TECHNIQUES FOR THE RECOGNITION OF SILHOUETTES

by

© David W. Capson, B.Sc.Eng.

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree
Master of Engineering

McMaster University

1981

MASTER OF ENGINEERING (1981) McMaster University
(Electrical Engineering) Hamilton, Ontario

TITLE: Techniques for the Recognition of Silhouettes

AUTHOR: David W. Capson, B.Sc.Eng. (University of New Brunswick)

SUPERVISORS: Dr. R. Kitai, Department of Electrical and Computer Engineering
Dr. J. Tlusty, Department of Mechanical Engineering

NUMBER OF PAGES: ix, 160

ABSTRACT

A representative set of binary image processing techniques selected from the literature is described. The measurement of shape as the fundamental information contained in silhouettes is examined. Operations on digital binary images are demonstrated including smoothing, connectivity analysis and determination of position and orientation. The effects of digitizing errors at the boundary of a silhouette are discussed and examples of industrial vision systems which use binary images are presented.

A binary image processing system has been designed and implemented. The apparatus is based on a General Electric TN2500 digital television camera and an Intel iSBC 86/12A microcomputer. Hardware for the acquisition of binary images from the camera is described followed by the software for calculating areas and centroids. The system is capable of "learning" a set of objects in a "Teach" mode and then making an identification based on their area in the "Run" mode.

ACKNOWLEDGEMENTS

The author wishes to express his sincere thanks to his supervisors, Dr. R. Kitai and Dr. J. Tlusty for their support and guidance throughout the course of this work.

Also, financial assistance from the Natural Sciences and Engineering Research Council and from McMaster University is greatly appreciated.

Finally, a special thank you to my wife who knows the reasons why.

TABLE OF CONTENTS

LIST OF FIGURES

Figure		Page
4.1	Machine Vision Applications	55
4.2	Binary Decision Tree	59
4.3	CONSIGHT-I System Configuration	62
4.4	CONSIGHT-I Lighting Arrangement	63
4.5	Improved Lighting Arrangement	64
4.6	Line of Light Generation	64
4.7	Camera and Strobe Arrangement for NBS Vision System	67
4.8	Example Objects and the Line Segment Patterns Formed by the Plane of Light as Seen by the Camera	68
4.9	The Calibration Chart for the NBS Vision System	69
5.1	Binary Image Processing System	71
5.2	System Hardware Configuration	73
5.3	Priority Resolution Between Modules	74
5.4	CID Sensor Geometry	74
5.5	DMA Controller Software	79
5.6	Image Data Format in RAM	80
5.7	Camera/DMA Interface Module	82
5.8	Relative Timing of VSYNC & $\overline{\text{SBLNK}}$	84
5.9	Sequential Mode Frame Presentation	84
5.10	Threshold Logic	85
5.11	Silhouette Generated for Input to the System	87
5.12	System Software Configuration	89

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Image processing is a rapidly emerging discipline which is gaining acceptance as a versatile tool in science and technology. Interactive or automated picture processing with the intent to detect, localise or classify the contents of images is receiving considerable attention in the literature. The co-evolution of the microprocessor has encouraged widespread research and development in this area. A recent bibliography by Rosenfeld [1], the tenth of a series, lists over 700 published works covering the year 1979 alone.

The methodologies of image processing have ascended from experimental laboratory apparatus to custom-designed computer vision systems. Applications are appearing in diverse activities ranging from meteorology (where the objects of interest, clouds for example, may be kilometers in size) to medicine (where subjects such as cells are of the order of micrometers).

Of primary contention among developers of automated vision is the selection of binary or grey-scaled images. Grey-scaled data has the potential of handling complex scenes in which lighting conditions may vary uncontrollably. Alternatively, binary images (silhouettes) generally offer quicker processing and smaller storage requirements, but their range of suitability is somewhat restricted. However, with the use of novel lighting techniques, silhouettes have proven appropriate in inherently binary applications such as shape measurement, text processing, engineering drawings, weather maps, etc.

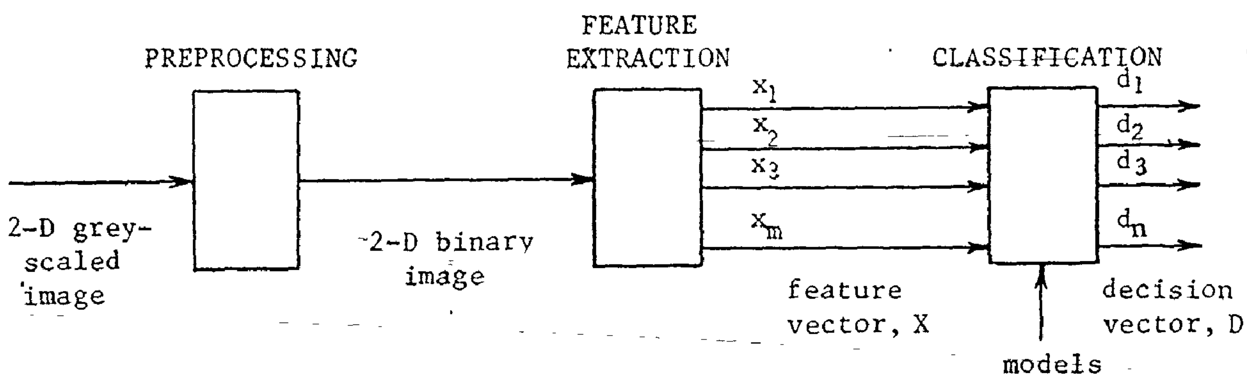


Figure 1.1 Silhouette Processing

Silhouette processing is graphically portrayed in fig. 1.1. The "preprocessing" device provides a conversion from the incoming grey-scaled image into a data structure containing the binary version of the image. A feature vector, X, is generated in the "feature extraction" phase and presented to the "classification" module. Based on the feature vector, and possibly some a priori models, a decision vector, D, is produced.

The feature vector represents various characteristics or traits of the image in question. The decision vector is designed to contain information for solution of the problem at hand.

With motivation from the marked increase of activity in picture processing, especially in the realm of automated manufacturing, this work presents a cross section of the strategies currently pursued in binary image processing. The objective is twofold; firstly to act as an introduction to the literature and secondly, to present the design and implementation of a binary image vision system.

Chapter 2 is a review of the quantitative study of shape and form. At this point, contours are assumed to be continuous, two-dimensional functions.

Chapter 3 deals with the manipulation of binary images after conversion to discrete two-dimensional arrays. Included is a discussion of the erroneous effects of digitization.

Chapter 4 describes representative examples of binary image processing schemes as are being applied to the industrial environment.

These chapters constitute the literature profile.

The design and implementation of a vision system is presented in Chapter 5. In its early stages, non-contact area measurements are taken from which various shapes are identified.

Chapter 6 completes the thesis with a discussion of conclusions and suggestions for topics of further investigations.

CHAPTER 2

SHAPE QUANTIZATION

2.1 Introduction

The quantization of shape is a primary problem in image processing and has received considerable attention in recent years. Unfortunately, shape is a very difficult characteristic to describe. Investigation of human shape perception has indicated that shape is a multi-dimensional quantity and it is, therefore, not surprising to find that a multitude of mathematical shape descriptors have been invented. Pavlidis has summarized a number of algorithms in [2].

Attneave and Arnoult produced one of the first publicized investigations in this area [3]. Their work resulted in the conclusion that the fundamental information which characterizes a shape for human recognition is contained in its perimeter and not within the enclosed area. Further, the greatest amount of information will be obtained from points of maximum change in the contour gradient (i.e. corners, vertices). Straight line segments and any circular sections of uniform curvature, therefore, represent redundant information. Some mathematical models for shape have made good use of these assertions.

In [4], Pavlidis has drawn a distinction between "external" (those concerned only with contours) and "internal" (those which include the enclosed area) shape descriptors. He also discriminates between "scalar transform" and "space domain" techniques. Scalar transforms extract an array of scalar features from a shape while space domain methods produce a new picture from the old.

To be a true function of shape, shape descriptors must be dimensionless and independent of scale or orientation. If they are related to specified ideal shapes (often circles), then they are designated as "shape factors". Finally, the analysis can be further subdivided into "information preserving" and information "non-preserving". Using a "non-preserving" shape descriptor, it will not be possible to reconstruct the original shape. Information preserving techniques permit the original shape to be reproduced.

The remainder of this chapter is devoted to a description of some of the more widely known shape descriptors. Further lists are given in Underwood [5] and more recently in Bookstein [6].

2.2 Information Non-Preserving Shape Descriptors

Each of the shape factors presented in this section are examples of the type "information non-preserving". They are useful individually for only a limited number of applications and clearly will be inappropriate for non-trivial shape measurement. Many widely different shapes can have identical shape factors. Performance can be improved however, by using various combinations of these descriptors.

P2A

A very popular shape factor, P2A, is defined as follows:

$$P2A = \frac{P^2}{4\pi A}$$

where, P = length of the perimeter
A = enclosed area

The factor $1/4\pi$ normalizes the quantity to unity for a circle and > 1 for other shapes. Thus, this ratio is sometimes referred to as the "circularity" of an object. Table 2.1 gives the P2A factor for some common figures. As each shape more closely approximates the circle, its corresponding shape factor approaches 1. Aside from its information non-preserving quality, there exists another drawback to this calculation. The quantity P2A is very sensitive to the resolution of the boundary. The perimeter squared can grow large as increasingly better resolution is introduced.

G

A recently proposed shape factor [7] has eliminated the problem of border resolution sensitivity. The method is based on the average distance between points in the object and its nearest border point. The factor "G" is given by:

$$G = \frac{A}{(d)^2}$$

where, A = enclosed area

$$d = \left[\iint_A x \, dA\right] / A$$

x = shortest distance from dA to border

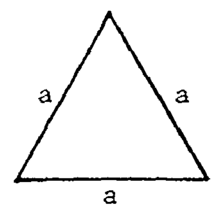
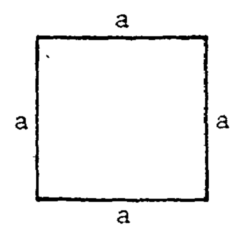
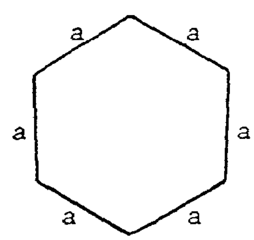
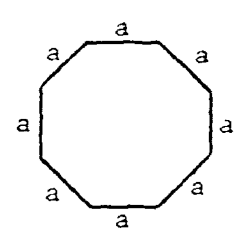
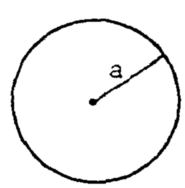
Shape		P2A Shape Factor
Equilateral Triangle		$\frac{3\sqrt{3}}{\pi} = 1.65$
Square		$\frac{4}{\pi} = 1.27$
Regular Hexagon		$\frac{6}{\sqrt{3}\pi} = 1.10$
Regular Octagon		$\frac{8}{(1+\sqrt{2})\pi} = 1.05$
Circle		1

Table 2.1 P2A Shape Factor for Some Common Figures

For a circle of radius "r",

$$d = \left[\iint_A dr\ dA\right] / \pi r^2$$

$$= [\iint dr\ 2\pi r\ dr]/\pi r^2 \qquad (dA = 2\pi r\ dr)$$

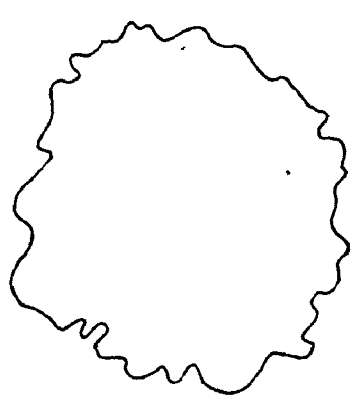
$$= \frac{1}{3} r$$

$$\therefore \quad G_o = \frac{\pi r^2}{(\frac{1}{3} r)^2} = 9\pi$$

The quantity G, normalized to unity for the circle is given by:

$$G = \frac{A}{9\pi\ (d)^2}$$

For all regular polygons, the factor G and P2A are identical. For assymetrical Figures, G and P2A differ, but G is insensitive to resolution. Danielsson gives figure 2.1 as an example:



P2A = 2.51

G = 1.56

Figure 2.1 Arbitrary Shape with Large Number of Boundary Changes

The shape factor G is an indication of "compactness". Clearly, the resemblance of figure 2.1 to a circle is better indicated by G than by P2A.

