarded as the initial tracking direction (Figure 3.13). The formal boundary tracking will then be launched. If no object pixel is found in all directions, the starting-point of the boundary is recognized as an isolated object pixel. In other words, it is noise. In this case, the boundary
tracking will be aborted and the search for new boundary will resume.

Figure 3.13 Determine Initial Tracking Direction.

According to the counter-clockwise rule of tracking, the tracking direction on the left border of the operation frame can only be direction 2, or SOUTH. Therefore, a special start of boundary tracking is needed when an object is found lying over the left border (Figure 3.14). The
tracking will immediately jump to the left border tracking. No search for initial tracking direction is required.

3-2-2-3 Tracking Straight

The general rule for the tracking to stay on the boundary is that there must be object pixels on the left side of the boundary and background pixels on the right side. Two nearest neighboring pixels of the last-tracked
boundary pixel are examined to determine whether the tracking should travel straight along the current tracking direction. One pixel \(C\) is in front of the last-tracked boundary pixel in the tracking direction. Another pixel \(A\) is in front of the last-tracked boundary pixel, with 45° to the right of the tracking direction (Figure 3.15).

If pixel \(C\) has the below-threshold gray level and pixel \(A\) has the above-threshold gray level, the tracking is right on the boundary and the tracking direction is pointing to the next boundary pixel. Therefore, the tracking will continue in the same direction.
3-2-2-4 Making Turns

If both pixel A and C have either the below-threshold gray level or the above-threshold gray level, a change of the tracking direction is needed. If both pixels have the above-threshold gray level, the tracking direction will make a 45° left turn by increasing the direction code by 1 (Figure 3.16a). The gray level of pixel C is checked again. If pixel C still has the above-threshold gray level, the tracking direction will make another 45° left turn (Figure 3.16b).

![Figure 3.16](image)

**Figure 3.16** Left Turns of Tracking Direction, (a) 45° Left Turn; (b) 90° Left Turn.

The gray-level checks and 45° left turns are repeated until pixel C has the below-threshold gray level (Figure 3.17).
If both pixel A and C have the below-threshold gray level, the tracking direction will turn right by reducing the direction code by 1 for each 45° turn until pixel A has the above-threshold gray level (Figure 3.18).

**Figure 3.17 Making Left Turns.**

**Figure 3.18 Right Turns of Tracking Direction,**

(a) 45° Right Turn; (b) 90° Right Turn; (c) Making Right Turns.
Several turn controls are required when making turns. If the direction code goes above 7 while doing left turns, the code will be set back to 0. Similarly, if the direction code goes below 0 while doing right turns, the code will be set to 7. If up to eight turns have been made but the tracking still can not advance to next boundary pixel, the tracking is considered to run into a dead end (Figure 3.19). The boundary tracking will be interrupted.

3-2-2-5 Removing Boundary Noise

There are often some bumps or holes on a boundary due to the image quality and the global threshold setting. If any of these bumps or holes has the size of one pixel, it
will be removed as noise to smooth the boundary. A bump or hole with more than one pixel in size will be considered as real boundary. In order to detect a boundary noise, a couple pixels have to be examined in addition to pixel C and A.

When the boundary tracking comes to a bump on a boundary, both pixel A and C have the below-threshold gray level and the tracking direction intends to turn right (see Section 3-2-2-4). Before making the turn, pixel B is examined (Figure 3.20).

![Diagram showing bump noise detection](image)

**Figure 3.20** Bump Noise Detection.

If pixel B has the above-threshold gray level, then pixel A is bump noise (Figure 3.21a). The bump noise will then be removed by setting pixel A with an above-threshold gray level. However, if pixel B has the below-threshold gray level, the right turn will be made (Figure 3.21b).
In case of a hole on the boundary, both pixel \(A\) and \(C\) have the above-threshold gray level. Before the tracking direction makes a left turn, pixel \(D\) is checked (Figure 3.22). If pixel \(D\) has the below-threshold gray level, \(\) is pixel \(D\) a remove pixel \(C\) object pixel? as hole noise both pixel \(A\) and \(C\) are background pixels \(\) pixel \(C\) is an object pixel pixel \(A\) is a background pixel \(45^\circ\) left turn

**Figure 3.22** Hole Noise Detection.
pixel C is a hole noise and will be removed by giving it a below-threshold gray level (Figure 3.23a). In contrast, if pixel D has the above-threshold gray level, the left turn will proceed (Figure 3.23b).

![Diagram](image)

**Figure 3.23** Detect and Remove Hole Noise, (a) A Hole Noise; (b) A Left Turn.

### 3-2-2-6 Tracking on the Operation Borders

A special procedure is designed for the boundary tracking on each border of the operation frame. When tracking reaches either border of the operation frame, the normal tracking algorithm is no longer effective. The tracking will cut into the object region instead of remaining on the boundary. In addition, the tracking can only follow the direction of the border depending on the
sides of the operation frame. The tracking directions on the top, the left, the bottom, and the right borders are 0, 2, 4, and 6, respectively. The tracking will keep traveling in the border direction until coming out of the object region. Then the normal tracking will resume.

Since the object region lies over the border, the object is considered to be an "incomplete" one. Therefore, the tracking will terminate with an error message.

3-2-2-7 Recording Tracking Directions

The chain-code of a boundary is a one-dimensional array which consists of the numerical codes of the tracking directions for each boundary pixel. The initial tracking direction at the starting-point of the boundary is the first element of the chain-code array. The last tracking direction at the finishing-point of the boundary is the last element. Other tracking directions on the boundary pixels between the starting-point and the finishing-point are sequentially arranged in the chain-code (Figure 3.24).

A tracking direction will be recorded into the chain-code for each successfully tracked boundary pixel as long as the boundary tracking is proceeding normally. The
tracking directions, however, will not be recorded if the boundary tracking is proceeding on the borders of the operation frame. Furthermore, a recorded chain-code will be cancelled if the tracked boundary belongs to an incomplete object which lies over one of the borders of the operation frame. Also, a chain-code will be cancelled if the boundary tracking goes out of control, which means that the chain-code length is too long or the tracking has run into a dead end.
3-2-2-8 Hole Boundary Tracking

The hole boundary tracking uses the same algorithm as the object boundary tracking does with two exceptions. 1) The hole pixels are treated as object pixels even though they have the above-threshold gray level. The object pixels are treated as background pixels even though they have the below-threshold gray level (Figure 3.25). 2) If hole-boundary tracking reaches either border of the operation frame, the tracking will be considered as being out of the control instead of an "incomplete hole" because a hole boundary does not appear on the border (see Figure 3.9).

(a) Object boundary tracking.  (b) Hole boundary tracking.

Figure 3.25 Object Boundary Tracking vs. Hole Boundary Tracking, (a) Object-Boundary Tracking; (b) Hole-Boundary Tracking.
3-2-2-9 Boundary Tracking Termination

Based on the fact that the starting-point and the finishing-point of a closed path is the same point, the boundary tracking stops when the last tracked boundary pixel meets the first tracked boundary pixel, i.e., the starting point of the boundary tracking. A temporary marker is dropped onto the first tracked boundary pixel. This temporary marker will signal that the tracking comes back to the location of the first tracked boundary pixel and the tracking should terminate (Figure 3.26). The boundary tracking also stops if the generated chain-code exceed a user-defined length or the tracking runs into a dead end.

**Figure 3.26** Boundary Tracking Termination.
The same temporary marker is also dropped onto every boundary pixel of the tracked boundary. The markers indicate that the pixels are tracked boundary pixels instead of a never-detected boundary pixels. This will prevent the boundary pixels from being tracked again. These temporary markers will be replaced by permanent markers after the boundary tracking has been successfully completed. The temporary markers will stay on the boundaries of the incomplete objects. Both the permanent markers and the temporary markers will keep the tracked boundary from mistaking to be new boundaries.

3-2-3 Error Handling

There are three kinds of errors from the boundary tracking; isolated pixels, incomplete object, and tracking out of control. These errors are handled after each boundary tracking which terminates with an error code. The boundary search will be resumed afterward.

1. Isolated Pixels:

The boundary search algorithm detects a single boundary pixel of an object or hole. This boundary pixel may turn out to be an isolated pixel instead of a real boundary pixel. Such a pixel is discovered at the
beginning of the boundary tracking (see Section 3-2-2-2). The pixel will be removed as noise by giving it the gray level of an "eraser". The "eraser" is an integer whose value is an above-threshold gray level if the isolated pixel is an object pixel, or a below-threshold gray level if the pixel is a hole pixel.

2. Incomplete Object:

If boundary tracking has reached one of the borders of the operation frame, the object is considered to be an incomplete object since it can not be seen as a whole within the operation frame. No additional processing will be done on such object. The recorded boundary chain-code will be cancelled. The temporary markers, however, will be left on the object boundary to prevent the boundary from being tracked again.

3. Tracking Out Of Control:

There are a few circumstances in which the boundary tracking would be out of control. The first one relates to the quality of the input image and the threshold setting. The combination of a poor quality image and an improperly set threshold can result in a very noisy image and very blurred object boundaries. As a result, the boundary search will fail or the boundary tracking will lose direction. Secondly, the boundary tracking may be blocked
or jammed if it runs into a dead end formed with an object pixel string of only one pixel wide (see Figure 3.19).

There are two controls to determine whether the boundary tracking is out of control. One is the length of the boundary chain-code. The other one is the number of turns required to track a single boundary pixel. If the chain length or turns exceed an user-defined value respectively, the tracking is considered to be out of control.

If the tracking is out of control, what ever have been tracked down has to be erased. The recorded chain-code is cancelled. The temporary markers on the tracked boundary are replaced by an "eraser". The eraser for an object boundary is an integer whose value is less than the threshold. The eraser for a hole boundary is an integer whose value is greater than the threshold. The propose of the eraser is to revert the marked boundary pixels to the gray level of the object or hole.
3-2-4 Marking Boundaries

A permanent marker is assigned to each successfully tracked boundary. There are different markers[1] for different boundaries. A negative integer less than -1 will be used to mark an object boundary, starting from -2[2] for first tracked object boundary. A positive integer greater than 64 will be used to mark a hole boundary, starting from 65[3] for the first tracked hole boundary. The permanent marker is placed on every boundary pixel while travelling around the boundary under the guidance of the generated chain-code.

3-2-5 Recording Chain-Codes

For each successfully tracked boundary, the chain-code will be recorded together with its starting-point location and the chain-code length. The chain-codes for object boundaries and hole boundaries are recorded separately. A serial number is assigned to each chain-code of the object or the hole. The starting-point location and the chain-

[1] Integer 0-63 will not be employed for markers because they represent the gray levels of the image pixels.
[2] Marker "-1" is reserved as a temporary marker for object boundaries.
[3] Marker "64" is reserved as a temporary marker for hole boundaries.
code length will be saved to a different field called "integer parameters". These integer parameters have the same serial number as their corresponding chain-code.

3-2-6 Computation of Object Moments\(^{[1]}\)

The moments of each object or hole are computed while travelling around the boundary during the same time of dropping permanent markers on the boundary. The computation is based on the concept of plane closed-path integration (see Eq.1-10). These moments will be used to determine the geometrical properties of the object or hole.

The perimeter of the object or hole is evaluated by counting the number of even directions and odd directions in the chain-code. Each even direction, 0, 2, 4, or 6, is given ONE unit which represents the distance between two nearest neighboring pixels in horizontal or vertical directions. Each odd direction, 1, 3, 5, or 7, is given \(\sqrt{2}\) unit since each odd direction indicates the distance between two nearest neighboring pixels in diagonal directions (Figure 3.27). The perimeter is:

\[ P = (\text{number of even directions}) + \sqrt{2}(\text{number of odd directions}) \]

\(^{[1]}\)All computations are based on the image-frame measure. The results will be converted to metric measure in "parameter arrangement".
Figure 3.27 Boundary Length-Unit on Even Directions and Odd Directions.

3-2-7 Hole-Object Relation

The hole-object relation is indicated by a numerical code. The code is placed to the fourth field in the integer parameter array of the object or hole boundary. The hole-object relation has a different meaning to the object and to the hole. To the object, the code tells how many holes are in the object or that there is no hole at all. To the hole, it tells which object the hole belongs to. The code to the object is one of the features for identifying the object. The code to the hole serves as an identifier when the host object tries to find its hole(s).
A hole can have one and only one host object, and a hole region must be enclosed in an object boundary. The object boundary is always tracked and marked before the hole boundary. The hole-object relation, therefore, can be determined by finding the marked object boundary outside the hole region.

The determination of the hole-object relation takes place after the completion of the hole boundary tracking. Beginning from the starting-point of the hole boundary and proceeding to the left, each pixel is examined until a marker of the object boundary is detected (Figure 3.28). The marker of the object boundary is a negative integer whose absolute value less 1, \(|\text{marker}| - 1\), is the serial number of the object. This serial number will be recorded as the hole-object relation code of the hole. One is added

![Figure 3.28 Determine Hole-Object Relation.](image-url)
to the hole-object relation code of the object to indicate that the object has one more hole discovered inside it. The hole moments will be computed and subtracted from the previously computed object moments to yield the over-all moments of the object.

3-2-8 Parameter Arrangement

Parameter arrangement is applied to every object upon the completion of the tracking of all boundaries. The raw moments computed for each object during the boundary tracking are utilized to obtain the centroid location and orientation angle of the object, as well as the area and principal values of the object. These raw moments computed by Eq.1-10 are with respect to the image origin, which is the upper-left corner of the image frame.

The object properties are determined by the equations mentioned in Section 1-3-6. First, the area of the object comes directly from Eq.1-10. The centroid of the object is given by Eq.1-1 and Eq.1-2. The moments of inertia and the product of inertia of the object is then transferred to the object centroid by Eq.1-11. The orientation angle of the object is determined by Eq.1-3. Finally, the principal values of the object is calculated by Eq.1-13.
The computation above is based on the measure of the image frame. One unit length is one pixel wide. The area and the principal values, as well as the perimeter, need to be converted from the measure of the image frame to the metric measure by multiplying a scale factor (see Section 3-2-9).

\[ \text{AREA}_{\text{metric}} = \text{AREA}_{\text{image}} \times \text{SCALE}^2 \]

\[ \text{PERIMETER}_{\text{metric}} = \text{PERIMETER}_{\text{image}} \times \text{SCALE} \]

\[ \text{PRINCIPAL VALUES}_{\text{metric}} = \text{PRINCIPAL VALUES}_{\text{image}} \times \text{SCALE}^3 \]

The starting-point location of each boundary is modified from the actual location in the image frame to a relative location corresponding to the object centroid. By subtracting the starting-point location from the object centroid location in both row and column direction, the displacement of the starting-point from the object centroid can be obtained. Such an arrangement makes it convenient to reconstruct the boundary at any location within the image frame when the object centroid is arbitrarily given.
3-2-9 Scale Factor

The scale factor, which is required to convert the object properties from the image-frame measure to the metric measure, is determined during the camera calibration. The procedure is as follows: (1) Carefully measure the area of a sample object; (2) Obtain the area of the object image; (3) Divide the real area by the image area; (4) Take the square root of the quotient to get the scale factor, SCALE.

\[
SCALE = \left( \frac{\text{Real Area of Sample Object}}{\text{Area of Object's Image}} \right)^{1/2}
\]

There is some variation on the values of the object properties while the same object is placed in different locations within the viewing window. Usually, the further the object from the center of the image frame, the smaller the values will be, although the variation is limited. This can be corrected by taking consideration of the angle formed by the object centroid location \((X_c, Y_c)\), the camera, and the aiming point of the camera \((I_c, J_c)\) as shown in Figure 3.29. In the figure, assuming the camera aims
straight downward, the displacement ($R_i$) of the object centroid ($X_c, Y_c$) from the camera aiming point, i.e., the center of the image frame ($I_c, J_c$), is determined by

$$R_i = \sqrt{(X_c - I_c)^2 + (Y_c - J_c)^2}$$

The spatial angle ($\beta$) is given by

$$\text{ATAN}(R_i/H_i)$$

where $H_i^{[1]}$ is the elevation of the camera.

---

[1] $H_i$ and $H_i$ are given in the measurement of the image, i.e., pixel units. The determination of $H_i$ is shown in next section.
above the work plane. A locational scale factor \((LS)\) will be determined by dividing the correction factor \(COS(\beta)\) into the scale factor, \(LS=SCALE/COS(\beta)\). This locational scale factor replaces the scale factor discussed in Section 3-2-8.

3-2-10 Determine Camera Elevation, \(H_i\)

The relationship between the image-frame measure and the robot-base measure is assumed to be linear when determining the height, \(H_i\), of the camera in the image-frame measure, i.e., in pixel units. The real elevation of the camera above the work plane \((HR)\) is 900 millimeters. A ruler is placed inside the viewing window of the camera. The ruler is aligned to the diagonal of the image frame when its live image is brought up to the graphic monitor (Figure 3.30). The real length of the ruler section within the viewing window \((LR)\) is measured. It is 370 millimeters for the elevation of the camera. The image length of the ruler section \((L_i)\) is \((256^2 + 320^2)^{\frac{1}{2}} = 410\) pixel units in the image-frame coordinates. The elevation of the camera in image-frame coordinates \((H_i)\), therefore, equals to 997 pixel units.

\[
H_i = HR \cdot \frac{L_i}{LR} = 900 \cdot \frac{410}{370} = 997
\]
In addition, the viewing angle of the camera ($\alpha$) can be calculated as below.

$$\alpha = \tan^{-1}\left(\frac{185}{900}\right) = 11.6^\circ$$

where 185 is half of $LR$.

![Diagram](image)

Figure 3.30 Determine Camera Elevation.

3-3 Learning Objects

The human user can command the robot vision system to memorize a new object by recording the information of the
object along with a given name into the computer memory. These object information includes the object's integer parameters, real-number invariants, and chain code. A new view of a known object can also be put into the computer memory by recording the information of the view. The vision system can identify that if the object or view is a new one before recording the information of the object or view. The basic operation in the learning routine is the comparison of the object parameters between the on-scene objects and the recorded objects. Other operations include deleting or renaming a recorded object and updating object parameters and invariants.

3-3-1 Invariants for Comparison

The parameters used in the comparison must be some invariants independent of the object location and orientation. The object area, the object perimeter, two object principal values, and number of holes, as well as the hole area, the hole perimeter, two hole principal values are the parameters for the comparison.

In order to limit the effect on object invariants variants due to the illumination, such as the object
shadow, the area and principal values are convert to the following forms:

\[ \frac{\text{AREA}}{\text{PERIMETER}}; \]
\[ \frac{\text{PRINCIPAL VALUE 1}}{\text{AREA}}; \]
\[ \frac{\text{PRINCIPAL VALUE 2}}{\text{AREA}}. \]

These quantities, instead of their originals, will be put into the comparison. However, the perimeter will keep its original form.

3-3-2 Objects in Comparison

For each comparison operation, one object is selected from the viewing window by the user. The invariants of this on-scene object is compared with the invariants of all the recorded objects to determine if the on-scene object matches any one of the recorded objects. If there is no match, the on-scene object is a new object. A name will be given by the user and all of the object parameters are recorded along with the given name, including the integer parameters, real-number invariants, boundary chain-code and hole's integer parameters, real-number invariants, chain-code if the object has hole(s). The boundary chain-code is not a feature to identify the object, but it is needed when reconstructing the object boundary for the user to examine the shape of the object. If there is a match, the on-scene
object is a known object and its parameters does not need to be put into the computer memory again. However, the recorded invariants of this object can be updated if the user wishes to do so.

3-3-3 Tolerance for Comparison

Due to the roughness of the boundaries, obtained real-number invariants of the objects and holes will vary slightly from time to time. There will be extremely few chances for an exact match of the real-number invariants between the on-scene object and the recorded objects. Therefore, a tolerance is allowed in the comparison of real-number invariants of objects and holes.

The on-scene object under consideration is the reference object in determining the tolerance domain for comparison. An acceptable lower limit ($LIMIT_1$) and an acceptable upper limit ($LIMIT_2$) are set for each real-number invariant of the reference object. The lower limits are defined by subtracting some percentage ($\alpha$) of the invariants from the invariants, $LIMIT_1 = RP(1-\alpha)$. The upper limits are defined by adding some percentage of the invariants to the invariants, $LIMIT_2 = RP(1+\alpha)$. The choice of the percentage depends on the statistical conclusion of
the experimental data and the accuracy of object identification the user desires. It might vary from 5% to 20%.

3-3-4 The Comparison

The number of holes is compared first between the on-scene object and the recorded objects. It is an integer number comparison and will proceed very fast. The comparison begins with recorded object No.1. If the hole number of the on-scene object does not match the hole number of the recorded object, the comparison moves to next recorded object. The same comparison continues until it comes to the last recorded object or a match is achieved (Figure 3.31). The comparison operation may terminate with a "no match" message if no successful match is achieved throughout the comparison of all recorded objects.

If the hole number is matched on one of the recorded objects, the comparison moves to the real-number invariants of the object. The real-number invariants of the recorded object are examined one by one in the sequence of area, perimeter, principal value 1, and principal value 2. Each of these invariants is checked if it falls into the interior between its acceptable upper limit and its
acceptable lower limit. If either one of them is outside the interior, the comparison will go back to the hole-number stage and repeat the same procedure. If all real-number invariants of the recorded object fall in between the acceptable limits, the recorded object is said to primarily match the on-scene object. The negative of the serial number of the recorded object is reserved. If the object has no hole, the recorded serial number will return to its positive original to indicate a complete match.

If there is any hole involved, additional comparison for the hole invariants is needed. The hole invariant comparison is similar to the object invariant comparison.
except that the hole-object relation has to be determined before the comparison. For both on-scene object and the recorded object, the hole-object relation is found by examining the relation code among the integer parameters of the holes. By searching through the relation codes of all the on-scene holes, the on-scene hole in the on-scene host object in question will be found. The acceptable upper and lower limits of the real-number invariants of the hole are determined in the same way as the object invariants. Then, by searching through the relation codes of all the recorded holes, the recorded hole in the recorded host object in question will be found. The real-number invariants of the recorded hole are checked one by one in the sequence of area, perimeter, principal value 1 and principal value 2, in the same way as in object invariant comparison. If all of the invariants falls in between their acceptable limits respectively, the recorded object is said to completely match the on-scene object. The negative serial number recorded from the primary match will go back to its positive original. If one of the invariants is outside the acceptable limits, the recorded serial number will keep its negative form to indicate the primary-only match.

The result of comparison can be a complete match, or a primary match, or no match. A complete match has a positive integer as its match code to indicate the serial
number of the matched recorded object. The code indicates the on-scene object in question is a known object and its parameters should not be recorded again. A primary match has a negative integer as its match code whose absolute value represents the serial number of the matched recorded object. The negative match code indicates that the match has succeeded only on the outer boundary of the recorded object, with no success on hole match. The recorded real-number invariants of the object and hole should be updated. No match has a zero as its match code which indicates the on-scene object is a new object. The parameters of the on-scene object can be recorded along with an object name given by the user.

3-3-5 Object Views

An object can be positioned in different ways and thus can produce a number of views totally different in shapes. For example, a machined bolt can have a T-shape view when it lies on a plane. The same bolt can also have a hexagonal view if it stands on the plane by one of its ends (Figure 3.32).
Figure 3.32 Different Views of a Bolt. 
(a) Side View; (b) Top View.

Obviously, these object views with different shapes will have different geometrical properties. The area, perimeter, principal values and number of holes from different view have no significant connection among the views. The images of the different views of the same object will, therefore, appear as the images of different objects.

In order to avoid mistaking the different views of an identical object for different objects, the same name is used to label the different views of the same recorded object. Therefore, a name check is needed before a given name being accepted. If the given name turns out to be a used name, the user has to confirm whether the name is for a different view of the same object or is mistakenly used for a different object. If the given name is for the
different view of the same object, the parameters of the object view will be recorded under the same name. If the given name is a mistake, the user has to enter a different name. This name check routine is also required when identifying an object from different views, except that the user assistance is not needed.

3-3-6 Invariants Update

The variation of the illumination condition on the viewing window of the camera may more or less cause some deviation in the real-number invariants of the objects. The recorded real-number invariants of an object can be updated if the user wishes to do so. The update operation averages the previously recorded invariants and the recently extracted invariants of the object. The same will be done to the hole(s) if there is any.

3-4 Object Identification

The operation of object identification has the same procedure as object learning. However, instead of comparing one on-scene object with all recorded objects, one recorded object is compared with all on-scene objects.
The same object and hole invariants as mentioned previously are put into the comparison.

3-4-1 Name Finding

The object identification begins with a name input, followed by a search among the recorded names. If there is a recorded name which matches the input name, the serial number of the recorded name is reserved for the invariant comparison. There may be more than one recorded name which match the input name. The number of matched names indicates the number of views previously recorded for the object. All serial numbers corresponding to the identical name will be reserved. The invariants under the serial numbers will all be put into the comparison operation, one view at a time.

3-4-2 The Comparison

The invariants of the recorded object will be the reference object to determine the acceptable lower and upper limits. The comparison operation is the same as that in object learning. The invariants of the on-scene objects will be checked if they fall in between the limits. If
more than one view has been recorded for the recorded object, the same comparison will applied on each view until a match view is found. A match indicates the requested object is inside the camera viewing window. If all recorded views have been checked but no match is achieved, the operation terminates with a "No-Match" code, i.e., a zero. A "No-Match" indicates the requested object is not found inside the viewing window or the object view has never been put into the computer memory.

3-5 Robot Guidance

In order to guide the robot hand to the recognized object, the location and the orientation of the object in the image-frame coordinates has to be transferred to the robot-base coordinates. Homogeneous Transformation Matrices are constructed to described the relationships between different coordinate systems. By matrix multiplication, the object-robot relation matrix can be determined. Then the object location and orientation can be determined in the robot-base coordinates.
3-5-1 Construction of Homogeneous Transformation Matrices

Three homogeneous transformation matrices have to be constructed; (1) object position in image-frame coordinates; (2) image-frame position in robot-base coordinates; (3) object position in robot-base coordinates.

3-5-1-1 Homogeneous Transformation Matrices

If the wrist of a robot first rolls $\phi_z$ about Z axis, and then pitches $\phi_y$ about Y axis, and finally yaws $\phi_x$ about X axis, the general form of homogeneous transformation matrix can be defined as: [Snyder, 1985]

$$[T] = \begin{bmatrix}
C\phi_z C\phi_y & C\phi_z S\phi_y S\phi_x - S\phi_z C\phi_x & C\phi_z S\phi_y C\phi_x + S\phi_z S\phi_x & X \\
S\phi_x C\phi_y & S\phi_x S\phi_y S\phi_x + C\phi_x C\phi_y & S\phi_x S\phi_y C\phi_x - C\phi_x S\phi_x & Y \\
-\sin \phi_y & C\phi_y S\phi_x & C\phi_y C\phi_x & Z \\
0 & 0 & 0 & 1
\end{bmatrix}$$

(Eq.3-3)
where \( x, y, z \) are the three displacements from the origin of the coordinate system. The symbol \( c \) is the abbreviation of \( \cosine \) and \( s \) is the abbreviation of \( \sine \).

Some simplification can be applied to the two-dimensional case. First, since all objects to be recognized will be placed on a horizontally fixed work plane, the object displacement in \( Z \) direction (vertical) is always a constant. Secondly, assuming that 1) the camera is placed directly above the work plane and is aimed vertically downward; 2) the objects lie horizontally on the work plane, there will be no rotation about the \( X \) or \( Y \) axis. That is \( \phi_x=\phi_y=0 \). The homogeneous transformation matrix can be reduced to the following form:

\[
[T] = \begin{bmatrix}
c\phi_z & -s\phi_z & 0 & x \\
s\phi_z & c\phi_z & 0 & y \\
0 & 0 & 1 & z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]  

(Eq.3-4)

3-5-1-2 Matrix of Object in Image-Frame Coordinates, \( c[T]_o \)

The construction of this matrix is straightforward. All required parameters have been derived from the image
processing operation. These parameters are given in the coordinates of the image frame whose origin is at its upper-left corner. The parameters are the X-Y coordinates \((c_{x_0} \text{ and } c_{y_0})\) of the object centroid and the object orientation angle about Z axis \((c_\phi)\). The matrix can be produced by simply plugging in the values.

\[
\begin{bmatrix}
c_{\phi_0} & -s_{\phi_0} & 0 & c_{x_0} \\
s_{\phi_0} & c_{\phi_0} & 0 & c_{y_0} \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]  
(Eq.3-5)

where \(z\) is set to be zero. It has no effect on the \(z\)-displacement in the robot-base coordinates since the work plane is stationary.

3-5-1-3 Matrix of Image Frame in Robot-Base Coordinates, \(^b[T]c\)

This matrix is constructed during the calibration of the camera position. It is a reverse operation of the construction of \(^b[T]o\). The following procedures will be taken: (1) Manually place an object to a location within the viewing window of the camera; (2) Measure the location
of the object centroid \((b_{x_o}, b_{y_o}, b_{z_o})\) with respect to the \(X\), \(Y\), and \(Z\) axes in the robot-base coordinates, and the object orientation angle \((b\phi_o)\) about \(Z\) axis; (3) Substitute the measurements into the reduced form of the transformation matrix, Eq. 3-2, to obtain \(B[T]o\);

\[
B[T]o = \begin{bmatrix}
  c^{b\phi_o} & -s^{b\phi_o} & 0 & b_{x_o} \\
  s^{b\phi_o} & c^{b\phi_o} & 0 & b_{y_o} \\
  0 & 0 & 1 & b_{z_o} \\
  0 & 0 & 0 & 1
\end{bmatrix}
\]

(Eq. 3-6)

(4) Perform the image processing to obtain the object location and orientation in the image-frame coordinates; (5) Use the obtained parameters to construct \(C[T]o\); (6) Inverse \(C[T]o\) to obtain \(O[T]c\); (7) Multiply \(B[T]o\) by \(O[T]c\) to get \(B[T]c\).

\[
B[T]c = B[T]o \cdot O[T]c
= B[T]o \cdot (C[T]o)^{-1}
\]

3-5-1-3 Matrix of Object in Robot-Base Coordinates, \(B[T]o\)

This matrix is the product of the two matrices mentioned above, \(B[T]c\) and \(C[T]o\).
\[ ^b[T]o = ^b[T]c \cdot ^c[T]o \]

\[
= \begin{bmatrix}
  c^b\phi_o & -s^b\phi_o & 0 & b_xo \\
  s^b\phi_o & c^b\phi_o & 0 & b_yo \\
  0 & 0 & 1 & b_zo \\
  0 & 0 & 0 & 1
\end{bmatrix} 
\]

(Eq.3-7)

3-5-2 Object Position in Robot-Base Coordinates

Upon obtaining the homogeneous transformation matrix of object position in robot-base coordinates, \(^b[T]o\) (Eq.3-7), the location and orientation angle of the object in robot-base coordinates can be determined. The orientation angle \((b\phi_o)\) equals to \(\text{ATAN}(s^b\phi_o/c^b\phi_o)\) in degrees. This angle is the rotation angle of the object about the \(Z\) axis of robot-base coordinates. The rotation angles about axis \(X\) and \(Y\) are set to be zero degree. The spatial location of the object is \((b_xo,b_yo,b_zo)\) as shown in Figure 3.33. The \(b_zo\) is the displacement of the horizontally fixed work plane along the \(Z\) axis of the robot-base coordinates.

A command string for defining the object location and orientation is sent to the robot controller in the format of VAL, the computer language designed specifically for use
with PUMA robot [UNIMATE 1982]. The VAL format of object location and orientation will look like the following:

\[
\begin{array}{cccccc}
X/JT1 & Y/JT2 & Z/JT3 & O/JT4 & A/JT5 & T/JT6 \\
34.75 & 530.47 & -23.06 & -167.459 & 57.437 & 114.247 \\
\end{array}
\]

The numbers displayed are the X-Y-Z coordinates of the hand (displacement in millimeters), and three angles (O, A, T) which define the orientation of the hand (Euler angles in degrees). A typical robot hand has two finger tips. An orientation vector points from one finger tip to the other. "O" specifies a rotation of the orientation vector about a vertical direction, i.e., Z-axis. "A" is a rotation about the orientation vector. "T" is a final rotation about the axis of the robot wrist. With A = 90° and T = 0°, the robot hand always points straight down. "O" is the only variable to orient the hand when picking up the object from directly above.

To define the location of a recognized object manually, the following commands will be issued[^1].

```
HERE_OBJECTR
X/JT1  Y/JT2  Z/JT3  O/JT4  A/JT5  T/JT6
34.75  530.47 -23.06 -167.459 57.437 114.247
CHANGE?
^R
```

[^1]: The underlined characters are typed by the user, and "R" represents the ENTER key.
Figure 3.33 An Object in Robot-Base Coordinates.
The following commands move the robot hand to the object and then retreat the hand to a position outside the viewing window of the camera (ORIGIN).

- APPRO_OBJECT,50
- MOVES_OBJECT
- DEPART 75
- MOVE_ORIGIN

1. Move to 50mm above the object
2. Move straight to the object
3. Withdraw 75mm from the object
4. Move to ORIGIN

A communication channel is built between the microcomputer system and the robot controller, with an IBM PC serves as a link between them (Figure 3.34). There are two add-on input/output boards inside the IBM PC. One of them is connected to the microcomputer system. The other one is connected to the robot controller. A communication program running in the IBM PC handles the communication between the two I/O boards. To implement the above task automatically, the same commands are sent from the microcomputer system to the robot controller with some "WRITE" statements. The microcomputer system first writes the commands to IBM PC through the I/O board. The communication program then passes the commands to another I/O board which forwards the commands to the robot controller.
Figure 3.34 Communication Channel between Microcomputer System and Robot Controller.
Chapter 4
Experimental System and Example Operations

The algorithms described in Chapter 3 have been programmed and tested on the robot vision system. In this chapter, the configuration of the vision system will be described in detail and some performance examples will be presented.

4-1 System Description

The robot vision system is set up and programmed to perform two-dimensional image processing and to determine the position of a recognized object in robot-base coordinates (see Figure 1.1). The central unit of the system is an Intel microcomputer system. The microcomputer receives images from the frame grabber and processes the images. It also identifies a user-specified object and determines the object position. The camera is positioned at a location over the work plane and aimed vertically toward the work plane, viewing the area within the
operational range of the robot. The frame grabber can
catch a 256 by 320 image from the camera as well as display
it on the graphic monitor.

This vision system does not need special illumination.
It works under the natural illumination, such as sun light
and indoor lighting. The average-gray-level technique is
used for the determination of a global threshold value for
the images. It is faster than most other algorithms, and
is particularly fast when sampling the image with masks.
The threshold value is determined during the camera
calibration. The threshold remains unchanged unless there
is some variation in the illumination condition or in the
light reflectivity of the object surface.

The system does not involve time-consuming operations.
The boundary tracking algorithm is directly applied to the
raw images input from the frame grabber. It requires only
one pass over the image to extract all desired geometrical
properties of the objects. The object boundaries in each
image, including the hole boundaries, are marked and chain-
coded. A combination of the rough search and fine search
speeds up the boundary search operation. As a result, only
a few more pixels have to be examined in addition to the
boundary pixels.
This system uses the geometrical properties for the object identification and object positioning. In order to avoid complicated calculations, a few necessary and significant geometrical invariants are selected for the object identification. They are the area, perimeter, two principal values and number of holes of the object. Some of the geometrical properties are obtained from the boundary tracking, such as perimeter, number of holes, and hole-object relation. Other are based on the moments computation with closed-path integration guided by the boundary chain codes. The moments of objects and holes are computed after their boundaries are successfully tracked. The obtained moments are then converted to the location and orientation of every object, as well as to the object area and the two principal values. An object can be recognized by the invariant comparison between this object and the recorded objects.

4-2 Equipment

A frame grabber receives images from a digital camera, stores them, and sent them to a graphic monitor. An Intel microcomputer system controls the frame grabber and inputs images from it or outputs images to it. An IBM compatible personal computer serves as a communication link which has
the accesses to both the Intel microcomputer system and the PUMA robot controller.

4-2-1 Digital Camera

The solid state CID surveillance and security camera (series TN2505) was made by General Electric Company. The camera is VLSI/LSI based, utilizing LSI circuits for timing, scan generation, pre-amplification and video processing.

4-2-2 Frame Grabber

The VG-121/221 frame grabber was made by Data Cube Inc. It is an add-on board in the Intel microcomputer system. It receives images from the camera and sends images to a graphic monitor. The operation and access are performed by eight 8-bit registers. The eight registers are either a memory device or an I/O device to the multibus at any base address dividable by eight. All data transfers between the VG-121/221 and the multibus are 8-bit bytes.
4-2-3 Intel Microcomputer System

The central unit of the vision system is an Intel made, 8086-microprocessor based, SYSTEM-310 series microcomputer system. It has 640K RAM and is equipped with a 8087 mathcoprocessor and 10MB Winchester hard disk. The operating system is iRMX 86, Release 7.0.

4-2-4 Operation Programs

The operation programs of this robot vision system are written in a very interactive fashion with FORTRAN language (FORTRAN-86 by Intel). The user can give most of the commands by responding to a YES/NO question or selecting option from a menu.

The vision system can not contain all designed operations into a single program due to the memory (RAM) limitation of the Intel microcomputer system. Therefore, two separate programs has been written to handle different operations. The first program calibrates the camera and processes images. It stores the on-scene object information into a data file. The second program recalls
the on-scene object information from the data file and performs object identification and object positioning.

4-2-5 Supporting Files

Along with the two operation programs, there are five supporting files, a data base file, a backup file for the data base, a data file for storing on-scene object information, and two trace files to record operations for the two operation programs, respectively.