$\underline{\varepsilon_o}$

The eccentricity, ε_o, of a shape is described by Bribiesca and Guzman [8]:

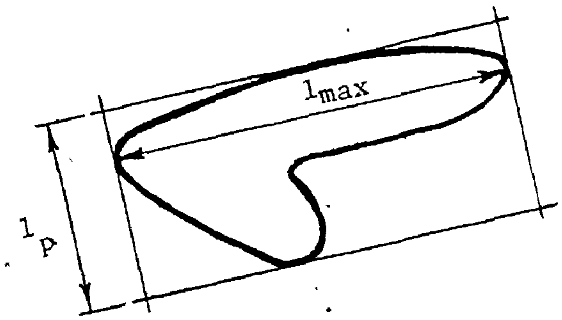
$$\varepsilon_o = \frac{l_{max}}{l_p}$$

where, l_{max} = distance between farthest neighbours of points on the contour

l_p = maximum distance, contained within the shape, perpendicular to l_{max}

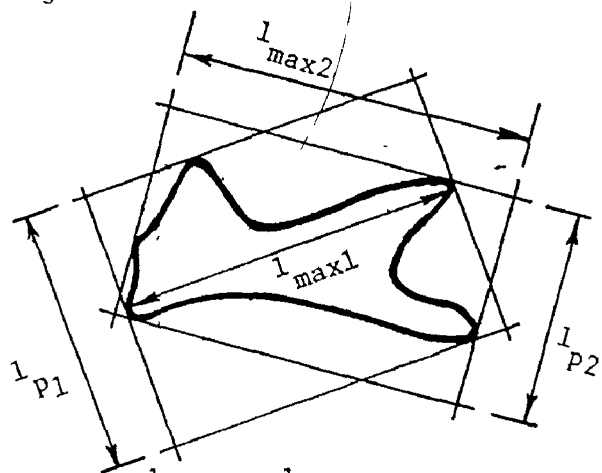
The distance "l_{max}" is the major axis of the region, "l_p" is the minor axis and the rectangle of sides l_{max} and l_p is termed the "basic rectangle". Figure 2.2 gives an example for an arbitrary shape. Eccentricity is an indication of "elongation" of an object, but there is a difficulty which makes its use undesirable. As in figure 2.3, l_{max} can be non-unique and two (or more) different values for ε_o can be obtained.

In this case, the authors [8] have suggested choosing l_{max} which has the shortest corresponding l_p. However, (l_{max}/l_p) is not necessarily unique either. A solution which alleviates this problem is to express the eccentricity in terms of the minimum area encasing rectangle. In this definition, eccentricity is expressed as the ratio of the longest to the shortest dimensions of the minimum area rectangle which just encloses the object.



$$\varepsilon_o = \frac{l_{max}}{l_p}$$

Figure 2.2 Basic Rectangle



$$l_{max1} = l_{max2}$$

$$\varepsilon_{o1} = \frac{l_{max1}}{l_{p1}} \neq \varepsilon_{o2} = \frac{l_{max2}}{l_{p2}}$$

Figure 2.3 Example of Non-Unique Eccentricity

$\underline{J}$

The second moment of area of an object is a function of the object's shape. It has its origin in the study of mechanics where it is usually termed "moment of inertia", although this is not correct when dealing with areas. (Moment of inertia should be used to denote integrals of mass). The polar moment of inertia is given by (see figure 2.4):

$$J = \int R^2 \, dA$$

For a circle of radius "r":

$$J_o = \int r^2 \, (2\pi r \, dr) \qquad (dA = 2\pi r \, dr)$$

$$= \frac{1}{2} \pi r^4$$

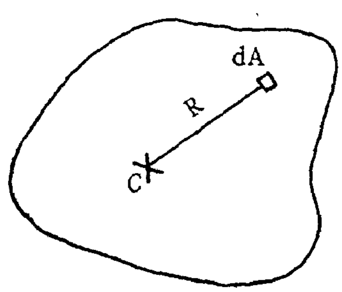
This leads to the normalized shape factor, J, defined by:

$$J = \frac{2\pi\ J_o}{A^2}$$

where, J_o = polar moment of inertia of a circle

A = enclosed area

For all shapes other than circles, $J > 1$. This shape factor is not sensitive to the resolution of the boundary, but pre-calculation of the center of area is necessary.



where, C is center of area

Figure 2.4 Calculation of Moment of Inertia

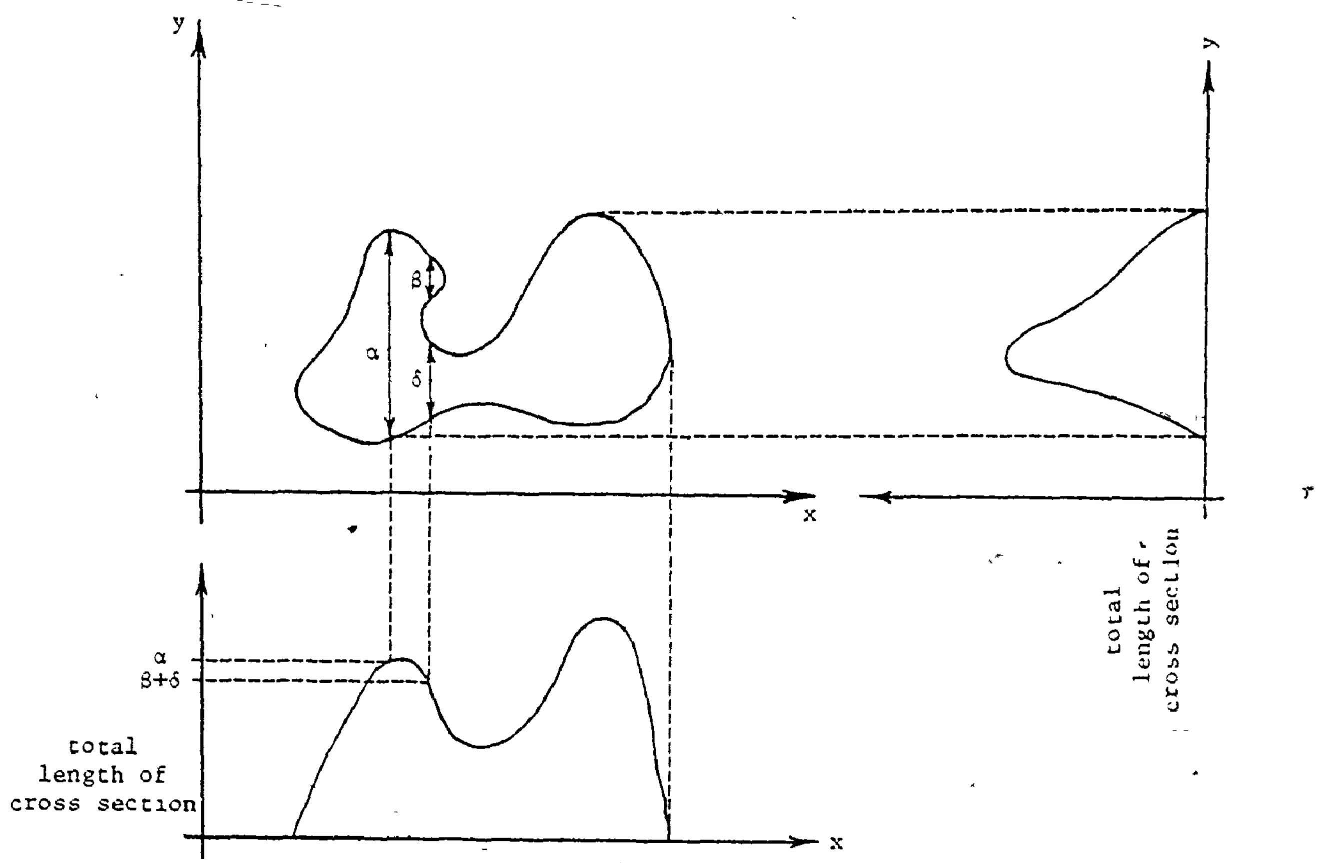


Figure 2.5 One-Dimensional Projections on x- and y-axes

Two-dimensional shape can be decomposed into one-dimensional distributions along the x- and y-axes (figure 2.5). These distributions can then be subjected to common statistical calculation of higher order moments. Zusne [9] has suggested the following interpretations of moments with respect to shape (Table 2.2).

	Moment	Interpretation
First:	$\int(x-\bar{x})\,dA$; $\int(y-\bar{y})\,dA$	Centroid
Second:	$\int(x-\bar{x})^2\,dA$, $\int(y-\bar{y})^2\,dA$	Compactness
Third:	$\int(x-\bar{x})^3\,dA$, $\int(y-\bar{y})^3\,dA$	Symmetry
Fourth:	$\int(x-\bar{x})^4\,dA$, $\int(y-\bar{y})^4\,dA$	Elongation

Table 2.2 Interpretation of Moments of Area

It is often convenient to allow the centroid of the object to correspond to the origin of the coordinate system such that $\bar{x} = \bar{y} = 0$. Dudani et. al. [10] have reported on the use of high order moments for the recognition of aircraft.

2.3 Information Preserving Shape Descriptors

Shape Numbers

A recent paper by Bribiesca and Guzman [8] summarizes a technique for finding the "shape number" of a closed contour. This method uses a grid to approximate the shape and information is extracted from the approximation. The information preserving property is regulated by selection of the appropriate grid resolution.

The procedure is as follows:

(a) The shape is approximated with a grid, oriented so as to coincide with the major axis of the object. Every cell on the grid of which more than 50% is contained within the contour, is shaded as a "black" region. The boundary of the entire black region is now the approximation. Figure 2.6 illustrates this procedure.

(b) The boundary of the black region is now encoded into a string of digits using a 1 for each convex corner, a 2 for each straight corner, and a 3 for each concave corner. There are as many strings of digits as there are digits independent of where on the boundary the encoding is started. Regarding each string of digits as one weighted number with symbols 1, 2, 3, the shape number is now arbitrarily taken as the minimum number from this set. (See figure 2.7).