4-2-5-1 Data Base and its Backup

The data-base file and its backup file store the parameters of the recorded objects and the parameters for operation controls. Both files have the following structure: (see next page)
(NUMBER OF RECORDED OBJECTS) (NUMBER OF RECORDED HOLES)
(NAME OF THE FIRST RECORDED OBJECT)
(BOUNDARY STARTING-POINT LOCATION: ROW) (COLUMN) (CHAIN CODE LENGTH) (NUMBER OF HOLES)
(THE COMPLETE CHAIN CODE)
(AREA/PERIMETER) (PERIMETER) (PRINCIPAL VALUE 1/AREA) (PRINCIPAL VALUE 2/AREA)
(NAME OF THE SECOND RECORDED OBJECT)
(BOUNDARY STARTING-POINT LOCATION: ROW) (COLUMN) (CHAIN CODE LENGTH) (NUMBER OF HOLES)
(THE COMPLETE CHAIN CODE)
(AREA/PERIMETER) (PERIMETER) (PRINCIPAL VALUE 1/AREA) (PRINCIPAL VALUE 2/AREA)
......
......
(NAME OF THE LAST RECORDED OBJECT)
(BOUNDARY STARTING-POINT LOCATION: ROW) (COLUMN) (CHAIN CODE LENGTH) (NUMBER OF HOLES)
(THE COMPLETE CHAIN CODE)
(AREA/PERIMETER) (PERIMETER) (PRINCIPAL VALUE 1/AREA) (PRINCIPAL VALUE 2/AREA)
BELOW IS FOR THE FIRST RECORDED HOLE
(BOUNDARY STARTING-POINT LOCATION: ROW) (COLUMN) (CHAIN CODE LENGTH) (NUMBER OF HOLES)
(THE COMPLETE CHAIN CODE)
(AREA/PERIMETER) (PERIMETER) (PRINCIPAL VALUE 1/AREA) (PRINCIPAL VALUE 2/AREA)
BELOW IS FOR THE SECOND RECORDED HOLE
(BOUNDARY STARTING-POINT LOCATION: ROW) (COLUMN) (CHAIN CODE LENGTH) (NUMBER OF HOLES)
(THE COMPLETE CHAIN CODE)
(AREA/PERIMETER) (PERIMETER) (PRINCIPAL VALUE 1/AREA) (PRINCIPAL VALUE 2/AREA)
......
......
BELOW IS FOR THE LAST RECORDED HOLE
(BOUNDARY STARTING-POINT LOCATION: ROW) (COLUMN) (CHAIN CODE LENGTH) (NUMBER OF HOLES)
(THE COMPLETE CHAIN CODE)
(AREA/PERIMETER) (PERIMETER) (PRINCIPAL VALUE 1/AREA) (PRINCIPAL VALUE 2/AREA)
THE 4X4 HOMOGENEOUS TRANSFORMATION MATRIX OF CAMERA IN ROBOT-BASE COORDINATES
SCALE FOR CONVERTING IMAGE MEASURE TO METRIC MEASURE (AREA OF THE STANDARD OBJECT)
TOLERANCE IN COMPARISON FOR (AREA) (PERIMETER) (PRINCIPAL VALUE 1) (PRINCIPAL VALUE 2)
(THRESHOLD) (SHIFT VALUE) (STEP LENGTH FOR SAMPLING PIXELS) (STEP LENGTH FOR BOUNDARY SEARCH)

Every time when the data-base file is read, the previously stored parameters in the data-base file are copied to the backup file.
The on-scene object storage file stores object information extracted from the image processing. The information includes the number of objects and holes inside the camera viewing window, chain codes of the object boundaries and hole boundaries, locations and orientations of the objects and holes, and the object and hole invariants.

There are two groups of real-number invariants for each object. The first group is for the over-all values of the object, which takes the holes into consideration. The second group is for the outer contour of the object, which does not consider the holes. Both groups of parameters can be used in identifying objects. However, if first group is used, the hole parameters will not be needed. If second group is used, the hole parameters will be put into object identification process. For the time being, the first group of real-number parameters is not used for object identification.
The on-scene storage file has the structure as follows:

(Number of On-scene Objects) (Number of On-scene Holes)

Below is for the first on-scene object

(Boundary Starting-Point Location: Row) (Column) (Chain Code Length) (Number of Holes)
(The Complete Chain Code)
(Area/Perimeter) (Perimeter) (Principal Value 1/Area) (Principal Value 2/Area)
(Location X) (Location Y) (Orientation Angle)

Below is for outer contour only

(Area/Perimeter) (Perimeter) (Principal Value 1/Area) (Principal Value 2/Area)
(Location X) (Location Y) (Orientation Angle)

Below is for the second on-scene object

(Boundary Starting-Point Location: Row) (Column) (Chain Code Length) (Number of Holes)
(The Complete Chain Code)
(Area/Perimeter) (Perimeter) (Principal Value 1/Area) (Principal Value 2/Area)
(Location X) (Location Y) (Orientation Angle)

Below is for outer contour only

(Area/Perimeter) (Perimeter) (Principal Value 1/Area) (Principal Value 2/Area)
(Location X) (Location Y) (Orientation Angle)

Below is for the last on-scene object

(Boundary Starting-Point Location: Row) (Column) (Chain Code Length) (Number of Holes)
(The Complete Chain Code)
(Area/Perimeter) (Perimeter) (Principal Value 1/Area) (Principal Value 2/Area)
(Location X) (Location Y) (Orientation Angle)

Below is for outer contour only

(Area/Perimeter) (Perimeter) (Principal Value 1/Area) (Principal Value 2/Area)
(Location X) (Location Y) (Orientation Angle)

Below is for the first on-scene hole

(Boundary Starting-Point Location: Row) (Column) (Chain Code Length) (Number of Holes)
(The Complete Chain Code)
(Area/Perimeter) (Perimeter) (Principal Value 1/Area) (Principal Value 2/Area)
(Location X) (Location Y) (Orientation Angle)

Below is for the second on-scene hole

(Boundary Starting-Point Location: Row) (Column) (Chain Code Length) (Number of Holes)
(The Complete Chain Code)
(Area/Perimeter) (Perimeter) (Principal Value 1/Area) (Principal Value 2/Area)
(Location X) (Location Y) (Orientation Angle)

......

Below is for the last on-scene hole

(Boundary Starting-Point Location: Row) (Column) (Chain Code Length) (Number of Holes)
(The Complete Chain Code)
(Area/Perimeter) (Perimeter) (Principal Value 1/Area) (Principal Value 2/Area)
(Location X) (Location Y) (Orientation Angle)
4-2-5-3 Trace Files

The trace files are used to record the operation procedures of each operation program. They are opened when the operation programs are loaded and they are closed when exiting the operation programs. The user can review the procedures by using an editor or by printing out the contents of the trace files.

4-3 Example Operation

Some operation examples are given below. The major operation of the system includes camera calibration, on-scene image processing, object learning, and object identifying and positioning.
4-3-1 Camera Calibration

There are two kinds of camera calibration, aperture and position. The position calibration is not necessary if the camera has not been moved since last operation. On the other hand, the camera aperture needs to be adjusted because the illumination condition may vary from time to time. The aperture should be checked when the system is turned on. If there is a malfunction during the image processing, aperture is the first factor to be checked. The following is a sample run which illustrates aperture and position calibrations.

4-3-1-1 Aperture Calibration

When the option for aperture calibration is selected, the system will bring up a live picture of the scene on the graphic monitor. After some objects are placed inside the viewing window, the system is ordered to grab the image, (Figure 4.1). The system samples the image and calculates the average gray level of the image. The average gray level of the image shown in Figure 4.1 is 40.
The camera aperture can be adjusted manually to yield a clear gray-level image when the live image is brought up. The average gray level alone may serve as the threshold value for the image processing. Most of the time, especially when it is difficult for the user to reach the camera, a proper shift value from the average gray level can assist the aperture to get a desired threshold value. The average can, therefore, serve as a reference for determining the threshold. The guideline for setting a proper threshold is to obtain a sharp binary image which correctly shows the details of the shapes and contours of the objects and the holes. Without the shift value, i.e.,
use the average as a threshold, the binary image may be as in Figure 4.2.

![Binary Image with Average Gray Level as Threshold.](image)

**Figure 4.2** Binary Image with Average Gray Level as Threshold.

The above binary image is unacceptable. By adding a shift value of -7 to the average gray level, the threshold is 33. The resulting binary image is shown in Figure 4.3(a). If a shift value of -15 is applied, the threshold is 25. The resulting binary image is shown in Figure 4.3(b) which is a clear image and 25 is the proper threshold value.
Figure 4.3 Binary Images with Threshold Values of (a) 33; and (b) 25.
4-3-1-2 Position Calibration

The position calibration of the camera includes the determination of a new conversion scale and the homogeneous transformation matrix of the camera position in the robot-base coordinates, $^B[T]_C$. When the system brings up the live picture, a standard object is placed inside the viewing window, and the image is grabbed. The standard object is a thick rectangular aluminum plate with a dark surface and an area of 50.7 by 25.5 millimeters. After the image is input, the system processes the image of the standard object. Then, the system calculates the conversion scale by dividing the real area of the standard object by the area of its image, and then taking the square root of the result (see Section 3-2-9). In this sample run, the area of the image of the standard object is 1382.11 square pixels. The new scale factor is: \[
[\frac{(50.7 \times 25.5)}{1382.11}]^{\frac{1}{2}} = 0.9672\text{(mm/pixel)}.
\]

In order to determine $^B[T]_C$, the user is asked to measure and input the position and orientation angle of the standard object relative to the robot-base coordinates. The system will automatically generate $^B[T]_O$ and $^C[T]_O$, and then obtain $^B[T]_C$ (see Section 3-5-1-3). In this sample run, the location of the standard object $(X,Y,Z)$ is $(236,328,-900)$ and the orientation angle is $-90^\circ$ in the
robot-base coordinates, and \((131.54, 152.03, 0.00)\) and \(0'\) in image-frame coordinates.

\[
\begin{bmatrix}
0.00 & 1.00 & 0.00 & 236.00 \\
-1.00 & 0.00 & 0.00 & 328.00 \\
0.00 & 0.00 & 1.00 & -900.00 \\
0.00 & 0.00 & 0.00 & 1.00 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1.00 & 0.00 & 0.00 & 131.54 \\
0.00 & 1.00 & 0.00 & 152.03 \\
0.00 & 0.00 & 1.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 1.00 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1.00 & 0.00 & 0.00 & -131.72 \\
0.00 & 1.00 & 0.00 & -151.87 \\
0.00 & 0.00 & 1.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 1.00 \\
\end{bmatrix}
\]

and

\[
\begin{bmatrix}
0.00 & 1.00 & 0.00 & 84.13 \\
-1.00 & 0.00 & 0.00 & 459.72 \\
0.00 & 0.00 & 1.00 & -900.00 \\
0.00 & 0.00 & 0.00 & 1.00 \\
\end{bmatrix}
\]
4-3-2 Image Processing

The image processing operation obtains the geometrical properties of the on-scene objects. The location and orientation angle of the objects are in the measure of the image-frame coordinates, i.e., the pixel units. The area, perimeter and two principal values are converted to the real measure, i.e., the metric units. In the following sample run, a nut, a bolt, a wrench, and a rectangular object are processed. All objects are placed inside the viewing window (Figure 4.4).

Figure 4.4 Objects to be Processed.
The following are the resulting integer parameters, chain-codes, real-number invariants, locations, and orientations, listed sequentially for the nut, bolt, wrench, and rectangle. These parameters are arranged in the format described in Section 4-2-5-2.

Note that the nut and the wrench have 1s as the last elements in their integer-parameter fields, which indicate that there is one hole inside each of the objects. The holes in the nut and wrench have the following parameters.
Note that the last integer in each integer-parameter field indicates the hole-object relation. Hole No.1 has 1 to indicate that the hole belongs to object No.1, which is the nut. Hole No.2 has 3 to indicate that it belongs to object No.3, which is the wrench.

After the image processing, the system displays the boundary outlines of the objects on the graphic monitor along with a principal axis for each object (Figure 4.5).
Figure 4.5 Boundaries of Processed Objects with their Principal Axes.
4-3-3 System Training

The vision system is trained by learning new objects, i.e., by storing new object parameters into the data-base. The training operation reads the object information from the on-scene object data-file obtained during the image-processing operation. The user can order the system to learn one of the objects or all the objects.

The system starts the training operation by displaying the boundary out-lines of all the on-scene objects on the graphic monitor. In this sample run, there are four objects in the viewing window (see Figure 4.4). For the first time of training, any object is new to the system because there is no object previously stored in the data base. ONE BY ONE option is chosen at the beginning of the training operation. Object locations are listed on the computer terminal. The objects are numbered in the order of top to bottom and left to right.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>65.47</td>
<td>149.44</td>
</tr>
<tr>
<td>2</td>
<td>122.17</td>
<td>66.21</td>
</tr>
<tr>
<td>3</td>
<td>144.02</td>
<td>248.48</td>
</tr>
<tr>
<td>4</td>
<td>171.27</td>
<td>152.20</td>
</tr>
</tbody>
</table>
An object is selected by entering its serial number to the system. Object 2 is selected for this illustration. Its image is then displayed on the graphic monitor (Figure 4.6). After the user confirms the selection, the system

![The Selected Object](image)

Figure 4.6 The Selected Object.

compares the object invariants with the invariants of the previously recorded objects in the data base. "NO MATCH" indicates it is a new object. The user is asked to input an object name. "BOLT" is the name for the object. The parameters of the BOLT are then stored into computer memory along with the name. In this illustration, object 4 is also selected and named "RECTANGLE".
For the second time of training, another image of the four objects is processed with a different view of the BOLT (Figure 4.7). All objects are selected this time. The

![Figure 4.7 Four Objects for Training. (Bolt has a New View.)](image)

object boundaries are displayed one by one on the graphic monitor. If an object is already known by the vision system, the system will reject it. If the object is new, the system will ask a name for it. When the name is given, the system checks the name against the recorded names. If no identical name is found, the object will be recorded as a new one. However, if the name is duplicated, the system
will display both the on-scene object and the recorded object on the graphic monitor (Figure 4.8). The system will ask whether the object is really a new one or actually a different view of a known object. If the object is new, a different name is requested. If the object is already known, it is recorded as a new view under the same name.

In this training operation, object 1 (the RECTANGLE) is a known object, object 2 (the WRENCH) and object 4 (the NUT) are new objects, and object 3 is a new view of the BOLT. After the above two training operations, the data base is as follows: (see next page).

Figure 4.8 An On-Scene BOLT (left) and a Recorded BOLT (right).
The format of the data base above is described in Section 4-2-5-1.
4-3-4 Object Identification and Positioning

This operation includes identifying an object requested by the user and determining the object location and orientation in the robot-base coordinates. The operation starts with an object name input by the user. The system checks the name against the recorded names to find out the number of views recorded for the object. The system picks up the invariants listed under the name in the data base. The recorded invariants are then compared with those of the on-scene objects obtained from image-processing operation. If there is a match, the object is said to be found in the camera viewing window. If more than one set of invariants are previously recorded under the same name, more than one view have been recorded for the object. All of the invariants will be put into the comparison, one set at a time, in order to achieve a match. If there is no match, the object is considered not to be inside the viewing window. In this sample run, the NUT, the RECTANGLE, and the BOLT are inside the viewing window and the WRENCH is not inside the viewing window (see Figure 4.9). The NUT, the RECTANGLE, and the BOLT are recognized even though their location and orientation differ from those at the time when the objects are learned during the training operations (compare Figure 4.5, 4.7, and 4.9). This outcome indicates that the object invariants, such as
the number of holes, area, perimeter, and principal values, are checked out OK. The **WRENCH**, the object at lower-right corner of the image, is actually inside the viewing window. The system could not identify it because this view of the **WRENCH** had never been learned.

**Figure 4.9** Object Identification.
**RECTANGLE**, **NUT**, and the Stand-Up **BOLT** are in the Viewing Window, but the Pre-Recorded **WRENCH** is not Inside the Viewing Window.

When an object is identified inside the viewing window, its location \((X,Y)\) and orientation angle \((\phi)\) is substituted into \(cX_0\), \(cY_0\), and \(c\phi_0\) in Eq.3-5 to construct the homogeneous transformation matrix \(c[T]o\). Then \(c[T]o\) is multiplied by \(b[T]c\) to obtain \(b[T]o\). \(b[T]c\) is determined
in camera position calibration (see Section 4-3-1-2). In the sample run, the RECTANGLE's location \((X,Y)\) is \((146.37, 67.44)\) and the orientation is \(10.47^\circ\) in the image-frame coordinates. The homogeneous transformation matrices are:

\[
\begin{bmatrix}
0.98 & -0.18 & 0.00 & 146.37 \\
0.18 & 0.98 & 0.00 & 67.44 \\
0.00 & 0.00 & 1.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 1.00
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.00 & 1.00 & 0.00 & 84.13 \\
-1.00 & 0.00 & 0.00 & 459.72 \\
0.00 & 0.00 & 1.00 & -900.00 \\
0.00 & 0.00 & 0.00 & 1.00
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.18 & 0.98 & 0.00 & 151.39 \\
-0.98 & -0.18 & 0.00 & 313.28 \\
0.00 & 0.00 & 1.00 & -900.00 \\
0.00 & 0.00 & 0.00 & 1.00
\end{bmatrix}
\]

The elements in matrix \(B[T]_o\) are utilized to determine the location and orientation angle of the RECTANGLE in the robot-base coordinates. The location is \((b_xo, b_yo, b_zo)\),
i.e., (151.39, 313.28, -900). The orientation angle is 
\[ \text{ATAN}(\sin \beta / \cos \beta) \text{, i.e., } -79.60^\circ \]. The image of the 
RECTANGLE is displayed alone on the graphic monitor (Figure 4.10).

![Figure 4.10 RECTANGLE is Located.](image)

4-4 System Reliability

The RECTANGLE is used to test the reliability of the 
system to identify objects. The image of the RECTANGLE is 
processed 44 times, with an unique view but different 
location and orientation (see Table 4.1). Table 4.1 
suggests that the objects can be recognized about 93.18%
with 5.41% tolerance if the averages are chosen to be the reference invariants.

Table 4.1 Multiple Runs of *RECTANGLE* and Match Percentage with Tolerance.

<table>
<thead>
<tr>
<th>RUN</th>
<th>LOCATION</th>
<th>ORIENTATION</th>
<th>AREA</th>
<th>PERIMETER</th>
<th>PRINCIPAL VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n</td>
<td>X</td>
<td>Y</td>
<td>φ</td>
<td>A/P</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>185.4</td>
<td>169.2</td>
<td>-77.9</td>
<td>8.11</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>164.5</td>
<td>186.8</td>
<td>11.1</td>
<td>8.00</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>104.5</td>
<td>98.2</td>
<td>15.0</td>
<td>8.13</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>93.1</td>
<td>105.3</td>
<td>-72.9</td>
<td>8.00</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>185.6</td>
<td>183.1</td>
<td>68.0</td>
<td>7.92</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>167.7</td>
<td>124.7</td>
<td>47</td>
<td>8.27</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>150.1</td>
<td>105.7</td>
<td>-46.4</td>
<td>8.63</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>127.5</td>
<td>113.0</td>
<td>-65.4</td>
<td>8.06</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>103.2</td>
<td>168.9</td>
<td>68.9</td>
<td>7.94</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>150.8</td>
<td>132.6</td>
<td>44</td>
<td>8.34</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>178.0</td>
<td>136.8</td>
<td>-87.4</td>
<td>8.35</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>147.0</td>
<td>221.0</td>
<td>48.5</td>
<td>8.43</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>173.3</td>
<td>168.9</td>
<td>-11.5</td>
<td>8.04</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>117.2</td>
<td>142.3</td>
<td>16.4</td>
<td>8.00</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>137.6</td>
<td>198.5</td>
<td>61.2</td>
<td>8.02</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>162.9</td>
<td>205.0</td>
<td>-30.0</td>
<td>7.95</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>183.6</td>
<td>147.1</td>
<td>-20.1</td>
<td>8.06</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>162.7</td>
<td>133.4</td>
<td>-41.3</td>
<td>8.51</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>116.0</td>
<td>144.2</td>
<td>-81.4</td>
<td>8.09</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>202.1</td>
<td>192.6</td>
<td>-60.4</td>
<td>7.99</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>152.1</td>
<td>199.9</td>
<td>60.2</td>
<td>8.03</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>112.0</td>
<td>174.6</td>
<td>69.6</td>
<td>7.93</td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>106.6</td>
<td>152.0</td>
<td>73.6</td>
<td>7.92</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>137.0</td>
<td>218.3</td>
<td>45.8</td>
<td>8.66</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>113.6</td>
<td>138.6</td>
<td>43.9</td>
<td>8.61</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
<td>98.6</td>
<td>193.3</td>
<td>74.7</td>
<td>7.95</td>
</tr>
<tr>
<td>27</td>
<td>27</td>
<td>108.8</td>
<td>242.0</td>
<td>-27.6</td>
<td>7.85</td>
</tr>
<tr>
<td>28</td>
<td>28</td>
<td>124.5</td>
<td>109.5</td>
<td>26.9</td>
<td>8.01</td>
</tr>
<tr>
<td>29</td>
<td>29</td>
<td>119.6</td>
<td>126.6</td>
<td>-85.7</td>
<td>8.32</td>
</tr>
</tbody>
</table>

(to be continued)
Table 4.1 (continued)

<table>
<thead>
<tr>
<th>RUN</th>
<th>LOCATION</th>
<th>ORIENTATION</th>
<th>AREA</th>
<th>PERIMETER</th>
<th>PRINCIPAL VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>Y</td>
<td>φ</td>
<td>A/P</td>
<td>I1/A</td>
</tr>
<tr>
<td>30</td>
<td>119.6</td>
<td>126.7</td>
<td>-85.8</td>
<td>8.28</td>
<td>156.37</td>
</tr>
<tr>
<td>31</td>
<td>114.4</td>
<td>190.6</td>
<td>-69.9</td>
<td>7.91</td>
<td>162.75</td>
</tr>
<tr>
<td>32</td>
<td>114.4</td>
<td>190.6</td>
<td>-69.9</td>
<td>7.91</td>
<td>161.95</td>
</tr>
<tr>
<td>33</td>
<td>148.8</td>
<td>197.1</td>
<td>4.7</td>
<td>8.22</td>
<td>155.23</td>
</tr>
<tr>
<td>34</td>
<td>152.9</td>
<td>214.7</td>
<td>-89.0</td>
<td>8.31</td>
<td>152.82</td>
</tr>
<tr>
<td>35</td>
<td>152.9</td>
<td>214.8</td>
<td>-89.4</td>
<td>8.21</td>
<td>154.76</td>
</tr>
<tr>
<td>36</td>
<td>150.8</td>
<td>210.4</td>
<td>-20.5</td>
<td>7.84</td>
<td>163.56</td>
</tr>
<tr>
<td>37</td>
<td>85.5</td>
<td>140.0</td>
<td>-65.4</td>
<td>7.98</td>
<td>163.13</td>
</tr>
<tr>
<td>38</td>
<td>85.5</td>
<td>140.0</td>
<td>-65.4</td>
<td>7.97</td>
<td>163.13</td>
</tr>
<tr>
<td>39</td>
<td>168.0</td>
<td>184.1</td>
<td>84.5</td>
<td>8.14</td>
<td>157.75</td>
</tr>
<tr>
<td>40</td>
<td>163.4</td>
<td>186.6</td>
<td>28.3</td>
<td>7.89</td>
<td>163.83</td>
</tr>
<tr>
<td>41</td>
<td>165.0</td>
<td>173.3</td>
<td>-38.7</td>
<td>8.31</td>
<td>157.14</td>
</tr>
<tr>
<td>42</td>
<td>163.1</td>
<td>176.3</td>
<td>75.4</td>
<td>7.92</td>
<td>162.38</td>
</tr>
<tr>
<td>43</td>
<td>150.5</td>
<td>187.9</td>
<td>-0.4</td>
<td>8.37</td>
<td>153.96</td>
</tr>
<tr>
<td>44</td>
<td>116.2</td>
<td>169.7</td>
<td>83.0</td>
<td>8.12</td>
<td>158.79</td>
</tr>
</tbody>
</table>

|             | MAXIMUM  | 8.66      | 164.17 | 226.11 | 54.81 |
|             | MINIMUM  | 7.84      | 152.82 | 213.46 | 51.18 |
|             | AVERAGE  | 8.13      | 159.59 | 220.23 | 52.87 |

STANDARD DEVIATION, $\delta_{n-1}$ | 0.22 | 3.73 | 2.50 | 0.80 |
\%
 FROM AVERAGE | 2.71 | 2.34 | 1.14 | 1.51 |
\%
 OF MATCH | 79.54 | 61.36 | 75.00 | 75.00 |

$2\cdot\delta_{n-1}$ | 0.44 | 7.46 | 5.00 | 1.60 |
\%
 FROM AVERAGE | 5.41 | 4.67 | 2.27 | 3.03 |
\%
 OF MATCH | 93.18 | 100.0 | 90.91 | 93.18 |

NOTE:
1) Object LOCATION ($X, Y$ in pixel units) and ORIENTATION ($\phi$ in degrees) are given in image-frame coordinates.
2) AREA, PERIMETER, and PRINCIPAL VALUES are given in the metric measure.
3) The real invariants of the RECTANGLE is measured with a vernier caliper in millimeters. The results are: $A/P = 8.48$; $P = 152.4$; $I_1/A = 214.21$; $I_2/A = 54.19$. 

138
Chapter 5

Conclusion

The example operations presented in Chapter 4 have indicated several advantages of the algorithms used in the experimental robot vision system. A brief system evaluation is given in this chapter. Some further tests and improvements are suggested.

5-1 System Evaluation

The developed robot vision system has the ability of identifying multiple objects with arbitrary shapes and sizes, and recognizing them from different views. The system can also determine the position of an object in the robot base coordinates at the same time as it identifies the object. Major image processing operations are concentrated on the boundary pixels. Most of the geometrical features of the objects are computed rapidly with closed-path integration. All required information for identifying and locating the objects is extracted within
only one pass on the raw gray-level images input from the frame grabber. The combination of the rough search and fine search for the boundaries further limits the average amount of operation on each image pixel. The process time for an image with four objects is about 5 seconds.

The system performs well under natural illumination. It can semi-automatically calibrate the camera aperture by providing a proper threshold value for various illumination conditions. The system can also semi-automatically produce a homogeneous transformation matrix for each new camera position, and a new scale for converting object's geometrical properties from image-frame measure to metric measure.

The vision system has a significant anti-noise and survival capability. It can detect and remove both background noise pixels and boundary noise pixels. The operation controls for boundary tracking can protect image processing from a crash caused by out-of-control tracking, a very narrow pixel-string, and cross-border objects.

The system's reliability of recognizing an identical object is about 95% with a 6% tolerance on object's real-number invariants. The accuracy of pinpointing the location of a recognized object is within 5 millimeters for
X-Y position and within 5° for Z-axis orientation, with the camera placed 900 millimeters above the work plane.

The system is designed in an option-selecting fashion for all operations. Therefore, no special training is required of a user before operating on the system.

5-2 Future Improvement

The communication link between the robot vision system and the PUMA robot controller was built but has not yet been tested due to a malfunction of the robot controller. When this communication link is operational, the robot arm should be able to move to an object under the guidance of the vision system. Furthermore, a completed two-way communication link may allow a fully automatic calibration of the conversion scale and camera position by siting the camera on the robot and feeding the camera position from the robot controller to the vision system. An auto-aperture device on the camera can achieve a fully automatic threshold determination. An auto-focus camera lens would permit image capture from variable distance and, therefore, a rough recognition and a fine recognition can cooperate to increase the accuracy and reliability of object identification.
A computer with enough RAM is desired to execute the entire program as a whole. Or the program size can be reduced by rewriting it, at least partially, with C language or assembly language.

One regret of the developed vision system is the lack of capability to distinguish overlapping objects. Applying other approaches rather than global thresholding, such as gradient edge detection or local thresholding may overcome this weakness. However, operation speed and system simplicity should remain as the first priority of a design.
References

Computer Vision. 
Prentice-Hall, Inc., 

On-Line X-Ray Inspection of 
Packaged Foods. 
Vision '86 Conference Proceedings, 
Detroit, Michigan, June 1986.

[Cognex 1987] Cognex Corporation. 
Product Summary. 
Vision '87, 
Detroit, Michigan, June 1987.

[Coreco 1987] Coreco, Inc. 
Product Summary. 

[Cunningham 1981] Cunningham, Robert. 
Segmenting Binary Images. 
Robotics Age, Vol.3, No.4, 
July/August, 1981.

[Data 1988] Data Transtation, Inc. 
1988 Product Summary. 
Marlboro, Massachusetts, 1988.

[Freeman 1989] Freeman, Herbert, Editor. 
Machine Vision for Inspection and 
Measurement. 
Academic Press, Inc., 
San Diego, California, 1989.

Machine Vision for Inspection and 
Measurement: Is Industry Ready for 
Machine Vision. 
Academic Press, Inc., 
San Diego, California, 1989.


*Moment-Preserving Thresholding: A New Approach.* 

[Tuma 1987] Tuma, Jan J. 
Second Edition, 
McGraw-Hill Book Company, 

[UNIMATE 1982] *User’s Guide to VAL.* 
Unimation Inc., A Condec Company, 
May 1982.

[Wilf 1981] Wilf, Joel M. 
*Chain-Code.* 
Robotics Age, Vol.3, No.2, 
March/April 1981.

*Machine Vision.* 
Fairmont Press Inc., 
Lilburn, Georgia, 1987

*Corner Detection Using the Facet Model.* 
IEEE, January, 1983.
Appendix A.

Operation Manual
This robot vision system contains two closely related programs. Program IP performs camera calibration and image processing for the on-scene objects. Program TP performs system training and object positioning. Now, let us turn on the robot vision system and tour through the two programs.

**Loading Programs**

Let’s plug in the adaptor of the camera, switch on the graphic monitor and the computer monitor, and turn on the Intel microcomputer system. After the computer is booted up, the programs can be loaded by command:[1]

FUNG/IP.RUN

or

FUNG/TP.RUN

for program IP or TP. Each program will open a trace file to record the operation courses, TRACE1 for IP and TRACE2 for TP. Both programs will read the same data file, FUNG/PARAMS. After the data are read, the main menu of operation options is displayed.

---

[1] The commands that we type will begin at first column. The massages displayed by the computer begin at second column or beyond.
A-1 Running Program IP

We will see the following messages when loading IP:

OPEN EYE.
READING DATA BASE.
DATA BASE IS LOADED.
MAKING A BACKUP.
SAVING DATA BASE.
DATA BASE IS SAVED.

The main menu of IP is as follows.

WHAT TO DO NEXT?
1. CAMERA CALIBRATION;
2. REVISE OPERATION PARAMETERS;
5. PROCESS ON SCENE OBJECTS;
6. SAVE ON SCENE OBJECTS;
8. SAVE DATA BASE;
9. EXIT TO OPERATION SYSTEM.

ENTER YOUR CHOICE:

We can start one of the operations by entering its code number. If we enter a number which is not in the list, the menu will be displayed again for a proper number to be entered.

A-1-1 CAMERA CALIBRATION

Let's choose the camera calibration by entering number "1". The menu for camera calibration will be displayed.

CAMERA CALIBRATION:
1. APERTURE AND NEW SCALE;
2. POSITION AND NEW SCALE;
3. BOTH APERTURE & POSITION;
9. EXIT.
ENTER YOUR CHOICE:

Let's select option 1 by entering "1".

CAMERA APERTURE CALIBRATION:

When the live image is brought up on the graphic monitor, we place some sample objects inside the viewing window of the camera.

ADJUST THE CAMERA'S APERTURE THEN PRESS ENTER KEY TO GRAB THIS PICTURE.

We can adjust the camera aperture manually if we wish. As soon as we press the ENTER key, the live image is grabbed and is displayed on the graphic monitor.

GRABBED IMAGE.

DISPLAYING.

IS IMAGE INPUT OK? (Y/N)

If the grabbed image is no good or we want another one, we will enter "N". The same procedure will be repeated to grab a new image. If we enter "Y", then

CURRENT THRESHOLD GRAY LEVEL IS 33

The system uses the previously stored threshold to convert the grabbed image into a binary image and display it.

IS THE IMAGE ACCEPTABLE? (Y/N)

A "N" is entered if the quality of the binary image should be improved.

IMAGE'S AVERAGE GRAY LEVEL IS: 40
CURRENT SHIFT VALUE IS: -7
ENTER A NEW ONE: -15
CURRENT THRESHOLD GRAY LEVEL IS 25

The binary image produced by this threshold is displayed.
IS THE IMAGE ACCEPTABLE? (Y/N)

The shift value should be adjusted until the binary image shows all details of the objects within the viewing window and background noise is limited. A "y" is then entered.

FINAL THRESHOLD VALUE IS: 25
APERTURE CALIBRATION HAS BEEN COMPLETED.
NOW FOR DETERMINING THE NEW SCALE.
PLACE THE STANDARD OBJECT AT THE CENTER OF THE VIEWING WINDOW.
PRESS ENTER KEY TO GRAB THIS PICTURE.
GRABBED IMAGE.
DISPLAYING.

The grabbed image is displayed.

IS IMAGE INPUT OK? (Y/N)
Y

The image is processed and the object boundaries is displayed on the graphic monitor with a list of their locations on the computer monitor.

WHICH ONE IS THE STANDARD OBJECT? (0=NOT THERE)
We must find the standard object, which is a rectangular object with a flat surface, and enter its serial number from the location list. If we can not find it, enter "0".

Be very careful, we can not afford to make a mistake.
Otherwise, the scale will turn out all wrong. It is better to place the standard object alone in the view window.

SCALE DETERMINATION IS COMPLETED.
We will find ourselves back to the calibration menu.

CAMERA CALIBRATION:
1. APERTURE NEW SCALE;
2. POSITION NEW SCALE;
3. BOTH APERTURE & POSITION;
9. EXIT.
ENTER YOUR CHOICE:

Let's choose option 2 this time.

PLACE THE STANDARD OBJECT AT THE CENTER OF THE VIEWING WINDOW.
PRESS ENTER KEY TO GRAB THIS PICTURE.
GRABBED IMAGE.
DISPLAYING.
IS IMAGE INPUT OK? (Y/N)
Y

The image is processed and the object boundaries are displayed on the graphic monitor with a list of their locations on the computer monitor.

WHICH ONE IS THE STANDARD OBJECT? (0=NOT THERE)
We enter the serial number of the standard object. The new scale is calculated. We have to measure the displacements of the standard object on X- and Y-axes of the robot base coordinates, and the orientation angle about the Z-axis.

INPUT BASE POSITION OF THE OBJECT.
INPUT LOCATION. (X,Y,Z IN MILLIMETERS)
236,328,-900
INPUT ORIENTATION ANGLE. (DEGREES)
-90
LOCATION:( 236.00, 328.00,-900.00); ORIENTATION:(-90.00).
ARE THESE VALUES CORRECT? (Y/N)

If there is a mistake, we can enter "N" to make a correction. If the input values are correct, we enter "Y". The system will automatically construct the homogeneous transformation matrix of the camera position in the robot-base coordinates. Then we are back to the calibration menu. We can, next time, choose option 3 to do the aperture calibration and the position calibration at the
same time. For now, we may enter "9" to quit the camera calibration and go back to the main menu.

**A-1-2 CHANGE OPERATION PARAMETERS**

Suppose we want to change the operation parameters, we can select option 2 from the main menu. We will see a new menu:

```plaintext
REVIEW OPERATION CONTROL PARAMETERS:
  1: OPERATION BORDERS;
  2: STEP LENGTH FOR BOUNDARY ROUGH-SEARCH;
  3: STEP LENGTH FOR GRAY LEVEL AVERAGE;
  9: EXIT.
ENTER CHOICE:
```

Let's choose option 1 to change operation borders.

```plaintext
CURRENT OPERATION BORDER:
  UP-LEFT CORNER: ( 16, 16)
  BOTTOM-RIGHT CORNER: (232,304)
ANY CHANGE? (Y/N)
```

Enter "Y" to make changes.

```plaintext
ENTER UP-LEFT CORNER: (ROW,COLUMN)
  8,8
ENTER BOTTOM-RIGHT CORNER: (ROW,COLUMN)
  240,312
CURRENT OPERATION BORDER:
  UP-LEFT CORNER: ( 8, 8)
  BOTTOM-RIGHT CORNER: (240,312)
ANY CHANGE? (Y/N)
```

Enter "N" to go back to the menu.

```plaintext
REVIEW OPERATION CONTROL PARAMETERS:
  1: OPERATION BORDERS;
  2: STEP LENGTH FOR BOUNDARY ROUGH-SEARCH;
  3: STEP LENGTH FOR GRAY LEVEL AVERAGE;
  9: EXIT.
ENTER CHOICE:
```
Let's enter "2".

CURRENT BOUNDARY SEARCH STEP LENGTH:  4
ANY CHANGE? (Y/N)

Enter "Y" to make a change.

ENTER NEW STEP LENGTH:
8
CURRENT BOUNDARY SEARCH STEP LENGTH:  8
ANY CHANGE? (Y/N)

Enter "N" to go back to the menu.

REVIEW OPERATION CONTROL PARAMETERS:
1: OPERATION BORDERS;
2: STEP LENGTH FOR BOUNDARY ROUGH-SEARCH;
3: STEP LENGTH FOR GRAY LEVEL AVERAGE;
9: EXIT.
ENTER CHOICE:

Enter "3".

CURRENT SAMPLE STEP LENGTH FOR AVERAGING GRAY LEVEL:  4
ANY CHANGE? (Y/N)

Enter "Y" to make change.

ENTER NEW STEP LENGTH:
8
CURRENT SAMPLE STEP LENGTH FOR AVERAGING GRAY LEVEL:  8
ANY CHANGE? (Y/N)

Enter "N" and back to the menu.

REVIEW OPERATION CONTROL PARAMETERS:
1: OPERATION BORDERS;
2: STEP LENGTH FOR BOUNDARY ROUGH-SEARCH;
3: STEP LENGTH FOR GRAY LEVEL AVERAGE;
9: EXIT.
ENTER CHOICE:

Enter "9" to go back to the main menu.
A-1-3 PROCESS ON-SCENE IMAGE

We have prepared the robot-vision system by calibrating the camera and defining the operation parameters. Now, let the system do some on-scene-object image processing. By choosing option 5 from the main menu, the images of the objects within the camera viewing window will be processed. When a live image is displayed on the graphic monitor, we can place some objects into the viewing window, e.g., a bolt, a wrench, a nut, and the rectangular object.

PRESS ENTER KEY TO GRAB THIS PICTURE.
GRABBED IMAGE.
IS IMAGE INPUT OK? (Y/N)
Y

The image processing starts. Every finding of a boundary will be signalled. "OUTER" is for the finding of a outer-boundary of an object, and "HOLE" is for a hole boundary. The signals are followed by some numbers. The first two numbers are the row and column number of the pixel, from it, the fine search for the boundary starts. The third number is the column number of the first found boundary pixel. The rest of the numbers are the gray levels of the pixels inside the fine search region. For an outer boundary of an object, the signal will be:

OUTER 100 60 62 32 30 24 19 16
The signal tells us that there is an outer boundary of an never-detected object; fine search starts at row 100 and column 60; first boundary pixel is found at row 100 and column 62; the gray level on the pixels within the fine search region are 32, 30, 24, 19, 16. We can see a big jump of gray level from 30 to 24, where the first boundary pixel was found. For a hole boundary, the signal will be: HOLE 172 212 216 16 16 17 21 29. We can see the first boundary pixel was found at row 172 and column 216, where was a gray-level jump from 21 to 29. The process will continue to the end unless it is interrupted by one of the following accidents:

1) TRACKING WENT WRONG.
2) THIS WAS A NOISE.
3) THIS WAS AN INCOMPLETE OBJECT.
   or THIS WAS A HOLE IN AN INCOMPLETE OBJECT.