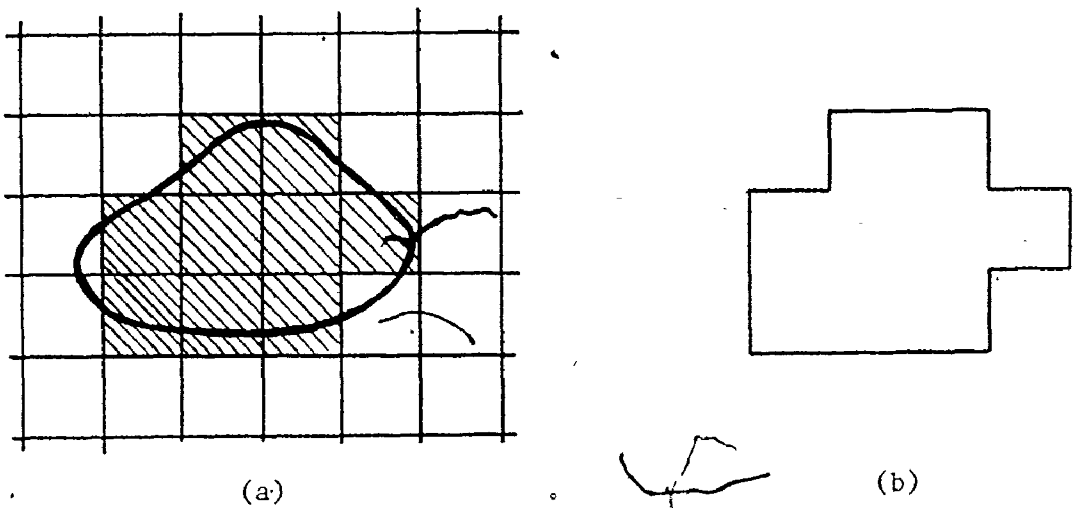
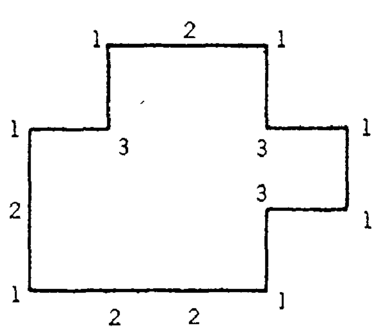


Figure 2.6 Extracting Shape Approximation

(a) original shape overlayed with grid

(b) boundary of shaded region



Boundary Codes (Clockwise)

1	3	1	2	1	3	1	1	3	1	2	2	1	2	
3	1	2	1	3	1	1	3	1	2	2	1	2	1	
1	2	1	3	1	1	3	1	2	2	1	2	1	3	
2	1	3	1	1	3	1	2	2	1	2	1	3	1	
1	3	1	1	3	1	2	2	1	2	1	3	1	2	
3	1	1	3	1	2	2	1	2	1	3	1	2	1	
1	1	3	1	2	2	1	2	1	3	1	2	1	3	← Shape Number
1	3	1	2	2	1	2	1	3	1	2	1	3	1	
3	1	2	2	1	2	1	3	1	2	1	3	1	1	
1	2	2	1	2	1	3	1	2	1	3	1	1	3	
2	2	1	2	1	3	1	2	1	3	1	1	3	1	
2	1	2	1	3	1	2	1	3	1	1	3	1	2	
1	2	1	3	1	2	1	3	1	1	3	1	2	2	
2	1	3	1	2	1	3	1	1	3	1	2	2	1	

Figure 2.7 Extracting Shape Number from Boundary

It is obvious that the size of the approximating grid determines how many digits will appear in the shape number, and more importantly, how accurately it represents the original region. The "order" of the shape number is the number of digits it consists of and may be chosen beforehand to yield any desired precision. The choice is made by selecting a grid size based on the eccentricity of the object. This procedure is covered in detail by the authors.

Shape numbers can be used for comparing two shapes. The degree of similarity is defined as the largest order for which the shape numbers of two contours remain equal.

Shape numbers may not exist for some contours. For example, if the shape contains a "neck" which is smaller than the grid size, then the approximating technique may produce two discrete shapes. A higher resolution grid must then be used.

Closed figures always have a shape number of even order. However, a shape number with an even number of digits is not necessarily a closed figure. The lowest order for closed contours is four (shape number = 1111) and represents a square.

Skeletons (Medial Axis)

Some recent efforts in the study of shape quantification have been directed towards the calculation of skeletons or medial axes of objects. (Badler and Dane [11], Shapiro, et. al. [12]). The idea was first proposed by Blum [13]. Other well-known work includes that of Rosenfeld and Pfaltz [14] and Montanari [15].

The approach is to reduce the given shape into a line drawing consisting of the points which do not have a unique nearest-neighbour on the contour. Each point of this line can then be labelled with its distance to the original boundary. Thus, reconstruction can be accomplished by tracing the envelope of a series of circles centered on the skeleton. Figure 2.8 demonstrates this procedure. The locus of points at which a wave front propagating from the contour intersects itself also defines the skeleton.

The medial axis is an example of a space transform. The original drawing is reduced, in this case, to a line drawing which characterizes the original shape. The output from a space transform is suitable for input to syntactic [16], [17], or structural [18] pattern recognition processes.

(a)

(b)

(c)

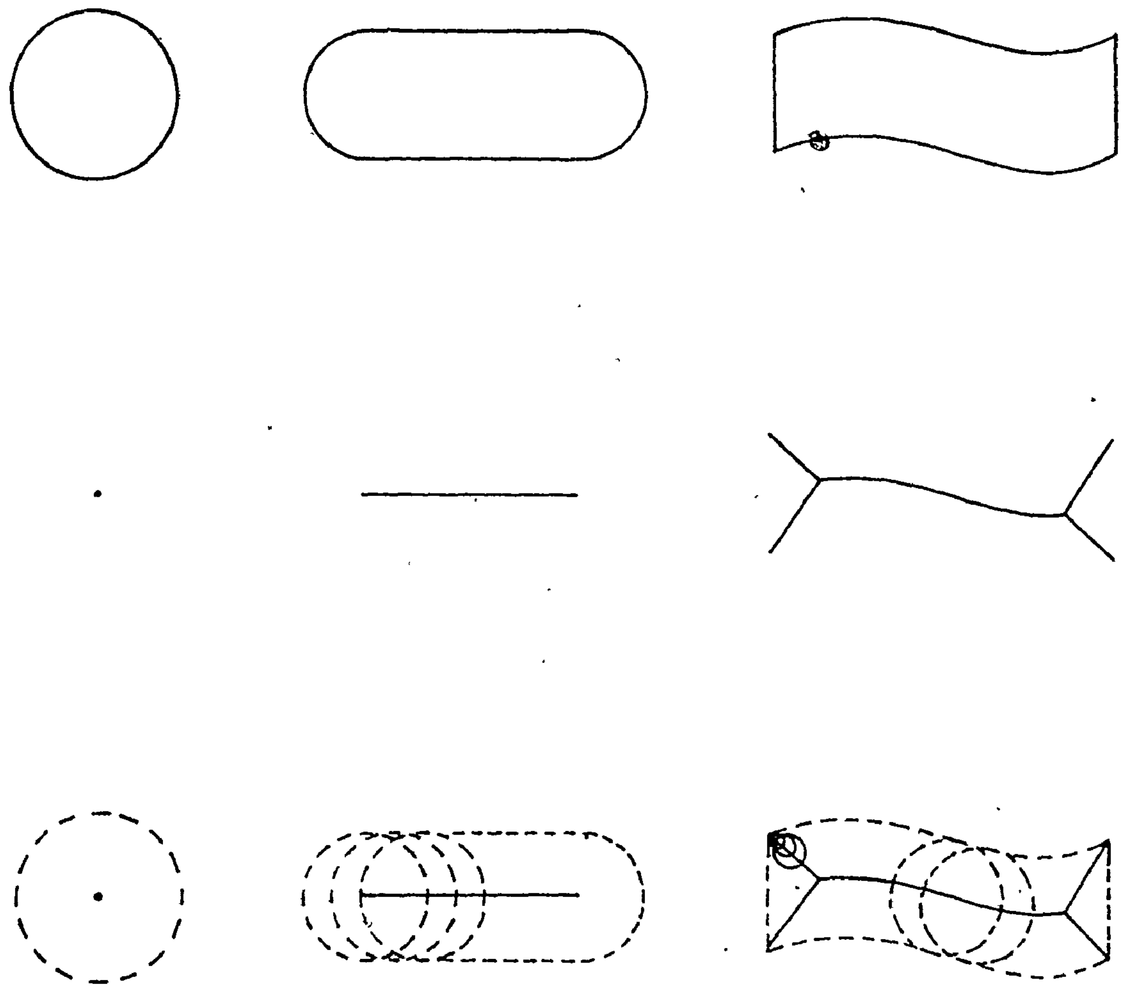


Figure 2.8 Skeletons

(a) contours

(b) skeletons

(c) reconstruction with a series of circles on skeleton

Fourier Shape Descriptors

The Fourier descriptors in Zahn and Roskies [19] are obtained from the Fourier transform of a parametric representation of the boundary of a shape. The angular direction of the contour is expressed as a function of arc length (see figure 2.9).

Assuming that the contour is smooth (i.e. the tangent exists everywhere), one point is selected as the starting point $(x(0), y(0))$. From this point the initial angle δ_o is determined and all remaining angles are expressed relative to this. That is:

$$\phi(\ell) = \theta(\ell) - \delta_o$$

where, $\theta(\ell)$ is the angular direction of the curve at $x(\ell)$, $y(\ell)$

$$\delta_o = \theta(0)$$

$$\phi(0) = 0$$

If the total length of the curve is "L", then $\phi(L) = -2\pi$ for clockwise traversal. Therefore, the function $\phi^*(t)$ is defined as:

$$\phi^*(t) = \phi\left(\frac{Lt}{2\pi}\right) + t$$

which normalizes any plane closed curve to the interval $[0, 2\pi]$ independent of translation, rotation or perimeter length. For a circle note that $\phi^*(t) = 0$ for all "t". Finally, $\phi^*(t)$ is expanded into its Fourier series:

$$\phi^*(t) = a_o + \sum_{k=1}^{\infty} (a_k \cos kt + b_k \sin kt)$$

In polar form: $$\phi^*(t) = a_o + \sum_{k=1}^{\infty} A_k \cos (kt-\alpha_k)$$

where (A_k, α_k) are the polar coordinates of (a_k, b_k)