In these cases, we are given a chance to examine the image.

NEED TO EXAMINE THE IMAGE? (Y/N) Y
EXAMINE IMAGE BY 20X20 PORTION.
ENTER CENTER OF THE PORTION: (ROW,COLUMN)

We can enter any combination of row number and column number to see what is in that portion of the image. The portion of the image will be listed.

MORE? (Y/N) N

The image-processing will continue. After all boundaries of objects and holes have been tracked, the boundary
outlines will be displayed along with the vectors indicating the object centroids and orientation angles.

BOUNDARIES OF ON-SCENE OBJECTS.
CLEARING PICTURE MAP. PLEASE WAIT.
CONSTRUCTING THE BOUNDARY OF AN ON-SCENE OBJECT.
CONSTRUCTING THE BOUNDARY OF AN ON-SCENE OBJECT.
CONSTRUCTING THE BOUNDARY OF AN ON-SCENE OBJECT.
CONSTRUCTING THE BOUNDARY OF AN ON-SCENE OBJECT.
(One statement for each object boundary. Here, four statements for four objects).
DISPLAYING.

We are then back to the main menu. At this point, we should choose option 6 to save the on-scene object for program TP to use. Before exiting to the operating system (option 9), we may want to save the data base (option 8) if we want to keep the updated operation parameters.

A-2 Running Program TP

When TP is loaded, we will see the following messages:

START WORKING.
READING DATA BASE.
DATA BASE IS LOADED.
MAKING A BACKUP.
SAVING DATA BASE.
DATA BASE IS SAVED.

The main menu of TP is the following.

WHAT DO YOU WISH TO DO NEXT?
3. TRAINING;
4. POSITION PUMA TO AN OBJECT;
7. REARRANGE DATA BASE;
8. SAVE DATA BASE;
9. EXIT TO OPERATING SYSTEM.
ENTER YOUR CHOICE:
Similar to program IP, we may enter any number to start one of the operations. If an incorrect number is entered, the menu will be displayed again for a proper number. Let's check what in the data base by enter "7".

TOTAL 2 OBJECTS AND 1 HOLES.
1-BOLT 2-WRENCH
REARRANGE RECORDED DATA:
1. DELETE AN OBJECT;
2. RENAME AN OBJECT;
3. REVISE TRANSFORMATION MATRIX;
4. NEW SCALE;
9. EXIT.
ENTER YOUR CHOICE:

We don't want to do anything to the data base right now.
Enter "9" to exit. Just keep in mind that there are two objects, a BOLT and a WRENCH, have been previously recorded.

A-2-1 TRAINING

The training operation begins with displaying the boundary out-lines of the on-scene objects on the graphic monitor. These object boundaries are obtained from the image processing by program IP.

BOUNDARIES OF ON-SCENE OBJECTS,
CLEARING PICTURE MAP. PLEASE WAIT.
CONSTRUCTION THE BOUNDARY OF AN ON-SCENE OBJECT.
CONSTRUCTION THE BOUNDARY OF AN ON-SCENE OBJECT.
CONSTRUCTION THE BOUNDARY OF AN ON-SCENE OBJECT.
CONSTRUCTION THE BOUNDARY OF AN ON-SCENE OBJECT.
(Four statements for four objects.)
DISPLAYING.
START TRAINING.
There are 4 objects with 2 holes on the scene. How to proceed? 1—one by one; 2—all of them; 9-exit.

Let's try one by one first. The locations of the objects will be listed.

<table>
<thead>
<tr>
<th>Location</th>
<th>Row</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>64.55</td>
<td>232.66</td>
</tr>
<tr>
<td>2</td>
<td>74.08</td>
<td>122.00</td>
</tr>
<tr>
<td>3</td>
<td>144.08</td>
<td>249.59</td>
</tr>
<tr>
<td>4</td>
<td>153.18</td>
<td>111.17</td>
</tr>
</tbody>
</table>

Enter object 1, 2, or 3,... (0=QUIT)

Let's select object 2, which is a nut and has never been learned before.

Display the object on graphic monitor.

The nut is displayed.

Do you mean this one? (Y/N)

If we did not choose the right one, enter "N". The locations of the objects will be listed again for a new selection. This option also provides us an opportunity to abort the operation. If the displayed object is the one we requested and we do want to continue the operation, enter "Y".

Matching an on-scene object to the recorded objects.
Match recorded object 0
This is a new object.
Enter the name for this object:
NUT

The NUT is put into the data base and we are brought back to select another training operation.

How to proceed? 1—one by one; 2—all of them; 9-exit.

Let's choose ALL OF THEM this time. Keep in mind that we already had a BOLT and a WRENCH in the data base and just
added the NUT to the data base. Starting with first object,

DISPLAY THE OBJECT ON GRAPHIC MONITOR.

The BOLT is displayed.

DO YOU MEAN THIS ONE? (Y/N)
Y (If we want to ignore this object, enter "N".)

MATCHING AN ON-SCENE OBJECT TO THE RECORDED OBJECTS.
MATCH RECORDED OBJECT 1
THIS IS A KNOWN OBJECT, CALLED BOLT . CORRECT? (Y/N)

Because the displayed object is truly a bolt, we enter "Y".

ENTER (Y) TO UPDATE DATA, OR ELSE TO PASS.

OK. Enter "Y" to update data.

NEXT OBJECT.
DISPLAY THE OBJECT ON GRAPHIC MONITOR.

The NUT is displayed.

DO YOU MEAN THIS ONE? (Y/N)
Y

MATCHING AN ON-SCENE OBJECT TO THE RECORDED OBJECTS.
MATCH RECORDED OBJECT 3
THIS IS A KNOWN OBJECT, CALLED NUT . CORRECT? (Y/N)

Let's pretend we don't know about the NUT. Enter "N".

DISPLAY BOTH OBJECTS ON SCREEN.
CLEARING PICTURE MAP. PLEASE WAIT.
CONSTRUCTION THE BOUNDARY OF AN ON-SCENE OBJECT.
CONSTRUCTION THE BOUNDARY OF A RECORDED OBJECT.
LEFT--ON SCENE; RIGHT--RECORDED.
DISPLAYING.
ARE THEY LOOKED THE SAME? (Y/N)

Let's try to fool the system, enter "N".

OBJECTS CAN NOT BE DISTINGUISHED.
NEXT OBJECT.
DISPLAY THE OBJECT ON GRAPHIC MONITOR.

The RECTANGLE is displayed.

DO YOU MEAN THIS ONE? (Y/N)
Y
MATCHING AN ON-SCENE OBJECT TO THE RECORDED OBJECTS.
MATCH RECORDED OBJECT 0
THIS IS A NEW OBJECT.
ENTER THE NAME FOR THIS OBJECT:
RECTANGLE

We have just added the RECTANGLE to the data base.

NEXT OBJECT.
DISPLAY THE OBJECT ON GRAPHIC MONITOR.

The WRENCH is displayed.

DO YOU MEAN THIS ONE? (Y/N)
Y
MATCHING AN ON-SCENE OBJECT TO THE RECORDED OBJECTS.
MATCH RECORDED OBJECT -2
ALMOST MATCH AN OBJECT CALLED WRENCH .

Well, the on-scene wrench only passed the primary match.

DISPLAY BOTH OBJECT ON SCREEN.
CLEARING PICTURE MAP. PLEASE WAIT.
CONSTRUCTION THE BOUNDARY OF AN ON-SCENE OBJECT.
CONSTRUCTION THE BOUNDARY OF A RECORDED OBJECT.
LEFT--ON SCENE; RIGHT--RECORDED.
DISPLAYING.
ARE THEY LOOKED SIMILAR? (Y/N)
Y (Yes, they do look almost the same.)
ENTER (Y) TO UPDATE DATA, OR ELSE TO PASS.
Y
NEXT OBJECT.
NO MORE.

All done. We go back to the main menu. Choose option 8 to save the data base. Then choose option 9 to exit to operating system.

Let's run program IP again to process another image of the four objects. At this time, place the WRENCH and the BOLT in a different way so that the different views of them can be obtained. Save the on-scene object parameters and exit to the operation system. Then run program TP again
and choose TRAINING. It doesn't matter we choose ONE BY ONE or ALL OF THEM option. When the BOLT is selected, we will see:

MATCH RECORDED OBJECT 0
THIS IS A NEW OBJECT.
ENTER THE NAME FOR THIS OBJECT:
BOLT
ONE OBJECT ALREADY HAS THIS NAME.
1 VIEW(S) HAD BEEN RECORDED.

The boundary of the recorded BOLT and the on-scene BOLT are displayed.

DISPLAY BOTH OBJECTS ON SCREEN.
CLEARING PICTURE MAP. PLEASE WAIT.
CONSTRUCTION THE BOUNDARY OF AN ON-SCENE OBJECT.
CONSTRUCTION THE BOUNDARY OF A RECORDED OBJECT.
LEFT--ON SCENE; RIGHT--RECORDED.
DISPLAYING.
IS THIS OBJECT
1. ACTUALLY A NEW OBJECT;
2. DIFFERENT VIEW OF SAME OBJECT;
3. SAME VIEW OF SAME OBJECT;
4. DISPLAY MORE VIEW;
9. QUIT.

We enter "2". The new view of the BOLT is stored. Next, let's select the WRENCH and play around a little.

MATCH RECORDED OBJECT 0
THIS IS A NEW OBJECT.
ENTER THE NAME FOR THIS OBJECT:
WRENCH
ONE OBJECT ALREADY HAS THIS NAME.
1 VIEW(S) HAD BEEN RECORDED.
DISPLAY BOTH OBJECTS ON SCREEN.
CLEARING PICTURE MAP. PLEASE WAIT.
CONSTRUCTION THE BOUNDARY OF AN ON-SCENE OBJECT.
CONSTRUCTION THE BOUNDARY OF A RECORDED OBJECT.
LEFT--ON SCENE; RIGHT--RECORDED.
DISPLAYING.
IS THIS OBJECT
1. ACTUALLY A NEW OBJECT;
2. DIFFERENT VIEW OF SAME OBJECT;
3. SAME VIEW OF SAME OBJECT;
4. DISPLAY MORE VIEW;
9. QUIT.
4 NO MORE. ENTER CHOICE 1, 2, 3, OR 9.
1 GIVE ANOTHER NAME:
   WRENCH2

The image of this on-scene wrench will be stored into the
data base as a new object instead of a new view of the
WRENCH. Of course we don't want this. We can rename
WRENCH2 to WRENCH later in the operation of REARRANGE DATA
BASE. Please note: if we chose option 3, the system would
be forced to make a mistake by mixing up the object
parameters of two completely different views. However, if
the previously recorded object parameters was so bad that
the system could not recognize the same view of the same
object, option 3 should be chosen to update the data.

We have had enough TRAINING. Let's do something new.

A-2-2 REARRANGE DATA BASE

Let's choose option 7 to do some data base
arrangement.

TOTAL 6 OBJECTS AND 2 HOLES.
   1-BOLT  2-WRENCH  3-WRENCH2  4-BOLT  5-NUT
6-RECTANGLE
REARRANGE RECORDED DATA:
   1. DELETE AN OBJECT;
   2. RENAME AN OBJECT;
   3. REVISE TRANSFORMATION MATRIX;
   4. NEW SCALE;
   5. MARGIN FOR RECOGNIZING OBJECTS;
   9. EXIT.
ENTER YOUR CHOICE:

Let's rename the "WRENCH2". Enter "2".

WHICH ONE IS TO BE RENAMED? (0=EXIT)
3
ENTER THE NEW NAME:
WRENCH (If we changed our mind, enter "WRENCH2" for no rename.)

We are back to the option list.

TOTAL 6 OBJECTS AND 2 HOLES.
1-BOLT 2-WRENCH 3-WRENCH 4-BOLT 5-NUT 6-RECTANGLE
REARRANGE RECORDED DATA:
1. DELETE AN OBJECT;
2. RENAME AN OBJECT;
3. REVISE TRANSFORMATION MATRIX;
5. MARGIN FOR RECOGNIZING OBJECTS;
4. NEW SCALE;
9. EXIT.
ENTER YOUR CHOICE:

We can rename any recorded object one by one and one view at a time. For now, let's delete the NUT. Enter "1".

WHICH ONE IS TO BE DELETED? (0=EXIT)
5
ARE YOU SURE? (Y/N)

If we change our mind, enter "N" to abort the deletion.

Let's give a go-ahead by entering "Y". The NUT along with its hole is deleted from the data base. We can delete all recorded object one by one and delete one view at a time for multiple-view objects. When last object is deleted, we will get a message "DATA BASE IS EMPTY.". We do not want to do this at this time.

TOTAL 5 OBJECTS AND 1 HOLES.
1-BOLT 2-WRENCH 3-WRENCH 4-BOLT 5-RECTANGLE
REARRANGE RECORDED DATA:
1. DELETE AN OBJECT;
2. RENAME AN OBJECT;
3. REVISE TRANSFORMATION MATRIX;
5. MARGIN FOR RECOGNIZING OBJECTS;
4. NEW SCALE;
9. EXIT.
ENTER YOUR CHOICE:

we are back to the option list. Let's adjust one of the margins for identifying objects. Enter "5".

CURRENT MARGINS FOR OBJECT ID:
1-AREA; 2-PERIMETER; 3-MAX I; 4-MIN I
0.05 0.05 0.15 0.15
WHICH ONE IS TO BE CHANGED? (0=QUIT)
3
INPUT NEW MARGIN (MUST < 1):
0.1
CURRENT MARGINS FOR OBJECT ID:
1-AREA; 2-PERIMETER; 3-MAX I; 4-MIN I
0.05 0.05 0.10 0.15
WHICH ONE IS TO BE CHANGED? (0=QUIT)
0
We are again back to selecting operation options. The rest options are REVISE TRANSFORMATION MATRIX and NEW SCALE.

Even though these parameters should be updated through the camera calibration. Here is an opportunity to do some minor adjustments. However, before making any change, we must fully understand the functions and implications of the homogeneous transformation matrix and the scale. Let's go through the procedures for a demonstration but do not change anything. Enter "3".

CURRENT TRANSFORMATION MATRIX IS:
-0.00120 1.00000 0.00000 84.13239
-1.00000 -0.00120 0.00000 459.72461
0.00000 0.00000 1.00000 -900.00000
0.00000 0.00000 0.00000 1.00000
WHICH ELEMENT IS TO BE REVISED: (ROW,COLUMN)
2,4
ENTER THE NEW VALUE:
459.72461
MORE? (Y/N)
N
Enter "4" to select NEW SCALE from the option list.

CURRENT SCALE IS: 0.96717
ENTER THE NEW ONE: 0.96717

We are all through the data-base arrangement operation. We can exit to the main menu of TP.

A-2-3 DETERMINE OBJECT POSITION

This is the time for positioning an object in the robot-base coordinates. We start the object positioning with selecting an object.

SELECTING AN OBJECT.
CHOICE AN OBJECT FROM MEMORY(1), SCENE(2), OR QUIT(9).

Let's choose an object from the data base, enter "1".

PICKING OBJECT BY NAME(1), OR FROM LIST(2), OR QUIT(9).
1
ENTER NAME OF THE OBJECT:
BOLT

MATCHING A RECORDED OBJECT TO THE ON-SCENE OBJECTS.
MATCH ON-SCENE OBJECT
THE OBJECT LOCATES AND ORIENTS AT:
331.44 110.32 -900.00 0.00 0.00
(The numbers above tell us that the bolt locates at X=331.44mm, Y=110.32mm, Z=-900mm and rotated 58.65° about the Z-axis in the robot base coordinates.)

The boundary of the bolt will be displayed on the graphic monitor at the position where it is located in the image frame, and with a vector indicating its orientation.

DISPLAY THE OBJECT.
CLEARING PICTURE MAP. PLEASE WAIT.
CONSTRUCTION THE BOUNDARY OF AN ON-SCENE OBJECT.
DISPLAYING.

Let's try another object. This time, select an object from the recorded name list.

CHOSE AN OBJECT FROM MEMORY(1), SCENE(2), OR QUIT(9).
1
PICKING OBJECT BY NAME(1), OR FROM LIST(2), OR QUIT(9).
2
1-BOLT  2-WRENCH  3-WRENCH  4-BOLT  5-RECTANGLE
ENTER OBJECT NUMBER (0=QUIT):
3
(There are two views for the wrench. It doesn't matter which number we enter.)
MATCHING A RECORDED OBJECT TO THE ON-SCENE OBJECTS.
MATCH ON-SCENE OBJECT -2
NO EXACT MATCH, BUT
THERE IS A SIMILAR ONE.

There is one of the on-scene objects passed the primary match. We will see what is it looked like.

DISPLAY BOTH OBJECTS ON SCENE.
CLEARING PICTURE MAP. PLEASE WAIT.
CONSTRUCTION THE BOUNDARY OF AN ON-SCENE OBJECT.
CONSTRUCTION THE BOUNDARY OF A RECORDED OBJECT.
LEFT--ON SCENE; RIGHT--RECORDED.
DISPLAYING.

The two boundaries are displayed on the graphic monitor.

We can see the two is very similar to each other.

THE OBJECT LOCATES AND ORIENTS AT:
  155.18  340.56  -900.00  0.00  0.00  0.00  66.13
DISPLAY THE OBJECT.
CLEARING PICTURE MAP. PLEASE WAIT.
CONSTRUCTION THE BOUNDARY OF AN ON-SCENE OBJECT.
DISPLAYING.

Let's try an object picked from the camera viewing window.

CHOOSE AN OBJECT FROM MEMORY(1), SCENE(2), OR QUIT(9).
2
THERE ARE TOTAL 4 OBJECTS ON THE SCENE.
LOCATION:   ROW   COLUMN
1       78.78    156.54
2       119.08    71.19
3       129.11    247.46
4       182.17    146.45

Note: these locations are given in the image frame coordinates, i.e., pixel units.

ENTER THE NUMBER OF THE OBJECT.
4

THE OBJECT LOCATES AND ORIENTS AT:
230.37    277.38    -900.00    0.00    0.00    9.97

DISPLAY THE OBJECT.
CLEARING PICTURE MAP. PLEASE WAIT.
CONSTRUCTION THE BOUNDARY OF AN ON-SCENE OBJECT.
DISPLAYING.
CHOOSE AN OBJECT FROM MEMORY(1), SCENE(2), OR QUIT(9).
9

We are back to the main Menu. Choose "8" to save the database because when successfully positioning an object selected from the database, its real-number invariants are updated. Then, let's choose "9" to quit program TP.
Appendix B.

Subroutine List
APERT(SL1, TH, SH)

INPUTS: SL1, SH
OUTPUTS: TH

This subroutine calibrates the camera aperture by obtaining a threshold value. The user manually adjusts the camera aperture until an acceptable live image is showed on the graphic monitor. Then the threshold value (TH) is determined by subroutine THRES.

BINARY(P, TH)

INPUTS: P, TH
OUTPUTS: N/A

This subroutine displays the 64 gray level image in a binary form without disturbing the raw image. The gray level of each pixel of the raw image P(ROW, COLUMN) is compared with the threshold value (TH). A gray level (63) representing WHITE will be sent to the frame grabber if P(ROW, COLUMN) is greater than TH, otherwise, a gray level of BLACK (5) will be sent.

BNDY(P, BP, CHAIN, RO, CO)

INPUTS: P, BP, CHAIN, RO, CO
OUTPUTS: P

This subroutine constructs a boundary of an object or a hole. Location (RO, CO) is the centroid of the object or hole. The starting point is determined by shifting RO and CO by the value of BP(1) and BP(2), respectively. Following the directions provided by the chain-codes (CHAIN), the boundary pixels are turned to BLACK on the WHITE background set by subroutine CLEAR.

BNDYOS(P, NUM, HNUM, OIP, HIP, OCH, HCH, RO, CO)

INPUTS: P, NUM, HNUM, OIP, HIP, OCH, HCH, RO, CO
OUTPUTS: P

This subroutine constructs a boundary of a selected on-scene object, as well as its hole(s). The boundary parameter of the selected object, whose sequential
number is NUM, are transferred to BP and CHAIN. Subroutine BNDY is called to construct the outer boundary. If there is any hole in the object, the hole-object relation code, HIP(#,4), is examined to determine which hole belongs to the object. When the hole is found, subroutine BNDY will be called to construct the hole boundary.

BNDYRD(P, RONUM, RHNUM, ROIP, RHIP, ROCH, RHCH, RO, CO)

INPUTS: P, RNUM, RHNUM, ROIP, RHIP, ROCH, RHCH, RO, CO
OUTPUTS: P

This subroutine constructs a boundary image of a recorded object, as well as its hole(s). The procedures are the same as in subroutine BNDYOS except the boundary parameters come from the recorded objects and holes.

BOTH(P, NUM, HNUM, OIP, HIP, OCH, HCH, RNUM, RHNUM, ROIP, RHIP, ROCH, RHCH)

INPUTS: NUM, HNUM, OIP, HIP, OCH, HCH, RNUM, RHNUM, ROIP, RHIP, ROCH, RHCH
OUTPUTS: P

This subroutine displays the boundaries of two selected objects for visual comparison by the user. The one on the left side is an on-scene object from the camera viewing window. The right one is a recorded object from the data base.

BOTTOM(P, R, C, JMAX, TH)

INPUTS: P, R, C, JMAX, TH
OUTPUTS: P, C

This subroutine continues the boundary tracking while the tracking reaches the bottom border of the operation frame. The tracking will keep going to the right within the object region where P(R,C+1) ≤ TH. Object pixels on the border will be marked with a temporary marker (-1) but the tracking directions will not be recorded into the chain-code. Normal tracking will resume when tracking leaves the object region, or right-tracking will take over if tracking reaches the right border (JMAX).
BTRACE(P,BORDER,CHAIN,CHLEN,R0,C0,TH,ERROR)

INPUTS: P,BORDER,R0,C0,TH
OUTPUTS: CHAIN,CHLEN,ERROR

This subroutine tracks an outer boundary of an object and produce a chain code, CHAIN. The boundary tracking begins at the starting point of the boundary, P(R0,C0). The tracking proceeds counter-clockwise after the initial tracking direction (D) is determined. The boundary pixels will be temporarily marked, chain-code will be generated, turns will be made in order to stay on the boundary, boundary noises will be removed, and tracking on operation borders will be specially handled. Tracking will terminate if (1) the starting point is actually an isolated object pixel; (2) too many turns required for a single tracking step; (3) chain-code grows too long (CHLEN>300); or (4) boundary tracking comes back to its starting point.

CALI(BORDER,SL,TH,SH,SL1,BTC,SCALE,STD)

INPUTS: BORDER,SL,SH,SL1
OUTPUTS: TH,BTC,SCALE

This subroutine calibrates the aperture and/or position of the camera with user assistance. One of the three options can be chosen by the user: aperture only, position only, or both.

CLEAR(P)

INPUTS: P
OUTPUTS: P

This subroutine clears the image map by putting a high gray-level value (63=color of white) to every pixel.

CONTRO(BORDER,SL,SL1)

INPUTS: N/A
OUTPUTS: BORDER,SL,SL1

This subroutine allows the user to review and set the operation parameters.
DBASE(RONUM,RHNUM,RNAMES,ROIP,RHIP,ROCH,RHCH,
     RORP,RHRP,BTC,MARGIN,SCALE)

    INPUTS: N/A
    OUTPUTS: RONUM,RHNUM,RNAMES,ROIP,RHIP,ROCH,RHCH,
             RORP,RHRP,BTC,MARGIN,SCALE

This subroutine manages the data base. Operations include deleting an object along its hole(s), renaming an object, and redefining new margins for object identification. The homogeneous transformation matrix of the camera position in the robot-base coordinates and the scale for converting object properties from image-frame measure to metric measure could also be manually adjusted, even though they are usually determined automatically during camera calibration.

DIREC2(D,DR,DC,OD)

    INPUTS: D
    OUTPUTS: DR,DC,OD

This subroutine sets control values of a tracking direction (D) for boundary construction and moment calculation. The odd-number-direction control code (OD) is also given a value for each tracking direction, odd direction has a code of 1 while even direction has 0.

DIRECT(D,DR,DC,CR,CC)

    INPUTS: D
    OUTPUTS: DR,DC,CR,CC

    this subroutine sets check point control values of a tracking direction (D) for boundary tracking.

DISP(P)

    INPUTS: P
    OUTPUTS: P

This subroutine displays an image on the graphic monitor for either a grabbed image, a binary image, or a boundary image.
DREAD(RONUM, RHNUM, RNAMES, ROIP, RHIP, ROCH, RHCH, RORP, RHRP, BTC, MARGIN, SCALE, STD, TH, SH, SL1, SL, BORDER)

**INPUTS:** N/A  
**OUTPUTS:** RONUM, RHNUM, RNAMES, ROIP, RHIP, ROCH, RHCH, RORP, RHRP, BTC, MARGIN, SCALE, STD, TH, SH, SL1, SL, BORDER

This subroutine reads recorded parameters of objects and holes, as well as the control parameters for operations, from the data file named PARAMS in directory FUNG. All parameters are backed up to a file called PARAMS.OLD by calling subroutine DSAVE with a control code of 0.

DSAVE(RONUM, RHNUM, RNAMES, ROIP, RHIP, ROCH, RHCH, RORP, RHRP, BTC, MARGIN, SCALE, STD, TH, SH, SL1, SL, BORDER, CONT)

**INPUTS:** RONUM, RHNUM, RNAMES, ROIP, RHIP, ROCH, RHCH, RORP, RHRP, BTC, MARGIN, SCALE, STD, TH, SH, SL1, SL, BORDER, CONT  
**OUTPUTS:** N/A

This subroutine saves recorded parameters of objects and holes, as well as the control parameters for operations, to the data file named PARAMS or PARAMS.OLD in directory FUNG. PARAMS.OLD or PARAMS will be opened according the control code, CONT. If control code is 0, PARAMS.OLD is opened to backup the previous data file. If control code is 1, PARAMS is opened to save the new data.

ERASE(P, BORDER, CHAIN, CHLEN, RO, CO, MARK, TH)

**INPUTS:** BORDER, CHAIN, CHLEN, RO, CO, MARK, TH  
**OUTPUTS:** P

This subroutine erases the markers on a tracked boundary by replacing the markers with an eraser. The eraser, ERASER, is set to an integer greater than threshold value (TH) for an outer boundary (MARK<0), or less than threshold for a hole boundary (MARK>63). The markers on operation borders will not be erased.
GRAB(P)

INPUTS: N/A
OUTPUTS: P

This subroutine grabs and inputs an image from the frame grabber. The frame grabber displays a live image on the graphic monitor. User orders the frame grabber to grab the image. The image is input to the computer with post-increment read. Then the grabbed image is displayed on the graphic monitor. The user determines if the image input is successful. If not, the image input will be repeated.

HTRACE(P,BORDER,CHAIN,CHLEN,R0,C0,TH,ERROR)

INPUTS: P,BORDER,R0,C0,TH
OUTPUTS: CHAIN,CHLEN,ERROR

This subroutine tracks a hole boundary and generate a chain-code. The operation for hole-boundary tracking is similar to outer-boundary tracking (BTRACE) except the followings. 1) pixels with below-threshold gray level are on the right of the boundary while pixels with above-threshold gray level are on the left; 2) hole boundary tracking will not start at or reach any operation borders.

INVERS(MATRX1,MATRX2)

INPUTS: MATRX1
OUTPUTS: MATRX2

This subroutine inverses a homogeneous transformation matrix.

IP

This is the main program of camera calibration and image processing. Trace file is opened to record operations. Data base is read. A operation menu is displayed. The user can choose either operation option from the menu.
LEFT(P,R,C,IMAX,TH)

INPUTS: P,R,C,IMAX,TH
OUTPUTS: P,R

This subroutine continues the boundary tracking while the tracking starts at or reaches the left border of the operation frame. The tracking will keep going downward within the object region where \( P(R+1,C) \leq TH \). Object pixels on the border will be marked with a temporary marker (-1) but the tracking directions will not be recorded into the chain-code. Normal tracking will resume when the tracking leaves the object region, or bottom-tracking will take over if reaches the bottom border (IMAX).

LINE(P,ORP,NUM)

INPUTS: ORP,NUM
OUTPUTS: P

This subroutine draws a line along the direction of the orientation angle (ORIENT) of an object, starting from the object centroid (RO,CO). The length of the line (L1) is about one fourth of the object perimeter.

MATCH1(NUM,OIP,ORP,HNUM,HIP,HRP,RONUM,ROIP, RORP,RHNUM,RHIP,RHRP,MARGIN,MATCH)

INPUTS: NUM,OIP,ORP,HNUM,HIP,HRP,RONUM,ROIP, RORP,RHNUM,RHIP,RHRP,MARGIN
OUTPUTS: MATCH

This subroutine matches an on-scene object to the recorded objects, i.e., one on-scene object vs. all recorded objects, in order to determine whether the selected on-scene object is a known object to the system. If there is no match, MATCH=0. A match is achieved if the recorded object parameters fall in between the allowed limits. If object parameters match, store the serial number (L) of the recorded object and set MATCH=-L as a primary match. If hole parameters also match, set MATCH=L as complete match.
MATCH2 (RNUM, ROIP, RORP, RHNUM, RHIP, RHRP, ONUM, OIP, ORP, HNUM, HIP, HRP, MARGIN, MATCH)

INPUTS: RNUM, ROIP, RORP, RHNUM, RHIP, RHRP, ONUM, OIP, ORP, HNUM, HIP, HRP, MARGIN
OUTPUTS: MATCH

This subroutine matches a recorded object to the on-scene objects, i.e., one recorded object versus all on-scene objects, in order to determine whether the selected recorded object is inside the camera viewing window.

MATCH3 (RONUM, RNAMES, NAME, NUMMAT, NMATCH)

INPUTS: RONUM, RNAMES, NAME
OUTPUTS: NUMMAT, NMATCH

This subroutine checks a name (NAME) against the recorded names (RNAMES). There may be more than one matches. A NUMMAT > 1 indicates that more than one view of the particular object had been recorded.

MOMENT (P, MARK, CHAIN, CHLEN, R0, C0, NODD, MT)

INPUTS: P, MARK, CHAIN, CHLEN, R0, C0
OUTPUTS: P, NODD, MT

This subroutine computes the moments for either an object or a hole, and marks its boundary as well. Start from the starting point (R0, C0) and follow the direction provided by the chain code (CHAIN), the boundary pixels are marked, P(R, C) = MARK. At the same time, the moments (MT) are calculated by the closed-path integration. Simultaneously, the total of odd number directions (NODD) is added up.

MOVE(P, MATCH, HNUM, OIP, HIP, OCH, HCH, ORP, BTC)

INPUTS: P, MATCH, HNUM, OIP, HIP, OCH, HCH, ORP, BTC
OUTPUTS: N/A

This subroutine transforms the object location and orientation from the image-frame coordinates to the robot-base coordinates.
MULTI(MATRX1, MATRX2, MATRX3)

INPUTS: MATRX1, MATRX2
OUTPUTS: MATRX3

This subroutine multiplies two matrices (MATRX1 and MATRX2) and output the product (MATRX3).

ONE(MARGIN, NUM, HNUM, OIP, HIP, OCH, HCH, ORP, ARP, HRP, R NAMES, RONUM, RHNUM, ROIP, RHIP, ROCH, RHCH, RORP, RHRP)

INPUTS: MARGIN, NUM, HNUM, OIP, HIP, OCH, HCH, ORP, ARP, HRP
OUTPUTS: R NAMES, RONUM, RHNUM, ROIP, RHIP, ROCH, RHCH, RORP, RHRP

This subroutine records one selected object from the viewing window to the data base. The selected on-scene object is checked if it is a new object. If it is a new one (MATCH=0), the object is recorded as a new object or a new view of a known object.

OSREAD(ONUM, HNUM, OIP, HIP, OCH, HCH, ORP, ARP, HRP)

INPUTS: N/A
OUTPUTS: ONUM, HNUM, OIP, HIP, OCH, HCH, ORP, ARP, HRP

This subroutine reads the parameters of on-scene objects from a file called OSOBS in directory FUNG.

OSSAVE(ONUM, HNUM, OIP, HIP, OCH, HCH, ORP, ARP, HRP)

INPUTS: ONUM, HNUM, OIP, HIP, OCH, HCH, ORP, ARP, HRP
OUTPUTS: N/A

This subroutine saves the parameters of on-scene objects to a file called OSOBS in directory FUNG.

PARAM1(NUM, RP, NODD, CH, CHAIN, CHLEN, IP, R0, C0)

INPUTS: NUM, NODD, CHAIN, CHLEN, R0, C0
OUTPUTS: RP, CH, IP

This subroutine organizes the integer parameters (IP) for the boundary of either an object or a hole, as well as its chain code (CH) and perimeter.
**PARAM2(\text{NUM,RP,SCALE})**

**INPUTS:** NUM,SCALE  
**OUTPUTS:** RP

This subroutine organizes the real-number invariants (RP) for the region of an object or a hole. The procedures are as follows. Calculate the area, two static moments, two moments of inertia, and one product of inertia. Determine the centroid location of the region. Shift the moments of inertia and the product of inertia to the region centroid coordinates. Compute principal values and orientation angle. Scale the parameters to metric measure. And finally normalize the parameters.

**PICMAP(P)**

**INPUTS:** P  
**OUTPUTS:** N/A

This subroutine lists the gray levels on the pixels of a 20x20 portion of an image. The center of the portion \((I_0,J_0)\) is given by the user. Repeat if requested.

**POSIT(BORDER,SL,TH,BTC,SCALE,STD,CONT)**

**INPUTS:** BORDER,SL,TH,STD,CONT  
**OUTPUTS:** BTC,SCALE

This subroutine calibrates the camera position by creating the homogeneous transformation matrix (BTC) of the image frame position in the robot-base coordinates and determining the scale (SCALE) for converting the object properties from the image measure to the metric measure. A reference object with a known area is placed inside the camera viewing window. Its position and orientation in the robot-base coordinates are carefully measured and input by the user when requested.
PROCES(BORDER, SL, TH, ONUM, HNUM, OIP, HIP,
OCH, HCH, ORP, ARP, HRP, SCALE)

INPUTS: BORDER, SL, TH, SCALE
OUTPUTS: ONUM, HNUM, OIP, HIP, OCH, HCH, ORP, ARP, HRP

This subroutine detects object and hole boundaries, as well as obtain their chain codes and the geometrical properties. The operation will take place within the operation frame. The combination of rough search (with a step-length SL) and fine search pinpoints the location of either an outer boundary or a hole boundary. Boundary tracking will be launched as soon as a boundary is found, and boundary search will resume afterward. The chain-codes (OCH and HCH), integer (OIP and HIP) and real (ARP, ORP, and HRP) number parameters will be stored for each successfully tracked boundary. Hole-object relation code, HIP(HNUM,4), and number of holes in its host object, OIP(ONUM,4), will be determined for each successfully tracked hole boundary.

PUMA(BORDER, SL, TH, MARGIN, SCALE, BTC, RONUM, RHNUM,
RNAMES, ROIP, RHIP, ROCH, RHCH, RORP, RHRP)

INPUTS: BORDER, SL, TH, MARGIN, SCALE, BTC, RONUM, RHNUM,
RNAMES, ROIP, RHIP, ROCH, RHCH, RORP, RHRP
OUTPUTS: N/A

This subroutine identifies a user-requested object inside the camera viewing window and determine its location and orientation in robot-base coordinates. User selects an object from the data base. The parameters of the recorded object are compared with the parameters of the on-scene objects. If there is a matched on-scene object, its location and orientation in the robot-base coordinates will be provided by subroutine MOVE.

The user can also select an object from the viewing window. The object location and orientation will be given directly by subroutine MOVE.
RIGHT(P,R,C,IMIN,TH)

INPUTS: P,R,C,IMIN,TH
OUTPUTS: P,R

This subroutine continues the boundary tracking while the tracking reaches the right border of the operation frame. The tracking will keep going upward within the object region where P(R-1,C) ≤ TH. Object pixels on the border will be marked with a temporary marker (-1) but the tracking directions will not be recorded into the chain-code. Normal tracking will resume when tracking leaves the object region, or top-tracking will take over if tracking reaches the top border (IMIN).

THRES(P,SL1,TH,SH)

INPUTS: P,SL1,SH
OUTPUTS: TH,SH

This subroutine determines the threshold value (TH) of the image. Some typical objects are placed inside the camera viewing window. The pixels in the image are sampled with a step-length (SL1). The threshold is obtained by adding a shift value (SH) to the average gray-level (AV) of the sampled pixels. A binary image is displayed and the user judges if all the details of the objects are correctly shown. If not, the SH is adjusted until the binary image is approved.

TOP(P,R,C,JMIN,TH)

INPUTS: P,R,C,JMIN,TH
OUTPUTS: P,C

This subroutine continues the boundary tracking while the tracking reaches the top border of the operation frame. The tracking will keep going to the left within the object region where P(R,C-1) ≤ TH. Object pixels on the border will be marked with a temporary marker (-1) but the tracking directions will not be recorded into the chain-code. Normal tracking will resume when tracking leaves the object region, or left-tracking will take over if tracking reaches the left border (JMIN).
This is the main program of system training and object positioning. Trace file is opened to record operations. Data base is read. A operation menu is displayed. The user can choose either operation option from the menu.

\[
\text{TRAIN}(\text{BORDER, MARGIN, SCALE, SL, TH, RONUM, RHNUM, RNAMES, ROIP, RHS, ROCH, RHCH, RORP, RHRP})
\]

- **INPUTS:** BORDER, MARGIN, SCALE, SL, TH, RONUM, RHNUM, RNAMES, ROIP, RHS, ROCH, RHCH, RORP, RHRP
- **OUTPUTS:** N/A

This subroutine places new objects into the data base. User can select one object at a time from the camera viewing window, or choose all on-scene objects. Subroutine ONE will decide which object should be stored into data base, which one is already there and its new view should be stored; and which object parameters should be updated.

\[
\text{UPDATE}(\text{NUM, ORP, RNUM, RORP, OIP, HNUM, HIP, HRP, RHNUM, RHS, RHRP})
\]

- **INPUTS:** NUM, ORP, RNUM, RORP, OIP, HNUM, HIP, HRP, RHNUM, RHS, RHRP
- **OUTPUTS:** ORP, RORP, OIP, HNUM, HIP, HRP, RHNUM, RHS, RHRP

This subroutine updates the real-number invariants for the specified object by averaging the previously recorded invariants and the newly-obtained invariants.
Appendix C.