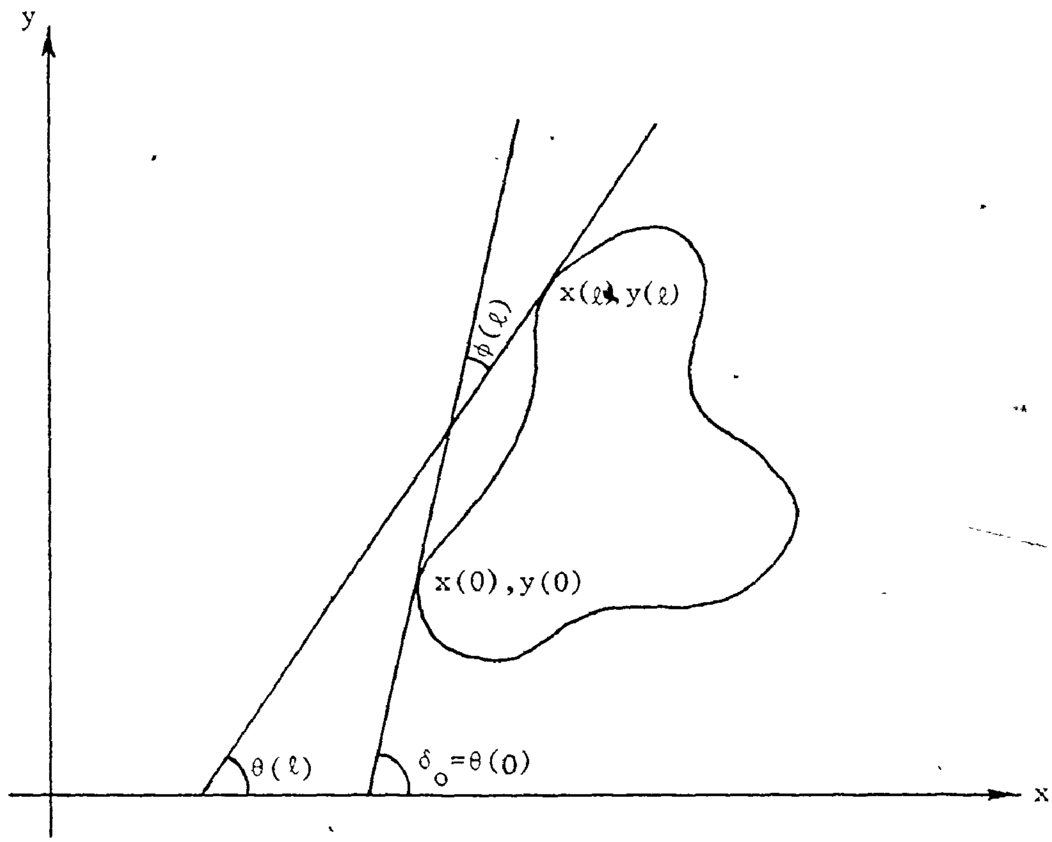


Figure 2.9 Parametric Representation of Contour

The Fourier descriptors are the pairs (A_k, α_k) where A_k is the k^{th} harmonic amplitude and α_k is the k^{th} harmonic phase angle. The set (A_1, α_1), (A_2, α_2) ... (A_n, α_n) will be unique for a given shape. By choosing "n" sufficiently large, shape information can be accurately retained.

Other well-known work in this area includes that of Persoon and Fu [20].

CHAPTER 3

PROCESSING OF 2-D BINARY IMAGE DATA

3.1 Introduction

Images that occur naturally are continuous functions of light intensity over a two-dimensional plane. To facilitate processing by digital computer, it is necessary to sample the data at known intervals and to restrict the value of the sampled point to within a known range. In the case of two-dimensional binary images (silhouettes) this range is [0, 1]. Conversion from continuous to discrete functions is generally the first step in image processing problems, of course not without some loss of information. The following discussion includes the definitions of terms.

Each "frame" of image data (one complete picture) is uniformly sampled at m rows of n picture points ("pixels"). The "grey-level" ~~of a pixel shall be designated~~ as $g(i,j)$ where $0 \leq i \leq (m-1)$ and $0 \leq j \leq (n-1)$. For silhouettes, $g(i,j) \in \{0, 1\}$ for all i, j. Furthermore, the data are often available in a "raster scan" fashion, that is, serially by consecutive rows. Referring to figure 3.1, the order is:

$$g(1,1),\ g(1,2),\ \ldots\ g(1,n),\ g(2,1),\ g(2,2),\ \ldots$$

$$g(2,n),\ \ldots\ g(m,1),\ g(m,2),\ \ldots\ g(m,n)$$

$$\begin{matrix} g(1,1) & g(1,2) & g(1,3) & \cdots & g(1,n) \\ g(2,1) & g(2,2) & g(2,3) & \cdots & g(2,n) \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ g(m,1) & g(m,2) & g(m,3) & \cdots & g(m,n) \end{matrix}$$

Figure 3.1 Frame Sampling

The distinction is drawn between 8-connected and 4-connected objects contained in this matrix. A silhouette represented by a region of "1's" in the matrix is said to be 8-connected if all 8 immediate neighbors of any pixel g(i,j) are considered to be 'connected' to that pixel. On the other hand, if only the neighbors g(i-1,j), g(i,j-1), g(i,j+1), and g(i+1,j) are to be taken as connected to g(i,j), then the object is said to be 4-connected.

In practice, the horizontal and vertical resolution (n and m) are often of the order 10^2 making the number of pixels of the order 10^4 [21] [22]. Pixel geometry is usually one of square, rectangular or hexagonal shape. The hexagonal grid has special merit in image processing because of its good symmetry properties. Smooth curves can be better approximated than with its square or rectangular counterparts. However, square and rectangular pixels are most common and the remainder of this work assumes this configuration.

3.2 Elementary Operations on Binary Images

The purpose of this section is to introduce a few of the basic operations that have been applied to binary images. Goldberg and Gougeon [23] have divided these techniques into three categories:

1. Point Operations: $g(i,j)$ of the resultant processed image depends only upon $g(i,j)$ of the original image. For silhouette processing this is not of much use.
2. Spatial Operations: $g(i,j)$ of the resultant processed image is a function of several points in the original image.
3. Multi-Image Operations: the processed $g(i,j)$ depends on the $g(i,j)$'s of more than one input image.

Examples of each type of operation are now presented.

1. Point Operations:

Because of the limited range of magnitude of $g(i,j)$ in binary images, point operations are not very useful. However, in obtaining binary values from grey-level images the thresholding technique is a point operation. The effect is to separate objects from the background.

$$g'(i,j) = \begin{cases} 1 \text{ if } g(i,j) \geq \delta \\ 0 \text{ otherwise} \end{cases}$$

where, $g'(i,j)$ is the resulting binary image matrix

$g(i,j)$ is the grey-level input image matrix

δ is the grey-level threshold. $0 \leq \delta \leq g(i,j)_{max}$

$g(i,j)_{max}$ is the maximum grey-level possible in $g(i,j)$

2. Spatial Operations

Spatial operations are widely used in image processing [24], [25]. A common application of these functions is that of image "smoothing" to reduce noise elements. The implementation involves moving an a x b window, centered on successive $g(i,j)$'s, throughout the image. The new $g(i,j)$ is a function of the pixels that fall within this window (often a = b = 3). A popular format for smoothing binary images is:

$$\begin{matrix} A & B & C \\ D & E & F \\ G & H & I \end{matrix} \qquad \text{window size: } 3 \times 3$$

$$g(i,j) = \begin{cases} 1 \text{ if } (A + B + C + D + F + G + H + I) \geq \delta \\ 0 \text{ otherwise} \end{cases}$$

where, A, B, ... I are elements in the original image

$g(i,j)$ is the resulting image matrix

δ is a threshold satisfying $0 \leq \delta \leq 8$

Figure 3.2(a) shows a sample binary image containing an object with a hole and several "noise" pixels. Figures 3.2(b) through (j) show the results of applying this smoothing operation for values of δ ranging from 0 to 8. For the lower values of δ (0 to 4) the smoothing function tends to "thicken" the image and noise is not deleted. The extreme case is $\delta = 0$ when the entire image is filled with 1's. Higher values of δ (5 to 8) introduce a "thinning" effect and small areas of noise are deleted. The extreme case is $\delta = 8$ where the resultant image is actually thinned so much that pieces of the object are deleted. As an intuitive selection, one might say that $\delta = 5$ produces adequate noise reduction with minimal effect on the desired image.

3. Multi-Image Operations

An example of a multi-image operation is "template matching" wherein an unknown image is compared to a previously stored image. The known image is translated or rotated until the correlation between the two is highest. A recent example (Perkins [26]) is described in which industrial parts are identified in a frame containing overlapping or incomplete parts. The technique is to attempt to match a template at various locations within the frame by superimposing models on a grey-level picture.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	1	1	1	1	1	0	0	0	0	1	1	1	1	1	0	0
0	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	0
0	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0	0
0	0	1	1	1	1	1	0	0	0	0	1	1	1	1	1	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(a) original image

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(b) $\delta = 0$

Figure 3.2 Binary Image Smoothing

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
0	0	1	0	0	0	0	1	1	1	1	0	0	0	1	1	1	0
0	1	1	0	0	1	1	1	1	1	1	1	1	0	1	1	1	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	1	1	1	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	1	1	1	0	0	0	1	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(c) $\delta = 1$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	1	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	1	1	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(d) $\delta = 2$

Figure 3.2 (continued)

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(e) $\delta = 3$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0	0	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	0	0	0	0	1	1	1	1	1	0	0
0	0	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	0	0	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	1	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(f) $\delta = 4$

Figure 3.2 (continued)

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0
0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0
0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

(g) $\delta = 5$

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0
0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0
0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0
0 0 0 1 1 1 1 1 0 0 1 1 1 1 1 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

(h) $\delta = 6$

Figure 3.2 (continued)

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0
0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0
0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0
0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0
0 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

(i) $\delta = 7$

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0
0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

(j) $\delta = 8$

Figure 3.2 (continued)

3.3 Connectivity Analysis

The connectivity of digital pictures has been formally treated by Rosenfeld in [27]. An outline of a connectivity algorithm as presented by Agin [28] is given below.

The purpose of connectivity analysis in binary images is to segregate the image into groups of pixels which form connected regions of uniform g(i,j).

Figure 3.3(a) shows a silhouette image consisting of a large "blob" with a hole and a smaller region (binary value "1") which intersects the right and bottom edges of the frame. In figure 3.3(b), a connectivity analysis has been applied and the image has been separated into components by giving each member of the connected regions the same label. Although the background (component "0") has been divided by component "3", both regions are labelled as background since any region of binary 0's are assumed to be background if they touch the edges of the frame. The hole (component "2") is not treated as background for this reason.

The algorithm makes use of an "active line" data structure which at any time consists of the pixel values in a single line of the image. A pointer keeps track of the next line in the image. To begin, the active line is assigned the string in line #1 and the pointer points to line #2. At this time, a comparison is made of the segments of binary 1's in the active line and those in the new line indicated by the pointer. One of three cases can occur:

column #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	line #
	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	2
	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	3
	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	4
	0	0	0	0	0	1	1	1	0	0	0	0	1	1	1	0	0	0	0	5
	0	0	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0	0	0	6
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	7
	0	0	0	1	1	1	1	1	0	0	0	0	1	1	1	0	0	0	0	8
	0	0	0	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	9
	0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	0	0	0	0	10
	0	0	0	0	0	1	1	1	0	1	1	1	1	1	0	0	1	1	0	11
	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	1	1	1	12
	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	13
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	14
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	15

(a) Original Binary Image

0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	2
0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	3
0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	4
0	0	0	0	0	1	1	1	0	0	0	0	1	1	1	0	0	0	0	5
0	0	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0	0	0	6
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	7
0	0	0	1	1	1	1	1	2	2	2	2	1	1	1	0	0	0	0	8
0	0	0	1	1	1	1	2	2	2	2	2	1	1	1	0	0	0	0	9
0	0	0	0	1	1	1	2	2	2	1	1	1	1	1	0	0	0	0	10
0	0	0	0	0	1	1	1	2	1	1	1	1	1	0	0	3	3	0	11
0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	3	3	3	12
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	3	3	0	0	13
0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3	0	0	0	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3	0	0	0	15

(b) Connectivity Labelling

Figure 3.3 Connectivity Analysis

Case 1: Segments do not overlap because the active line segment is to the left of the current segment (that which is indicated by the pointer).

e.g. Active line: 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0

Current line: 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0

Case 2: Segments do not overlap because the active line segment is to the right of the current segment.

e.g. Active line: 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0

Current line: 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0

Case 3: The segments overlap.

e.g. Active line: 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0

Current line: 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0

For Case 3, overlap depends on whether the image is to be considered 4- or 8-connected. For example:

4-connected overlap: $\begin{bmatrix} 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \end{bmatrix}$

8-connected overlap: $\begin{bmatrix} 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \end{bmatrix}$

Successive pairs of lines in the image are examined. At each comparison, it is decided which of the three cases applies and the pixels are labelled according to which region they are connected.

As an example, again consider figure 3.3(a). Line #1 is initialized as active and the current line is line #2. In line #1, the segments of 0's are labelled as 0 since they are background, and the 1's are labelled 1 as the first object found. When compared with line #2, Case 3 applies and the segment of 1's in line #2 receive the same label as those in line #1 since they are connected. (In this instance the label is "1"). Also, the pixels of value 0 in line 2 are classified as background. The process is repeated for lines #2 and #3.

When comparing lines #3 and #4 the following situation exists:

```
  col.#                       6 7               1415              line #

                    0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0           1
                    0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0           2
 active line   →    0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0           3
current line   →    0 0 0 0 0 1 1 0 0 0 0 0 0 2 2 0 0 0 0           4
```

The pixels in columns #6 and #7 are labelled as 1's as described above, but those in columns #14 and #15 must be temporarily labelled "2" since it is not yet known that they are part of the same "blob". The algorithm builds two separate regions (in addition to background) until lines #6 and #7 are compared. At this point the following labels have been assigned:

```
  col.#                       6 7               1415              line #

                    0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0           1
                    0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0           2
                    0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0           3
                    0 0 0 0 0 1 1 0 0 0 0 0 0 2 2 0 0 0 0           4
                    0 0 0 0 0 1 1 1 0 0 0 0 2 2 2 0 0 0 0           5
 active line   →    0 0 0 0 1 1 1 1 1 0 0 2 2 2 2 2 0 0 0           6
current line   →    0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0           7
```

In this situation, a Case 3 overlap has occurred with line #7 and two segments in line #6. It is now known that the region "2" is connected to region "1" and all the labels "2" should be replaced with "1's". The two connected components are merged.

As processing continues and the hole is encountered (line #8), unique labels are applied until further along it is determined that this region is disconnected from the background. The labels remain and identify the hole as a distinct component of the image.

After (or during) the labelling procedure, components of the image can then be described with the various characterizing features (area, centroid, P2A, etc.).

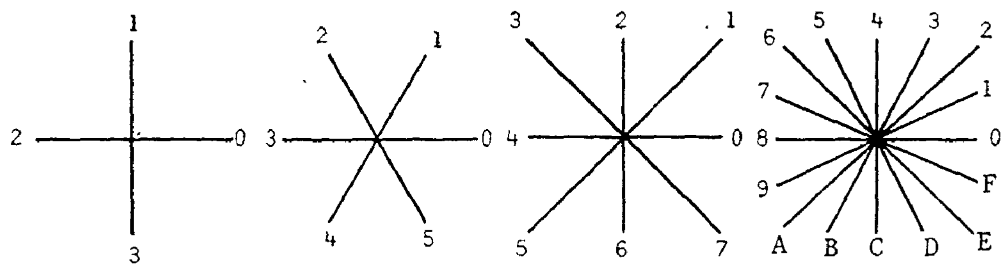
3.4 Encoding Techniques

Freeman Chain Code

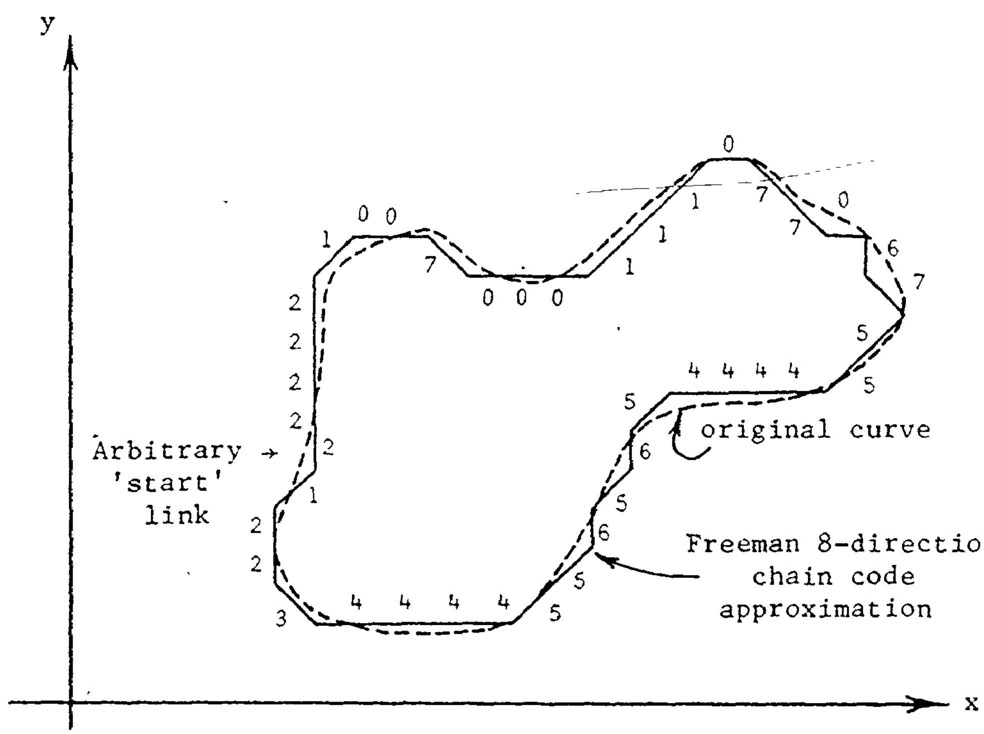
A widely known method for encoding geometric configurations was described by Freeman [29]. Since its introduction in the early 1960's, it has been a common technique for representing contours in many image processing applications. It has given rise to several variations of not only representation, but manipulation of shape boundaries and is tailored for digital computers. The basic process is one of approximating a curve with a series of line segments whose lengths are determined by the desired resolution and whose directions are limited to a finite set of values. The resulting curve is a piece-wise linear approximation of the original contour (figure 3.4(a)).

For example, in figure 3.4(b), an arbitrary plane curve is approximated by overlaying a grid and connecting, in sequence, the grid nodes which lie closest to the curve. In this case, the 8-direction links are used and the resulting chain code is given in figure 3.4(c). Each octal digit in the code can be represented by only 3 binary digits. For a closer approximation of the boundary, a finer grid can be selected.

Once the chain code is ascertained, there exists a variety of operations utilizing its information. The perimeter of the boundary can be calculated as the number of even octal digits plus the square root of two times the number of odd digits. (This assumes a square grid size of unity).



(a) Left to Right: Chain code links for 4, 6, 8 and 16 directions



(b) An arbitrary curve and its chain-code approximation

22222 10070 00111 07706 75544 44565 65544 44432 21

(c) 8-direction chain code of the above boundary

Figure 3.4 Freeman Chain Code

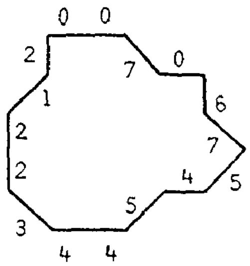
Elementary expansion or contraction can be realized readily by an appropriate change of grid size. Expansion can also be performed by replacing chain code digits with multiple digits of the same value. (i.e. a 2 x expansion of the code 07654321 is 0077665544332211).

Rotation in multiples of 90° is accomplished by adding (modulo 8) the corresponding even digit to each digit in the code. Rotations through odd multiples of 45° are obtained by adding the correct odd digit. These rotations however, except in trivial cases, result in distortion.

Figure 3.5 illustrates some of these operations. Others are described in [29].

Using these techniques, shape descriptors such as P2A, eccentricity, etc. can be extracted from Freeman chains. In [30] a method is proposed for finding the minimum-area encasing rectangle for an arbitrary closed curve. The eccentricity of this rectangle can be used as a shape descriptor. In addition, a detailed review of other operations on chain coded curves is given in [31]. A variation of the chain code which takes advantage of raster scan presentation devices is given in [32].

start →



Chain Code: 22120 07067
54544 3

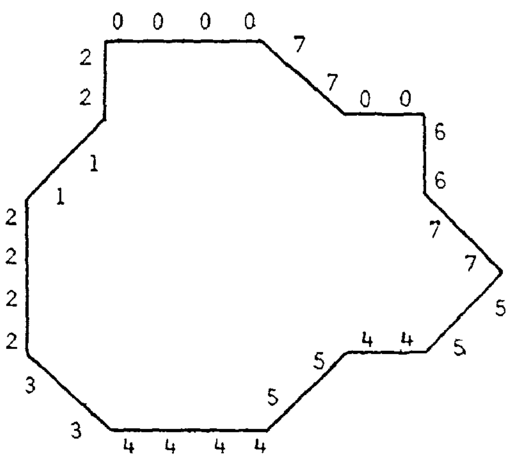
(a) Original contour and its chain code

Perimeter length:

\# even octal digits: 10
\# odd octal digits: 6

perimeter length = 10 + ($\sqrt{2}$) (6)
= 18.49 grid units

(b) Calculation of perimeter length

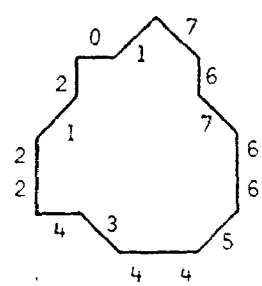


Chain code:

22221 12200 00770 06677
55445 54444 33

(c) Expansion by a factor of 2

Chain Code: 44342 21201
76766 5



(d) Rotation by +90°: add (modulo 8) 2 to each digit

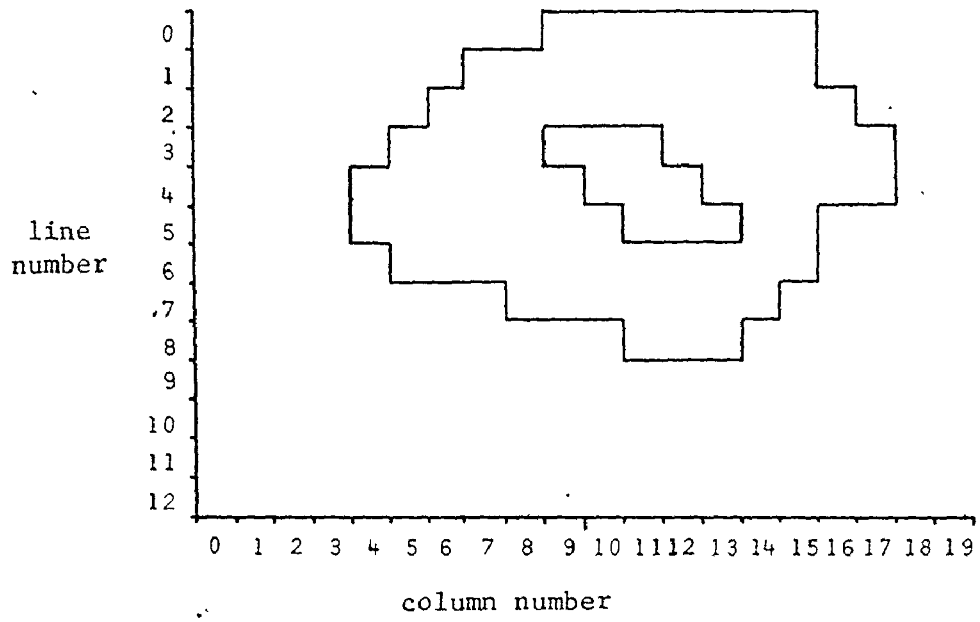
Figure 3.5 Some Operations on a Freeman Chain

Run-Length Coding

Run-length coding is a method for representing silhouettes which profits from the redundancies inherent in binary images. Only the edge transitions in each line of data are noted for future processing.