Flow Charts
APERT

ENTER

ADJUST CAMERA

GRAB

GRAB IMAGE

THRES

DETERMINE THRESHOLD

RETURN
BINARY

ENTER

SETUP FRAME GRABBER FOR DISPLAYING

COMMAND POST-INCREMENT WRITE

LOOP FOR ENTIRE IMAGE

COMPARE PIXEL GRAY-LEVEL VALUE WITH THRESHOLD VALUE

GREATER

WRITE VALUE OF BLACK COLOR TO FRAME GRABBER

N END LOOP

Y RETURN

WRITE VALUE OF WHITE COLOR TO FRAME GRABBER
BNDY

ENTER

SET LOCATION OF FIRST BOUNDARY PIXEL & LENGTH OF CHAIN CODE

BLACKED FIRST BOUNDARY PIXEL

LOOP FOR WHOLE CHAIN CODE

SET DIRECTION CODE

DIREC2

DIRECTION CONTROLS

SET LOCATION OF BOUNDARY PIXEL

BLACKEN BOUNDARY PIXEL

END LOOP

? N Y

RETURN
BNDYOS &
BNDYRD

ENTER

TRANSFER STARTING-POINT LOCATION AND CHAIN CODE OF OUTER BOUNDARY

CONSTRUCT OUTER BOUNDARY

ANY HOLE?

Y

LOOP FOR ALL HOLES

CHECK HOLE-OBJECT RELATION CODE AGAINST OBJECT SERIAL NUMBER

FIND HOLE?

N

TRANSFER STARTING-POINT LOCATION AND CHAIN CODE OF HOLE BOUNDARY

CONSTRUCT HOLE BOUNDARY

MORE HOLE?

Y

RETURN

N
ENTER

CLEAR IMAGE MAP

BNDYOS

CONSTRUCT ON-SCENE OBJECT BOUNDARY.

BNDYRD

CONSTRUCT RECORDED OBJECT BOUNDARY.

DISP

DISPLAY IMAGE

RETURN
BOTTOM

1. Enter
2. Check Pixel \((r, c+1)\)
   - If Object Pixel?
     - Mark Pixel \((r, c)\)
     - Move to Next Pixel on the Right
     - Reach Right Border?
       - Yes: Return
       - No: Go back to Check Pixel \((r, c+1)\)
BTRACE

ENTER

SET OPERATION BORDER LIMITS

CHECK STARTING POINT LOCATION

Y
LEFT BORDER ?
N

BTRACE-1

INITIAL TRACKING DIRECTION

NOISE ?

Y
RECORD FIRST TRACKING DIRECTION; MARK STARTING-POINT; MOVE TO NEXT PIXEL ALONG TRACKING DIRECTION

N

BTRACE-2

TRACKING BOUNDARY.

DEAD END ?

Y
RECORD TRACKING DIRECTION; MARK BOUNDARY PIXEL; MOVE TO NEXT PIXEL ALONG TRACKING DIRECTION.

N

BTRACE-3

BORDER OPERATION

REACH BORDER ?

Y

N

BTRACE-4

TRACKING STOPS ?
**BTRACE-1**

**INITIAL TRACKING DIRECTION**

- **ENTER**
  - SET TRACKING DIRECTION TO 1, (SOUTH-WEST)
  - DIRECT CHECK POINT CONTROLS
  - \( \square = \mathbf{B} \)
    - \( \square = \mathbf{B} \) ?
      - \( \text{marked boundary?} \)
        - CHECK IF \( \mathbf{A} \) IS MARKED HOLE-BOUNDARY PIXEL
          - \( \text{YES/NO} \) ?
            - \( \text{CHECK IF} \mathbf{\square} \text{ IS MARKED OUTER-BOUNDARY PIXEL} \)
              - \( \text{YES/NO} \) ?
                - \( \mathbf{F} \) IS NOISE
  - \( \square \) IS THE PIXEL CURRENTLY BEING EXAMINED;
  - \( \mathbf{A} \) AND \( \mathbf{B} \) ARE CHECK POINTS.

- **EXIT**
  - READ PIXEL A, PIXEL B, AND SO ON;
  - \( \square \) = \( \mathbf{B} \) READS PIXEL C IS A BACKGROUND PIXEL, OR PIXEL C HAS THE COLOR OF BACKGROUND
  - LOCATION OF PIXELS: TRACKING DIRECTION
  - \( \square \) IS THE PIXEL CURRENTLY BEING EXAMINED;
  - \( \mathbf{A} \) AND \( \mathbf{B} \) ARE CHECK POINTS.

- **LEFT TURN**
  - DIRECTION CODE \( \geq 7 \) ?
    - \( \text{NO} \)
      - \( \mathbf{F} \) IS NOISE
    - \( \text{YES} \)
      - \( \text{EXIT} \)
**BTRACE-2**

**TRACKING BOUNDARY**

- **A**, **A**, **A**, **B**, **P**. READ PIXEL A, PIXEL B, AND SO ON;
- **P** = **BP** READS PIXEL B IS A BACKGROUND PIXEL, OR
  PIXEL B HAS THE COLOR OF BACKGROUND;
- **C** = **OP** READS PIXEL C IS AN OBJECT PIXEL, OR
  PIXEL C HAS THE COLOR OF OBJECT;

LOCATION OF PIXELS: TRACKING DIRECTION

- **A** IS THE PIXEL CURRENTLY UNDER EXAMINATION,
  OTHERS ARE CHECK POINTS

---

**Flowchart Diagram**

- ENTER
- **C** = **OP**?
  - **Y**
    - **A** = **OP**?
      - **Y**
        - **B** = **BP**?
          - **Y**
            - **NO TURN**
          - **N**
            - **Y**
              - **REMOVE A AS BUMP NOISE (A = BP)**
            - **N**
              - **N**
                - **RIGHT TURN**
        - **N**
          - **REMOVE C AS HOLE NOISE (C = OP)**
      - **N**
        - **NO TURN**
  - **N**
    - **D** = **OP**?
      - **Y**
        - **N**
          - **P** = **BP**?
            - **Y**
              - **LEFT TURN**
            - **N**
              - **REMOVE P AS BUMP NOISE (P = BP)**
            - **N**
              - **N**
                - **DIRECTION CONTROLS**
      - **N**
        - **TOO MANY TURNS**?
          - **Y**
            - **DEAD END**
          - **N**
            - **TOO MANY TURNS**?
              - **Y**
                - **DEAD END**
              - **N**
                - **DIRECTION CONTROLS**

---

**Exit**
BTRACE-3

BORDER TRACKING

ENTER

CHECK BORDER

LEFT

? N

Y

ERROR CODE

LEFT BORDER TRACKING

SET DIRECTION CODE TO 3

DIRECT

CHECK-POINT CONTROLS

EXIT

BOTTOM

? N

Y

ERROR CODE

BOTTOM BORDER TRACKING

SET DIRECTION CODE TO 5

DIRECT

CHECK-POINT CONTROLS

RIGHT

? N

Y

ERROR CODE

RIGHT BORDER TRACKING

SET DIRECTION CODE TO 7

DIRECT

CHECK-POINT CONTROLS

TOP

? N

Y

ERROR CODE

TOP BORDER TRACKING

SET DIRECTION CODE TO 1

DIRECT

CHECK-POINT CONTROLS
BTRACE-4
TRACKING TERMINATION

ENTER

CHECK PIXEL (R,C)

MEET STARTING-POINT?

Y

BOUNDARY TRACKING IS ACCOMPLISHED

N

INCREASE CHAIN LENGTH BY 1

CHECK CHAIN LENGTH

TOO LONG?

Y

SET ERROR CODE

N

CONTINUE TRACKING

EXIT
CLEAR

1. ENTER
2. LOOP FOR ALL PIXELS IN IMAGE
3. SET PIXEL TO GRAY-LEVEL OF WHITE
4. END LOOP?
5. Y → RETURN
6. N → LOOP FOR ALL PIXELS IN IMAGE
CONTRO

ENTER

DISPLAY MENU

SELECT OPTION

OPTION 1

Y

NEW OPERATION BORDERS

N

OPTION 2

Y

NEW STEP-LENGTH FOR BOUNDARY SEARCH

N

OPTION 3

Y

NEW STEP-LENGTH FOR GRAY-LEVEL AVERAGE

N

OPTION 9

Y

RETURN
CONTRO-1
NEW OPERATION BORDERS

CONTRO-2
NEW STEP-LENGTH FOR BOUNDARY SEARCH

CONTRO-3
NEW STEP-LENGTH FOR GRAY-LEVEL AVERAGE
DISPLAY MENU

INPUT CHOICE

CHOICE 1

CHOICE 2

CHOICE 3

CHOICE 4

CHOICE 9

RETURN

DELETE AN OBJECT

RENAME AN OBJECT

ADJUST TRANSFORMATION MATRIX OF CAMERA IN ROBOT COORDINATES

NEW SCALE FOR CONVERTING OBJECT INVARIANTS
DBASE-1
DELETE A
RECORDED
OBJECT

DECREASE TOTAL
NUMBER OF RE-
CORDED HOLES
BY THE NUMBER
OF HOLES IN THE
RECORDED OBJECT

INDICATE THE OBJECT
TO BE DELETED

DECREASE NUMBER OF
RECORDED OBJECTS BY 1

CHECK SERIAL NUMBER
OF THE OBJECT

LAST OBJECT?

ANY HOLE(S)?

DELETE HOLE(S)

LOW FOR ALL RECORDED
OBJECTS BEYOND THE
DELETED OBJECT

DECREASE OBJECT
SERIAL NUMBER BY 1

END LOOP?

ORGANIZE HOLE-
OBJECT RELATION CODE

EMPTY DATA
BASE?

EXIT

EMPTY DATA
BASE MESSAGE

N

N
DELETE RECORDED HOLES

DBASE-1-1

ENTER

SET OPERATION FOR AT LEAST ONE HOLE

LOOP FOR ALL RECORDED HOLES

CHECK HOLE-OBJECT RELATION CODE AGAINST OBJECT SERIAL NUMBER

FIND HOLE

N

DECREASE NUMBER OF RECORDED HOLES BY 1

LOOP FOR ALL HOLES BEYOND DELETED HOLE

DECREASE HOLE SERIAL NUMBER BY 1

END LOOP

Y

END

EXIT

N

MORE HOLE

Y

N

Y
ORGANIZE
HOLE–OBJECT
RELATION CODE

ENTER

LOOP FOR ALL RECORDED HOLES

CHECK HOLE–OBJECT RELATION CODE AGAINST OBJECT SERIAL NUMBER

FIND HOLE?

N

END LOOP?

Y

LOOP FOR ALL HOLES BEYOND THIS HOLE

DECREASE HOLE–OBJECT RELATION CODE BY 1

END LOOP?

N

EXIT

Y
DBASE-2

RENAME A
RECORDED OBJECT

ENTER

LIST RECORDED OBJECTS

INDICATE SERIAL NUMBER OF
THE OBJECT TO BE DELETED

INPUT NEW NAME

REPLACE OLD NAME
WITH NEW NAME

EXIT
DBASE-3
ADJUST HOMOGENEOUS TRANSFORMATION MATRIX

ENTER

LIST ELEMENTS OF CURRENT TRANSFORMATION MATRIX

INDICATE THE ELEMENT TO BE ADJUSTED

INPUT NEW VALUE

REPLACE OLD VALUE WITH NEW VALUE

EXIT

DBASE-4
CHANGE SCALE

ENTER

LIST CURRENT SCALE VALUE

INPUT NEW VALUE

REPLACE OLD VALUE WITH NEW VALUE

EXIT
CHECK DIRECTION CODE

CODE=0
  Y  SET DIRECTION CONTROLS FOR DIRECTION 0
  N

CODE=1
  Y  SET DIRECTION CONTROLS FOR DIRECTION 1
  N

CODE=2
  Y  SET DIRECTION CONTROLS FOR DIRECTION 2
  N

CODE=3
  Y  SET DIRECTION CONTROLS FOR DIRECTION 3
  N

CODE=4
  Y  SET DIRECTION CONTROLS FOR DIRECTION 4
  N

CODE=5
  Y  SET DIRECTION CONTROLS FOR DIRECTION 5
  N

CODE=6
  Y  SET DIRECTION CONTROLS FOR DIRECTION 6
  N

CODE=7
  Y  SET DIRECTION CONTROLS FOR DIRECTION 7
  N

RETURN
DIRECT

CHECK DIRECTION CODE

CODE = 0

Y

SET CHECK-POINT CONTROLS FOR DIRECTION 0

N

CODE = 1

Y

SET CHECK-POINT CONTROLS FOR DIRECTION 1

N

CODE = 2

Y

SET CHECK-POINT CONTROLS FOR DIRECTION 2

N

CODE = 3

Y

SET CHECK-POINT CONTROLS FOR DIRECTION 3

N

CODE = 4

Y

SET CHECK-POINT CONTROLS FOR DIRECTION 4

N

CODE = 5

Y

SET CHECK-POINT CONTROLS FOR DIRECTION 5

N

CODE = 6

Y

SET CHECK-POINT CONTROLS FOR DIRECTION 6

N

CODE = 7

Y

SET CHECK-POINT CONTROLS FOR DIRECTION 7

N

RETURN
SETUP FRAME GRABBER FOR DISPLAYING

COMMAND POST-INCREMENT WRITE

LOOP FOR ALL PIXELS IN THE IMAGE

OUTPUT PIXEL TO FRAME GRABBER

END LOOP?

RETURN
DREAD

ENTER

OPEN DATA FILE

READ NUMBERS OF RECORDED OBJECTS AND HOLES

ANY OBJECTS?

Y

READ RECORDED NAMES, OUTER BOUNDARY'S INTEGER PARAMETERS, CHAIN CODES, AND REAL PARAMETERS

N

READ RECORDED HOLE'S INTEGER PARAMETERS, CHAIN CODES, AND REAL-NUMBER PARAMETERS

ANY HOLES?

Y

READ OPERATION AND CONTROL PARAMETERS

N

BACKUP DATA FILE

DSAVE

RETURN
DSAVE

ENTER

CHECK OPEN FILE CONTROL CODE

OPEN BACKUP FILE

Y

CODE=0

? N

OPEN DATA FILE

WRITE NUMBERS OF RECORDED OBJECTS AND HOLES

ANY OBJECTS

N

Y

WRITE RECORDED NAMES, OUTER BOUNDARY'S INTEGER PARAMETERS, CHAIN CODES, AND REAL PARAMETERS

ANY HOLES

N

? Y

WRITE RECORDED HOLE'S INTEGER PARAMETERS, CHAIN CODES, AND REAL-NUMBER PARAMETERS

WRITE OPERATION AND CONTROL PARAMETERS

RETURN
ERASE

ENTER

CHECK TYPE OF TEMPORARY MARKER

Y

MARKER < 0

N

SET ERASER FOR OUTER BOUNDARY

SET ERASER FOR HOLE BOUNDARY

REPLACE STARTING-POINT WITH ERASER

LOOP FOR ENTIRE CHAIN CODE

RETURN

MOVE TO NEXT PIXEL ON SAME DIRECTION

Y

MARKED PIXEL

N

REPLACE BOUNDARY PIXEL WITH ERASER

END LOOP

N

REACH BORDER

Y

RETURN
GRAB

ENTER

SET UP FRAME GRABBER FOR IMAGE INPUT

DISPLAY LIVE IMAGE

COMMAND FREEZE IMAGE

COMMAND POST-INCREMENT READ

LOOP FOR ENTIRE IMAGE

INPUT ONE PIXEL

END LOOP

? Y

DISPLAY INPUT IMAGE

GOOD IMAGE

N Y

RETURN
HTRACE

ENTER

HTRACE-1

INITIAL TRACKING DIRECTION

Y

NOISE

?

N

RECORD FIRST TRACKING DIRECTION;
MARK STARTING-POINT;
MOVE TO NEXT PIXEL ALONG
FIRST TRACKING DIRECTION.

HTRACE-2

CONTINUE BOUNDARY TRACKING

Y

DEAD

END

?

N

RECORD TRACKING DIRECTION;
MARK BOUNDARY PIXEL;
MOVE TO NEXT PIXEL ALONG
TRACKING DIRECTION.

Y

REACH

BORDER

?

N

N

HTRACE-3

TRACKING OUT
OF CONTROL.

Y

TRACKINGSTOPS

?

N

RETURN

N

CHAIN
CODE
TOO
LONG

?
**HTRACE-1**

*INITIAL TRACKING DIRECTION*

A, B, C: READ PIXEL A, PIXEL B, AND SO ON;  
C = OP READS PIXEL C IS AN OBJECT PIXEL, OR PIXEL C HAS THE COLOR OF OBJECT;  

LOCATION OF PIXELS:  

A, B, C: THE PIXEL CURRENTLY BEING EXAMINED;  
A AND C ARE CHECK POINTS.

```
[Diagram]
```

EXIT
HTRACE-2
TRACKING BOUNDARY

A, B, C, P, F, R, read pixel A, pixel B, and so on;

= BP reads pixel C is a background pixel, or
pixel C has the color of background;

= OP reads pixel B is an object pixel, or
pixel B has the color of object;

Location of pixels: Tracking direction

is the pixel currently under examination,
others are check points

enter

\[ C = BP \]
\[ \text{?} \]
\[ \text{?} \]
\[ \text{?} \]
\[ \text{?} \]
\[ \text{?} \]
\[ \text{?} \]
\[ \text{?} \]

remove C as hole noise ( C = BP )

remove A as bump noise ( A = OP )

right turn

too many turns?

dead end

too many turns?

left turn

dead end

direction controls

dead end

direction controls

direct
HTRACE-3

TRACKING TERMINATION

[Flowchart diagram]

ENTER

CHECK PIXEL (R,C)

MEET STARTING POINT?

Y → BOUNDARY TRACKING IS ACCOMPLISHED

N → INCREASE CHAIN LENGTH BY 1

CHECK CHAIN LENGTH

TOO LONG?

Y → SET ERROR CODE

N → CONTINUE TRACKING

EXIT
INVERS

ENTER

LOOP FOR 3 TIMES (L=1 TO 3)

TRANSPOSE COLUMN L OF MATRIX 1 TO ROW L OF MATRIX 2

SET ELEMENT (L,4) OF MATRIX 2 TO ZERO

OBTAIN ELEMENT (L,4) OF MATRIX 2 BY MULTIPLYING ROW L OF MATRIX 2 TO COLUMN 4 OF MATRIX 1

SET ELEMENT (4,L) OF MATRIX 2 TO ZERO

END LOOP

SET ELEMENT (4,4) OF MATRIX 2 TO ONE

RETURN
C-34

CAMERA CALIBRATION & IMAGE PROCESSING

BEGIN
OPEN FILE FOR OPERATION RECORD
INPUT DATABASE
SET OBJECT & HOLE PARAMETERS
DISPLAY MENU
SELECT OPTION

CALLE
CAMERA CALIBRATION

CONTRO
REVISE OPERATION PARAMETERS

PROCES
PROCESS IMAGE

OSAVE
SAVE OBJECT INFORMATION

DSAVE
SAVE DATA BASE

STOP
LEFT

1. **ENTER**

2. **REACH BOTTOM BORDER**
   - **Y**
     - **RETURN**
   - **N**
     - **CHECK PIXEL(R,C)**

3. **MARKED PIXEL**
   - **Y**
     - **RETURN**
   - **N**
     - **CHECK PIXEL(R+1,C)**

4. **OBJECT PIXEL**
   - **N**
     - **CHECK PIXEL(R+1,C)**
   - **Y**
     - **MARK PIXEL(R,C)**

5. **MOVE DOWN TO NEXT PIXEL**
LINE

ENTER

SET: NUMBER OF PIXELS ON THE LINE; LINE'S STARTING-POINT LOCATION; LINE DIRECTION.

LOOP FOR ALL PIXELS ON THE LINE.

DETERMINE LOCATION OF A PIXEL ON THE LINE

BLACKEN THE PIXEL

END LOOP

RETURN
MATCH!

SET MATCH CODE TO NO MATCH

DETERMINE LIMITS FOR EACH INVARIANT OF THE OUTER BOUNDARY OF ON-SCENE OBJECT

LOOP FOR ALL RECORDED OBJECTS

COMPARE NUMBER OF HOLES

LOOP FOR ALL INVARIANTS

COMPARE ONE INVARIANT TO ITS LIMITS.

END LOOP?

SET MATCH CODE TO PRIMARY MATCH

ANY HOLE?

MATCH1 = 1

HOLES MATCH

SET MATCH CODE TO TOTAL MATCH

RETURN
**MATCH1-1**

*HOLE MATCH*

```
MATCH CODE

ENTER

LOOP FOR ALL
ON-SCENE HOLES

CHECK HOLE-
OBJECT RELATION

FIND HOLE

Y

DETERMINE LIMITS
FOR INVARIANTS OF
ON-SCENE HOLE

LOOP FOR ALL
RECORDED HOLES

CHECK HOLE-
OBJECT RELATION

END LOOP

N

FIND HOLE

N

Y

COMPARE ONE
INARIANT TO
ITS LIMITS

WITHIN
LIMITS

N

Y

END LOOP

Y

SET MATCH CODE
TO TOTAL MATCH

MORE HOLE

N

EXIT

N

Y
```
MATCH2

ENTER

SET MATCH CODE TO NO MATCH

DETERMINE LIMITS FOR EACH INVARIANTS OF THE OUTER BOUNDARY OF RECORDED OBJECT

LOOP FOR ALL ON-SCENE OBJECTS

COMPARE NUMBER OF HOLES

LOOP FOR ALL INVARIANTS

EQUAL?

COMPARE ONE INVARIANT TO ITS LIMITS.

END LOOP?

SET MATCH CODE TO PRIMARY MATCH

ANY HOLE?

HOLE MATCH

MATCH2-1

SET MATCH CODE TO TOTAL MATCH

RETURN
**MATCH2-1**

**HOLE MATCH**

- **ENTER**
- **LOOP FOR ALL RECORDED HOLES**
- **CHECK HOLE-OBJECT RELATION**
- **FIND HOLE**
- **DETERMINE LIMITS FOR INVARIANTS OF RECORDED HOLE**
- **LOOP FOR ALL ON-SCENE HOLES**
- **CHECK HOLE-OBJECT RELATION**
- **FIND HOLE**
- **COMPUTE LIMITS FOR INVARIANTS**
- **LOOP FOR ALL INVARIANTS**
- **COMPARE ONE INVARIANT TO ITS LIMITS**
- **WITHIN LIMITS**
- **END LOOP**
- **SET MATCH CODE TO TOTAL MATCH**
- **END LOOP**
- **MORE HOLE**
- **EXIT**
MATCH3

ENTER

SET TOTAL OF MATCHES TO ZERO

LOOP FOR ALL RECORDED NAMES

COMPARE INPUT NAME TO A RECORDED NAME

ADD 1 TO TOTAL OF MATCHES AND RECORD THE SERIAL NUMBER

MATCH

END LOOP

RETURN
SET MOMENTS TO ZERO

PUT MARKER ON FIRST BOUNDARY PIXEL

LOOP FOR ENTIRE CHAIN CODE

DIRECTION CONTROLS

ADD TOTAL OF ODD DIRECTIONS

MOVE TO NEXT BOUNDARY PIXEL AND MARK IT

CONVERT INTEGER TO REAL NUMBER

CALCULATE MOMENTS

END LOOP?

RETURN
MOVE

ENTER

SET [CTO] TO IDENTICAL MATRIX

CONSTRUCT [CTO]

MULTI

[BTO] = [BTC]•[CTO]

OBTAIN OBJECT LOCATION AND ORIENTATION ON ROBOT COORDINATES

DISP

DISPLAY THE OBJECT

RETURN
MULTI

ENTER

LOOP FOR ALL ROWS OF FIRST MATRIX (L1)

LOOP FOR ALL COLUMNS OF SECOND MATRIX (L2)

SET ZERO TO ELEMENT (L1,L2) OF THIRD MATRIX

LOOP FOR ROW L1 OF FIRST MATRIX AND COLUMN L2 OF SECOND MATRIX (L3)

SUM THE PRODUCTS OF THE ELEMENTS OF FIRST MATRIX (L1,L3) AND THE ELEMENTS OF SECOND MATRIX (L3,L2)

END LOOP

Y

ASSIGN THE SUMMATION TO ELEMENT (L1,L2) OF THIRD MATRIX

N

END LOOP

Y

RETURN
ONE

ENTER

DISPLAY SPECIFIED ON-SCENE OBJECT

COMPARE THE OBJECT TO RECORDED OBJECTS

MATCH 1

EXAMINE MATCHED OBJECTS AND UPDATE DATA

ANY MATCH?

Y

INPUT NAME FOR NEW OBJECT

MATCH 3

USED NAME?

Y

DISPLAY BOTH OBJECTS INPUT USER JUDGEMENT

MORE VIEW?

Y

ANOTHER NAME

N

NEW OBJECT?

Y

UPDATE DATA

N

SAME VIEW?

Y

NEW VIEW?

N

RETURN

RECORD NEW OBJECT OR NEW VIEW
EXAMINE MATCHED OBJECTS AND UPDATE DATA

ENTER

CHECK STATUS OF MATCH

TOTAL MATCH?

Y

CORRECT NAME?

Y

DISPLAY BOTH OBJECTS

DISPLAY BOTH OBJECTS

N

UPDATE

UPDATE DATA

SIMILAR LOOK?

Y

SAME LOOK?

Y

OBJECT CAN NOT BE DISTINGUISHED

N

UPDATE DATA

EXIT
ONE-2

RECORD NEW OBJECT
OR NEW VIEW

ENTER

INCREASE NUMBER OF RECORDED OBJECTS BY 1

CHECK AVAILABLE SPACE FOR OBJECTS

ANY MORE

N

Y

RECORD OBJECT NAME; RECORD INVARlANTS; RECORD CHAIN CODES.

ANY HOLE

N

Y

RECORD HOLE(S)

EXIT

ANY

MORE

Y

CANCEL RECORDED OBJECT

INCREASE NUMBER OF RECORDED HOLE BY 1

CHECK HOLE-OBJECT RELATION

FIND HOLE

N

Y

CHECK AVAILABLE SPACE FOR HOLES

ANY MORE

N

Y

LOOP FOR ALL ON-SCENE HOLE(S)
OSREAD

1. Enter

2. Open on-scene object file

3. Read numbers of on-scene objects and holes

4. Check for any objects:
   - If no objects, return
   - If objects, read object's integer parameters, chain codes, and real-number parameters

5. Check for any holes:
   - If no holes, return
   - If holes, read hole's integer parameters, chain codes, and real-number parameters

6. Return
OSSAVE

ENTER

OPEN ON-SCENE OBJECT FILE

SAVE NUMBERS OF ON-SCENE OBJECTS AND HOLES

ANY OBJECTS?

N

SAVE OBJECT'S INTEGER PARAMETERS, CHAIN CODES, AND REAL-NUMBER PARAMETERS.

Y

ANY HOLES?

N

SAVE HOLE'S INTEGER PARAMETERS, CHAIN CODES, AND REAL-NUMBER PARAMETERS.

Y

RETURN
PARAM1

ENTER

TRANSFER CHAIN CODE

TRANSFER STARTING-POINT LOCATION AND CHAIN-CODE LENGTH

CALCULATE PERIMETER OF THE BOUNDARY

RETURN
PARAM2

1. ENTER
2. CHECK MOMENT VALUE
3. VALID
   - N
4. RETURN
5. Y
   - DETERMINE MOMENTS TO CAMERA'S COORDINATES
   - DETERMINE LOCATION OF OBJECT OR HOLE CENTROID
   - MODIFY MOMENTS TO THE CENTROID COORDINATES
   - CALCULATE PRINCIPAL VALUES
   - DETERMINE ORIENTATION ANGLE
   - SCALE MOMENTS TO REAL MEASURE
   - NORMALIZE MOMENTS
   - RETURN
INPUT CENTER POSITION OF THE IMAGE PORTION

DETERMINE ENDS OF THE PORTION

ENSURE PORTION NOT OVER IMAGE BORDER

PRINT OUT THE PORTION

MORE?

RETURN
POSIT

ENTER

PROCESS IMAGE WITHOUT SCALING

INDICATE THE STANDARD OBJECT

DETERMINE NEW SCALE

CHECK CONTROL CODE

SCALE ONLY

INPUT STANDARD-OBJECT LOCATION & ORIENTATION IN ROBOT COORDINATES

CONSTRUCT HOMOGENEOUS TRANSFORMATION MATRIXES [BTO] AND [CTO]

DETERMINE [OTC] BY INVERSING [CTO]

OBTAIN [BTC] BY MULTIPLYING [BTO] TO [OTC]

RETURN
PROCES

ENTER

GRAB

GRAB IMAGE

SET OPERATION PARAMETER

START BOUNDARY SEARCH
AT TOP-LEFT CORNER
OF THE IMAGE BORDER

PROCES-1

SEARCH FOR OUTER BOUNDARY

FIND BOUNDARY ?

Y

TRACKED BOUNDARY

Y

BTRACE

TRACK OUTER BOUNDARY

NOISE ?

Y

TRACING FAILURE ?

N

TRACKING REACH BORDER

Y

ERASE MARKED BOUNDARY

Y

ERASE

A

B

C

D

E
PROCES (cont.)

OUTER BOUNDARY MOMENTS AND INTEGER PARAMETERS

MOVE ONE STEP TO THE RIGHT

REACH RIGHT BORDER

MOVE ONE STEP DOWN

BOTTOM BORDER?

SEARCH FOR HOLE BOUNDARY

FIND BOUNDARY?

TRACKED BOUNDARY?

TRACK HOLE BOUNDARY

NOISE?

C-55
PROCES (cont.)

- TRACKING FAILURE
  - Y: ERASE MARKED BOUNDARY
  - N: PROCESES-3

PROCESES-3

- SET HOLE-OBJECT RELATION CODE
- HOLE IN A CROSS-BORDER OBJECT
  - Y: PROCESES-3
  - N: HOLE MOMENTS AND INTEGER PARAMETERS

- MOVE ONE STEP TO THE RIGHT
- REACH RIGHT BORDER
  - Y: MOVE ONE STEP DOWN
  - N: PROCESES-4

PROCESES-4

- REAL-NUMBER INVARIANTS OF OUTER & HOLE BOUNDARIES

- DISPLAY BOUNDARIES

RETURN
**Process - 1**

*Search for Outer Boundary*

1. **Enter**
2. Check pixel on top or left border
   - **N**
   - **Y**
     - **N**
     - **Y**
     - **N**
     - **Y**
   - **N**
   - **Y**
3. Check pixel one step to the right
4. **N**
5. **N**
6. Find a cross-border object
7. **Exit**

**Decision Points:**
- **Check Background Pixel:**
  - **Y**
  - **N**
- **Marked Boundary Pixel:**
  - **Y**
  - **N**

**Flow:**
- Check pixel on top or left border
- If not marked boundary pixel, check next pixel
- If marked boundary pixel, find a cross-border object
- If no new boundary is found, exit.
PROCES-2
SEARCH FOR HOLE BOUNDARY

ENTER

CHECK PIXEL ONE STEP TO THE RIGHT

NO BOUNDARY IS FOUND

OBJECT PIXEL?

Y

CHECK PIXEL ON NEXT COLUMN

OBJECT PIXEL?

N

CHECK SAME PIXEL AGAIN

NOT A NEW BOUNDARY

MARKED BOUNDARY PIXEL?

Y

N

EXIT
**PROCES-3**

**HOLE-OBJECT RELATION AND HOLE MOMENTS**

ENTER

SAVE TRACKER LOCATION

CHECK PIXEL ONE COLUMN TO LEFT

MARKED PIXEL?

Y

CHECK TYPE OF MARKER

HOLE BOUNDARY?

N

CROSS BORDER OBJECT?

N

SET HOLE MARKER AND HOLE NUMBER

HOLE-OBJECT RELATION CODE NUMBER

HOLE MOMENTS AND INTEGER PARAMETERS

SUBTRACT HOLE MOMENTS FROM OBJECT'S OVER ALL MOMENTS

EXIT

ANOTHER HOLE IN A CROSS BORDER OBJECT

EXIT
PROCES-4
PARAMETERS

ENTER

ANY OBJECT ?

N

Y

LOOP FOR ALL ON-SCENE OBJECTS

PARAM2

OUTER BOUNDARY PARAMETERS

PARAM2

OBJECT'S OVER-ALL PARAMETERS

LOCATION OF STARTING-POINT OF OBJECT'S OUTER BOUNDARIES

END LOOP ?

N

Y

ANY HOLE ?

N

Y

LOOP FOR ALL ON-SCENE HOLES

PARAM2

HOLE PARAMETERS

STARTING-POINT LOCATION OF HOLE BOUNDARIES.

END LOOP ?

N

Y

DISPLAY BOUNDARIES
IDENTIFY OBJECT SELECTED BY NAME

PuMA-1

ENTER

INPUT OBJECT NAME

COMPARE THE NAME TO RECORDED NAMES

ANY MATCH?

Y

PROCES READ

PROCESS IMAGE OR INPUT ON-SCENE OBJECT PARAMETERS

LOOP FOR ALL RECORDED VIEWS WITH THE NAME

COMPARE ONE RECORDED VIEW TO ON-SCENE OBJECTS

ANY MATCH?

Y

END LOOP

N

ON MATCH

N

EXIT

UNKNOWN OBJECT NAME
IDENTIFY OBJECT SELECTED FROM LIST

ENTER

LIST NAMES OF ALL RECORDED OBJECTS

SELECT ONE OBJECT

ANY SELECTION?

? Y

FIND ALL RECORDED VIEWS WITH THE SAME NAME

PROCESS IMAGE OR INPUT ONE-SCENE OBJECT PARAMETERS

LOOP FOR ALL VIEWS WITH THE SAME NAME.

COMPARE ONE RECORDED VIEW TO ONE-SCENE OBJECTS

ANY MATCH?

? Y

END LOOP?

N
RIGHT

ENTER

CHECK PIXEL(R-1,C)

OBJECT PIXEL?

Y

MARK PIXEL(R,C)

MOVE UP TO NEXT PIXEL

REACH TOP BORDER?

Y

N

RETURN
THRES

1. Enter
2. Set gray-level summation and total number of picked pixels to zero
3. Loop for entire image, advancing by sample step-length
4. Sum gray-level value and number of picked pixels
5. END LOOP
6. Determine gray-level average
7. Determine threshold
8. Input new shift value
9. Display binary image
10. GOOD IMAGE
   10.1 Y: RETURN
   10.2 N: Go back to step 1
CHECK PIXEL(R,C-1)

OBJECT PIXEL?

MARK PIXEL(R,C)

MOVE TO NEXT PIXEL TO THE LEFT

REACH LEFT BORDER?

RETURN

ENTER
TP

SYSTEM TRAINING &
OBJECT POSITIONING

BEGIN
OPEN FILE FOR
OPERATION RECORD
DREAD
INPUT DATA BASE
DISPLAY MENU
SELECT OPTION

TRAIN
SYSTEM TRAINING

Y

CHOOSE
3

N

PUMA
OBJECT POSITIONING

Y

CHOOSE
4

N

DBASE
MANAGE DATA BASE

Y

CHOOSE
7

N

SAVE DATA
BASE

Y

CHOOSE
8

N

CHOSE
9

N

STOP
UPDATE

AVERAGE OUTER-BOUNDARY INVARIANTS OF THE RECORDED AND THE ON-SCENE OBJECTS

ANY HOLE?

LOOP FOR ALL ON-SCENE HOLES

CHECK HOLE-OBJECT RELATION

END LOOP?

FIND HOLE?

LOOP FOR ALL RECORDED HOLES

CHECK HOLE-OBJECT RELATION

END LOOP?

FIND HOLE?

AVERAGE HOLE-BOUNDARY INVARIANTS OF THE RECORDED AND THE ON-SCENE HOLES

RETURN
Appendix D.

Table of Variables
A

A integer for initializing the frame grabber.
AL unit factor for closed-path integration.
ARP(5,10) over-all real-number parameters of on-scene objects, with the deduction of its holes.
   (#,1): area;
   (#,2): perimeter;
   (#,3): principle value 1;
   (#,4): principle value 2;
   (#,8): centroid position in ROW direction in image-frame coordinates;
   (#,9): centroid position in COLUMN direction in image-frame coordinates;
   (#,10): orientation angle about Z-axis of image-frame coordinates.
AV average gray level of an image.

B

B integer for initializing the frame grabber.
BASE object orientation angle about X- or Y- axis of the robot-base coordinates.
BORDER(4) border limits of operation frame for boundary searching and tracking.
   (1): top border = IMIN;
   (2): left border = JMIN;
   (3): bottom border = IMAX;
   (4): right border = JMAX.
BP(3) boundary integer parameters of objects or holes.
   (1): starting-point position related to object centroid in ROW direction;
   (2): starting-point position related to object centroid in COLUMN direction;
   (3): length of the chain code.
BTC(4,4) homogeneous transformation matrix expressing camera position in robot-base coordinates.
BTO(4,4) homogeneous transformation matrix expressing object position in robot-base coordinates.

C

C0 column number of the starting point of a boundary.
C column number of a boundary pixel.
CC column increment from last-tracked boundary pixel, for determining check-point location.
CH(5,300) chain codes of on-scene object, for outer boundaries or for hole boundaries during
boundary tracking.

**CH(5,300)** chain codes of on-scene outer boundaries or hole boundaries.

**CHAIN(300)** a single chain code of the boundary of an object or a hole.

**CHLEN** length of boundary chain code.

**CHOICE1** same as **CHOICE**.

**CHOICE** selected option from available operation options.

**CO** column number of object centroid.

**CONT** control code for operations.

**CR** row increment from last-tracked boundary pixel, for determining check-point location during boundary tracking.

**CTO(4,4)** homogeneous transformation matrix expressing the object position in camera coordinates.

**D**

**D** direction code for boundary tracking.

**DC** column increment from last-tracked boundary pixel, for determining check-point locations during boundary tracking.

**DIST** distance between the object centroid to the center of the image frame.

**DR** row increment from last-tracked boundary pixel, for determining check-point locations during boundary tracking.

**DX** real-number form of **DR**.

**DY** real-number form of **DC**.

**E**

**E** integer for initializing the frame grabber.

**ERASER** an integer to replace the markers on a tracked boundary.

**ERROR** error control code for boundary tracking.

**H**

**HCH(5,300)** chain codes of the hole boundaries of the on-scene objects.

**HIP(5,4)** integer number parameters of the hole boundaries of the on-scene objects.

- (**#,1**): starting-point position related to object centroid in ROW direction;
- (**#,2**): starting-point position related to object centroid in COLUMN direction;
- (**#,3**): length of the boundary chain code;
- (**#,4**): serial number of the host object.
HLIM1(4) & HLIM2(4) LOWER limits and UPPER limits of HOLE's real-number invariants for the comparison during object recognition.
(1): area;
(2): perimeter;
(3): principle value 1;
(4): principle value 2.

HM permanent marker for hole boundaries.

HNUM total number of holes in on-scene objects.

HRP(5,10) real-number parameters of the region enclosed by the hole boundaries of on-scene objects.
(#,1): area of the region;
(#,2): perimeter of the region;
(#,3): principle value 1 of the region;
(#,4): principle value 2 of the region;
(#,8): centroid position of the region in ROW direction in image-frame coordinates;
(#,9): centroid position of the region in COLUMN direction in image-frame coordinates;
(#,10): orientation angle about Z-axes of image-frame coordinates.

I

I0,I1,I2 control integers in ROW direction for printing out a portion of the image.
I0: center of the portion;
I1: top edge of the portion;
I2: bottom edge of the portion.

IMAX row number of the bottom border of the operation frame.

IMIN row number of the top border of the operation frame.

IP(5,4) integer-number parameters of the boundaries of on-scene objects or holes.
(#,1): starting-point position related to object centroid in ROW direction;
(#,2): starting-point position related to object centroid in COLUMN direction;
(#,3): length of the boundary chain code;
(#,4): for OBJECT: number of holes;
for HOLE: serial number of its host object.
JO, J1, J2 control integers in COLUMN direction for printing out a portion of the image.
  J0: center of the portion;
  J1: left edge of the portion;
  J2: right edge of the portion.
J column number of a pixel.
JA, JC two byte integers for DO LOOPs, >127.
JMAX column number of the right border of the operation frame.
JMIN column number of the left border of the operation frame.

K

KA, KB four byte integers, >32767.
  KA: the summation of gray values of picked pixels for gray-level average;
  KB: number of picked pixels.
K, K1, K2 two byte integers, >127.

L

L, L1, L2, L3, L4, L5 one byte integers, <127.
LS locale scale factor with respect to object centroid location, (see SCALE).

M

MARGIN allowed tolerance in object recognition.
  (1): for area;
  (2): for perimeter;
  (3): for principal value 1;
  (4): for principal value 2.
MARK marker to be put on a tracked boundary.
MATCH serial number of the matched object among the recorded objects or the on-scene objects.
MATRIX1(4,4) & MATRIX2(4,4) & MATRIX3(4,4) matrixes in the operations of matrix multiplication and matrix inversion.
MT(6) calculated moments of an object or hole by the closed-path integration.

N

NAME user-input object name.
NODD total number of odd-number tracking direction in a boundary chain code.
NUM serial number of an on-scene object.
NUMMAT  total number of identical object names in
data base, meaning more than one view have
been recorded for the object with the name.

NMATCH(6)  serial numbers of the matched name(s)
of the requested object.

OCH(5,300)  chain codes of the outer boundaries of
on-scene objects.

OD  code of odd-number tracking direction,
used in determining perimeter.
   OD=0 for even number direction (0,2,4,6);
   OD=1 for odd number direction (1,3,5,7).

OIP(5,4)  integer-number parameters of outer boundaries
of on-scene objects.
   (#,1): starting-point position related to
      object centroid in ROW direction;
   (#,2): starting-point position related to
      object centroid in COLUMN direction;
   (#,3): length of the chain code;
   (#,4): number of holes in object.

OLIM1(4) & OLIM2(4)  LOWER limits and UPPER limits of OBJECT’S
real-number invariants for the comparison
during object recognition.
   (1): area;
   (2): perimeter;
   (3): principle value 1;
   (4): principle value 2.

OM  permanent marker for outer boundaries.

ONUM  total number of on-scene objects.

ORIENT  object orientation angle about Z-axis of the
image frame or the robot-base coordinates.

ORP(5,10)  real-number parameters of the region enclosed
by the outer boundaries of on-scene objects.
   (#,1): area;
   (#,2): perimeter;
   (#,3): principle value 1;
   (#,4): principle value 2;
   (#,8): centroid position in ROW direction in
      image-frame coordinates;
   (#,9): centroid position in COLUMN direction
      in image-frame coordinates;
   (#,10): orientation angle about Z-axis of
      image-frame coordinates.

OTC(4,4)  homogeneous transformation matrix expressing
the camera position in object coordinates.
   It is the inverse of CTO.
P

P(256,320) pixels of the image.
P(R,C) a pixel at row R and column C in the image.
P(R+DR,C+DC) the pixel in front of P(R,C).
P(R+CR,C+CC) the pixel in front of and to the right of P(R+DR,C+DC).
P(R+CR+DR, C+CC+DC) the pixel in front of P(R+CR,C+CC).
P(R+DR+DR, C+DC+DC) the pixel in front of P(R+DR,C+DC).
P(R-DC,C+DR) the pixel to the left of P(R,C).

R

RO row number of the starting point of a boundary.
R row number of a boundary pixel.
RHCH(10,300) chain codes of recorded hole boundaries.
RHIP(10,4) integer parameters of the recorded hole boundaries.
  (#,1): starting-point position related to the host-object centroid in ROW direction;
  (#,2): starting-point position related to the host-object centroid in COLUMN direction;
  (#,3): length of the chain code;
  (#,4): serial number of recorded host-object.
RNUM total number of recorded holes.
RHRP(10,4) real-number invariants of recorded holes.
  (#,1): area;
  (#,2): perimeter;
  (#,3): principle value 1;
  (#,4): principle value 2.
RNAMES(10) recorded object names.
RNUM serial number of a recorded object.
RO row number of the object centroid.
ROCH(10,300) chain codes of the outer boundaries of the recorded objects.
ROIP(10,4) integer-number parameters of the outer boundaries of the recorded objects.
  (#,1): starting-point position related to the object centroid in ROW direction;
  (#,2): starting-point position related to the object centroid in COLUMN direction;
  (#,3): length of the chain code;
  (#,4): number of holes in the object.
RONUM total number of recorded objects.
RORP(10,4) real-number invariants of recorded objects.
  (#,1): area;
  (#,2): perimeter;
RP(5,10) real-number parameters of object or hole boundaries.
(#,1):area;
(#,2):perimeter;
(#,3):principle value 1;
(#,4):principle value 2;
(#,8):centroid position in ROW direction in image-frame coordinates;
(#,9):centroid position in COLUMN direction in image-frame coordinates;
(#,10):orientation angle about Z-axis of image-frame coordinates.

S

SA spatial angle formed by the object centroid, the camera, and the aiming point of camera.

SCALE scale for converting the real-number invariants of objects and holes from the image-frame measure to the metric measure.

SH shift value from the average gray-level of an image to obtain the threshold.

SL step length of rough search for boundaries.

SL1 step length of sampling for gray-level average of the image.

STD area of the reference object used for camera calibration.

SUM a summation used during matrix multiplication.

T

T1,T2 intermediate values used for calculation.

TH threshold for distinguishing objects from their background.

TURNS number of turns required for one single tracking step.

U

U1,U2 intermediate values used for calculation.

X

X real-number form of R.

XBASE displacement of the object on X-axis of the robot-base coordinates.
Y

Y real-number form of C.
YBASE displacement of the object on Y-axis of the robot-base coordinates.
YN Yes/No control character.

Z

ZBASE displacement of the object on Z-axis of the robot-base coordinates.
Appendix E.

Program Listing
The contents of Appendix E are as follows:

<table>
<thead>
<tr>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure of Program EYE</td>
<td>E-2</td>
</tr>
<tr>
<td>Structure of Program IP</td>
<td>E-3</td>
</tr>
<tr>
<td>Structure of Program TP</td>
<td>E-4</td>
</tr>
<tr>
<td>Program Listing of IP</td>
<td>E-5</td>
</tr>
<tr>
<td>Program Listing of TP</td>
<td>E-56</td>
</tr>
<tr>
<td>Modification for EYE</td>
<td>E-95</td>
</tr>
</tbody>
</table>

Program EYE is the original design of the entire program of the robot vision system. Due to the insufficiency of computer memory (RAM), the original program is divided into two smaller programs, IP and TP. When a computer with enough RAM is available, the original program can be easily reconstructed with some minor modifications. These modifications are listed at the end of this appendix.
The numbers in parentheses are the line numbers reserved for operations of GOTOs and DO LOOPs in each subroutine.
PROGRAM IP
C
C MAIN PROGRAM FOR CAMERA CALIBRATION AND IMAGE PROCESSING.
C
INTEGER*1 ONUM,HNUM,OCH(9,300),HCH(9,300)
INTEGER*1 RONUM,RHNUM,ROCH(19,300),RHCH(19,300)
INTEGER*1 CHOICE,TH,SH,SL1,SL
INTEGER*2 IP(9,4),HIP(9,4),BORDER(4)
INTEGER*2 ROIP(19,4),RHIP(19,4)
REAL ORP(9,10),ARP(9,10),HRP(9,10)
REAL MARGIN(4),RORP(19,4),RHRP(19,4)
REAL BTC(4,4),SCALE,STD
CHARACTER*10 RNAMES(19)
C
C******************************************************C
C OPEN OPERATION TRACE FILE & LOAD DATA BASE C
C******************************************************C
OPEN(8,FILE='FUNG/TRACE1~,RECL=80)
WRITE(6,*) ' OPEN EYE.'
WRITE(8,*) ' OPEN EYE.'
CALL DREAD(RONUM,RHNUM,RNAMES,ROIP,RHIP,ROCH,RHCH,
$ RORP,RHRP,BTC,MARGIN,SCALE,STD,TH,SH,SL1,SL,BORDER)
C
C******************************************************************************C
C DISPLAY OPERATION MENU C
C******************************************************************************C
1 WRITE(6,*) ' WHAT DO YOU WISH TO DO NEXT?'
   WRITE(6,*) ' 1. CAMERA CALIBRATION;'
   WRITE(6,*) ' 2. REVISE OPERATION PARAMETERS;'
   WRITE(6,*) ' 5. PROCESS ON-SCENE IMAGE;'
   WRITE(6,*) ' 6. SAVE ON-SCENE OBJECTS;'
   WRITE(6,*) ' 8. SAVE DATA BASE;'
   WRITE(6,*) ' 9. EXIT TO OPERATING SYSTEM.'
   WRITE(6,*) ' ENTER YOUR CHOICE:'
WRITE(8,*) ' WHAT DO YOU WISH TO DO NEXT?'
WRITE(8,*) ' 1. CAMERA CALIBRATION;'
WRITE(8,*) ' 2. REVISE OPERATION PARAMETERS;'
WRITE(8,*) ' 5. PROCESS ON SCENE IMAGE;'
WRITE(8,*) ' 6. SAVE ON SCENE OBJECTS;'
WRITE(8,*) ' 8. SAVE DATA BASE;'
WRITE(8,*) ' 9. EXIT TO OPERATING SYSTEM.'
READ(S,*) CHOICE
WRITE(8,*) ' ENTER YOUR CHOICE:',CHOICE
C
C******************************************************************************C
C SELECT OPERATION OPTION C
C******************************************************************************C
IF(CHOICE.EQ.1) CALL CALI(BORDER,SL,TH,SH,SL1,
$ BTC,SCALE,STD)
IF(CHOICE.EQ.2) CALL CONTRO(BORDER,SL,SL1)
IF(CHOICE.EQ.5) THEN
    CALL PROCES(BORDER,SL,TH,ONUM,HNUM,OIP,HIP,
                 OCH,HCH,ORP,ARP,HRP,SCALE)
ENDIF
IF(CHOICE.EQ.6) CALL OSSAVE(ONUM,HNUM,OIP,HIP,
                             OCH,HCH,ORP,ARP,HRP)
IF(CHOICE.EQ.8) THEN
    CALL DSAVE(RONUM,RHNUM,RNAMES,ROIP,RHIP,ROCH,RHCH,
               RORP,RHRP,BTC,MARGIN,SCALE,STD,TH,SH,SL1,SL,BORDER,1)
ENDIF
IF(CHOICE.NE.9) GOTO 1
C
WRITE(6,*) 'CLOSE EYE.'
WRITE(8,*) 'CLOSE EYE.'
C
STOP
END
C
SUBROUTINE APERT(SL1, TH, SH)

C THIS SUBROUTINE CALIBRATES THE APERTURE OF THE CAMERA
C UNTIL A SATISFACTORY BINARY IMAGE IS ACHIEVED.

INTEGER*1 P(256, 320), TH, SH, SL1

WRITE(8, *) ' CAMERA APERTURE CALIBRATION:'
WRITE(6, *) ' CAMERA APERTURE CALIBRATION:'
WRITE(8, *) ' ADJUST THE CAMERA'S APERTURE THEN'
WRITE(6, *) ' ADJUST THEカメラ'S APERTURE THEN'
CALL GRAB(P)
CALL THRES(P, SL1, TH, SH)

WRITE(8, *) ' APERTURE CALIBRATION HAS BEEN', $
WRITE(6, *) ' APERTURE CALIBRATION HAS BEEN', $
RETURN
END
SUBROUTINE BINARY(P,TH)
C
C THIS SUBROUTINE DISPLAYS THE IMAGE IN BINARY FORM
C WITHOUT DISTURRING THE ORIGINAL GRAY LEVEL IMAGE.
C
INTEGER A,B,E
INTEGER*1 P(256,320),TH,L
INTEGER*2 I,J
C
C******************************************************
C
C SET UP FRAME GRABBER FOR POST-INCREMENT WRITE
C
C******************************************************
A=#4000H
B=#00H
DO 41 L=1,7
   E=A+L
   CALL OUTPUT(E,B)
41 CONTINUE
C
CALL OUTPUT(#4007H,#01H)
CALL OUTPUT(#4003H,#d1H)
DO 42 L=1,20
   CALL OUTPUT(#4001H,#02H)
42 CONTINUE
C
C************************************************************
C OUTPUT BINARY IMAGE TO FRAME GRABBER
C************************************************************
DO 43 I=1,256
   DO 43 J=1,320
      IF(P(I,J).GT.TH) THEN
         CALL OUTPUT(#4000H,#3fH)
      ELSE
         CALL OUTPUT(#4000H,#05H)
      ENDIF
43 CONTINUE
C
RETURN
END
C
E-8
SUBROUTINE BNDY(P,BP,CHAIN,RO,CO)
C
C THIS SUBROUTINE USES A CHAIN CODE TO CONSTRUCT
C A BOUNDARY OUTLINE OF AN OBJECT OR A HOLE.
C
INTEGER*1 P(256,320),CHAIN(300),D,DR,DC,OD
INTEGER*2 R,C,RO,CO,BP(3),CHLEN,K2
C
R=RO+BP(1)
C=CO+BP(2)
CHLEN=BP(3)
C
C**********************************************************************C
C CONSTRUCTING BOUNDARY
C**********************************************************************C
P(R,C)=1
DO 820 K2=1,CHLEN
   D=CHAIN(K2)
   CALL DIREC2(D,DR,DC,OD)
   R=R+DR
   C=C+DC
   P(R,C)=1
820 CONTINUE
C
RETURN
END
C
SUBROUTINE BNDYOS(P,NUM,HNUM,OIP,HIP,OCH,HCH,RO,CO)

C THIS SUBROUTINE CONSTRUCTS A BOUNDARY IMAGE OF
C AN ON-SCENE OBJECT, AS WELL AS ITS HOLE(S).

INTEGER*1 P(256,320),OCH(9,300),HCH(9,300)
INTEGER*1 CHAIN(300)
INTEGER*1 NUM,HNUM,L,L1
INTEGER*2 RO,CO,OIP(9,4),HIP(9,4),BP(3)
INTEGER*2 CHLEN,K2

WRITE(6,*) ' CONSTRUCTING THE BOUNDARY OF',
$ ' AN ON-SCENE OBJECT.'
WRITE(8,*) ' CONSTRUCTING THE BOUNDARY OF',
$ ' AN ON-SCENE OBJECT.'

C CONSTRUCTING OBJECT BOUNDARY
C
BP(1)=OIP(NUM,1)
BP(2)=OIP(NUM,2)
BP(3)=OIP(NUM,3)
CHLEN=BP(3)

DO 830 K2=1,CHLEN
   CHAIN(K2)=OCH(NUM,K2)
830 CONTINUE
CALL BNDY(P,BP,CHAIN,RO,CO)

C CONSTRUCTING HOLE BOUNDARY IF THERE IS ANY
C
IF(OIP(NUM,4).GT.0) THEN
   L1=1
   DO 831 L=1,HNUM
      IF(HIP(L,4).EQ.NUM) THEN
         BP(1)=HIP(L,1)
         BP(2)=HIP(L,2)
         BP(3)=HIP(L,3)
         CHLEN=BP(3)
         DO 833 K2=1,CHLEN
            CHAIN(K2)=HCH(L,K2)
833    CONTINUE
         CALL BNDY(P,BP,CHAIN,RO,CO)
         L1=L1+1
      ENDIF
831    CONTINUE
   IF(L1.GT.OIP(NUM,4)) GOTO 839
ENDIF
839 RETURN
END
SUBROUTINE BOTTOM(P,R,C,JMAX,TH)

C THIS SUBROUTINE CONTINUES THE BOUNDARY TRACKING WHILE THE
C TRACKING REACH THE BOTTOM BORDER OF THE OPERATION.
C
C INTEGER*1 P(256,320), TH
INTEGER*2 JMAX, R, C
C
826 IF(P(R,C+1).LE.TH) THEN ; AN OBJECT PIXEL?
P(R,C)=-1
C=C+1
IF(C.GE.JMAX) GOTO 830 ; REACH RIGHT BORDER?
GOTO 826
ENDIF
C
830 RETURN
END
C
SUBROUTINE BTRACE(P,BORDER,CHAIN,
$       CHLEN,RO,CO,TH,ERROR)
C
THIS SUBROUTINE TRACKS THE OUTER BOUNDARY OF AN OBJECT.
C
INTEGER*1 P(256,320),CHAIN(300),TH,D,DR,DC,CR,CC
INTEGER*1 ERROR,TURNS
INTEGER*2 R,C,RO,CO,CHLEN
INTEGER*2 BORDER(4),IMIN,JMIN,IMAX,JMAX
C
C************************************************************************
C SET OPERATION BORDERS
C************************************************************************
IMIN=80RDER(1)
JMIN=BORDER(2)
IMAX=80RDER(3)
JMAX=BORDER(4)
C
C************************************************************************
C TRACKING STARTS AT LEFT BORDER?
C************************************************************************
IF(CO.EQ.JMIN) THEN
   R=RO
   C=CO
   CHLEN=0
   GOTO 626
ENDIF
C
C************************************************************************
C INITIAL TRACKING DIRECTION
C************************************************************************
D=1
610 CALL DIRECT(D,DR,DC,CR,CC)
   IF(P(RO+DR,CO+DC).GT.TH) THEN               ;C=BP?
      D=D+1
      IF(D.GT.7) THEN
         ERROR=2
         GOTO 649
      ENDIF
      GOTO 610
   ELSE
   ENDIF
   GOTO 610
ELSE
C
C************************************************************************
C TRACKED BOUNDARY?
C************************************************************************
IF(P(RO+CR,CO+CC).GT.63.OR.P(RO+DR,CO+DC).LT.0) THEN
   ERROR=2
   GOTO 649
ENDIF
ENDIF
C
E-13

C*******************************************************************************C
C START TRACKING
C*******************************************************************************C
CHAIN(1)=D
P(RO,CO)=-1
R=RO+DR
C=CO+DC
ERROR=0
TURNS=0
CHLEN=2
C
C*******************************************************************************C
C TRACKING TURNS RIGHT
C*******************************************************************************C
611 IF(P(R+DR,C+DC).LE.TH) THEN ;C=OP?
612 IF(P(R+CR,C+CC).LE.TH) THEN ;A=OP?
   IF(P(R+CR+DR,C+CC+DC).GT.TH) THEN ;B=BP?
      P(R+CR,C+CC)=TH+1 ;BUMP NOISE
      GOTO 615
   ENDIF
   D=D-1 ;RIGHT TURN
   TURNS=TURNS+1
   IF(TURNS.GT.8) THEN ;TOO MANY TURNS?
      ERROR=1
      GOTO 649
   ENDIF
   IF(D.EQ.-1) D=7
   CALL DIRECT(D,DR,DC,CR,CC)
   GOTO 612
   ENDIF
   GOTO 615
ENDIF
C
IF(P(R+DR+DR,C+DC+DC).LE.TH) THEN ;D=OP?
   P(R+DR,C+DC)=TH-1 ;HOLE NOISE
   GOTO 615
ENDIF
C
IF(P(R-DC,C+DR).GT.TH) THEN ;F=BP?
   P(R,C)=TH+1 ;BUMP NOISE
   R=R-DR
   C=C-DC
   CHLEN=CHLEN-1
   GOTO 611
ENDIF
C
C*******************************************************************************C
C TRACKING TURNS LEFT
C*******************************************************************************C
D=D+1 ;LEFT TURN
TURNS=TURNS+1
C
C*******************************************************************************C
IF(TURNS.GT.8) THEN ; TOO MANY TURNS?
   ERROR=1
   GOTO 649
ENDIF
IF(D.EQ.8) D=0
CALL DIRECT(D,DR,DC,CR,CC)
GOTO 611

C******************************************************
C RECORD ONE DIRECTION CODE
C******************************************************
615 CHAIN(CHLEN)=D
   P(R,C)=-1 ; MARK BOUNDARY
   R=R+DR
   C=C+DC
   TURNS=0

C******************************************************
C BORDER OPERATION IF NECESSARY
C******************************************************
626 IF(C.LE.JMIN) THEN ; LEFT BORDER
   ERROR=3
   CALL LEFT(P,R,C,IMAX,TH)
   CHLEN=CHLEN+1
   CHAIN(CHLEN)=2
   D=3
   CALL DIRECT(D,DR,DC,CR,CC)
ENDIF

C  IF(R.GE.IMAX) THEN ; BOTTOM BORDER
   ERROR=3
   CALL BOTTOM(P,R,C,JMAX,TH)
   CHLEN=CHLEN+1
   CHAIN(CHLEN)=4
   D=5
   CALL DIRECT(D,DR,DC,CR,CC)
ENDIF

C  IF(C.GE.JMAX) THEN ; RIGHT BORDER
   ERROR=3
   CALL RIGHT(P,R,C,IMIN,TH)
   CHLEN=CHLEN+1
   CHAIN(CHLEN)=6
   D=7
   CALL DIRECT(D,DR,DC,CR,CC)
ENDIF

C  IF(R.LE.IMIT) THEN ; TOP BORDER
   ERROR=3
   CALL TOP(P,R,C,JMIN,TH)
   CHLEN=CHLEN+1
CHAIN(CHLEN)=0
D=1
CALL DIRECT(D,DR,DC,CR,CC)
ENDIF

C******************************************************
C TRACKING TERMINATION
C******************************************************

IF(P(R,C).NE.-1) THEN
  CHLEN=CHLEN+1
  IF(CHLEN.GT.300) THEN ;CHAIN-CODE TOO LONG?
    ERROR=1
  ELSE
    GOTO 611
  ENDIF
ENDIF

C
649 RETURN
END

C
E-16

SUBROUTINE CALI(BORDER, SL, TH, SH, SL1, BTC, SCALE, STD)
C
C THIS SUBROUTINE CALIBRATES THE CAMERA WITH USER'S
C ASSISTANCE, FOR CAMERA APERTURE AND/OR POSITION.
C
INTEGER*1 CHOICE, TH, SH, SL1, SL
INTEGER*2 BORDER(4)
REAL BTC(4, 4), SCALE, STD
C
C******************************************************************************
C DISPLAY MENU
C******************************************************************************
WRITE(8, *) ' CAMERA CALIBRATION:
WRITE(8, *)' 1. APERTURE ONLY, AND NEW SCALE;
WRITE(8, *)' 2. POSITION ONLY, AND NEW SCALE;
WRITE(8, *)' 3. BOTH APERTURE & POSITION;
WRITE(8, *)' 9. EXIT.'

11 WRITE(6, *) ' CAMERA CALIBRATION:
WRITE(6, *)' 1. APERTURE ONLY, AND NEW SCALE;
WRITE(6, *)' 2. POSITION ONLY, AND NEW SCALE;
WRITE(6, *)' 3. BOTH APERTURE & POSITION;
WRITE(6, *)' 9. EXIT.'
WRITE(6, *)' ENTER YOU CHOICE:'
READ(5, *) CHOICE
WRITE(8, *)' ENTER YOU CHOICE:', CHOICE

C******************************************************************************
C SELECT OPTION
C******************************************************************************
IF (CHOICE.EQ.1) THEN
   CALL APERT(SL1, TH, SH)
   WRITE(6, '*') ' NOW FOR DETERMINING THE NEW SCALE.'
   WRITE(8, '*')' NOW FOR DETERMINING THE NEW SCALE.'
   CALL POSIT(BORDER, SL, TH, BTC, SCALE, STD, 0)
ENDIF
IF (CHOICE.EQ.2) THEN
   CALL POSIT(BORDER, SL, TH, BTC, SCALE, STD, 1)
ENDIF
IF (CHOICE.EQ.3) THEN
   CALL APERT(SL1, TH, SH)
   CALL POSIT(BORDER, SL, TH, BTC, SCALE, STD, 1)
ENDIF
IF (CHOICE.EQ.9) GOTO 19
GOTO 11
C
19 RETURN
END
SUBROUTINE CLEAR(P)
C
C THIS SUBROUTINE CLEARS THE IMAGE MAP.
C
INTEGER*1 P(256,320)
INTEGER*2 I,J
C
C 63: COLOR OF WHITE.
C
WRITE(6,*) ' CLEARING PICTURE MAP. PLEASE WAIT.'
DO 90 I=1,256
  DO 90 J=1,320
    P(I,J)=63
  90 CONTINUE
C
RETURN
END
SUBROUTINE CONTRO(BORDER,SL,SL1)

C
C THIS SUBROUTINE SETS NEW OPERATION CONTROL PARAMETERS.
C
INTEGER*1 SL,SL1,CHOICE
INTEGER*2 BORDER(4)
CHARACTER*1 YN

C******************************************************
C DISPLAY MENU
C******************************************************
WRITE(B,*) ' REVIEW OPERATION CONTROL PARAMETERS:
WRITE(8,*) ' 1: OPERATION BORDERS;
WRITE(8,*) ' 2: SAMPLE STEP LENGTH FOR BOUNDARY SEARCH;
WRITE(8,*) ' 3: SAMPLE STEP LENGTH FOR GRAY LEVEL AVERAGE;
WRITE(8,*) ' 9: EXIT.'
111 WRITE(6,*) ' REVIEW OPERATION CONTROL PARAMETERS:
WRITE(6,*) ' 1: OPERATION BORDERS;
WRITE(6,*) ' 2: SAMPLE STEP LENGTH FOR BOUNDARY SEARCH;
WRITE(6,*) ' 3: SAMPLE STEP LENGTH FOR GRAY LEVEL AVERAGE;
WRITE(6,*) ' 9: EXIT.'
WRITE(6,*) ' ENTER CHOICE:'
READ(S,*) CHOICE
WRITE(8,*) ' ENTER CHOICE:', CHOICE
C******************************************************
C SET NEW OPERATION BORDERS
C******************************************************
IF(CHOICE.EQ.1) THEN
  WRITE(8,*) ' CURRENT OPERATION BORDER:'
  WRITE(8,1111) BORDER(1),BORDER(2)
  WRITE(8,1112) BORDER(3),BORDER(4)
  WRITE(8,1113) ' ANY CHANGE? (Y/N)'
112 WRITE(6,*) ' CURRENT OPERATION BORDER:'
  WRITE(6,1111) BORDER(1),BORDER(2)
  WRITE(6,1112) BORDER(3),BORDER(4)
  WRITE(6,1113) ' ANY CHANGE? (Y/N)'
READ(5,1110) YN
WRITE(8,1110) YN
IF(YN.EQ.'Y') THEN
  WRITE(6,*) ' ENTER UP-LEFT CORNER: (ROW,COLUMN)'
  READ(5,*) BORDER(1),BORDER(2)
  WRITE(8,*) ' ENTER UP-LEFT CORNER: (ROW,COLUMN)',
  BORDER(1),BORDER(2)
  WRITE(6,*) ' ENTER BOTTOM-RIGHT CORNER:
  (ROW,COLUMN)'
  READ(5,*) BORDER(3),BORDER(4)
  WRITE(8,*) ' ENTER BOTTOM-RIGHT CORNER:',

C NEW STEP-LENGTH FOR BOUNDARY SEARCH
C
IF(CHOICE.EQ.2) THEN
  WRITE(8,*)' CURRENT BOUNDARY SEARCH STEP LENGTH:',SL
  WRITE(8,*)' ANY CHANGE? (Y/N)'
  WRITE(6,*)' CURRENT BOUNDARY SEARCH STEP LENGTH:',SL
  WRITE(6,*)' ANY CHANGE? (Y/N)'
  READ(5,1110) YN
  WRITE(8,1110) YN
  IF(YN.EQ.'Y') THEN
    WRITE(8,*)' ENTER NEW STEP LENGTH:'
    WRITE(6,*)' ENTER NEW STEP LENGTH:'
    READ(5,*) SL
    WRITE(8,*)' NEW STEP LENGTH:',SL
  GOTO 113
ENDIF
ENDIF
C NEW STEP-LENGTH FOR GRAY-LEVEL SAMPLING FOR AVERAGE
C
IF(CHOICE.EQ.3) THEN
  WRITE(8,*)' CURRENT SAMPLE STEP LENGTH FOR', $  
  ' AVERAGING GRAY LEVEL:',SL1
  WRITE(8,*)' ANY CHANGE? (Y/N)'
  WRITE(6,*)' CURRENT SAMPLE STEP LENGTH FOR', $  
  ' AVERAGING GRAY LEVEL:',SL1
  WRITE(6,*)' ANY CHANGE? (Y/N)'
  READ(5,1110) YN
  WRITE(8,1110) YN
  IF(YN.EQ.'Y') THEN
    WRITE(8,*)' ENTER NEW STEP LENGTH:'
    WRITE(6,*)' ENTER NEW STEP LENGTH:'
    READ(5,*) SL1
    WRITE(8,*)' NEW STEP LENGTH:',SL1
  GOTO 114
ENDIF
ENDIF
C
IF(CHOICE.NE.9) GOTO 111
C
1110 FORMAT(A1)
1111 FORMAT(’ UP-LEFT CORNER: (’,I3,’,’I3,’))’
1112 FORMAT(’BOTTOM-RIGHT CORNER: (’,I3,’,’I3,’))’
C
RETURN
END
SUBROUTINE DIREC2(D,DR,DC,OD)
C
C THIS SUBROUTINE GIVES ROW AND COLUMN INCREMENT CONTROLS
C FOR BOUNDARY CONSTRUCTION AND MOMENT CALCULATION.
C
INTEGER*1 D,DR,DC,OD
C
IF(D.EQ.0) THEN
   DR=0
   DC=-1
   OD=0
   GOTO 779
ENDIF
C
IF(D.EQ.1) THEN
   DR=1
   DC=-1
   OD=1
   GOTO 779
ENDIF
C
IF(D.EQ.2) THEN
   DR=1
   DC=0
   OD=0
   GOTO 779
ENDIF
C
IF(D.EQ.3) THEN
   DR=1
   DC=1
   OD=1
   GOTO 779
ENDIF
C
IF(D.EQ.4) THEN
   DR=0
   DC=1
   OD=0
   GOTO 779
ENDIF
C
IF(D.EQ.5) THEN
   DR=-1
   DC=1
   OD=1
   GOTO 779
ENDIF
C
IF(D.EQ.6) THEN
   DR=-1
   DC=1
   OD=1
   GOTO 779
ENDIF
C
END
DC=0
OD=0
GOTO 779
ENDIF
C

IF(D.EQ.7) THEN
  DR=-1
  DC=-1
  OD=1
ENDIF
C

779 RETURN
END
C
SUBROUTINE DIRECT(D,DR,DC,CR,CC)
C THIS SUBROUTINE GIVES ROW AND COLUMN INCREMENT CONTROLS
C OF CHECK POINTS FOR BOUNDARY TRACKING.
C
INTEGER*1 D,DR,DC,CR,CC
C
IF(D.EQ.0) THEN
  DR=0
  DC=-1
  CR=-1
  CC=-1
  GOTO 759
ENDIF
C
IF(D.EQ.1) THEN
  DR=1
  DC=-1
  CR=0
  CC=-1
  GOTO 759
ENDIF
C
IF(D.EQ.2) THEN
  DR=1
  DC=0
  CR=1
  CC=-1
  GOTO 759
ENDIF
C
IF(D.EQ.3) THEN
  DR=1
  DC=1
  CR=1
  CC=0
  GOTO 759
ENDIF
C
IF(D.EQ.4) THEN
  DR=0
  DC=1
  CR=1
  CC=1
  GOTO 759
ENDIF
C
IF(D.EQ.5) THEN
  DR=-1
  DC=1
  CR=0
  GOTO 759
CC=1
GOTO 759
ENDIF
C

IF(D.EQ.6) THEN
  DR=-1
  DC=0
  CR=-1
  CC=1
  GOTO 759
ENDIF
C

IF(D.EQ.7) THEN
  DR=-1
  DC=-1
  CR=-1
  CC=0
ENDIF
C

759 RETURN
END
C
SUBROUTINE DISP(P)
C
C THIS SUBROUTINE DISPLAYS AN IMAGE ON THE MONITOR FOR
C EITHER A GRAY LEVEL ONE, A BINARY ONE OR BOUNDARIES.
C
INTEGER A,B,E
INTEGER*1 P(256,320),L
INTEGER*2 I,J
C
WRITE(6,*) 'DISPLAYING.'
C******************************************************
C SET UP FRAME GRABBER FOR DISPLAYING
C******************************************************
A=#4000H
B=#00H
DO 41 L=1,7
   E=A+L
   CALL OUTPUT(E,B)
41 CONTINUE
C
CALL OUTPUT(#4007H,#01H)
CALL OUTPUT(#4003H,#01H)
do 42 L=1,20
42 CONTINUE
CALL OUTPUT(#4001H,#02H)
C
C******************************************************
C OUTPUT IMAGE TO FRAME GRABBER
C******************************************************
DO 43 I=1,256
   DO 43 J=1,320
      CALL OUTPUT(#4000H,P(I,J))
43 CONTINUE
C
RETURN
END
C
SUBROUTINE DREAD(RONUM,RHNUM,RNAMES,ROIP,RHIP,RCH, $  RHCH,RORP,RHRP,BTC,MARGIN,SCALE,STD, $  TH,SH,SL1,SL,BORDER)

C THIS SUBROUTINE READS DATA BASE.

C INTEGER*1 TH,SH,SL1,SL,L,L1
INTEGER*1 RONUM,RHNUM,ROCH(19,300),RHCH(19,300)
INTEGER*2 ROIP(19,4),RHIP(19,4),BORDER(4),CHLEN,K1
REAL RORP(19,4),RHRP(19,4),BTC(4,4)
REAL MARGIN(4),SCALE,STD
CHARACTER*10 RNAMES(19)

WRITE(6,*) ' READING DATA BASE.'
WRITE(*,*) ' READING DATA BASE.'

OPEN(2,FILE='FUNG/PARAMS',RECL=80)

C**********************************************************************C
C READ NUMBERS OF RECORDED OBJECTS AND HOLES C
C**********************************************************************C
READ(2,3201) RONUM,RHNUM

C**********************************************************************C
C READ OBJECT NAMES AND OBJECT PARAMETERS C
C**********************************************************************C
IF(RONUM.GT.0) THEN
  DO 320 L=1,RONUM
    READ(2,3200) RNAMES(L)
    READ(2,3201) (ROIP(L,L1),L1=1,4)
    CHLEN=ROIP(L,3)
    READ(2,3203) (ROCH(L,K1),K1=1,CHLEN)
    READ(2,3202) (RORP(L,L1),L1=1,4)
  320 CONTINUE
ENDIF

C**********************************************************************C
C READ HOLE PARAMETERS C
C**********************************************************************C
IF(RHNUM.GT.0) THEN
  DO 321 L=1,RHNUM
    READ(2,3201) (RHIP(L,L1),L1=1,4)
    CHLEN=RHIP(L,3)
    READ(2,3203) (RHCH(L,K1),K1=1,CHLEN)
    READ(2,3202) (RHRP(L,L1),L1=1,4)
  321 CONTINUE
ENDIF

C**********************************************************************C
C READ OPERATION CONTROL PARAMETERS C
C**********************************************************************C
DO 322 L=1,4
   READ(2,3202) (BTC(L,L1),L1=1,4)
322 CONTINUE
   READ(2,3202) SCALE,STD
   READ(2,3202) (MARGIN(L),L=1,4)
   READ(2,3201) TH,SH,SL1,SL
   READ(2,3201) (BORDER(L),L=1,4)
C
   CLOSE(2)
C
   WRITE(6,*),' DATA BASE IS LOADED.'
   WRITE(8,*),' DATA BASE IS LOADED.'
C
3200 FORMAT(A10)
3201 FORMAT(4I10)
3202 FORMAT(4F15.5)
3203 FORMAT(60I1)
C
C******************************************************************************C
C BACK UP DATA FILE
C******************************************************************************C
   WRITE(6,*),' MAKING A BACKUP.'
   WRITE(8,*),' MAKING A BACKUP.'
   CALL DSAVE(RONUM,RHNUM,RNAMES,ROIP,RHIP,ROCH,
   $   RHCH,RORP,RHRP,BTC,MARGIN,SCALE,STD,
   $   TH,SH,SL1,SL,BORDER,0)
C
   RETURN
END
SUBROUTINE DSAVE(RONUM,RHNUM,RNAMES,ROIP,RHIP,ROCH,
$\text{RHCH, RORP, RHRP, BTC, MARGIN, SCALE, STD,}
$\text{TH, SH, SL1, SL, BORDER, CONT)}$
C
C THIS SUBROUTINE SAVES DATA BASE.
C
INTEGER*1 RONUM, RHNUM, TH, SH, SL1, SL, L, L1, CONT
INTEGER*1 ROCH(19, 300), RHCH(19, 300)
INTEGER*2 ROIP(19, 4), RHIP(19, 4), BORDER(4), CHLEN, K1
REAL RORP(19, 4), RHRP(19, 4), BTC(4, 4)
REAL MARGIN(4), SCALE, STD
CHARACTER*10 RNAMES(19)
C
WRITE(6,*) 'SAVING DATA BASE.'
WRITE(8,*) 'SAVING DATA BASE.'
C
C******************************************************C
C OPEN DATA FILE OR BACKUP FILE
C******************************************************C
IF(CONT.EQ.0) THEN
  OPEN(2, FILE='FUNG/PARAMSMOLD', RECL=80)
ELSE
  OPEN(2, FILE='FUNG/PARAMS', RECL=80)
ENDIF
C
C******************************************************C
C SAVE NUMBERS OF RECORDED OBJECTS AND HOLES
C******************************************************C
WRITE(2, 3501) RONUM, RHNUM
C
C******************************************************C
C SAVE OBJECT NAMES AND OBJECT PARAMETERS
C******************************************************C
IF(RONUM.GT.0) THEN
  DO 350 L = 1, RONUM
    WRITE(2, 3500) RNAMES(L)
    WRITE(2, 3501) (ROIP(L, L1), L1 = 1, 4)
    CHLEN = ROIP(L, 3)
    WRITE(2, 3503) (ROCH(L, K1), K1 = 1, CHLEN)
    WRITE(2, 3502) (RORP(L, L1), L1 = 1, 4)
  350 CONTINUE
ENDIF
C
C******************************************************C
C SAVE HOLE PARAMETERS
C******************************************************C
IF(RHNUM.GT.0) THEN
  DO 351 L = 1, RHNUM
    WRITE(2, 3501) (RHIP(L, L1), L1 = 1, 4)
    CHLEN = RHIP(L, 3)
    WRITE(2, 3503) (RHCH(L, K1), K1 = 1, CHLEN)
  351 CONTINUE
ENDIF
WRITE(2,3502) (RHRP(L,L1),L1=1,4)

351 CONTINUE
ENDIF

C
C******************************************************
C SAVE OPERATION CONTROL PARAMETERS
C******************************************************

DO 352 L=1,4
    WRITE(2,3502) (BTC(L,L1),L1=1,4)
352 CONTINUE

WRITE(2,3502) SCALE,STD
WRITE(2,3502) (MARGIN(L),L=1,4)
WRITE(2,3501) TH,SH,SL1,SL
WRITE(2,3501) (BORDER(L),L=1,4)

C
CLOSE(2)

C
3500 FORMAT(A10)
3501 FORMAT(4I10)
3502 FORMAT(4F15.5)
3503 FORMAT(60I1)

C
WRITE(6,*) ' DATA BASE IS SAVED.'
WRITE(8,*) ' DATA BASE IS SAVED.'