Consider figure 3.6 as an example. For each line in the original matrix, the column numbers for every transition from background to object (0 to 1) or object to background (1 to 0) are recorded. It is assumed that all the pixels between each pair of column numbers are 1's. As shown, when a silhouette contains a hole (or there are multiple silhouettes) the technique simply notes multiple pairs for the affected lines. To handle this situation with a chain code, a chain for each boundary is formed, including holes.

Generally, run-length coding greatly reduces the amount of data to be processed. The worst case occurs when storing a checkerboard pattern where alternate pixels are of the same value, as can appear in noisy images. Run-length codes are well suited for input to connectivity analysis routines. A well known scheme is described by Agin in [28].



(a) A digitized silhouette (with a hole).

Line Number	Run Length Code	
0	9, 16	
1	7, 16	
2	6, 17	
3	5, 9	12, 18
4	4, 10	13, 18
5	4, 11	14, 16
6	5, 16	
7	8, 15	
8	11, 14	

(b) The corresponding run-length codes

Figure 3.6 Run-Length Coding Example

Neighborhood Coding

There are many instances in which the 8 immediate neighbors of a pixel are used in binary image processing, for example, filtering and contour following. Sobel [33] has proposed an interesting variation for storing binary images upon which these operations are to be performed.

Since much of the processing time is used to access the eight neighbors of each pixel, it would be advantageous to store the neighbors of each pixel assembled as one byte (8-bits). The savings result from reducing to a single access, the problem of examining all 8 neighbors. The technique stores each pixel 8 times however, and results in an 8-fold increase in the required buffer memory.

Hardware in the form of a "tapped shift register" is described which allows this information to be extracted from a raster scan input. Figure 3.7 shows the configuration, given in [33].

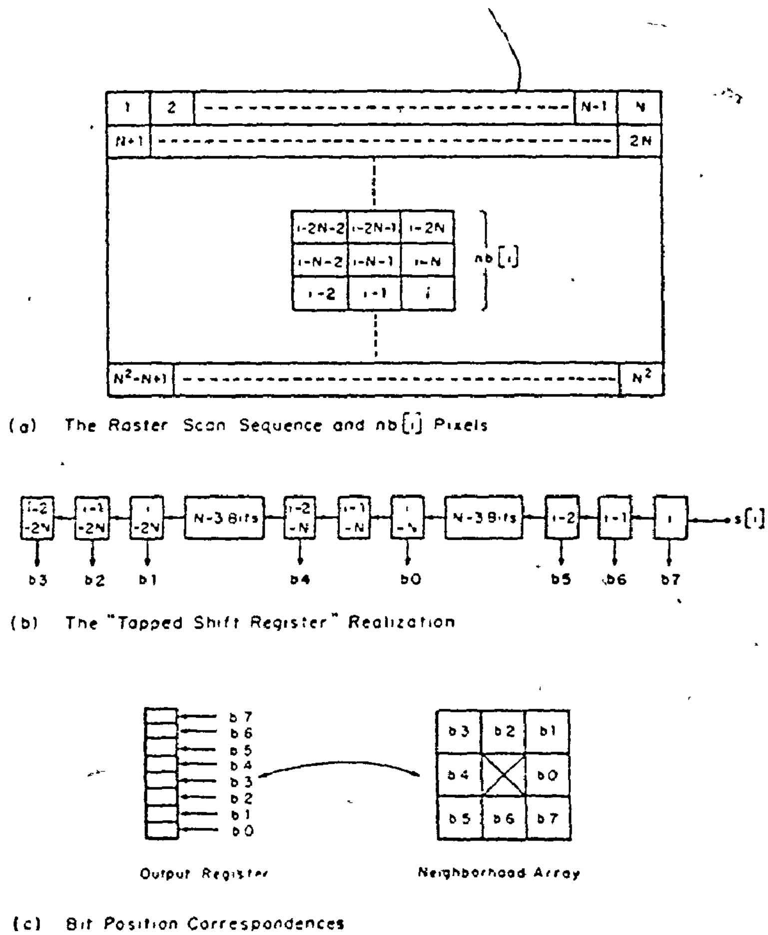


Figure 3.7 Neighborhood Encoder (reproduced from [33])

3.5 Determining Location and Angular Orientation

To determine the location of an object within an image frame it is necessary to specify the coordinates of a unique point on the object. A suitable point is the centroid, or center of area. For a digital binary image, the coordinates of this point (x_c, y_c) are given by:

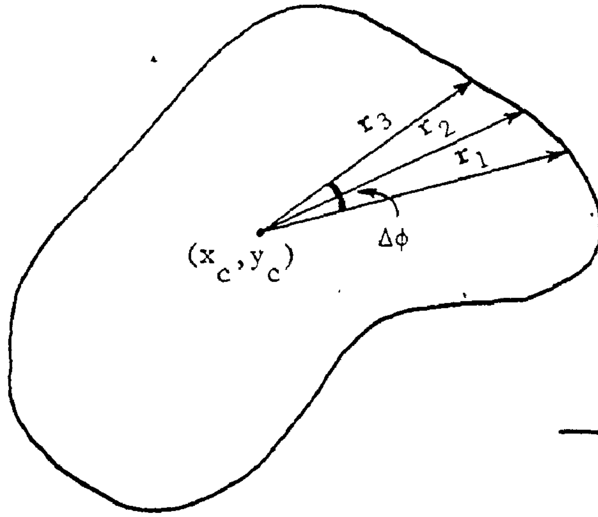
$$x_c = \frac{1}{\text{Area}} \sum_{j=0}^{m-1} \sum_{i=0}^{n-1} i \cdot g(i,j)$$

$$y_c = \frac{1}{\text{Area}} \sum_{j=0}^{m-1} \sum_{i=0}^{n-1} j \cdot g(i,j)$$

where, $\text{Area} = \sum_{j=0}^{m-1} \sum_{i=0}^{n-1} g(i,j)$

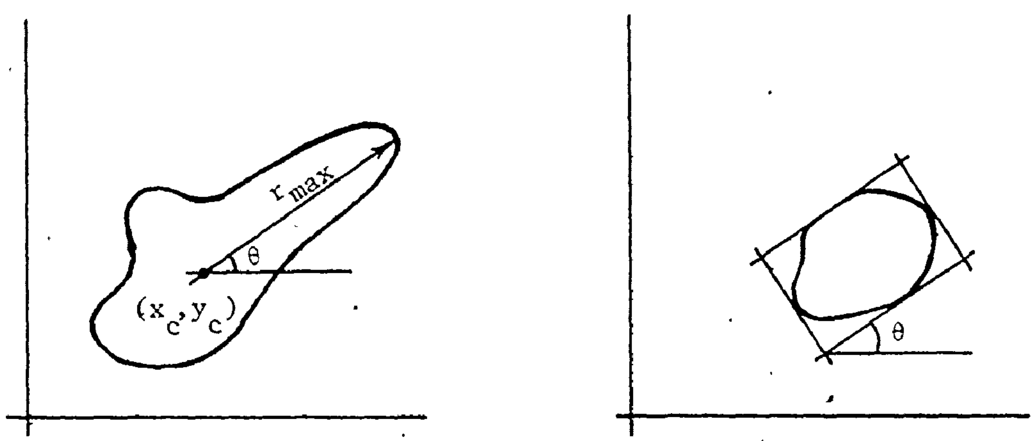
$g(i,j)$ is an element of the binary matrix of size $m \times n$

The angular orientation of an object can be evaluated by several different methods (Martini and Nehr [34]). For example, a shape can be radially scanned from the center of area and the distance to the contour expressed as a function of angle can be stored (fig. 3.8). The function $R(\phi)$ is invariant relative to the position of the object, and displays a phase shift for different angular displacements. $R(\phi)$ from an unknown object can be compared to that of a previously stored object and the angular distance between the two can be found by determining the phase shift. However, there are problems with this method. When the object is not



ϕ	R(ϕ)
0	r_1
$\Delta\phi$	r_2
$2\Delta\phi$	r_3
$3\Delta\phi$	r_4
⋮	⋮

Figure 3.8 Distance From Centroid to Contour as a Function of Angle (Polar Signature)



(a) Maximum radius from centroid (b) Minimum-area encasing rectangle

Figure 3.9 Methods for Determining Angular Displacement, θ convex or it contains holes, $R(\phi)$ assumes multiple values. Also, knowledge of the border points is necessary.

De Coulon and Kammenos [35] have solved the first problem by summing the radial distance from the centroid to each contour intersection. This approach, it is reported, tends to account for the contribution of features such as holes, minor protrusions, etc.

A cross-correlation of the signatures of a reference object and an unknown is given in [35] as:

$$C(a) = \frac{1}{n} \sum_{i=1}^{n} R_1(\theta_i) \cdot R_2(\theta_i + a)$$

where $C(a)$ is the cross-correlation for phase shift "a"

n is the number of sample points

$R_2(\theta_i)$ is the unknown polar signature

$R_1(\theta_i)$ is the reference polar signature

The angular position of the peak of this function gives the relative angular displacement of the two objects.

Also described in [35] is the absolute difference function:

$$C_d(a) = \sum_{i=1}^{n} R_1(\theta_i) - R_2(\theta_i + a)$$

where, $C_d(a)$ is the absolute difference function at phase shift "a"

n is the number of sample points

$R_2(\theta_i)$ is the unknown polar signature

$R_1(\theta_i)$ is the reference polar signature

The angular displacement of $R_2(\theta)$ with respect to $R_1(\theta)$ is then given by "a" for which $C_d(a)$ is minimum.

Other methods for determining angular orientation are illustrated in fig. 3.9. Agin [41] uses the angle of the maximum radius, r_{max}, from the centroid. Alternatively, the orientation of the major side of the minimum-area encasing rectangle could be used.

3.6 Boundary Errors in Quantized Binary Images

Uncertainty in digitized images arises at border points and is dependent on the resolution used in the quantizing process and the positioning of objects over the grid. Consider the examples shown in figure 3.10. A 4 x 4 square positioned parallel to the axes in 3.10(a) will cover 16 grid points. In 3.10(b), however, depending on the quantizing process, a different number of grid points can be classified as belonging to the square, in this case, 18. The discrepancy arises at the edges of the square where it is not clear whether or not a grid point will be included as part of the object (i.e. their areas are different). Even more uncertain is the measurement of the perimeter. Rosenfeld [36] has proposed two methods of measuring perimeters in digital pictures:

(1) The perimeter, P, is taken as the area of the border of the object, where border points are those which have any 4-connected neighbor not belonging to the object.