C
RETURN
END

C
SUBROUTINE ERASE(P,BORDER,CHAIN,CHLEN,RO,CO,MARK,TH)
C
C THIS SUBROUTINE ERASES MARKERS ON TRACKED PIXELS.
C
INTEGER*1 P(256,320),CHAIN(300)
INTEGER*1 MARK,TH,ERASER,D,DR,DC,OD
INTEGER*2 BORDER(4),R,C,RO,CO,CHLEN,K1
C
C******************************************************
C SET ERASER
C******************************************************
IF(MARK.LT.0) THEN
  ERASER=TH+1
ELSE
  ERASER=TH-1
ENDIF
C
C******************************************************
C REPLACE TRACKED PIXELS WITH ERASER
C******************************************************
R=RO
C=CO
P(R,C)=ERASER
DO 90 K1=1,CHLEN
  D=CHAIN(K1)
  CALL DIREC2(D,DR,DC,OD)
C
C******************************************************
C BORDER OPERATION
C******************************************************
IF(R.LE.BORDER(1).OR.R.GE.BORDER(3).OR.
  C.LE.BORDER(2).OR.C.GE.BORDER(4)) THEN
  IF(P(R,C).EQ.MARK) THEN
    R=R+DR
    C=C+DC
    GOTO 91
  ENDIF
ENDIF
R=R+DR
C=C+DC
P(R,C)=ERASER
90 CONTINUE
C
RETURN
END
C
SUBROUTINE GRA8(P)

C
C THIS SUBROUTINE GRABS AND INPUTS AN IMAGE
C FROM THE FRAME GRABBER.
C
INTEGER A,B,E
INTEGER*1 P(256,320),L
INTEGER*2 I,J
CHARACTER*1 YN

C******************************************************
C SET UP FRAME GRABBER FOR POST-INCREMENT READ
C******************************************************
A=#4000H
B=#00H
20 CONTINUE
DO 21 L=1,7
E=A+L
CALL OUTPUT(E,B)
21 CONTINUE

CALL OUTPUT(#4002H,#00H)
CALL OUTPUT(#4003H,#d0H)
DO 22 L=1,20
22 CONTINUE
CALL OUTPUT(#4001H,#04H)
DO 23 I=1,10000
23 CONTINUE

C******************************************************
C GRAB THE IMAGE
C******************************************************
WRITE(6,*) 'PRESS ENTER KEY TO GRAB THIS PICTURE.'
WRITE(8,*) 'PRESS ENTER KEY TO GRAB THIS PICTURE.'
READ(5,2000) YN
2000 FORMAT(A1)
CALL OUTPUT(#4001H,#00H)
DO 24 I=10000
24 CONTINUE

C******************************************************
C INPUT THE IMAGE
C******************************************************
WRITE(6,*) 'INPUTTING IMAGE. PLEASE WAIT.'
CALL OUTPUT(#4001H,#01H)
DO 25 I=1,256
DO 25 J=1,320
CALL INPUT(#4000H,P(I,J))
25 CONTINUE

C**********************************************************************
C DISPLAY THE GRABBED IMAGE
C**********************************************************************
WRITE(6,*) 'GRABBED IMAGE.'
CALL DISP(P)
WRITE(6,*) 'IF IMAGE INPUT OK? (Y/N)'
READ(5,2000) YN
IF(YN.NE.'Y') GOTO 20
RETURN
END
C
SUBROUTINE HTRACE(P,BORDER,CHAIN,
  $  CHLEN,RO,CO,TH,ERROR)
C
C THIS SUBROUTINE TRACKS THE HOLE BOUNDARY INSIDE AN OBJECT
C REGION.
C
INTEGER*1 P(256,320),CHAIN(300),TH,ERROR,TURNS
INTEGER*1 D,DR,DC,CR,CC
INTEGER*2 BORDER(4),R,C,RO,CO,CHLEN
C
C***************************************************************************************
C INITIAL TRACKING DIRECTION
C***************************************************************************************
D=1
660 CALL DIRECT(D,DR,DC,CR,CC)
  IF(P(RO+DR,CO+DC).LE.TH) THEN ;C=OP?
    D=D+1
    IF(D.GT.7) THEN ;ALL DIRECTIONS CHECKED?
      ERROR=2
    GOTO 699
  ENDIF
  GOTO 660
ELSE
C***************************************************************************************
C TRACKED BOUNDARY?
C***************************************************************************************
  IF(P(RO+CR,CO+CC).LT.0.OR.P(RO+CR,CO+CC).GT.63) THEN
      ERROR=2
  GOTO 699
ENDIF
C***************************************************************************************
C START TRACKING
C***************************************************************************************
  CHAIN(1)=D
  P(RO,CO)=64
  R=RO+DR
  C=CO+DC
  ERROR=0
  TURNS=0
  CHLEN=2
C
C***************************************************************************************
C TRACKING TURNS RIGHT
C***************************************************************************************
  IF(P(R+DR,C+DC).GT.TH) THEN ;C=BP?
    IF(P(R+CR,C+CC).GT.TH) THEN ;A=BP?
      IF(P(R+CR+DR,C+CC+DC).LE.TH) THEN ;B=OP?
        P(R+CR,C+CC)=TH-1 ;BUMP NOISE
      GOTO 665
ENDIF
D=D-1 ;LEFT TURN
TURNS=TURNS+1
IF(TURNS.GT.8) THEN ;TOO MANY TURNS?
      ERROR=1
      GOTO 699
ENDIF
IF(D.EQ.-1) D=7
CALL DIRECT(D,DR,DC,CR,CC)
GOTO 662
ENDIF
GOTO 665
ENDIF

C
IF(P(R+DR+DR,C+DC+DC).GT.TH) THEN ;D=BP?
P(R+DR,C+DC)=TH+1 ;HOLE NOISE
GOTO 665
ENDIF

C
IF(P(R-DC,C+DR).LE.TH) THEN ;F=OP?
P(R,C)=TH-1 ;BUMP NOISE
R=R-DR
C=C-DC
CHLEN=CHLEN-1
GOTO 661
ENDIF

C
C**************************************************************************C
C TRACKING TURNS LEFT
C**************************************************************************C
D=D+1 ;LEFT TURN
TURNS=TURNS+1
IF(TURNS.GT.8) THEN ;TOO MANY TURNS?
      ERROR=1
      GOTO 699
ENDIF
IF(D.EQ.8) D=0
CALL DIRECT(D,DR,DC,CR,CC)
GOTO 661

C
C**************************************************************************C
C RECORD ONE DIRECTION CODE
C**************************************************************************C
665 CHAIN(CHLEN)=D
P(R,C)=64 ;MARK BOUNDARY
R=R+DR
C=C+DC
TURNS=0
C
C******************************************************************************C
C REACH OPERAtION BORDER
C******************************************************************************C
    IF(R.LE.BORDER(1).OR.R.GE.BORDER(3).OR.
       $   C.LE.BORDER(2).OR.C.GE.BORDER(4)) THEN
          ERROR=1
       GOTO 699
       ENDIF
C******************************************************************************C
C TRACKING TERMINATION
C******************************************************************************C
    IF(P(R,C).NE.64) THEN
       CHLEN=CHLEN+1
       IF(CHLEN.GT.300) THEN ;TRACKING RUNS AWAY?
          ERROR=1
       GOTO 699
       ENDIF
       GOTO 661
    ENDIF
    699 RETURN
    END
SUBROUTINE INVERS(MATRIX1,MATRIX2)
C
C THIS SUBROUTINE DETERMINES THE INVERSE (MATRIX2) OF A HOMOGENEOUS TRANSFORMATION MATRIX (MATRIX1).
C
INTEGER*1 L,L1,L2
REAL MATRIX1(4,4),MATRIX2(4,4)
C
C******************************************************
C TRANSPOSE ROTATION MATRIX (3 x 3)
C******************************************************
DO 125 L::1,3
   DO 126 L1=1,3
      MATRIX2(L,L1)=MATRIX1(L1,L)
   126 CONTINUE
C******************************************************
C MULTIPLY THE TRANSPOSED MATRIX (3x3) TO FIRST THREE ELEMENTS IN COLUMN 4 OF THE ORIGINAL MATRIX
C******************************************************
   MATRIX2(L,4)=0.0
   DO 127 L2=1,3
      MATRIX2(L,4)=MATRIX2(L,4)-MATRIX2(L,L2)*MATRIX1(L2,4)
   127 CONTINUE
   MATRIX2(4,L)=0.0
   MATRIX2(4,4)=1.0
C
RETURN
END
C
SUBROUTINE LEFT(P,R,C,IMAX,TH)
C
C THIS SUBROUTINE CONTINUES THE BOUNDARY TRACKING WHEN
C TRACKING REACHES THE LEFT BORDER OF THE OPERATION FRAME.
C
INTEGER*1 P(256,320),TH
INTEGER*2 IMAX,R,C
C
C*************************************************************************
C REACH BOTTOM BORDER OR STARTING POINT?  
C*************************************************************************
831 IF(R.GE.IMAX.OR.P(R,C).EQ.-1) GOTO 835
   IF(P(R+1,C).LE.TH) THEN ;AN OBJECT PIXEL?
       P(R,C)=-1
       R=R+1
     GOTO 831
   ENDIF
C
835 RETURN
END
C
SUBROUTINE LINE(P,ORP,NUM)

C THIS SUBROUTINE DRAW A VECTOR ALONG THE DIRECTION OF THE
C OBJECT ORIENTATION ANGLE, BEGINNING FROM OBJECT CENTROID.
C
INTEGER*1 P(256,320),NUM,L,L1
INTEGER*2 RO,CO,R,C
REAL ORP(9,10),ORIENT

L1=INT(0.25*ORP(NUM,2)) ;VECTOR LENGTH
RO=INT(ORP(NUM,8)) ;START LOCATION
CO=INT(ORP(NUM,9))
P(RO,CO)=5
ORIENT=ORP(NUM,10)*3.14159/180 ;DIRECTION ANGLE

C******************************************************************************C
C DRAWING THE VECTOR C
C******************************************************************************C
DO 810 L=1,L1
   R=RO+INT(L*1.4*COS(ORIENT))
   C=CO+INT(L*1.4*SIN(ORIENT))
   P(R,C)=5
810 CONTINUE

RETURN
END
SUBROUTINE MOMENT(P,MARK,CHAIN,CHLEN,RO,CO,NODD,MT)
C
C THIS SUBROUTINE COMPUTES THE MOMENTS OF AN OBJECT
C OR A HOLE, AND MARK ITS BOUNDARY AS WELL.
C
INTEGER*1 P(256,320),CHAIN(300),MARK,D,DR,DC,OD,L
INTEGER*2 R,C,RO,CO,CHLEN,NODD,K1
REAL X,Y,DX,DY,AL,MT(6)
C
C**********************************************************************
C PREPARATION
C**********************************************************************
DO 96 L=1,6
   MT(L)=0.0
96 CONTINUE
NODD=0
R=RO
C=CO
P(R,C)=MARK
C**********************************************************************
C BEGIN INTEGRATION
C**********************************************************************
DO 95 K1=1,CHLEN
   D=CHAIN(K1)
   CALL DIREC2(D,DR,DC,OD)
   NODD=NODD+OD ;ODD NUMBER CHAIN CODE
   R=R+DR
   C=C+DC
   P(R,C)=MARK ;MARK BOUNDARY.
   X=FLOAT(R) ;CONVERT INTEGER
   Y=FLOAT(C) ;TO REAL NUMBER.
   DX=FLOAT(DR)
   DY=FLOAT(DC)
   AL=X*DY-Y*DX
C**********************************************************************
C MT(1-6) :: A,MX,MY,IXX,IYY,IXY, RESPETIVELY
C**********************************************************************
   MT(1)=MT(1)+AL
   MT(2)=MT(2)+AL*(Y-0.5*DY)
   MT(3)=MT(3)+AL*(X-0.5*DX)
   MT(4)=MT(4)+AL*(Y**2-Y*DY+(DY**2)/3.)
   MT(5)=MT(5)+AL*(X**2-X*DX+(DX**2)/3.)
   MT(6)=MT(6)+AL*(X*Y-0.5*(X*DY+Y*DX)+(DX*DY)/3.)
95 CONTINUE
C
RETURN
END
SUBROUTINE MULTI(MATRX1,MATRX2,MATRX3)

C
C THIS SUBROUTINE MULTIPLIES TWO MATRIXES AND OUTPUT
C THE PRODUCT.
C
REAL MATRX1(4,4), MATRX2(4,4), MATRX3(4,4), SUM
INTEGER*1 L1,L2,L3

C
DO 30 L1=1,4
    DO 31 L2=1,4
        SUM=0
        DO 32 L3=1,4
            SUM=SUM+MATRX1(L1,L3)*MATRX2(L3,L2)
        CONTINUE
    MATRX3(L1,L2)=SUM
    CONTINUE
71 CONTINUE
30 CONTINUE
C
RETURN
END
SUBROUTINE OSSAVE(ONUM,HNUM,OIP,HIP,  
$  
OCH,HCH,ORP,ARP,HRP)

C THIS SUBROUTINE SAVES THE ON-SCENE OBJECT PARAMETERS.
C
INTEGER*1 ONUM,HNUM,OCH(9,300),HCH(9,300),L,L1  
INTEGER*2 OIP(9,4),HIP(9,4),CHLEN,K1  
REAL ORP(9,10),ARP(9,10),HRP(9,10)

OPEN(3,FILE='FUNG/OSOBS',RECL=80)  
WRITE(3,3451) ONUM,HNUM

C***********************************************************************
C IF ANY OBJECT, SAVE THE OBJECT PARAMETERS
C***********************************************************************
IF(ONUM.GT.0) THEN
  DO 345 L=1,ONUM
    WRITE(3,3451) (OIP(L,L1),L1=1,4)  
    CHLEN=OIP(L,3)
    WRITE(3,3453) (OCH(L,K1),K1=1,CHLEN)
    WRITE(3,3452) (ARP(L,L1),L1=1,4)  
    WRITE(3,3452) (ARP(L,L1),L1=8,10)  
    WRITE(3,3452) (ORP(L,L1),L1=1,4)  
    WRITE(3,3452) (ORP(L,L1),L1=8,10)
  345 CONTINUE

C***********************************************************************
C IF ANY HOLE, SAVE THE HOLE PARAMETERS
C***********************************************************************
IF(HNUM.GT.0) THEN
  DO 346 L=1,HNUM
    WRITE(3,3451) (HIP(L,L1),L1=1,4)  
    CHLEN=HIP(L,3)
    WRITE(3,3453) (HCH(L,K1),K1=1,CHLEN)
    WRITE(3,3452) (HRP(L,L1),L1=1,4)  
    WRITE(3,3452) (HRP(L,L1),L1=8,10)
  346 CONTINUE
ENDIF
ENDIF

CLOSE(3)

3451 FORMAT(4I10)
3452 FORMAT(4F15.5)
3453 FORMAT(60I1)

RETURN
END
SUBROUTINE PARAM1(NUM, RP, NODD, CH,
CHAIN, CHLEN, IP, RO, CO)

C THIS SUBROUTINE ORGANIZES THE INTEGER-NUMBER PARAMETERS,
AND THE PERIMETER OF THE TRACKED BOUNDARY.

INTEGER*1 NUM, CH(9,300), CHAIN(300)
INTEGER*2 IP(9,4), RO, CO, CHLEN, NODD, K2
REAL RP(9,10)

DO 811 K2=1, CHLEN
  CH(NUM,K2) = CHAIN(K2)
811 CONTINUE

WRITE(8,*)( CHLEN )
WRITE(8,8110) ( CHAIN(K2), K2=1, CHLEN )

C******************************************************
C BOUNDARY STARTING-POINT, CHAIN LENGTH AND PERIMETER
C******************************************************
IP(NUM,1) = RO
IP(NUM,2) = CO
IP(NUM,3) = CHLEN
RP(NUM,7) = REAL(CHLEN-NODD)+SQRT(2.0)*NODD

8110 FORMAT(2X,60I1)
RETURN
END
SUBROUTINE PARAM2(NUM, RP, SCALE)

C
C THIS SUBROUTINE ORGANIZES THE REAL-NUMBER PARAMETERS
C OF THE TRACKED BOUNDARY OF AN OBJECT OR A HOLE.
C
INTEGER*1 NUM, L
REAL RP(9, 10), SCALE, T1, T2, U1, U2, LS, SA, DIST

WRITE(8,*) 'ORIGINAL'
DO 812 L = 1, 6
   WRITE(8, 8120) L, RP(NUM, L)
812 CONTINUE
8120 FORMAT(I3, F20.2)

IF(RP(NUM, 1) > 0) THEN
C*************************************************************
C RP(NUM, 1-6) = A, MX, MY, IXX, IYY, IXY, RESPECTIVELY, C
C CORRESPONDING TO IMAGE-FRAME COORDINATES C
C*************************************************************
   RP(NUM, 1) = RP(NUM, 1) / 2.
   RP(NUM, 2) = RP(NUM, 2) / 3.
   RP(NUM, 3) = RP(NUM, 3) / 3.
   RP(NUM, 4) = RP(NUM, 4) / 4.
   RP(NUM, 5) = RP(NUM, 5) / 4.
   RP(NUM, 6) = RP(NUM, 6) / 4.
WRITE(8,*) 'MODIFIED'
DO 813 L = 1, 6
   WRITE(8, 8120) L, RP(NUM, L)
813 WRITE(8, 8120) L, RP(NUM, L)
C*************************************************************
C RP(NUM, 8 & 9) = XC, YC, RESPECTIVELY. C
C THE OBJECT CENTROID OR HOLE CENTROID. C
C*************************************************************
   RP(NUM, 8) = RP(NUM, 3) / RP(NUM, 1)
   RP(NUM, 9) = RP(NUM, 2) / RP(NUM, 1)
C*************************************************************
C RP(NUM, 4-6) = CIXX, CIYY, CIXY, RESPECTIVELY. C
C CORRESPONDING TO OBJECT OR HOLE CENTROID. C
C CIYY=IYY-A*XC**2. C
C CIXY=IXY-A*YC**2. C
C CIYY=IXY-A*XC*YC. C
C*************************************************************
   RP(NUM, 4) = RP(NUM, 4) - RP(NUM, 1) * RP(NUM, 9) ** 2
   RP(NUM, 5) = RP(NUM, 5) - RP(NUM, 1) * RP(NUM, 8) ** 2
   RP(NUM, 6) = RP(NUM, 6) - RP(NUM, 1) * RP(NUM, 8) * RP(NUM, 9)
WRITE(8,*) 'SHIFTED'
DO 814 L = 4, 6
   WRITE(8, 8120) L, RP(NUM, L)
814 WRITE(8, 8120) L, RP(NUM, L)
C*************************************************************
C PRINCIPAL VALUES CORRESPONDING TO CENTROID C
C*************************************************************
   T1 = (RP(NUM, 4) + RP(NUM, 5)) / 2.
T2 = 0.5 * SQRT(((RP(NUM,4) - RP(NUM,5))**2 + 4 * RP(NUM,6)**2)
U1 = 2 * RP(NUM,6)
U2 = RP(NUM,5) - RP(NUM,4)

C
RP(NUM,3) = T1 + T2
RP(NUM,4) = T1 - T2
WRITE(8,*)) 'PRINCIPALS'
WRITE(8,*)) RP(NUM,3), RP(NUM,4)
C******************************************************C
C ORIENTATION ANGLE
C******************************************************C
IF(U2 .NE. 0) THEN
  RP(NUM,10) = 0.5 * ATAN(U1/U2) * 180 / 3.14159
IF(U2 .LT. 0) THEN
  IF(U1 .GE. 0) THEN
    RP(NUM,10) = RP(NUM,10) + 90
  ELSE
    RP(NUM,10) = RP(NUM,10) - 90
  ENDIF
ELSE
  IF(U1 .GE. 0) THEN
    RP(NUM,10) = 45.
  ELSE
    RP(NUM,10) = -45.
  ENDIF
ENDIF
C
RP(NUM,2) = RP(NUM,7) ; PERIMETER
C******************************************************C
C SCALE PARAMETERS TO REAL MEASURE
C******************************************************C
DIST = SQRT((RP(NUM,8) - 128.)**2 + (RP(NUM,9) - 160.)**2)
SA = ATAN(DIST/997.)
LS = SCALE/COS(SA)
C
RP(NUM,1) = RP(NUM,1) * (LS**2)
RP(NUM,2) = RP(NUM,2) * LS
RP(NUM,3) = RP(NUM,3) * (LS**4)
RP(NUM,4) = RP(NUM,4) * (LS**4)
C******************************************************C
C NORMALIZE PARAMETERS
C******************************************************C
RP(NUM,3) = RP(NUM,3) / RP(NUM,1)
RP(NUM,4) = RP(NUM,4) / RP(NUM,1)
RP(NUM,1) = RP(NUM,1) / RP(NUM,2)
ENDIF
C
RETURN
END
SUBROUTINE PICMAP(P)
C
C THIS SUBROUTINE PRINTS OUT A PORTION OF THE IMAGE.
C
INTEGER*1 P(256,320)
INTEGER*2 I0,I1,I2,J0,J1,J2
CHARACTER*1 YN
C
WRITE(6,*) ' EXAMINE IMAGE BY 20x20 PORTION.'
WRITE(8,*) ' EXAMINE IMAGE BY 20x20 PORTION.'
342 WRITE(6,*) ' ENTER CENTER OF THE PORTION:',
$ ' (ROW,COLUMN)'
WRITE(8,*) ' ENTER CENTER OF THE PORTION:',
$ ' (ROW,COLUMN)'
READ(5,*) I0,J0
WRITE(8,*) I0,J0
C
C******************************************************************
C EDGES OF THE PORTION C
C******************************************************************
11=I0-10
12=I0+10
J1=J0-10
J2=J0+10
C
IF(I1.LT.1) I1=1
IF(I2.GT.256) I2=256
IF(J1.LT.1) J1=1
IF(J2.GT.320) J2=320
C
C******************************************************************
C PRINT OUT THE PORTION C
C******************************************************************
WRITE(6,*) I1,I2,J1,J2
WRITE(8,*) I1,I2,J1,J2
DO 341 I0=I1,I2
   WRITE(6,3410) (P(I0,JO),JO=J1,J2)
   WRITE(8,3410) (P(I0,JO),JO=J1,J2)
341 CONTINUE
C
WRITE(6,*) ' MORE? (Y/N)'
WRITE(8,*) ' MORE? (Y/N)'
READ(5,3411) YN
WRITE(8,3411) 'YN
IF(YN.NE.'N') GOTO 342
C
3410 FORMAT(21I3)
3411 FORMAT(A1)
RETURN
END
SUBROUTINE POSIT(BORDER, SL, TH, BTC, SCALE, STD, CONT)
C
C THIS SUBROUTINE CALIBRATES THE CAMERA POSITION.
C
INTEGER*1 ONUM, HNUM, NUM, SL, TH, L, L1, CONT
INTEGER*1 OCH(9,300), HCH(9,300)
INTEGER*2 BORDER(4), OIP(9,4), HIP(9,4)
REAL ORP(9,10), ARP(9,10), HRP(9,10)
REAL BTO(4,4), CTO(4,4), OTC(4,4), BTC(4,4)
REAL XBASE, YBASE, ZBASE, ORIENT, SCALE, STD
CHARACTER*1 YN

124 WRITE(6,*) 'PLACE THE STANDARD OBJECT AT THE CENTER',
$ 'OF THE VIEWING WINDOW.'
CALL PROCES(BORDER, SL, TH, ONUM, HNUM, OIP, HIP,
$ OCH, HCH, ORP, ARP, HRP, 1.0)
WRITE(6,1201) ONUM
DO 120 L=1, ONUM
WRITE(6,1200) L, ORP(L,8), ORP(L,9)
120 CONTINUE
WRITE(6,*) 'WHICH ONE IS THE STANDARD OBJECT?',
$ '(O=NOT THERE)'
READ(5,*) NUM
IF(NUM.EQ.0) GOTO 124
C******************************************************
C NEW CONVERSION SCALE
C******************************************************
ORP(NUM,1)=ORP(NUM,1)*ORP(NUM,2) ;DE-NORMALIZE
SCALE=SQRT(STD/ORP(NUM,1)) ;NEW SCALE
WRITE(8,*) 'AREA =', ORP(NUM,1), 'SCALE =', SCALE
IF(CONT.EQ.0) THEN
WRITE(6,*) 'SCALE DETERMINATION IS COMPLETED.'
WRITE(8,*) 'SCALE DETERMINATION IS COMPLETED.'
GOTO 129
ENDIF
C******************************************************
C PREPARE MATRIXES
C******************************************************
DO 121 L=1,4
  DO 122 L1=1,4
    IF(L.EQ.L1) THEN
      BTO(L,L1)=1.0
      CTO(L,L1)=1.0
    ELSE
      BTO(L,L1)=0.0
      CTO(L,L1)=0.0
    ENDIF
  122 CONTINUE
121 CONTINUE
C LOCATION & ORIENTATION OF REFERENCE OBJECT, C
IN ROBOT-BASE COORDINATES. C

123 WRITE(6,*) ' INPUT BASE POSITION OF THE OBJECT.'
WRITE(6,*) ' INPUT LOCATION. (X,Y,Z IN MILLIMETRES)'
READ(5,*) XBASE,YBASE,ZBASE
WRITE(6,*) ' INPUT ORIENTATION ANGLE. (DEGREE)'
READ(5,*) ORIENT
WRITE(6,1202) XBASE,YBASE,ZBASE,ORIENT
WRITE(6,*) ' ARE THESE VALUES CORRECT? (Y/N)' 
READ(5,1203) YN
IF(YN.NE.'y') GOTO 123
WRITE(8,1202) XBASE,YBASE,ZBASE,ORIENT
C CONSTRUCT HOMOGENEOUS TRANSFORMATION MATRIXES C

ORIENT=ORIENT*3.14159/180.0
BTO(1,1)=COS(ORIENT) ;OBJECT IN ROBOT-BASE
BTO(2,2)=BTO(1,1)
BTO(2,1)=SIN(ORIENT)
BTO(1,2)=-BTO(2,1)
BTO(1,4)=XBASE
BTO(2,4)=YBASE
BTO(3,4)=ZBASE
WRITE(8,*) ' BTO ='
DO 125 L=1,4
125 WRITE(8,1204) (BTO(L,L1),L1=1,4)

ORIENT=ORP(NUM,10)*3.14159/180.0
CTO(1,1)=COS(ORIENT) ;OBJECT IN CAMERA
CTO(2,2)=CTO(1,1)
CTO(2,1)=SIN(ORIENT)
CTO(1,2)=-CTO(2,1)
CTO(1,4)=ORP(NUM,8)
CTO(2,4)=ORP(NUM,9)
WRITE(8,*) ' CTO ='
DO 126 L=1,4
126 WRITE(8,1204) (CTO(L,L1),L1=1,4)
CALL INVERS(CTO,OTC) ;CAMERA IN OBJECT
WRITE(8,*) ' OTC ='
DO 127 L=1,4
127 WRITE(8,1204) (OTC(L,L1),L1=1,4)
CALL MULTI(BTO,OTC,BTC) ;CAMERA IN ROBOT-BASE
WRITE(8,*) ' BTC ='
DO 128 L=1,4
128 WRITE(8,1204) (BTC(L,L1),L1=1,4)
C
1200 FORMAT(10X,I10,2F10.2)
1201 FORMAT( ' THERE ARE',I2, ' OBJECTS IN THE WINDOW.')
1202 FORMAT(' LOCATION: (',F7.2,',',F7.2,',',F7.2,
$       '); ORIENTATION: (',F6.2,')' )
1203 FORMAT(A1)
1204 FORMAT(4F10.2)
C
   129 RETURN
END
SUBROUTINE PROCES(BORDER, SL, TH, ONUM, HNUM, OIP, HIP, 
  OCH, HCH, ORP, ARP, HRP, SCALE)

C THIS SUBROUTINE PROCESSES A GRABBED IMAGE TO OBTAIN 
C BOUNDARY CHAIN CODES, AS WELL AS THE INTEGER- AND 
C REAL-NUMBER PARAMETERS OF THE OBJECTS AND HOLES.

INTEGER*1 P(256, 320), OCH(9, 300), HCH(9, 300), CHAIN(300)
INTEGER*1 TH, SL, OM, HM, ONUM, HNUM, ERROR, L, L1
INTEGER*2 OIP(9, 4), HIP(9, 4), I, J, C, CHLEN, NODD, K
INTEGER*2 BORDER(4), IMIN, JMIN, IMAX, JMAX, RO, CO
INTEGER*2 JA, JC
REAL ORP(9, 10), ARP(9, 10), HRP(9, 10), M(6), SCALE
CHARACTER*1 YN

CALL GRAB(P) ;INPUT IMAGE

C******************************************************
C PREPARATION
C******************************************************
IMIN = BORDER(1)
JMIN = BORDER(2)
IMAX = BORDER(3)
JMAX = BORDER(4)
ONUM = 0
HNUM = 0
DO 405 I = 1, 5
  DO 406 J = 1, 4
    OIP(I, J) = 0
    HIP(I, J) = 0
  CONTINUE
DO 407 J = 1, 10
  ORP(I, J) = 0.0
  ARP(I, J) = 0.0
  HRP(I, J) = 0.0
CONTINUE
405 CONTINUE
OM = -1
HM = 64
I = IMIN
J = JMIN

C******************************************************
C IF ANY OBJECT ACROSS THE BORDER
C******************************************************
401 IF (P(I, J) .LE. TH) THEN
  IF (P(I, J) .LT. 0) GOTO 421
  C = J
  GOTO 414
ENDIF

C
C**********************************************
C SEARCHING FOR OBJECT BOUNDARY
C**********************************************
400 IF(P(I,J+SL).GT.TH) GOTO 440 ;ROUGH SEARCH
   C=J+1
410 IF(P(I,C).GT.TH) THEN ;FINE SEARCH
   C=C+1
   GOTO 410
ENDIF
C**********************************************
C A MARKED BOUNDARY?
C**********************************************
IF(P(I,C).LT.0.OR.P(I,C-1).GT.63) GOTO 420
C**********************************************
C OBJECT BOUNDARY TRACKING
C**********************************************
414 JC=J+SL
   WRITE(6,4111) 'OUTER',I,J,C,(P(I,JA),JA=J,JC)
   WRITE(8,4111) 'OUTER',I,J,C,(P(I,JA),JA=J,JC)
   CALL BTRACE(P,BORDER,CHAIN,CHLEN,I,C,TH,ERROR)
IF(ERROR.EQ.1) THEN
   WRITE(6,*) 'TRACKING WENT WRONG, IGNORE AND GO ON.'
   WRITE(8,*) 'TRACKING WENT WRONG.'
   WRITE(6,*) 'NEED TO EXAMINE THE IMAGE? (Y/N)'
   READ(5,4000) YN
   IF(YN.EQ.'Y') CALL PICMAP(P)
   CALL ERASE(P,BORDER,CHAIN,CHLEN,I,C,-1,TH)
   GOTO 420
ENDIF
IF(ERROR.EQ.2) THEN
   WRITE(6,*) 'THIS WAS A NOISE, IGNORE AND GO ON.'
   WRITE(8,*) 'NOISE.'
   WRITE(6,*) 'NEED TO EXAMINE THE IMAGE? (Y/N)'
   READ(5,4000) YN
   IF(YN.EQ.'Y') CALL PICMAP(P)
   P(I,C)=TH-1
   GOTO 440
ENDIF
IF(ERROR.EQ.3) THEN
   WRITE(6,*) 'THIS WAS AN INCOMPLETE OBJECT.'
   WRITE(8,*) 'INCOMPLETE OBJECT.'
   WRITE(6,*) 'NEED TO EXAMINE THE IMAGE? (Y/N)'
   READ(5,4000) YN
   IF(YN.EQ.'Y') CALL PICMAP(P)
   GOTO 421
ENDIF
E-50

C**************************************************************
C OBJECT BOUNDARY TRACKING SUCCEEDS; COMPUTE MOMENTS
C**************************************************************
WRITE(6,*)' THIS IS A COMPLETE OBJECT.'
OM=OM-1
CALL MOMENT(P,OM,CHAIN,CHLEN,I,C,NODD,MT)
ONUM=-OM-1
DO 412 L=1,6
   ORP(ONUM,L)=MT(L)
   ARP(ONUM,L)=MT(L)
412 CONTINUE
CALL PARAM1(ONUM,ORP,NODD,OCH,CHAIN,CHLEN,OIP,I,C)
   ARP(ONUM,7)=ORP(ONUM,7)
C**************************************************************
C MOVE ONE STEP ALONG COLUMN OR ROW DIRECTION
C**************************************************************
420 J=J+SL
IF(J.GE.(JMAX-SL)) THEN
   I=I+SL
IF(I.GE.(IMAX-SL)) GOTO 459
   J=JMIN
GOTO 401
ENDIF
C**************************************************************
C SEARCHING FOR HOLE BOUNDARY
C**************************************************************
421 IF(P(I,J+SL).LE.TH) GOTO 420 ;ROUGH SEARCH
   C=J+1
430 IF(P(I,C).LE.TH) THEN ;FINE SEARCH
   C=C+1
GOTO 430
ENDIF
C**************************************************************
C A MARKED BOUNDARY?
C**************************************************************
IF(P(I,C).GT.63.0R.P(I,C-1).LT.0) GOTO 440
C**************************************************************
C HOLE BOUNDARY TRACKING
C**************************************************************
JC=J+SL
WRITE(6,4111) ' HOLE',I,J,C,(P(I,JA),JA=J,JC)
WRITE(8,4111) ' HOLE',I,J,C,(P(I,JA),JA=J,JC)
CALL HTRACE(P,BORDER,CHAIN,CHLEN,I,C,TH,ERROR)
C
   IF(ERROR.EQ.1) THEN
      WRITE(6,*) ' TRACKING WENT WRONG.',
      WRITE(8,*) ' TRACKING WENT WRONG.'
      WRITE(6,*) ' NEED TO EXAMINE THE IMAGE? (Y/N)'
      READ(S,4000) YN
      IF(YN.EQ.'Y') CALL PICMAP(P)
   ELSE
      CALL EXAMINE(P,CHAIN,CHLEN,I,C,TH,ERROR)
      IF(ERROR.EQ.1) THEN
         WRITE(6,*) ' TRACKING WENT WRONG.'
         WRITE(8,*) ' TRACKING WENT WRONG.'
         WRITE(6,*) ' NEED TO EXAMINE THE IMAGE? (Y/N)'
         READ(S,4000) YN
         IF(YN.EQ.'Y') CALL PICMAP(P)
   ENDIF
C**************************************************************
C OBJECT BOUNDARY TRACKING SUCCEEDS; COMPUTE MOMENTS
C**************************************************************
CALL ERASE(P,BORDER,CHAIN,CHLEN,I,C,64,TH)
GOTO 440
ENDIF
IF(ERROR.EQ.2) THEN
WRITE(6,*) 'THIS WAS A NOISE, IGNORE AND GO ON.'
WRITE(8,*) 'NOISE.'
WRITE(6,*) 'NEED TO EXAMINE THE IMAGE? (Y/N)'
READ(5,4000)YN
IF(YN.EQ.'Y') CALL PICMAP(P)
P(I,C)=TH+1
GOTO 420
ENDIF
C******************************************************c
C HOLE-OBJECT RELATION C
C******************************************************c
K=C
434 K=K-1
IF(P(I,K).GE.0) THEN
GOTO 434
ELSE
IF(P(I,K).EQ.-1) THEN
ERROR=3
ELSE
C******************************************************c
C HOLE BOUNDARY TRACKING SUCCEEDS; COMPUTE MOMENTS C
C******************************************************c
HM=HM+1
HNUM=HNUM-64
HIP(HNUM,4)=-P(I,K)-1
L=HIP(HNUM,4)
OIP(L,4)=OIP(L,4)+1
WRITE(6,*)'THIS WAS A COMPLETE HOLE.'
CALL MOMENT(P,HM,CHAIN,CHLEN,I,C,NODD,MT)
DO 433 L1=1,6
HRP(HNUM,L1)=MT(L1)
ARP(L,L1)=ARP(L,L1)-MT(L1)
433 CONTINUE
CALL PARAM1(HNUM,HRP,NODD,HCH,CHAIN,CHLEN,
HIP,I,C)
ENDIF
ENDIF
C
IF(ERROR.EQ.3) THEN
WRITE(6,*) 'THIS WAS A HOLE IN AN INCOMPLETE ',
'OBJECT.'
WRITE(6,*) 'INCOMPLETE HOLE.'
WRITE(6,*) 'NEED TO EXAMINE THE IMAGE? (Y/N)'
READ(5,4000)YN
IF(YN.EQ.'Y') CALL PICMAP(P)
GOTO 440
ENDIF
C**********************************************************************C
C MOVE ONE STEP ALONG COLUMN OR ROW DIRECTION
C**********************************************************************C

440  J=J+SL
     IF(J.GE.(JMAX-SL)) THEN
       I=I+SL
       IF(I.GE.(IMAX-SL)) GOTO 459
       J=JMIN
     GOTO 401
     ENDIF
     GOTO 400

C**********************************************************************C
C BOUNDARY SEARCH STOPS; ORGANIZE PARAMETERS
C**********************************************************************C

459  IF(ONUM.EQ.0) GOTO 490
     DO 461 L=1,ONUM
       WRITE(8,*) ' OUTER PARAMS'
       CALL PARAM2(L,ORP,SCALE)
       WRITE(8,*) ' OBJECT PARAMS'
       CALL PARAM2(L,ARP,SCALE)
       OIP(L,1)=OIP(L,1)-INT(ORP(L,8))
       OIP(L,2)=OIP(L,2)-INT(ORP(L,9))
       WRITE(8,*) L,(OIP(L,L1),L1=1,4)
     461  CONTINUE
     IF(HNUM.EQ.0) GO TO 489
     DO 462 L=1,HNUM
       WRITE(8,*) ' HOLE PARAMS'
       CALL PARAM2(L,HRP,SCALE)
       L1=HIP(L,4)
       HIP(L,1)=HIP(L,1)-INT(ORP(L1,8))
       HIP(L,2)=HIP(L,2)-INT(ORP(L1,9))
       WRITE(8,*) L,(HIP(L,L1),L1=1,4)
     462  CONTINUE

C**********************************************************************C
C DISPLAY BOUNDARIES
C**********************************************************************C

489  WRITE(6,*), ' BOUNDARIES OF ON-SCENE OBJECTS.'
     CALL CLEAR(P)
     DO 470 L=1,ONUM
       RO=INT(ORP(L,8))
       CO=INT(ORP(L,9))
       CALL BNDYOS(P,L,HNUM,OIP,HIP,OCH,HCH,RO,CO)
       CALL LINE(P,ORP,L)
     470  CONTINUE
     CALL DISP(P)

C
4000 FORMAT(A1)
4111 FORMAT(A6,3I4,9I3)
490  RETURN
END
SUBROUTINE RIGHT(P,R,C,IMIN,TH)
C
C THIS SUBROUTINE CONTINUES THE BOUNDARY TRACKING WHILE THE
C TRACKING REACH THE RIGHT BORDER OF THE OPERATION FRAME.
C
INTEGER*1 P(256,320),TH
INTEGER*2 IMIN,R,C
C
836 IF(P(R-1,C).LE.TH) THEN ; AN OBJECT PIXEL?
    P(R,C)=-1
    R=R-1
    IF(R.LE.IMIN) GOTO 840 ; REACH TOP BORDER?
    GOTO 836
ENDIF
C
840 RETURN
END
C
SUBROUTINE THRES(P,SL1,TH,SH)

C THIS SUBROUTINE DETERMINES THE THRESHOLD VALUE.

INTEGER*1 P(256,320),AV,TH,SH,SL1
INTEGER*2 I,J
INTEGER*4 KA,KB
CHARACTER*1 YN

C******************************************************
C ADD UP SAMPLED PIXEL GRAY-LEVELS & NUMBER OF PIXELS
C******************************************************
KA=0
KB=0
DO 130 I=1,256,SL1
  DO 130 J=1,320,SL1
    KA=KA+P(I,J)
    KB=KB+1
130 CONTINUE
AV=INT(KA/KB) ;AVERAGE GRAY LEVEL

C******************************************************
C OBTAIN THRESHOLD BY ADJUSTING SHIFT VALUE
C******************************************************
TH=AV+SH
WRITE(8,*) ' CURRENT THRESHOLD GRAY LEVEL IS',TH
WRITE(6,*) ' CURRENT THRESHOLD GRAY LEVEL IS',TH
CALL BINARY(P,TH)
WRITE(8,*) ' IS THE IMAGE ACCEPTABLE? (Y/N)'
WRITE(6,*) ' IS THE IMAGE ACCEPTABLE? (Y/N)'
READ(5,1310) YN
WRITE(8,1310) YN
IF(YN.NE.'Y') THEN
  WRITE(8,*) ' IMAGE'S AVERAGE GRAY LEVEL IS:',AV
  WRITE(6,*) ' IMAGE'S AVERAGE GRAY LEVEL IS:',AV
  WRITE(8,*) ' CURRENT SHIFT VALUE IS:',SH
  WRITE(6,*) ' CURRENT SHIFT VALUE IS:',SH
  WRITE(8,*) ' ENTER A NEW ONE:'
  WRITE(6,*) ' ENTER A NEW ONE:'
  READ(5,*) SH
  WRITE(8,*) SH
  GOTO 131
ENDIF

C 1310 FORMAT(A1)
C
RETURN
END
SUBROUTINE TOP(P,R,C,JMIN,TH)
C
C THIS SUBROUTINE CONTINUES THE BOUNDARY TRACKING WHILE THE
C TRACKING REACH THE TOP BORDER OF THE OPERATION FRAME.
C
INTEGER*1 P(256,320),TH
INTEGER*2 JMIN,R,C
C
821 IF(P(R,C-1).LE.TH) THEN ;AN OBJECT PIXEL?
    P(R,C)=-1
    C=C-1
    IF(C.LE.JMIN) GOTO 825 ;REACH LEFT BORDER?
    GOTO 821
ENDIF
C
825 RETURN
END
PROGRAM TP

C
C MAIN PROGRAM OF SYSTEM TRAINING AND OBJECT POSITIONING.
C
INTEGER*1 RONUM,RHNUM,ROCH(19,300),RHCH(19,300)
INTEGER*1 CHOICE,TH,SH,SL1,SL
INTEGER*2 ROIP(19,4),RHIP(19,4),BORDER(4)
REAL RORP(19,4),RHRP(19,4)
REAL BTC(4,4),MARGIN(4),SCALE,STD
CHARACTER*10 RNAMES(19)

C
C PREFIX R = RECORDED DATA.
C
C******************************************************
C OPEN OPERATION TRACE FILE & LOAD DATA BASE
C******************************************************
OPEN(8,FILE='FUNG/TRACE2',RECL=80)
WRITE(6,*) ' START WORKING.'
WRITE(8,*) ' START WORKING.'
CALL DREAD(RONUM,RHNUM,RNAMES,ROIP,RHIP,ROCH,RHCH,
RORP,RHRP,BTC,MARGIN,SCALE,STD,TH,SH,SL1,SL,BORDER)

C******************************************************
C DISPLAY OPERATION MENU
C******************************************************
1 WRITE(6,*) ' WHAT DO YOU WISH TO DO NEXT?'
WRITE(6,*) ' 3. TRAINING,'
WRITE(6,*) ' 4. POSITION PUMA TO AN OBJECT,'
WRITE(6,*) ' 7. REARRANGE DATA BASE,'
WRITE(6,*) ' 8. SAVE DATA BASE,'
WRITE(6,*) ' 9. EXIT TO OPERATION SYSTEM.'
WRITE(6,*) ' ENTER YOUR CHOICE:

WRITE(8,*) ' WHAT DO YOU WISH TO DO NEXT?'
WRITE(8,*) ' 3. TRAINING,'
WRITE(8,*) ' 4. POSITION PUMA TO AN OBJECT,'
WRITE(8,*) ' 7. REARRANGE DATA BASE,'
WRITE(8,*) ' 8. SAVE DATA BASE,'
WRITE(8,*) ' 9. EXIT TO OPERATING SYSTEM.'
READ(5,*) CHOICE
WRITE(8,*) ' ENTER YOUR CHOICE:',CHOICE

C******************************************************
C SELECT OPERATION OPTION
C******************************************************
IF(CHOICE.EQ.3) THEN
    CALL TRAIN(BORDER,MARGIN,SCALE,SL,TH,RONUM,RHNUM,
RORP,RHRP,BTC,RONUM,RHNUM,RNAMES,ROIP,RHIP,ROCH,RHCH,
RORP,RHRP)
ENDIF
IF(CHOICE.EQ.4) CALL PUMA(BORDER,SL,TH,MARGIN,SCALE,
BTC,RONUM,RHNUM,RNAMES,ROIP,RHIP,ROCH,RHCH,
RORP,RHRP)
IF(CHOICE.EQ.7) CALL DBASE(RONUM,RHNUM,RNAMES,ROIP, $ RHIP,ROCH,RHCH,RORP,RHRP,BTC,MARGIN,SCALE)
  IF(CHOICE.EQ.8) THEN
    CALL DSAVE(RONUM,RHNUM,RNAMES,ROIP,RHIP,ROCH,RHCH, $ RORP,RHRP,BTC,MARGIN,SCALE,STD,TH,SH,SL1,SL,BORDER,1)
  ENDIF
IF(CHOICE.NE.9) GOTO 1
C
WRITE(6,*) ' BREAK TIME.'
WRITE(8,*) ' BREAK TIME.'
C
STOP
END
SUBROUTINE BNDY(P,BP,CHAIN,RO,CO)
C
C THIS SUBROUTINE USES A CHAIN CODE TO CONSTRUCT
C A BOUNDARY OUTLINE FOR AN OBJECT OR A HOLE.
C
INTEGER*1 P(256,320),CHAIN(300),D,DR,DC,OD
INTEGER*2 R,C,RO,CO,BP(3),CHLEN,K2
C
R=RO+BP(1)
C=CO+BP(2)
CHLEN=BP(3)
C
C******************************************************
C CONSTRUCTING BOUNDARY
C******************************************************
P(R,C)=1
DO 820 K2=1,CHLEN
   D=CHAIN(K2)
   CALL DIREC2(D,DR,DC,OD)
   R=R+DR
   C=C+DC
   P(R,C)=1
820 CONTINUE
C
RETURN
END
SUBROUTINE BNDYOS(P,NUM,HNUM,OIP,HIP,OCH,HCH,RO,CO)
C
C THIS SUBROUTINE CONSTRUCTS A BOUNDARY IMAGE OF
C AN ON-SCENE OBJECT, AS WELL AS ITS HOLE(S).
C
INTEGER*1 P(256,320),NUM,HNUM,L,L1
INTEGER*1 OCH(9,300),HCH(9,300),CHAIN(300)
INTEGER*2 RO,CO,OIP(9,4),HIP(9,4),BP(3)
INTEGER*2 CHLEN,K2
C
WRITE(6,*)
1 CONSTRUCTING THE BOUNDARY OF AN’,
$ ’ ON SCENE OBJECT.’
C*******************************************************
C CONSTRUCTING OBJECT BOUNDARY
C*******************************************************
BP(1)=OIP(NUM,1)
BP(2)=OIP(NUM,2)
BP(3)=OIP(NUM,3)
CHLEN=BP(3)
C
DO 830 K2=1,CHLEN
   CHAIN(K2)=OCH(NUM,K2)
830 CONTINUE
CALL BNDY(P,BP,CHAIN,RO,CO)
C
C*******************************************************
C CONSTRUCTING HOLE BOUNDARY IF THERE IS ANY
C*******************************************************
IF(OIP(NUM,4).GT.0) THEN
   L1=1
   DO 831 L1=1,HNUM
      IF(HIP(L,4).EQ.NUM) THEN
         BP(1)=HIP(L,1)
         BP(2)=HIP(L,2)
         BP(3)=HIP(L,3)
         CHLEN=BP(3)
         DO 833 K2=1,CHLEN
            CHAIN(K2)=HCH(L,K2)
833 CONTINUE
         CALL BNDY(P,BP,CHAIN,RO,CO)
      ENDIF
      L1=L1+1
      IF(L1.GT.OIP(NUM,4)) GOTO 839
   ENDIF
831 CONTINUE
839 RETURN
END
C
SUBROUTINE BNDYRD(P,RNUM,RHNUM,ROIP,RHIP,
$       ROCH,RHCH,RO,CO)
C
C THIS SUBROUTINE CONSTRUCTS A BOUNDARY IMAGE OF
C A RECORDED OBJECT, AS WELL AS HOLES.
C
INTEGER*1 P(256,320),RNUM,RHNUM,L,L1
INTEGER*1 ROC(19,300),RHCH(19,300),CHAIN(300)
INTEGER*2 RO,CO,ROIP(19,4),RHIP(19,4),BP(3)
INTEGER*2 CHLEN,K2
C
WRITE(6,*) 'CONSTRUCTING THE BOUNDARY OF A',
$   'RECORDED OBJECT.'
C******************************************************c
C CONSTRUCTING OBJECT BOUNDARY
C******************************************************c
BP(1)=ROIP(RNUM,1)
BP(2)=ROIP(RNUM,2)
BP(3)=ROIP(RNUM,3)
CHLEN=BP(3)
C
DO 840 K2=1,CHLEN
   CHAIN(K2)=ROCH(RNUM,K2)
840 CONTINUE
CALL BNDY(P,BP,CHAIN,RO,CO)
C
C******************************************************c
C CONSTRUCTING HOLE BOUNDARY IF THERE IS ANY
C******************************************************c
IF(ROIP(RNUM,4).NE.0) THEN
   L1=1
   DO 841 L=1,RHNUM
      IF(RHIP(L,4).EQ.RNUM) THEN
         BP(1)=RHIP(L,1)
         BP(2)=RHIP(L,2)
         BP(3)=RHIP(L,3)
         CHLEN=BP(3)
         DO 843 K2=1,CHLEN
            CHAIN(K2)=RHCH(L,K2)
843     CONTINUE
      CALL BNDY(P,BP,CHAIN,RO,CO)
      L1=L1+1
      IF(L1.GT.ROIP(RNUM,4)) GOTO 849
841    CONTINUE
   ENDIF
849 RETURN
END
SUBROUTINE BOTH(P, NUM, HNUM, OIP, HIP, OCH, HCH, RNUM, RHNUM, ROIP, RHIP, ROCH, RHCH)

C THIS SUBROUTINE DISPLAYS THE BOUNDARY IMAGES OF
C AN ON-SCENE OBJECT AND A RECORDED OBJECT.
C
INTEGER*1 P(256,320), OCH(9,300), HCH(9,300), NUM, HNUM
INTEGER*1 ROCH(19,300), RHCH(19,300), RNUM, RHNUM
INTEGER*2 OIP(9,4), HIP(9,4), ROIP(19,4), RHIP(19,4)

WRITE(6,*) ' DISPLAY BOTH OBJECTS ON SCREEN.'
WRITE(8,*) ' DISPLAY BOTH OBJECTS ON SCREEN.'
CALL CLEAR(P)
CALL BNDYOS(P, NUM, HNUM, OIP, HIP, OCH, HCH, 128, 80)
CALL BNDYRD(P, RNUM, RHNUM, ROIP, RHIP, ROCH, RHCH, 128, 240)
WRITE(6,*) ' LEFT--ON SCENE; RIGHT--RECORDED.'
WRITE(8,*) ' LEFT--ON SCENE; RIGHT--RECORDED.'
CALL DISP(P)

RETURN
END
SUBROUTINE CLEAR(P)

C THIS SUBROUTINE CLEARS THE IMAGE MAP.

INTEGER*1 P(256,320)
INTEGER*2 I,J

C 63: COLOR OF WHITE.

WRITE(6,*) 'CLEARING IMAGE MAP. PLEASE WAIT.'
DO 90 I=1,256
   DO 90 J=1,320
      P(I,J)=63
90 CONTINUE

RETURN
END
SUBROUTINE DBASE(RONUM,RHNUM,RNAMES,ROIP,RHIP,
   $ ROCH,RHCH,RORP,RHRP,BTC,MARGIN,SCALE)

C THIS SUBROUTINE MODIFIES THE DATA BASE.
C
INTEGER*1 RONUM,RHNUM,ROCH(19,300),RHCH(19,300)
INTEGER*1 CHOICE,L,L1,L2,L3,L4,L5
INTEGER*2 ROIP(19,4),RHIP(19,4),CHLEN,K1
REAL RORP(19,4),RHRP(19,4),BTC(4,4),MARGIN(4),SCALE
CHARACTER*10 RNAMES(19)
CHARACTER*1 YN

C******************************************************
C DISPLAY RECORDED OBJECT NAMES & OPERATION MENU
C********************************************************
WRITE(8,2000) RONUM,RHNUM
WRITE(8,2001) (L,RNAMES(L),L=1,RONUM)
WRITE(8,*), 'REARRANGE RECORDED DATAS:'
WRITE(8,*),' 1. DELETE AN OBJECT:'
WRITE(8,*),' 2. RENAME AN OBJECT:'
WRITE(8,*),' 3. REVISE TRANSFORMATION MATRIX:'
WRITE(8,*),' 4. NEW SCALE:'
WRITE(8,*),' 5. MARGIN FOR RECOGNIZE OBJECTS:'
WRITE(8,*),' 9. EXIT.'

207 WRITE(6,2000) RONUM,RHNUM
WRITE(6,2001) (L,RNAMES(L),L=1,RONUM)
WRITE(6,*), 'REARRANGE RECORDED DATAS:'
WRITE(6,*),' 1. DELETE AN OBJECT:'
WRITE(6,*),' 2. RENAME AN OBJECT:'
WRITE(6,*),' 3. REVISE TRANSFORMATION MATRIX:'
WRITE(6,*),' 4. NEW SCALE:'
WRITE(6,*),' 5. MARGIN FOR RECOGNIZE OBJECTS:'
WRITE(6,*),' 9. EXIT.'
WRITE(6,*),' ENTER YOUR CHOICE:'
READ(5,*) CHOICE
WRITE(8,*), 'ENTER YOUR CHOICE:',CHOICE

C********************************************************
C SELECTING OPERATION OPTION
C********************************************************
IF(CHOICE.EQ.1) THEN
211 WRITE(6,*), 'WHICH ONE IS TO BE DELETED? (O=EXIT)'
   READ(5,*),L
   IF(L.EQ.O) GOTO 207
   WRITE(6,*), 'ARE YOU SURE? (Y/N)'
   READ(5,2004),YN
   IF(YN.NE.'Y') GOTO 211
   WRITE(8,*), 'WHICH ONE IS TO BE DELETED?',L
C*********************************************************************************************C
C DELETING THE OBJECT AT THE END OF THE LIST C
C*********************************************************************************************C
RONUM=RONUM-1
IF((L-1).EQ.RONUM) THEN
    RNUM=RNUM-ROIP(L,4)
    IF(RONUM.EQ.0) THEN
        WRITE(6,*), ' DATA BASE IS EMPTY.'
        RNUM=0
    ENDIF
    GOTO 207
ENDIF
C*********************************************************************************************C
C DELETING HOLE BOUNDARY IF THERE IS ANY C
C*********************************************************************************************C
IF(ROIP(L,4).GT.0) THEN
    L2=1
    DO 203 L3=1,RHNUM
        IF(RHIP(L3,4).EQ.L) THEN
            RHNlJM=RHNUM-1
            DO 204 L4=L3,RHNUM
                ROIP(L4,L2)=ROIP(L4+1,L2)
                RORP(L4,L2)=RORP(L4+1,L2)
            204 CONTINUE
            CHLEN=ROIP(L4,3)
            DO 206 K1=1,CHLEN
                ROCH(L4,K1)=ROCH(L4+1,K1)
            206 CONTINUE
        ELSE
            GOTO 212
        ENDIF
    203 CONTINUE
ENDIF
C
C*********************************************************************************************C
C DELETING OBJECT BOUNDARY C
C*********************************************************************************************C
213 DO 200 L1=1,RONUM
    DO 201 L2=1,4
        ROIP(L1,L2)=ROIP(L1+1,L2)
        RORP(L1,L2)=RORP(L1+1,L2)
    201 CONTINUE
    CHLEN=ROIP(L1,3)
    DO 202 K1=1,CHLEN
        ROCH(L1,K1)=ROCH(L1+1,K1)
    202 CONTINUE
CONTINUE
RNAMES(L1)=RNAMES(L1+1) ;DELETE NAME
CONTINUE

C******************************************************
C ADJUSTING HOLE-OBJECT RELATION CODES
C******************************************************
DO 215 L2=1,RHNUM
   IF(RHIP(L2,4).GT.L) THEN
      DO 216 L3=L2,RHNUM
         RHIP(L3,4)=RHIP(L3,4)-1
      CONTINUE
   ENDIF
  216 CONTINUE
  GOTO 207
ENDIF

C******************************************************
C RENAME AN OBJECT
C******************************************************
IF(CHOICE.EQ.2) THEN
   WRITE(6,*) ' WHICH ONE IS TO BE RENAMED? (O=EXIT)'
   READ(5,*) L
   IF(L .EQ. O) THEN
      WRITE(6,*) ' NO RENAME.'
   ENDIF
   WRITE(8,*) ' WHICH ONE IS TO BE RENAMED?',L
   WRITE(6,*) ' ENTER THE NEW NAME:'
   READ(5,2002) RNAMES(L)
   WRITE(8,*) ' ENTER THE NEW NAME:',RNAMES(L)
ENDIF

C******************************************************
C CHANGE TRANSFORMATION MATRIX
C******************************************************
IF(CHOICE.EQ.3) THEN
   WRITE(8,*) ' CURRENT TRANSFORMATION MATRIX IS:'
   WRITE(6,*) ' CURRENT TRANSFORMATION MATRIX IS:'
   DO 208 L=1,4
      WRITE(8,2003) (BTC(L,L1),L1=1,4)
      WRITE(6,2003) (BTC(L,L1),L1=1,4)
   CONTINUE
   WRITE(8,*) ' WHICH ELEMENT IS TO BE REVISED:',(ROW,COLUMN)
   WRITE(6,*) ' WHICH ELEMENT IS TO BE REVISED:',(ROW,COLUMN)
   READ(5,*) L2,L3
   WRITE(8,*) ' ENTER THE NEW VALUE:'
   WRITE(6,*) ' ENTER THE NEW VALUE:'
   READ(5,2003) BTC(L2,L3)
   WRITE(8,*) ' NEW VALUE',BTC(L2,L3)
WRITE(8,*), MORE? (Y/N)
WRITE(6,*), MORE? (Y/N)
READ(5,2004) YN
WRITE(8,2004) YN
IF(YN.NE.'N') GOTO 210
ENDIF
C******************************************************C
C NEW CONVERSION SCALE
C******************************************************C
IF(CHOICE.EQ.4) THEN
WRITE(8,*), CURRENT SCALE IS:, SCALE
WRITE(8,*), ENTER THE NEW ONE:
WRITE(6,*), CURRENT SCALE IS:, SCALE
WRITE(6,*), ENTER THE NEW ONE:
READ(5,2003) SCALE
WRITE(8,2003) SCALE
ENDIF
C******************************************************C
C NEW COMPARISON MARGINS
C******************************************************C
IF(CHOICE.EQ.5) THEN
WRITE(8,*), CURRENT MARGIN FOR OBJECT ID:
WRITE(8,*), 1-AREA; 2-PERIMATER; 3-MAX I; 4-MIN I
WRITE(8,2003) (MARGIN(L),L=1,4)
WRITE(8,*), WHICH ONE IS TO BE CHANGED? (O=EXIT)
WRITE(8,*) L
WRITE(8,*), WHICH ONE IS TO BE CHANGED? (O=EXIT)
WRITE(6,*) L
READ(5,*) L
IF(L.NE.0) THEN
WRITE(8,*), INPUT NEW MARGIN, MUST < 1:
WRITE(6,*), INPUT NEW MARGIN, MUST < 1:
READ(5,*) MARGIN(L)
GOTO 214
ENDIF
ENDIF
C
IF(CHOICE.NE.9) GOTO 207
C
2000 FORMAT(' TOTAL ',I2,' OBJECTS AND ',I2,' HOLES. ')
2001 FORMAT(5(I2,-',A10))
2002 FORMAT(A10)
2003 FORMAT(4F15.5)
2004 FORMAT(A1)
RETURN
END
C
SUBROUTINE DIREC2(D,DR,DC,OD)

C THIS SUBROUTINE GIVES ROW AND COLUMN ADVANCES FROM A
C PIXEL FOR BOUNDARY RECONSTRUCTION AND MOMENT CALCULATION.

C
C INTEGER*1 D,DR,DC,OD

C IF(D.EQ.0) THEN
DR=0
DC=-1
OD=0
GOTO 779
ENDIF

C IF(D.EQ.1) THEN
DR=1
DC=-1
OD=1
GOTO 779
ENDIF

C IF(D.EQ.2) THEN
DR=1
DC=0
OD=0
GOTO 779
ENDIF

C IF(D.EQ.3) THEN
DR=1
DC=1
OD=1
GOTO 779
ENDIF

C IF(D.EQ.4) THEN
DR=0
DC=1
OD=0
GOTO 779
ENDIF

C IF(D.EQ.5) THEN
DR=-1
DC=1
OD=1
GOTO 779
ENDIF

C IF(D.EQ.6) THEN
DR=-1
DC=0
OD=0
GOTO 779
ENDIF

C
 IF(D.EQ.7) THEN
   DR=-1
   DC=-1
   OD=1
 ENDIF

C
  779 RETURN
 END

C
SUBROUTINE DrSp(p)
C
C THIS SUBROUTINE DISPLAYS AN IMAGE ON THE MONITOR FOR
C EITHER A GRAY LEVEL ONE, A BINARY ONE OR BOUNDARIES.
C
INTEGER A,B,E
INTEGER*1 P(256,320),L
INTEGER*2 I,J
C
WRITE(6,*) ' DISPLAYING.'
C******************************************************c
C SET UP FRAME GRABBER FOR DISPLAYING
C******************************************************c
A=#4000H
B=#00H
DO 41 L=1,7
   E=A+L
   CALL OUTPUT(E,B)
41 CONTINUE
C
CALL OUTPUT(#4007H,#01H)
CALL OUTPUT(#4003H,#d1H)
DO 42 L=1,20
   CALL OUTPUT(#4001H,#02H)
42 CONTINUE
C
C******************************************************c
C OUTPUT IMAGE TO FRAME GRABBER
C******************************************************c
DO 43 I=1,256
   DO 43 J=1,320
      CALL OUTPUT(#4000H,P(I,J))
43 CONTINUE
C
RETURN
END
C
SUBROUTINE DREAD(RONUM,RHNUM,RNAMES,ROIP,RHIP,ROCH,
$   RHCH,RORP,RHRP,BTC,MARGIN,SCALE,
$   STD,TH,SH,SL1,SL,BORDER)
C
C THIS SUBROUTINE READS DATA BASE.
C
INTEGER*1 RONUM,RHNUM,TH,SH,SL1,SL,L,L1
INTEGER*1 ROCH(19,300),RHCH(19,300)
INTEGER*2 ROIP(19,4),RHIP(19,4),BORDER(4),CHLEN,K1
REAL RORP(19,4),RHRP(19,4),BTC(4,4)
REAL MARGIN(4),SCALE,STD
CHARACTER*10 RNAMES(19)
C
WRITE(6,*),' READING DATA BASE.'
WRITE(8,*),' READING DATA BASE.'
C
OPEN(2,FILE='FUNG/PARAMS',RECL=80)
C**********************************************************************
C READ NUMBERS OF RECORDED OBJECTS AND HOLES
C**********************************************************************
READ(2,3201) RONUM,RHNUM
C**********************************************************************
C READ OBJECT NAMES AND OBJECT PARAMETERS
C**********************************************************************
IF(RONUM.GT.0) THEN
   DO 320 L=1,RONUM
      READ(2,3200) RNAMES(L)
      READ(2,3201) (ROIP(L,L1),L1=1,4)
      CHLEN=ROIP(L,3)
      READ(2,3203) (ROCH(L,K1),K1=1,CHLEN)
      READ(2,3202) (RORP(L,L1),L1=1,4)
   320 CONTINUE
ENDIF
C**********************************************************************
C READ HOLE PARAMETERS
C**********************************************************************
IF(RHNUM.GT.0) THEN
   DO 321 L=1,RHNUM
      READ(2,3201) (RHIP(L,L1),L1=1,4)
      CHLEN=RHIP(L,3)
      READ(2,3203) (RHCH(L,K1),K1=1,CHLEN)
      READ(2,3202) (RHRP(L,L1),L1=1,4)
   321 CONTINUE
ENDIF
C******************************RE**ND**ATION CONTROL PARAMETERS******************************
DO 322 L=1,4
   READ(2,3202) (BTC(L,L1),L1=1,4)
   CONTINUE
READ(2,3202) SCALE,STD
READ(2,3202) (MARGIN(L),L=1,4)
READ(2,3201) TH,SH,SL1,SL
READ(2,3201) (ORDER(L),L=1,4)
CLOSE(2)
WRITE(6,*) 'DATA BASE IS LOADED.'
WRITE(8,*) 'DATA BASE IS LOADED.'
3200 FORMAT(A10)
3201 FORMAT(4I10)
3202 FORMAT(4F15.5)
3203 FORMAT(6011)
C******************************BACK UP DATA FILE******************************
WRITE(6,*) 'MAKING A BACKUP.'
WRITE(8,*) 'MAKING A BACKUP.'
CALL DSAVE(RONUM,RHNUM,RNAMES,ROIP,RHIP,RONCH,RHCH,$RORP,RHRP,BTC,MARGIN,SCALE,STD,TH,SH,SL1,SL,BORDER,0)
RETURN
END
SUBROUTINE DSAVE(RONUM,RHNUM,RNAMES,ROIP,RHIP,ROCH,
$  RHCH,RORP,RHRP,BTC,MARGIN,SCALE,
$  STD,TH,SH,SL1,SL,BORDER,CONT)

C THIS SUBROUTINE SAVES DATA BASE.

INTEGER*1 RONUM,RHNUM,TH,SH,SL1,SL,L,L1,CONT
INTEGER*1 ROCH(19,300),RHCH(19,300)
INTEGER*2 ROIP(19,4),RHIP(19,4),BORDER(4),CHLEN,K1
REAL RORP(19,4),RHRP(19,4),BTC(4,4)
REAL MARGIN(4),SCALE,STD
CHARACTER*10 RNAMES(19)

WRITE(6,*) 'SAVING DATA BASE.'
WRITE(8,*) 'SAVING DATA BASE.'

C******************************************************
C OPEN DATA FILE OR BACKUP FILE
C******************************************************
IF(CONT.EQ.0) THEN
  OPEN(2,FILE='FUNG/PARAMS.OLD',RECL=80)
ELSE
  OPEN(2,FILE='FUNG/PARAMS',RECL=80)
ENDIF

C******************************************************
C SAVE NUMBERS OF RECORDED OBJECTS AND HOLES
C******************************************************
WRITE(2,3501) RONUM,RHNUM

C******************************************************
C SAVE OBJECT NAMES AND OBJECT PARAMETERS
C******************************************************
IF(RONUM.GT.0) THEN
  DO 350 L=1,RONUM
    WRITE(2,3500) RNAMES(L)
    WRITE(2,3501) (ROIP(L,L1),L1=1,4)
    CHLEN=ROIP(L,3)
    WRITE(2,3503) (ROCH(L,K1),K1=1,CHLEN)
    WRITE(2,3502) (RORP(L,L1),L1=1,4)
  CONTINUE
ENDIF

C******************************************************
C SAVE HOLE PARAMETERS
C******************************************************
IF(RHNUM.GT.0) THEN
  DO 351 L=1,RHNUM
    WRITE(2,3501) (RHIP(L,L1),L1=1,4)
    CHLEN=RHIP(L,3)
    WRITE(2,3503) (RHCH(L,K1),K1=1,CHLEN)
    WRITE(2,3502) (RHRP(L,L1),L1=1,4)
  CONTINUE
ENDIF
CONTINUE ENDIF ENDIF

C******************************************************************************C
C SAVE OPERATION CONTROL PARAMETERS C******************************************************************************C
DO 352 L=1,4
  WRITE(2,3502) (BTC(L,L1),L1=1,4)
  CONTINUE
WRITE(2,3502) SCALE,STD
WRITE(2,3502) (MARGIN(L),L=1,4)
WRITE(2,3501) TH,SH,SL1,SL
WRITE(2,3501) (BORDER(L),L=1,4)

CLOSE(2)

3500 FORMAT(A10)
3501 FORMAT(4I10)
3502 FORMAT(4F15.5)
3503 FORMAT(60I1)

WRITE(6,*) ' DATA BASE IS SAVED.'
WRITE(8,*) ' DATA BASE IS SAVED.'

RETURN
END
SUBROUTINE LINE(P,ORP,NUM)

C THIS SUBROUTINE DRAW A VECTOR ALONG THE DIRECTION OF THE
C OBJECT ORIENTATION ANGLE, BEGINNING FROM OBJECT CENTROID.
C
INTEGER*1 P(256,320),NUM,L,L1
INTEGER*2 RO,CO,R,C
REAL ORP(9,10),ORIENT
C
L1=INT(0.25*ORP(NUM,2)) ;VECTOR LENGTH
RO=INT(ORP(NUM,8)) ;START LOCATION
CO=INT(ORP(NUM,9))
P(RO,CO)=5
ORIENT=ORP(NUM,10)*3.14159/180.0 ;DIRECTION ANGLE
C
C********************************************************************
C DRAWING THE VECTOR
C********************************************************************
DO 810 L=1,L1
   R=RO+INT(L*1.4*COS(ORIENT))
   C=CO+INT(L*1.4*SIN(ORIENT))
   P(R,C)=5
810 CONTINUE
C
RETURN
END
SUBROUTINE MATCH1(NUM,OIP,ORP,HNUM,HIP,HRP,RNUM,RHNUM,$
ROIP,RORP,RHNUM,RHIP,HRHP,MARGIN,MATCH)

C THIS SUBROUTINE MATCHES AN ON-SCENE OBJECT TO THE
C RECORDED OBJECTS, TO SEE IF THE SELECTED OBJECT IS A
C NEW ONE.
C
INTEGER*1 NUM,HNUM,RNUM,RHNUM
INTEGER*1 MATCH,L,L1,L2,L3,L4,L5
INTEGER*2 OIP(9,4),HIP(9,4),ROIP(19,4),RHIP(19,4)
REAL ORP(9,10),HRP(9,10),RORP(19,4),RHRP(19,4)
REAL OLIM1(4),OLIM2(4),HLIM1(4),HLIM2(4),MARGIN(4)

WRITE(6,'(A)')' MATCHING AN ON-SCENE OBJECT',
$ ' TO THE RECORDED OBJECTS.'
WRITE(8,*)' MATCHING AN ON-SCENE OBJECT',
$ ' TO THE RECORDED OBJECTS.'