(2) Each border point is visited in succession and P is calculated by counting 1 for each horizontal or vertical move and $\sqrt{2}$ for each diagonal move.

Clearly, these definitions can give widely different results. For the examples in figure 3.10, the results of the two methods are:

	Method 1	Method 2	Actual Perimeter
Figure 3.10(a)	12	12	16
Figure 3.10(b)	10	$2 + 8\sqrt{2} \simeq 13.3$	16

Kulpa [37] and Sankar and Krishnamurthy [38] have suggested that the area of digitized objects be evaluated using Pick's theorem:

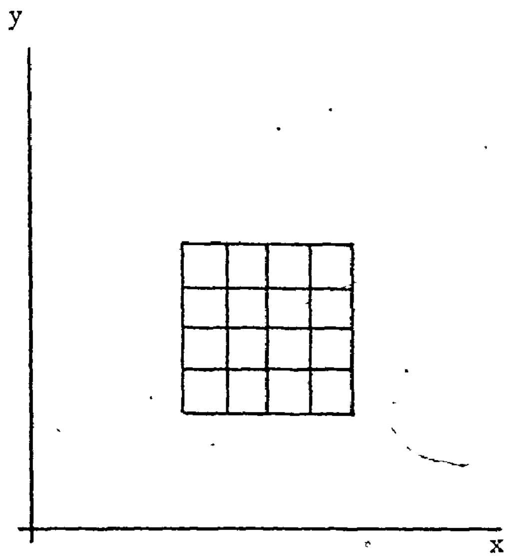
$$A_p = i + b/2 - 1$$

where, A_p = area of polygon

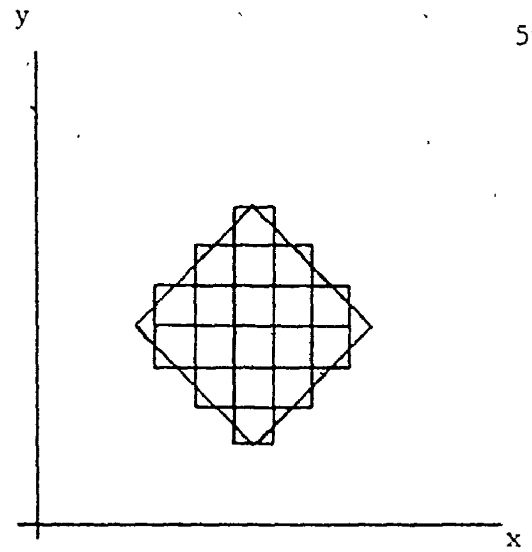
i = number of pixels interior to the object

b = number of pixels on the boundary

For the example of figure 3.10(a), Pick's theorem gives $A_p = 9$ and in figure 3.10(b), $A_p = 12$, Again, it is necessary to be clear as to which points comprise the actual boundary. Ellis and Proffitt [39] have proposed the use of lattice points which specify the centers of pixels as opposed to grid points. The technique is illustrated in figure 3.11. The lattice is a mesh of the



(a) Oriented parallel to x and y axes



(b) Rotated by $\sim 45^{\circ}$

Figure 3.10 A 4 x 4 Square with Different Orientations on a Quantizing Grid

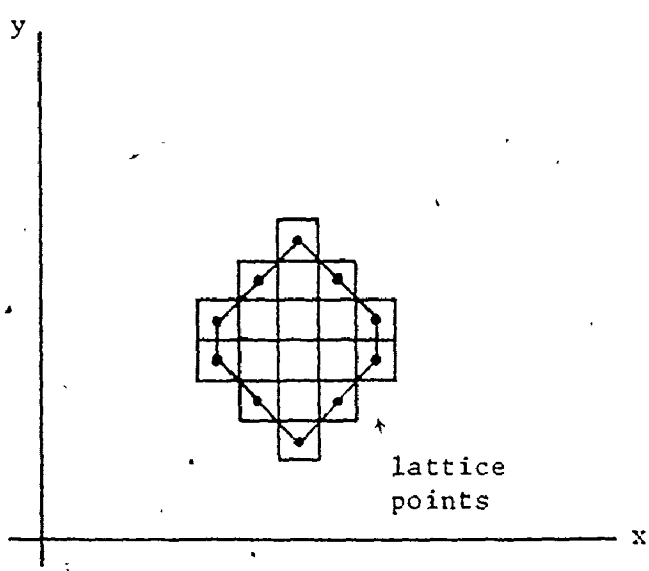


Figure 3.11 The Lattice Points of a 4 x 4 Square

same size and shape as the grid. However, a grid coordinate specifies a particular pixel and its area, whereas a lattice coordinate only indicates a point on the object. (Lattice points need not necessarily be centered on the grid). Lattice points, also, do not give a true measurement of perimeter in Pick's theorem. For figure 3.11:

$$A_p = 12$$

The errors introduced with image digitization give rise to uncertainty in measuring the lengths of curves of which users must be aware. For a particular application, the definition of perimeter must be established beforehand. In addition, the area can vary depending on an object's orientation on the quantizing grid.

These errors are most prominent for small objects. For larger ones which cover greater areas on the sampling grid, relative errors decrease. Depending on the method of evaluating perimeter, measurements such as P2A can take on widely different values.

CHAPTER 4

COMPUTER VISION FOR INDUSTRIAL APPLICATIONS

4.1 Introduction

Industrial vision has evolved from photocells simply counting objects as they move along an assembly line, to full scale television monitoring systems relaying information to computers.

A demand for machines to extract and interpret information from video signals has appeared due to the concern for increased productivity in automated manufacturing. The influx of microcomputer-based equipments has made such apparatus attractive to assembly line operators where manual tasks are unreliable and expensive [40].

Research and development in this area is an ~~acti~~ve and growing effort. Progress towards practical industrial vision stems from the science of digital picture and image processing. A survey of the literature shows active programs at such organizations as Stanford Research Institute (SRI), [41], [42], [43], [44], [45], University of Rhode Island [46], [47], [48], [49], [50], [51], University of Nottingham [52], [60], General Motors [26], [53], [54], [55], [56], [66], and University of Edinburgh [57]. The state-of-the-art in industrial vision has been summarized by Stewart [58].

The following section in this chapter gives a brief overview of the functional requirements of machine vision systems. In keeping within the scope of this thesis, section 4.3 describes some representative systems which operate on binary pictures. Since the introduction of the SRI Vision Module, several similar schemes have been reported [52], [59], [60], [61], [62]. The General Motors CONSIGHT-I system has been included because of its unique lighting arrangement. The NBS system illustrates the use of binary image processing in servo applications.

4.2 Functional Requirements of Industrial Vision

Rosen [63] has classified industrial machine vision applications into two categories. In the first instance, visual capabilities provide information for manipulative functions, for example, determining the location and orientation of randomly positioned items on a conveyor belt. The second group pertains to inspection operations where non-contact examination measures, for example, the dimensions of a workpiece. Figure 4.1 (taken from [63]) summarizes these groupings.

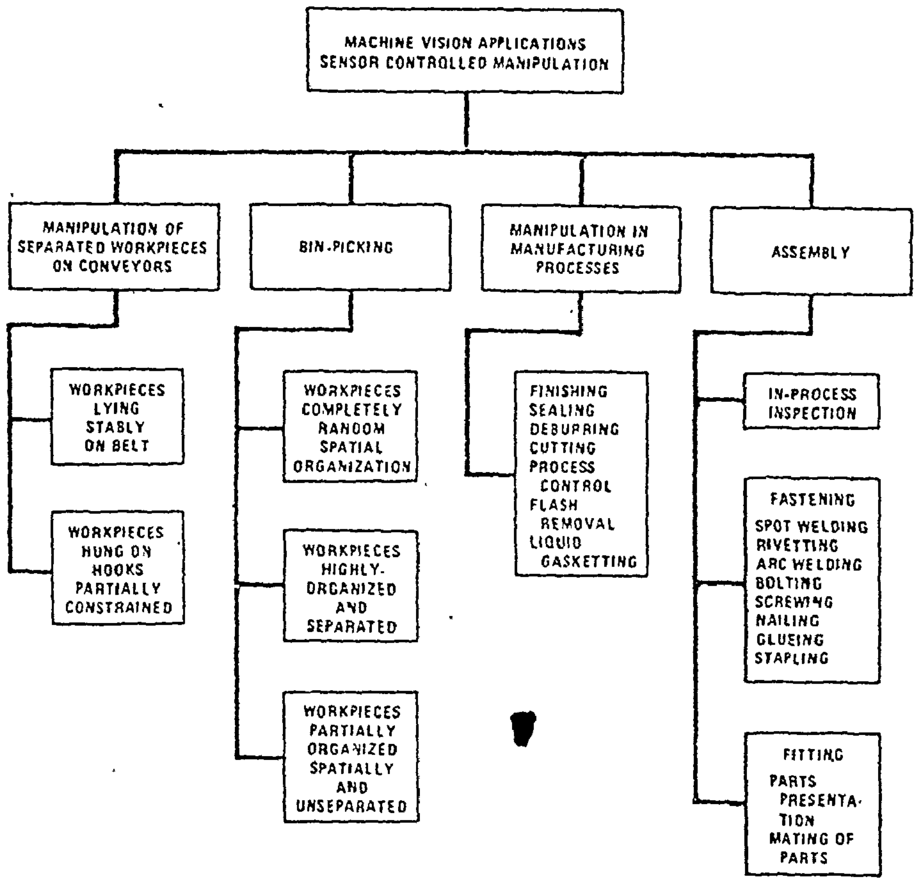
(a) Sensor-Controlled Manipulation

Most automated manipulators without sensory feedback rely on the precise alignment of objects presented to them. In many cases, these part presentation devices can be eliminated with the use of a machine vision system [46], [50], [52], [57]. Visual feedback has allowed the automation of intricate operations such as integrated circuit wire bonding [64].

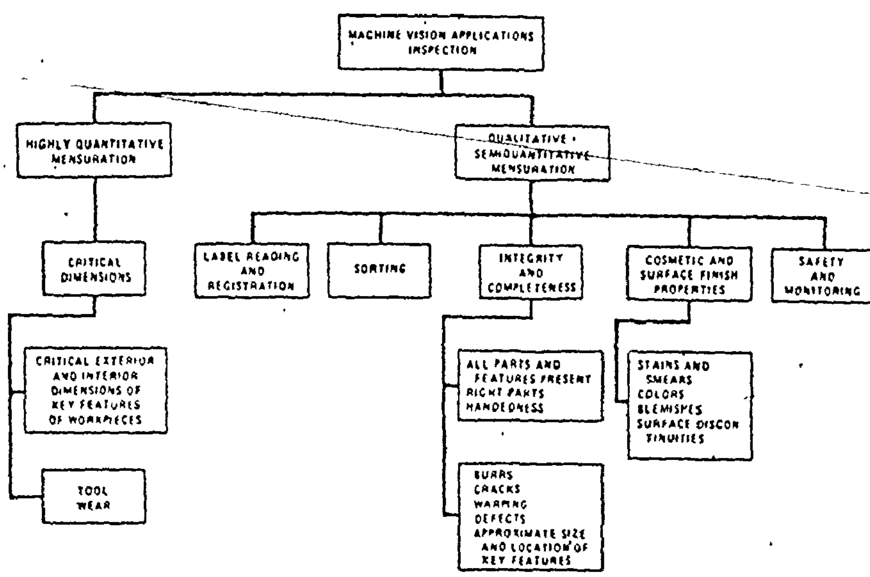
In figure 4.1(a), manipulative tasks in automated manufacturing have been grouped into 4 situations which can be enhanced with machine vision. The first two cases deal with the presentation of workpieces to manipulators. The second two illustrate the role of vision in feedback for assembly and manufacturing processes.