MATCH=0

C*******************************************************************************
C SET LOWER & UPPER LIMITS FOR OBJECT INVARIANTS
C*******************************************************************************
DO 60 L=1,4
OLIM1(L)=ORP(NUM,L)*(1-MARGIN(L))
OLIM2(L)=ORP(NUM,L)*(1+MARGIN(L))
CONTINUE
WRITE(8,6001) 'OUTER:',(ORP(NUM,L),L=1,4)
WRITE(8,6001) 'OLIM1:',(OLIM1(L),L=1,4)
WRITE(8,6001) 'OLIM2:',(OLIM2(L),L=1,4)
WRITE(8,*)' NUMBER OF HOLES:',OIP(NUM,4)

C*******************************************************************************
C OBJECT INVARIANT COMPARISON
C*******************************************************************************
WRITE(8,*)' BEGIN OUTER COMPARE:
DO 61 L=1,RNUM
WRITE(8,*)' NUMBER OF HOLES:',ROIP(L,4)
IF(ROIP(L,4).NE.OIP(NUM,4))GOTO 61
WRITE(8,6001) 'OUTER:',(RORP(L,L1),L1=1,4)
DO 62 L1=1,4
IF(RORP(L,L1).LT.OLIM1(L1)).OR.
RORP(L,L1).GT.OLIM2(L1))GOTO 61
CONTINUE
MATCH=-L

C*******************************************************************************
C HOLE MATCH IF THERE IS ANY
C*******************************************************************************
IF(OIP(NUM,4).GT.0) THEN
L4=0
L5=1
DO 63 L1=1,HNUM
   IF(HIP(L1,4).EQ.NUM) THEN ;FIND ON-SCENE HOLE
      WRITE(8,*) ' FIND HOLE:',L1
   CONTINUE
C**********************************************************************C
C SET LOWER & UPPER LIMITS HOLE INVARIANTS
C**********************************************************************C
   DO 64 L2=1,4
      HLIM1(L2)=HRP(L1,L2)*(1-MARGIN(L2))
      HLIM2(L2)=HRP(L1,L2)*(1+MARGIN(L2))
   CONTINUE
   WRITE(8,6001) ' HOLE: ',(HRP(L1,L2),L2=1,4)
   WRITE(8,6001) ' HLIM1: ',(HLIM1(L2),L2=1,4)
   WRITE(8,6001) ' HLIM2: ',(HLIM2(L2),L2=1,4)
   DO 65 L2=L5,RHNUM
      IF(RHIP(L2,4).EQ.L) THEN ;FIND RECORDED HOLE
         WRITE(8,*) ' BEGIN HOLE COMPARISON: ',L2
      CONTINUE
C**********************************************************************C
C HOLE INVARIANT COMPARISON
C**********************************************************************C
      WRITE(8,6001) ' HOLE: ',(RHRP(L2,L3),L3=1,4)
      D(J 66 L3:: 1 , 4
      IF(RHRP(L2,L3)_LT'HLIM1(L3)_OR.
         RHRP(L2,L3).GT.HLIM2(L3)) GOTO 61
      CONTINUE
      L4=L4+1
      IF(L4.GE.OIP(NUM,4)) THEN ;NO MORE HOLE
         MATCH=L ;COMPLETE MATCH
         GOTO 69
      ELSE
         L5=L2+1
         GOTO 63
      ENDIF
   ENDIF
   CONTINUE
63    CONTINUE
65    CONTINUE
66    CONTINUE
   ELSE
      MATCH=L
      GOTO 69
   ENDIF
C
61    CONTINUE
C
6001 FORMAT(A8,1X,4F10.2)
C
69    RETURN
END
SUBROUTINE MATCH2(RNUM,ROIP,RORP,RNUM,RHIP,RHRP,
$ ONUM,OIP,ORP,HNUM,HIP,HRP,MARGIN,MATCH)
C
C THIS SUBROUTINE MATCHES A RECORDED OBJECT TO THE ON-SCENE
C OBJECTS, TO SEE IF THE SELECTED OBJECT IS IN THE VIEWING
C WINDOW OF THE CAMERA.
C
INTEGER*1 RNUM,RHNUM,ONUM,HNUM
INTEGER*1 MATCH,L,L1,L2,L3,L4,L5
INTEGER*2 OIP(9,4),HIP(9,4),ROIP(19,4),RHIP(19,4)
REAL ORP(9,10),HRP(9,10),RORP(19,4),RHRP(19,4)
REAL OLIM1(4),OLIM2(4),HLIM1(4),HLIM2(4),MARGIN(4)
C
WRITE(6,*) , MATCHING A RECORDED OBJECT',
$ , TO THE ON SCENE OBJECTS.'
WRITE(8,*) , MATCHING A RECORDED OBJECT',
$ , TO THE ON SCENE OBJECTS.'
C
MATCH=0
C**********************************************************************C
C SET LOWER & UPPER LIMITS OBJECT INVARIANTS  C
C**********************************************************************C
DO 70 L=1,4
   OLIM1(L)=RORP(RNUM,L)*(1-MARGIN(L))
   OLIM2(L)=RORP(RNUM,L)*(1+MARGIN(L))
70 CONTINUE
WRITE(8,7001) , OUTER:' ,(RORP(RNUM,L),L=1,4)
WRITE(8,7001) , OLIM1:' ,(OLIM1(L),L=1,4)
WRITE(8,7001) , OLIM2:' ,(OLIM2(L),L=1,4)
WRITE(8,*) , NUMBER OF HOLES:' ,ROIP(RNUM,4)
C**********************************************************************C
C OBJECT INVARIANT COMPARISON  C
C**********************************************************************C
WRITE(8,*) , BEGIN OUTER COMPARE:'
DO 71 L=1,ONUM
   WRITE(8,*) , NUMBER OF HOLES:' ,OIP(L,4)
   IF(OIP(L,4).NE.ROIP(RNUM,4)) GOTO 71
   WRITE(8,7001) , OUTER:' ,(ORP(L,L1),L1=1,4)
   DO 72 L1=1,4
      IF(ORP(L,L1).LT.OLIM1(L1).OR.
          ORP(L,L1).GT.OLIM2(L1)) GOTO 71
   72 CONTINUE
   MATCH=-L ;PRIMARY MATCH
C**********************************************************************C
C HOLE MATCH IF THERE IS ANY  C
C**********************************************************************C
IF(ROIP(RNUM,4).NE.O) THEN
   L4=0
   L5=1
   DO 73 L1=1,RHNUM
IF(RHIP(L1,4).EQ.RNUM) THEN ;FIND RECORDED HOLE
WRITE(8,*), 'FIND HOLE:', L1
C********************************************************************C
C SET LOWER & UPPER LIMITS FOR HOLE INVARIANTS C
C********************************************************************C
DO 74 L2=1,4
   HLIM1(L2)=RHRP(L1,L2)*(1-MARGIN(L2))
   HLIM2(L2)=RHRP(L1,L2)*(1+MARGIN(L2))
CONTINUE
WRITE(8,7001), 'HOLE:', (RHRP(L1,L2), L2=1,4)
WRITE(8,7001), 'HLIM1:', (HLIM1(L2), L2=1,4)
WRITE(8,7001), 'HLIM2:', (HLIM2(L2), L2=1,4)
DO 75 L2=L5,HNUM
  IF(HIP(L2,4).EQ.L) THEN ;FIND ON-SCENE HOLE
    WRITE(8,*), 'BEGIN HOLE COMPARISON:', L2
C**********************************************************C
C HOLE INVARIANT MATCH C
C**********************************************************C
  WRITE(8,7001), 'HOLE:', (HRP(L2,L3), L3=1,4)
  DO 76 L3=1,4
    IF(HRP(L2,L3).LT.HLIM1(L3).OR.$
       HRP(L2,L3).GT.HLIM2(L3)) GOTO 71
  CONTINUE
  L4=L4+1
  IF(L4.GE.ROIP(RNUM,4)) THEN ;NO MORE HOLE
    MATCH=L ;COMPLETE MATCH
    GOTO 79
  ELSE ;MORE HOLE
    L5=L2+1
    GOTO 73
  ENDFD
ENDF
ELSE
  MATCH=L ;COMPLETE MATCH
  GOTO 79
ENDIF
ENDF
CONTINUE
71 CONTINUE
C
7001 FORMAT(A8,1X,4F10.2)
C
79 RETURN
END
SUBROUTINE MATCH3(RONUM,RNAMES,NAME,NUMMAT,NMATCH)

C THIS SUBROUTINE MATCHES A NEW NAME TO THE RRCORDED NAMES. 
C THERE MAYBE MORE THAN ONE MATCH. THAT MEANS THERE WERE 
C MORE THAN ONE VIEW OF THE OBJECT HAD BEEN RECORDED. 
C
INTEGER*1 RONUM,NUMMAT,NMATC(6),L 
CHARACTER*10 RNAMES(19),NAME 

NUMMAT=0 
DO 80 L=1,RONUM 
   IF(NAME.EQ.RNAMES(L)) THEN 
      NUMMAT=NUMMAT+1 ; ATOL NUMBER OF MATCHES 
      NMATCH(NUMMAT)=L ; SERIAL NUMBERS OF MATCHED NAMES 
   ENDIF 
80 CONTINUE 
C 
RETURN 
END
C
SUBROUTINE MOVE(P, MATCH, HNUM, DIP, HIP, OCH, HCH, ORP, BTC)

C
C THIS SUBROUTINE TRANSFORMS THE OBJECT POSITION FROM
C CAMERA COORDINATES TO ROBOT-BASE COORDINATES.
C
INTEGER*1 P(256, 320), MATCH, HNUM, L, L1
INTEGER*1 OCH(9, 300), HCH(9, 300)
INTEGER*2 DIP(9, 4), HIP(9, 4), RO, CO
REAL ORP(9, 10), BTO(4, 4), BTC(4, 4), CTO(4, 4)
REAL XBASE, YBASE, ZBASE, ORIENT, BASE

C******************************************************
C PREPARATION C
C******************************************************

BASE = 0.0
DO 220 L = 1, 4
   DO 220 L1 = 1, 4
      IF (L .NE. L1) THEN
         CTO(L, L1) = 0.0
      ELSE
         CTO(L, L1) = 1.0
      ENDIF
   220 CONTINUE

C******************************************************
C CONSTRUCTING MATRIX OF OBJECT IN CAMERA COORDINATES C
C******************************************************

WRITE(8,*) ' OBJECT IN IMAGE FRAME: X, Y, ORIENT'
WRITE(8, 2200) (ORP(MATCH, L), L = 8, 10)
ORIENT = ORP(MATCH, 10) * 3.14159 / 180.0
CTO(1, 1) = COS(ORIENT)
CTO(2, 2) = CTO(1, 1)
CTO(2, 1) = SIN(ORIENT)
CTO(1, 2) = -CTO(2, 1)
CTO(1, 4) = ORP(MATCH, 8)
CTO(2, 4) = ORP(MATCH, 9)
WRITE(8,*) ' CTO ='
DO 221 L = 1, 4
   221 WRITE(8, 2200) (CTO(L, L1), L1 = 1, 4)

C******************************************************
C OBJECT POSITION IN ROBOT-BASE COORDINATES C
C******************************************************

CALL MULTI(BTC, CTO, BTO)
WRITE(8,*) ' BTC ='
DO 222 L = 1, 4
   222 WRITE(8, 2200) (BTC(L, L1), L1 = 1, 4)
   WRITE(8,*) ' BTO ='
   DO 223 L = 1, 4
      223 WRITE(8, 2200) (BTO(L, L1), L1 = 1, 4)
   C
ORIENT=180.0/3.14159*ATAN(BTO(2,1)/BTO(1,1))
XBASE=BTO(1,4)
YBASE=BTO(2,4)
ZBASE=BTO(3,4)

WRITE(6,*) ' THE OBJECT LOCATES AND ORIENTS AT:'
WRITE(6,2200) XBASE,YBASE,ZBASE,BASE,BASE,ORIENT
WRITE(8,*) ' THE OBJECT LOCATES AND ORIENTS AT:'
WRITE(8,2200) XBASE,YBASE,ZBASE,BASE,BASE,ORIENT

C**************************************************************
C DISPLAYING THE OBJECT
C**************************************************************
WRITE(6,*) ' DISPLAY THE OBJECT.'
CALL CLEAR(P)
RO=INT(ORP(MATCH,8))
CO=INT(ORP(MATCH,9))
CALL BNDYOS(P,MATCH,HNUM,OIP,HIP,OCH,HCH,RO,CO)
CALL LINE(P,ORP,MATCH)
CALL DISP(P)

2200 FORMAT( ' ',6F10.2)

RETURN
END
SUBROUTINE MULTI(MATRX1,MATRX2,MATRX3)

C THIS SUBROUTINE MULTIPLIES TWO MATRIXES AND OUTPUT
C THE PRODUCT.

C REAL MATRX1(4,4), MATRX2(4,4), MATRX3(4,4),SUM
C INTEGER*1 L1,L2,L3

DO 30 L1=1,4
  DO 31 L2=1,4
    SUM=0
    DO 32 L3=1,4
      SUM=SUM+MATRX1(L1,L3)*MATRX2(L3,L2)
    32 CONTINUE
    MATRX3(L1,L2)=SUM
  31 CONTINUE
  30 CONTINUE

C RETURN
C END
SUBROUTINE ONE(MARGIN, NUM, HNUM, OIP, HIP, OCH, HCH, ORP, $ ARP, HRP, RNAMES, RONUM, RHNUM, ROIP, RHIP, ROCH, RHCH, $ RORP, RHRP)
C
C THIS SUBROUTINE RECORDS ONE SPECIFIED OBJECT
C FROM THE SCENE TO THE DATA BASE.
C
INTEGER*1 P(256,320), OCH(9,300), HCH(9,300)
INTEGER*1 ROCH(19,300), RHCH(19,300), NMATCH(6), MATCH
INTEGER*1 ROCH(9,300), RHCH(9,300), NUM, HNUM, RONUM, RHNUM, L, L1, L2, CHOICE, NUMMAT
INTEGER*2 OIP(9,4), HIP(9,4), ROIP(19,4), RHIP(19,4)
INTEGER*2 RO, CO, CHLEN, K1
REAL ARP(9,10), ORP(9,10), HRP(9,10)
REAL RORP(19,4), RHRP(19,4), MARGIN(4)
CHARACTER*1 YN
CHARACTER*10 NAME, RNAMES(19)
C
C******************************************************
C
C DISPLAYING THE SELECTED OBJECT
C
C******************************************************
WRITE(6,*) , DISPLAY THE OBJECT ON GRAPHIC MONITOR'
WRITE(8,*) , DISPLAY THE OBJECT ON GRAPHIC MONITOR'
CALL CLEAR(P)
RO=INT(ORP(NUM,8))
CO=INT(ORP(NUM,9))
CALL BNDYOS(P, NUM, HNUM, OIP, HIP, OCH, HCH, RO, CO)
CALL DISP(P)
WRITE(6,*) , DO YOU MEAN THIS ONE? (Y/N)'
WRITE(8,*) , DO YOU MEAN THIS ONE? (Y/N)'
READ(5,5001) YN
WRITE(8,5001) YN
IF(YN.EQ.'N') GOTO 509
C
CALL MATCH1(NUM, OIP, ORP, HNUM, HIP, HRP, RONUM, ROIP, $ RORP, RHNUM, RHIP, RHRP, MARGIN, MATCH)
C******************************************************
C
C NOT A NEW OBJECT; UPDATE DATA
C
C******************************************************
IF(MATCH.NE.0) THEN
  IF(MATCH.LT.0) THEN ; PRIMARY MATCHED
    MATCH=-MATCH
    WRITE(6,5003) RNAMES(MATCH)
    WRITE(8,5003) RNAMES(MATCH)
    CALL BOTH(P, NUM, HNUM, OIP, HIP, OCH, HCH, $ MATCH, RHNUM, ROIP, RHIP, ROCH, RHCH)
    WRITE(6,*) , ARE THEY LOOKED SIMILAR? (Y/N)'
    WRITE(8,*) , ARE THEY LOOKED SIMILAR? (Y/N)'
  ELSE ; COMPLETE MATCHED
    WRITE(6,5000) RNAMES(MATCH)
    WRITE(8,5000) RNAMES(MATCH)
    READ(5,5001) YN
  END IF
ENDIF
WRITE(8,5001) YN

IF(YN.EQ.'Y') GOTO 502
IF(YN.EQ.'N') THEN
   CALL BOTH(P,NUM,HNUM,OIP,CHIP,OCHE,HCH,
   $ MATCH,RHNUM,RHIP,ROCH,RHCH)
   WRITE(6,*) ' ARE THEY LOOKED THE SAME? (Y/N)'
   WRITE(8,*) ' ARE THEY LOOKED THE SAME? (Y/N)'
ENDIF
ENDIF

READ(5,5001) YN
WRITE(8,5001) YN
IF(YN.EQ.'N') THEN
   WRITE(6,*) ' OBJECTS CAN NOT BE DISTINGUISHED.'
   WRITE(8,*) ' OBJECTS CAN NOT BE DISTINGUISHED.'
GOTO 509
ENDIF

502 WRITE(6,*) ' ENTER (Y) TO UPDATE DATA,','
$ OR ELSE TO PASS.'
WRITE(8,*) ' ENTER (Y) TO UPDATE DATA,','
$ OR ELSE TO PASS.'
READ(5,5001) YN
WRITE(8,5001) YN
IF(YN.EQ.'Y') CALL UPDATE(NUM,ORP,MATCH,RORP,
$ OIP,HNUM,CHIP,HRP,RHNUM,RHIP,RHRP)
GOTO 509
ELSE
C******************************************************
C UNKNOWN OBJECT; RECORD OBJECT OR UPDATE DATA       C
C******************************************************
WRITE(6,*) ' THIS IS A NEW OBJECT.'
WRITE(8,*) ' THIS IS A NEW OBJECT.'
WRITE(6,*) ' ENTER THE NAME FOR THIS OBJECT:'
WRITE(8,*) ' ENTER THE NAME FOR THIS OBJECT:'
508 READ(5,5002) NAME
WRITE(8,5002) NAME
CALL MATCH3(RONUM,RNAMES,NAME,NUMMAT,NMATCH)
C******************************************************
C OPTIONS FOR A KNOWN NAME                             C
C******************************************************
IF(NUMMAT.NE.0) THEN
   WRITE(6,*) ' ONE OBJECT ALREADY HAS THIS NAME.'
   WRITE(8,*) ' ONE OBJECT ALREADY HAS THIS NAME.'
   WRITE(6,5004) NUMMAT
   WRITE(8,5004) NUMMAT
   DO 501 L=1,NUMMAT
      MATCH=NMATCH(L)
      CALL BOTH(P,NUM,HNUM,OIP,CHIP,OCHE,HCH,
MATCH,RHNUM,RHIP,ROCH,RHCH)
WRITE(6,*) ' IS THIS OBJECT'
WRITE(6,*) ' 1. ACTUALLY A NEW OBJECT,'
WRITE(6,*) ' 2. DIFFERENT VIEW OF SAME OBJECT,'
WRITE(6,*) ' 3. SAME VIEW OF SAME OBJECT,'
WRITE(6,*) ' 4. DISPLAY MORE VIEWS,'
WRITE(6,*) ' 9. QUIT,'
WRITE(8,*) ' IS THIS OBJECT'
WRITE(8,*) ' 1. ACTUALLY A NEW OBJECT,'
WRITE(8,*) ' 2. DIFFERENT VIEW OF SAME OBJECT,'
WRITE(8,*) ' 3. SAME VIEW OF SAME OBJECT,'
WRITE(8,*) ' 4. DISPLAY MORE VIEWS,'
WRITE(8,*) ' 9. QUIT,'
READ(5,*) CHOICE
WRITE(8,*) CHOICE
IF(CHOICE.NE.4) GOTO 511
501 CONTINUE
WRITE(6,*)' NO MORE. ENTER CHOICE 1, 2, 3, OR 9.'
WRITE(8,*)' NO MORE. ENTER CHOICE 1, 2, 3, OR 9.'
READ(5,*) CHOICE
WRITE(8,*) CHOICE
C 511 IF(CHOICE.EQ.1) THEN
  WRITE(6,*)' GIVE ANOTHER NAME:'
  WRITE(8,*)' GIVE ANOTHER NAME:'
  GOTO 508
ENDIF
C IF(CHOICE.EQ.3) THEN
  WRITE(6,*)' ENTER (Y) TO UPDATE DATA,',
  WRITE(6,*)' OR ELSE TO PASS.'
  WRITE(8,*)' ENTER (Y) TO UPDATE DATA,',
  WRITE(8,*)' OR ELSE TO PASS.'
  READ(5,5001) YN
  WRITE(8,5001) YN
$ IF(YN.EQ.'y') CALL UPDATE(NUM,ORP,MATCH,RORP,
  OIP,HNUM,HIP,HRP,RHNUM,HRH,HRP)
$  GOTO 509
ENDIF
C IF(CHOICE.NE.2) GOTO 509
WRITE(6,*)' RECORD A NEW VIEW OF THE OBJECT.'
WRITE(8,*)' RECORD A NEW VIEW OF THE OBJECT.'
ENDIF
C**********************************************************************
C RECORD OBJECT IF THERE IS AVAILABLE SPACE
C**********************************************************************
510 IF(RONUM.GE.10) THEN
  WRITE(6,*)' NO MORE MEMORY FOR OBJECTS.'
  WRITE(8,*)' NO MORE MEMORY FOR OBJECTS.'
  GOTO 509
ENDIF
RONUM=RONUM+1
RNAMES(RONUM)=NAME
DO 503 L1=1,4
   RORP(RONUM,L1)=ORP(NUM,L1)
   ROIP(RONUM,L1)=OIP(NUM,L1)
503 CONTINUE
CHLEN=OIP(NUM,3)
DO 504 K1=1,CHLEN
   ROCH(RONUM,K1)=OCH(NUM,K1)
504 CONTINUE
C******************************************************************************
C RECORD HOLE IF THERE IS ANY HOLE & AVAILABLE SPACE
C******************************************************************************
IF(OIP(NUM,4).GT.0) THEN
   IF((RHNUM+OIP(NUM,4)).GT.10) THEN
      WRITE(6,*) ' NO MORE MEMORY FOR HOLES.'
      WRITE(8,*) ' NO MORE MEMORY FOR HOLES.'
      RONUM=RONUM-1
      GOTO 509
   ENDIF
L2=0
DO 505 L=1,RHNUM
   IF(HIP(L,4).EQ.NUM) THEN
      RHNUM=RHNUM+1
      DO 506 L1=1,4
         RHRP(RHNUM,L1)=HRP(L,L1)
         RHIP(RHNUM,L1)=HIP(L,L1)
      506 CONTINUE
      RHIP(RHNUM,4)=RONUM
      CHLEN=HIP(L,3)
      DO 507 K1=1,CHLEN
         RHCH(RHNUM,K1)=HCH(L,K1)
      507 CONTINUE
      L2=L2+1
      IF(L2.GE.OIP(NUM,4)) GOTO 509
   ENDIF
505 CONTINUE
ENDIF

C******************************************************************************
C 5000 FORMAT( ' THIS IS A KNOWN OBJECT, CALLED ',A10,
               '$', . CORRECT? (Y/N)', )
5001 FORMAT(A1)
5002 FORMAT(A10)
5003 FORMAT( ' ALMOST MATCH AN OBJECT CALLED ',A10,'.' )
5004 FORMAT(I2, ' VIEW(S) HAD BEEN RECORDED.' )
C
509 RETURN
END
SUBROUTINE OSREAD(ONUM,HNUM,OIP,HIP,
OCH,HCH,ORP,ARP,HRP)

C THIS SUBROUTINE READS ON-SCENE OBJECTS PARAMETERS.
C
INTEGER*1 ONUM,HNUM,OCH(9,300),HCH(9,300),L,L1
INTEGER*2 OIP(9,4),HIP(9,4),CHLEN,K1
REAL ARP(9,10),ORP(9,10),HRP(9,10)

OPEN(3,FILE='FUNG/OSOBS',RECL=80)
READ(3,3401) ONUM,HNUM

C***********************************************************************
C IF ANY OBJECT, READ THE OBJECT PARAMETERS
C***********************************************************************
IF(ONUM.GT.0) THEN
DO 340 L=1,ONUM
    READ(3,3401) (OIP(L,L1),L1=1,4)
    CHLEN=OIP(L,3)
    READ(3,3403) (OCH(L,K1),K1=1,CHLEN)
    READ(3,3402) (ARP(L,L1),L1=1,4)
    READ(3,3402) (ARP(L,L1),L1=8,10)
    READ(3,3402) (ORP(L,L1),L1=1,4)
    READ(3,3402) (ORP(L,L1),L1=8,10)
340 CONTINUE

C***********************************************************************
C IF ANY HOLE, READ THE HOLE PARAMETERS
C***********************************************************************
IF(HNUM.GT.0) THEN
DO 341 L=1,HNUM
    READ(3,3401) (HIP(L,L1),L1=1,4)
    CHLEN=HIP(L,3)
    READ(3,3403) (HCH(L,K1),K1=1,CHLEN)
    READ(3,3402) (HRP(L,L1),L1=1,4)
    READ(3,3402) (HRP(L,L1),L1=8,10)
341 CONTINUE
ENDIF
ENDIF
CLOSE(3)

3401 FORMAT(4I10)
3402 FORMAT(4F15.5)
3403 FORMAT(60I1)

RETURN
END
SUBROUTINE PUMA(BORDER, SL, TH, MARGIN, SCALE, BTC, RONUM, $ RHNUM, RNAMES, ROIP, RHIP, ROCH, RHCH, RORP, RHRP)

C THIS SUBROUTINE DETERMINES THE POSITION OF A SELECTED
C OBJECT IN ROBOT-BASE COORDINATES.

INTEGER*1 P(256, 320), OCH(9, 300), HCH(9, 300)
INTEGER*1 ROCH(19, 300), RHCH(19, 300)
INTEGER*1 TH, SL, RONUM, NUMMAT, NMATCH(6), MATCH
INTEGER*1 CHOICE, CHOIC1, NUM, RNUM, L, L1, L2, ONUM, HNUM
INTEGER*2 OIP(9, 4), HIP(9, 4), ROIP(19, 4), RHIP(19, 4)
INTEGER*2 BORDER(4)
REAL ORP(9, 10), ARP(9, 10), HRP(9, 10)
REAL RORP(19, 4), RHRP(19, 4), BTC(4, 4), MARGIN(4), SCALE

C******************************************************************************
C OPTIONS OF SELECTING AN OBJECT
C******************************************************************************

WRITE(6, *) 'SELECTING AN OBJECT',
WRITE(8, *) 'SELECTING AN OBJECT',
100 WRITE(6, *) 'CHOOSE AN OBJECT FROM',
$ 'MEMORY(1), SCENE(2), OR QUIT(9).',
WRITE(8, *) 'CHOOSE AN OBJECT FROM',
$ 'MEMORY(1), SCENE(2), OR QUIT(9).',
READ(5, *) CHOICE
WRITE(8, *) CHOICE

IF(CHOICE.EQ.1) THEN

C******************************************************************************
C WAYS TO SELECT AN OBJECT
C******************************************************************************

WRITE(6, *) 'PICKING OBJECT BY NAME(1), OR',
$ 'FROM LIST(2), OR QUIT(9).',
WRITE(8, *) 'PICKING OBJECT BY NAME(1), OR',
$ 'FROM LIST(2), OR QUIT(9).',
READ(5, *) CHOIC1
WRITE(8, *) CHOIC1
MATCH = 0

IF(CHOIC1.EQ.1) THEN

WRITE(6, *) 'ENTER NAME OF THE OBJECT:',
WRITE(8, *) 'ENTER NAME OF THE OBJECT:',
READ(5, 1002) NAME
WRITE(8, 1002) NAME
CALL MATCH3(RONUM, RNAMES, NAME, NUMMAT, NMATCH)

IF(NUMMAT.EQ.0) THEN

WRITE(6, *) 'SORRY, UNKNOWN OBJECT.,'
WRITE(8, *) 'SORRY, UNKNOWN OBJECT.,'
GOTO 100
ELSE
    CALL OSREAD(ONUM,HNUM,OIP,HIP,OCH,HCH,
        ORP,ARP,HRP)
ENDIF
C***************************************************************************C
C FINDING THE OBJECT IN CAMERA VIEWING WINDOW  C
C***************************************************************************C
DO 104 L=1,NUMMAT
    RNUM=NMATCH(L)
    CALL MATCH2(RNUM,ROIP,RORP,RHNUM,RHIP,RHRP,
        ONUM,OIP,ORP,HNUM,HIP,HRP,MARGIN,MATCH)
    IF(MATCH.NE.0) GOTO 105
104 CONTINUE
ENDIF
C
IF(CHOIC1.EQ.2) THEN
C***************************************************************************C
C LISTING ALL NAMES OF RECORDED OBJECTS  C
C***************************************************************************C
    L1=1
    L2=0
101   IF(L1.GT.RONUM) GOTO 102
    L2=L2+5
    IF(L2.GT.RONUM) L2=RONUM
    WRITE(6,1000) (L,'-',RNAMES(L),L=L1,L2)
    WRITE(8,1000) (L,'-',RNAMES(L),L=L1,L2)
    L1=L1+5
    GOTO 101
C
102   WRITE(6,*),'ENTER OBJECT NUMBER (0=QUIT):'
    WRITE(8,*),'ENTER OBJECT NUMBER (0=QUIT):'
    READ(5,*) RNUM
    WRITE(8,*),RNUM
    IF(RNUM.GT.0.AND.RNUM.LE.RONUM) THEN
        NAME=RNAMES(RNUM)
        CALL MATCH3(RNUM,RNAMES,NAME,NUMMAT,NMATCH)
        CALL OSREAD(ONUM,HNUM,OIP,HIP,OCH,HCH,
            ORP,ARP,HRP)
        C***************************************************************************C
        C FINDING THE OBJECT IN CAMERA VIEWING WINDOW  C
        C***************************************************************************C
        DO 106 L=1,NUMMAT
            RNUM=NMATCH(L)
            CALL MATCH2(RNUM,ROIP,RORP,RHNUM,RHIP,RHRP,
                ONUM,OIP,ORP,HNUM,HIP,HRP,MARGIN,MATCH)
            IF(MATCH.NE.0) GOTO 105
106     CONTINUE
    ENDIF
    GOTO 100
ENDIF
C
E-90
C******************************************************************************C
C OBJECT IDENTIFICATION
C******************************************************************************C
105 IF(MATCH.EQ.0) THEN ;NO MATCH
WRITE(6,*)' SORRY, NO SUCH OBJECT IS DETECTED.'
WRITE(8,*)' SORRY, NO SUCH OBJECT IS DETECTED.'
ELSE
IF(MATCH.LT.0) THEN ;PRIMARY MATCH ONLY
MATCH=-MATCH
WRITE(6,*)' NO EXACT MATCH. BUT'
WRITE(6,*)' THERE IS A SIMILAR ONE.'
WRITE(8,*)' NO EXACT MATCH. BUT'
WRITE(8,*)' THERE IS A SIMILAR ONE.'
CALL BOTH(P,MATCH,HNUM,OIP,HIP,OCH,HCH,
$   RNUM,RHNUM,ROIP,RHIP,ROCH,RHCH)
CALL MOVE(P,MATCH,HNUM,OIP,HIP,OCH,HCH,ORP,BTC)
CALL UPDATE(MATCH,ORP,RNUM,RORP,OIP,HNUM,
$   HIP,HRP,RHNUM,RHIP,RHRP)
ELSE ;COMPLETE MATCH
CALL MOVE(P,MATCH,HNUM,OIP,HIP,OCH,HCH,ORP,BTC)
CALL UPDATE(MATCH,ORP,RNUM,RORP,OIP,HNUM,
$   HIP,HRP,RHNUM,RHIP,RHRP)
ENDIF
ENDIF
GOTO 100
ENDIF
C
IF(CHOICE.EQ.2) THEN
CALL OSREAD(ONUM,HNUM,OIP,HIP,OCH,HCH,
$   ORP,ARP,HRP)
C******************************************************************************C
C SELECT OBJECT FROM CAMERA VIEWING WINDOW
C******************************************************************************C
WRITE(6,1001) ONUM
WRITE(8,1001) ONUM
WRITE(6,*) ' LOCATION: ROW COLUMN'
WRITE(8,*) ' LOCATION: ROW COLUMN'
DO 103 L=1,ONUM
   WRITE(6,1004) L,ORP(L,8),ORP(L,9)
   WRITE(8,1004) L,ORP(L,8),ORP(L,9)
103 CONTINUE
WRITE(6,*) ' ENTER THE NUMBER OF THE OBJECT.'
WRITE(8,*) ' ENTER THE NUMBER OF THE OBJECT.'
READ(5,*) CHOICE1
WRITE(8,*) CHOICE1
CALL MOVE(P,CHOICE1,HNUM,OIP,HIP,OCH,HCH,ORP,BTC)
GOTO 100
ENDIF
C
IF(CHOICE.NE.9) GOTO 100
C
1000 FORMAT(5(I1, A1, A10))
1001 FORMAT(’ THERE ARE TOTAL’ ,I2,’ OBJECTS ON THE’ , $ 
 ’ SCENE.’)
1002 FORMAT(A10)
1004 FORMAT(I5,2F15.2)
C
   RETURN
C
END
C
SUBROUTINE TRAIN(BORDER,MARGIN,SCALE,SL,TH,RONUM, 
$ \quad \text{RHNUM,RNAMES,ROIP,RHIP,ROCH,RORP,RHRP})$

C

C THIS SUBROUTINE PUTS NEW OBJECTS INTO DATA BASE.
C
INTEGER*1 P(256,320),TH,SL,OCH(9,300),HCH(9,300)
INTEGER*1 ONUM,HNUM,RONUM,RHNUM,CHOICE,L,NUM
INTEGER*1 ROCH(19,300),HCH(19,300)
INTEGER*2 OIP(9,4),HIP(9,4),ROIP(19,4),RHIP(19,4)
INTEGER*2 BORDER(4),RO,CO
REAL ARP(9,10),ORP(9,10),HRP(9,10),SCALE
REAL RORP(19,4),RHRP(19,4),MARGIN(4)
CHARACTER*10 RNAMES(19)

WRITE(6,*) 'START TRAINING,'
WRITE(8,*) 'START TRAINING,'

C***********************************************************************C
C INPUT OBJECT PARAMETERS AND DISPLAY BOUNDARIES C
C***********************************************************************C
CALL OSREAD(ONUM,HNUM,RONUM,RHNUM,CHOICE,L,NUM,
  OIP(9,4),HIP(9,4),ROIP(19,4),RHIP(19,4),
  BORDER(4),RO,CO,
  ARP(9,10),ORP(9,10),HRP(9,10),SCALE,
  REAL RORP(19,4),RHRP(19,4),MARGIN(4),
  CHARACTER*10 RNAMES(19))

WRITE(6,*) 'BOUNDARIES OF ON-SCENE OBJECTS,'
WRITE(8,*) 'BOUNDARIES OF ON-SCENE OBJECTS,'
CALL CLEAR(P)
DO 304 L=1,ONUM
  RO=INT(ORP(L,8))
  CO=INT(ORP(L,9))
  CALL BNDYOS(P,L,HNUM,OIP,HIP,OCH,HCH,RO,CO)
  CALL LINE(P,ORP,L)
304 CONTINUE
CALL DISP(P)

C***********************************************************************C
C SELECTING TRAINING OPTION C
C***********************************************************************C
WRITE(6,3000) ONUM,HNUM
WRITE(8,3000) ONUM,HNUM
WRITE(6,*) 'HOW TO PROCEED? 1-ONE BY ONE;',
$ '2-ALL OF THEM; 9-EXIT.'
WRITE(8,*) 'HOW TO PROCEED? 1-ONE BY ONE;',
$ '2-ALL OF THEM; 9-EXIT.'
READ(5,*) CHOICE
WRITE(8,*) CHOICE

IF(CHOICE.EQ.9) GOTO 309

IF(CHOICE.EQ.1) THEN
  WRITE(6,*) 'LOCATION: ROW COLUMN'
  WRITE(8,*) 'LOCATION: ROW COLUMN'
  DO 300 L=1,ONUM
    WRITE(6,3001) L,ORP(L,8),ORP(L,9)
    WRITE(8,3001) L,ORP(L,8),ORP(L,9)
 300 CONTINUE
CONTINUE
WRITE(6,*) ' ENTER OBJECT 1, OR 2, OR',
$ ' 3,...(O=QUIT)' WRITE(8,*) ' ENTER OBJECT 1, OR 2, OR',
$ ' 3,...(O=QUIT)' READ(5,*) NUM WRITE(8,*) NUM IF(NUM.EQ.0) GOTO 301 IF(NUM.GT.ONUM.OR.NUM.LT.O) GOTO 302 CALL ONE(MARGIN,NUM,HNUM,OIP,HIP,OCH,HCH,ORP,ARP,
$HRP,RNAMES,RONUM,RHNUM,ROIP,RHIP,ROCH,RHCH,RORP,RHRP) GOTO 301 ENDF

IF(CHOICE.EQ.2) THEN DO 303 L=1,NUM
  CALL ONE(MARGIN,L,HNUM,OIP,HIP,OCH,HCH,ORP,ARP,
$HRP,RNAMES,RONUM,RHNUM,ROIP,RHIP,ROCH,RHCH,RORP,RHRP)
  WRITE(6,*) ' NEXT OBJECT.'
  WRITE(8,*) ' NEXT OBJECT.'
303 CONTINUE WRITE(6,*) ' NO MORE.' WRITE(8,*) ' NO MORE.' ENDF

THERE ARE ',I3,' OBJECTS WITH ',I3,
$ ' HOLES ON THE SCENE.') FORMAT(I5,2F15.2) RETURN
SUBROUTINE UPDATE(NUM, ORP, RNUM, RORP, OIP, HNUM,
    HIP, HRP, RHNUM, RHIP, RHRP)
C
C THIS SUBROUTINE UPDATES THE PARAMETERS OF THE SPECIFIED
C OBJECT.
C
INTEGER*1 NUM, RNUM, HNUM, RHNUM, L, L1, L2
INTEGER*2 OIP(9, 4), HIP(9, 4), RHIP(19, 4)
REAL ORP(9, 10), HRP(9, 10), RORP(19, 4), RHRP(19, 4)
C
C*******************************************************************
C AVERAGING OBJECT INVARIANTS
C*******************************************************************
DO 520 L=1,4
    RORP(RNUM,L)=(RORP(RNUM,L)+ORP(NUM,L))/2.
520 CONTINUE
C
C*******************************************************************
C FINDING HOLES IF THERE IS ANY
C*******************************************************************
IF(OIP(NUM,4).GT.0) THEN
    DO 521 L=1,HNUM
        IF(HIP(L,4).EQ.NUM) GOTO 522 ;FIND ON-SCENE HOLE
521 CONTINUE
C
522 DO 523 L1=1,RHNUM
    IF(RHIP(L1,4).EQ.RNUM) GOTO 524 ;FIND RECORDED HOLE
523 CONTINUE
C
C*******************************************************************
C AVERAGING HOLE INVARIANTS
C*******************************************************************
524 DO 525 L2=1,4
    RHRP(L1,L2)=(RHRP(L1,L2)+HRP(L,L2))/2.
525 CONTINUE
ENDIF
C
RETURN
END
PROGRAM EYE
C
C THE MAIN PROGRAM OF THE ROBOT VISION SYSTEM.
C
INTEGER*1 RONUM, RHNUM, ROCH(19, 300), RHCH(19, 300)
INTEGER*1 CHOICE, TH, SH, SL1, SL
INTEGER*2 ROIP(19, 4), RHIP(19, 4), BORDER(4)
REAL RORP(19, 4), RHRP(19, 4)
REAL BTC(4, 4), MARGIN(4), SCALE, STD
CHARACTER*10 RNAMES(19)

C******************************************************
C OPEN OPERATION TRACE FILE & LOAD DATA BASE C
C******************************************************
WRITE(6,*) ', OPENING EYE.'
WRITE(8,*) ', OPENING EYE.'
OPEN(8, FILE='FUNG/TRACE', RECL=80)
CALL DREAD(RONUM, RHNUM, RNAMES, ROIP, RHIP, ROCH, RHCH,
$ RORP, RHRP, BTC, MARGIN, SCALE, STD, TH, SH, SL1, SL, BORDER)

C******************************************************
C DISPLAY OPERATION MENU C
C******************************************************
   WRITE(6,*) ', WHAT DO YOU WISH TO DO NEXT?'
   WRITE(6,*) ', 1. CAMERA CALIBRATION;'
   WRITE(6,*) ', 2. REVISE OPERATION PARAMETERS;'
   WRITE(6,*) ', 3. TRAINING;'
   WRITE(6,*) ', 4. POSITION PUMA TO AN OBJECT;'
   WRITE(6,*) ', 5. REARRANGE DATA BASE;'
   WRITE(6,*) ', 6. SAVE DATA BASE;'
   WRITE(6,*) ', 9. EXIT TO OPERATION SYSTEM;'
   WRITE(6,*) ', ENTER YOUR CHOICE:'
WRITE(8,*) ', WHAT DO YOU WISH TO DO NEXT?'
WRITE(8,*) ', 1. CAMERA CALIBRATION;'
WRITE(8,*) ', 2. REVISE OPERATION PARAMETERS;'
WRITE(8,*) ', 3. TRAINING;'
WRITE(8,*) ', 4. POSITION PUMA TO AN OBJECT;'
WRITE(8,*) ', 5. REARRANGE DATA BASE;'
WRITE(8,*) ', 6. SAVE DATA BASE;'
WRITE(8,*) ', 9. EXIT TO OPERATING SYSTEM;'
READ(5,*) CHOICE
WRITE(8,*) ', ENTER YOUR CHOICE:', CHOICE

C******************************************************
C SELECT OPERATION OPTION C
C******************************************************
IF(CHOICE.EQ.1) CALL CALI(BORDER, SL, TH, SH, SL1,
$ BTC, SCALE, STD)
IF(CHOICE.EQ.2) CALL CONTRO(BORDER, SL, SL1)
IF(CHOICE.EQ.3) THEN
CALL TRAIN(BORDER,MARGIN,SCALE,SL,TH,RONUM,RHNUM, $ RNAMES,ROIP,RHIP,ROCH,RHCH,RORP,RHRP)
ENDIF
IF(CHOICE.EQ.4) CALL PUMA(BORDER,SL,TH,MARGIN,SCALE, $BTC,RONUM,RHNUM,RNAMES,ROIP,RHIP,ROCH,RHCH,RORP,RHRP)
IF(CHOICE.EQ.7) CALL DBASE(RONUM,RHNUM,RNAMES,ROIP, $ RORP,RHRP,BTC,MARGIN,SCALE)
IF(CHOICE.EQ.8) THEN
  CALL DSAVE(RONUM,RHNUM,RNAMES,ROIP,RHIP,ROCH,RHCH, $ RORP,RHRP,BTC,MARGIN,SCALE,STD,TH,SH,SL1,SL,BORDER,1)
ENDIF
IF(CHOICE.NE.9) GOTO 1
C
WRITE(6,*) 'EYE CLOSED.'
WRITE(8,*) 'EYE CLOSED.'
C
STOP
END

SUBROUTINE PUMA
Change CALL OSREAD(......) to CALL PROCES(......).

SUBROUTINE TRAIN
Change CALL OSREAD(......) to CALL PROCES(......) and delete following lines:

CALL CLEAR(P)
DO 304 L=1,ONUM
  RO=INT(ORP(L,8))
  CO=INT(ORP(L,9))
  CALL BNDYOS(P,L,HNUM,OIP,HIP,OCH,HCH,RO,CO)
  CALL LINE(P,ORP,L)
304 CONTINUE
CALL DISP(P)