(b) Inspection

In figure 4.1(b), inspection applications are divided into two categories; highly quantitative mensuration and qualitative/semi-quantitative mensuration. Highly quantitative operations are those in which machine vision provides non-contact measurement of critical dimensions, tool wear, etc. Qualitative mensuration involves sorting, label reading, etc. A survey of inspection automation is given in [65].



(a) Sensor Controlled Manipulation



(b) Inspection

Figure 4.1 Machine Vision Applications
(reproduced from [63])

4.3 SRI Vision Module

The SRI Vision Module is an LSI-11 microcomputer-based binary image processing system [41]. The system was designed as a cost-effective solution to many industrial problems.

Input to the system is provided by one of a number of different television cameras with resolutions of 128 x 128 or 256 x 256 pixels. Since processing is achieved line-by-line, a one-dimensional diode array camera is also suitable.

Images are thresholded to binary values to give high contrast silhouettes. The binary data is run-length encoded to reduce processing time and memory requirements, and the run-length codes are used as inputs to connectivity analysis software (see [28]).

Table 4.1 (taken from [41]) shows some of the useful features extracted from the image data. The connectivity analysis allows more than one object to be in the field of view, providing no contour touches another.

Position Features:

(a) Center of gravity
(b) Center of bounding rectangle

Shape Features:

(a) Area
(b) Square root of area
(c) Perimeter length
(d) Maximum, minimum and average length of radius vector from centroid to perimeter
(e) Number of holes in the figure
(f) Total area of all holes in the figure
(g) Number of local maxima and minima in the radius vector (corners and notches)
(h) Dimensions of bounding rectangle

Orientation Features:

(a) Angle of axis of least moment of inertia
(b) Angle of maximum and minimum length radius vectors from centroid to perimeter
(c) Orientation obtained from best match of corners and notches in the figure with those of a prototype

Table 4.1 Features Used in SRI Vision Module

Prototype objects are "taught" to the system beforehand. Interactive software allows additions, deletions or changes to the set of known objects. When a particular workpiece can have multiple stable positions beneath the camera, each state is taught as a different shape.

Identification of unknown objects is realized with a "nearest-neighbor" technique. A distance from the unknown part to each prototype is calculated as the sum of the squares of the differences of each measured feature. The object in question is then identified as the prototype from which this distance measure is minimum. A disadvantage of this technique is that it is not always clear which features should be measured to give acceptable discrimination among similar shapes. The alternative used is a binary decision tree as shown in figure 4.2 (from [41]). A large number of features, (say x_1, ... x_n), are considered in generating the tree, but final recognition is based on a selected few, (x_1, x_2, x_3, x_4, x_5, in figure 4.2). For example, of all measured features, one is selected (x_3) which divides the known prototypes into two classifications. Each of these are similarly divided. Finally, when an identification is requested, a sequence of queries is instigated as in figure 4.2.

Various lighting techniques have been used to ensure clean (low noise) binary images. These include back-lit tables, fluorescent backgrounds illuminated by ultraviolet lamps, and color filters. Cycle times are reported to be in the vicinity of 2-5 seconds. Specifications for the commercially available version of the system are included in Appendix "B".

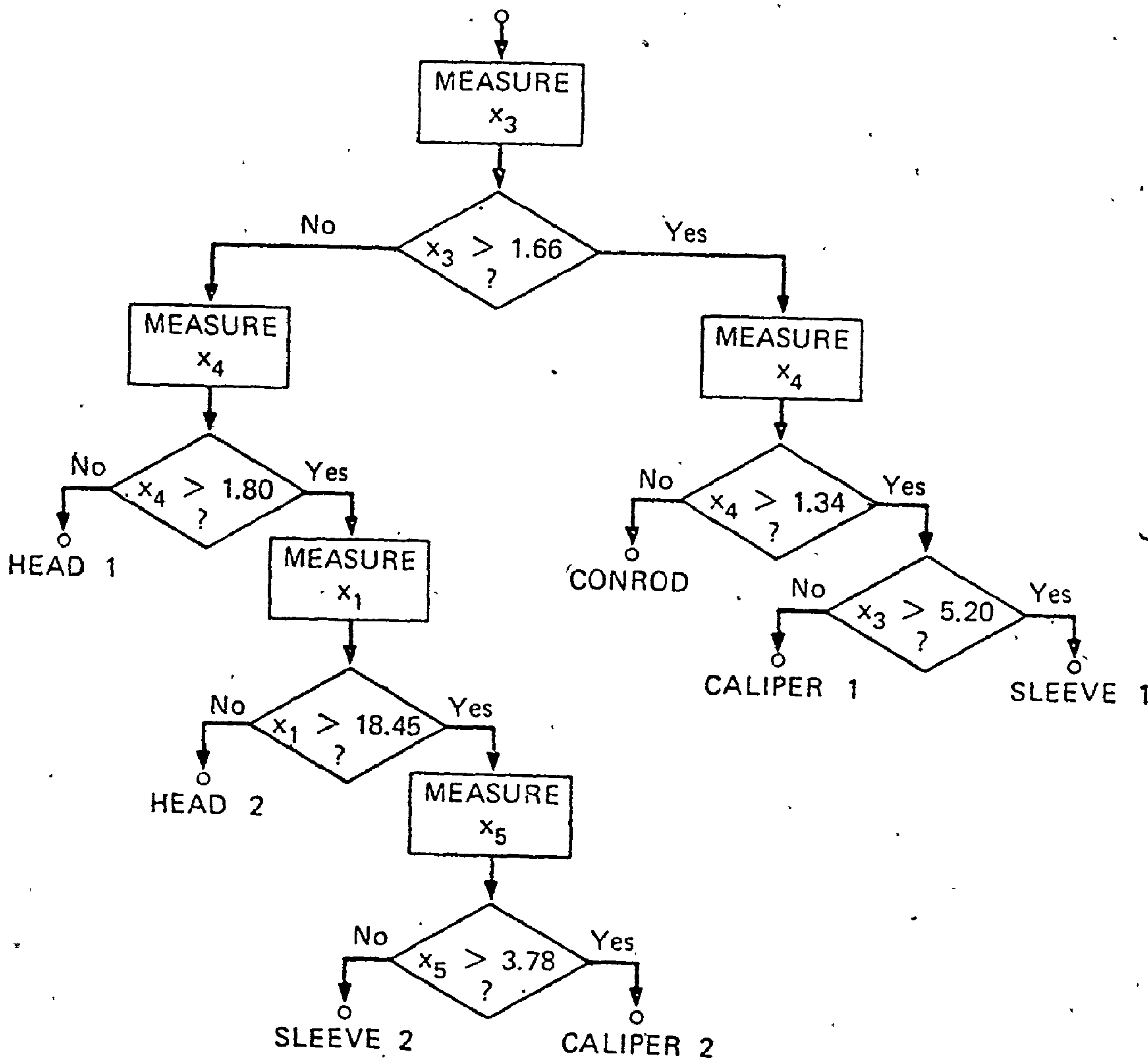


Figure 4.2 Binary Decision Tree
(reproduced from [41])

4.4 General Motors CONSIGHT-I

The CONSIGHT-I system is a vision-based robot that picks up parts from a moving conveyor belt [55]. Its distinguishing characteristic is its ability to work on visually noisy images through the use of a unique lighting technique.

The vision hardware consists of a 256 element line camera and a PDP-11 minicomputer. The linear array is placed perpendicular to the direction of motion of the conveyor belt whose position is reported to the computer by a belt position/speed indicator. A special lighting arrangement projects a single line of light across the belt directly beneath the linear array camera. Objects intersect this light source to provide shape information to the computer. The overall system configuration is shown in figure 4.3 and details of the lighting arrangement in figures 4.4, 4.5, and 4.6.

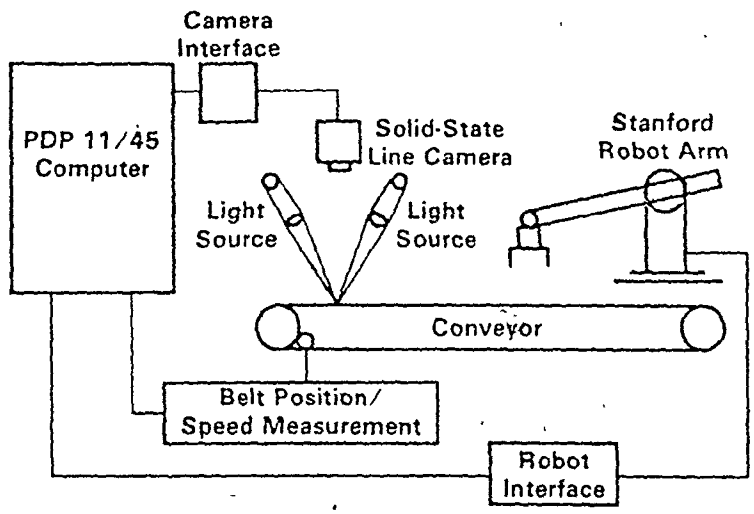
Similar to the SRI Vision Module, there are two modes of operation: a setup mode in which the system learns new parts, and an operate mode in which part transfer operations are performed. The following component descriptors (features) are used for object identification and location:

1. external position reference
2. color (black or white)
3. count of pixels (area)
4. sum of x-coordinates
5. sum of y-coordinates
6. sum of product of x- and y-coordinates

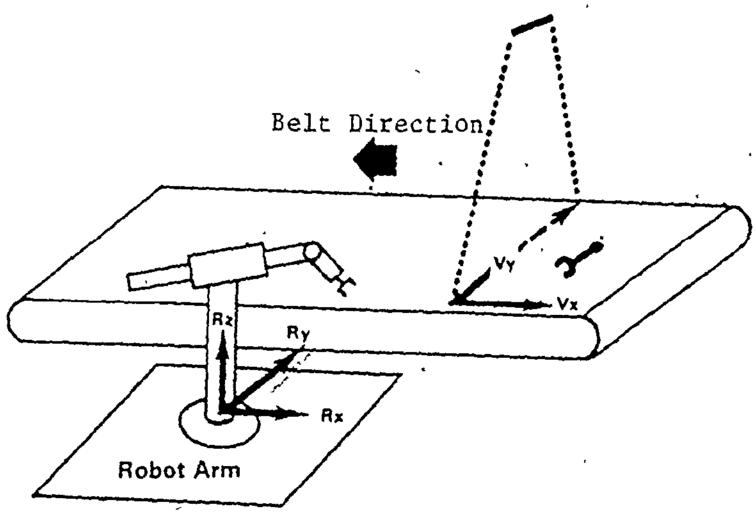
7. sum of x-coordinates squared
8. sum of y-coordinates squared
9. min x-coordinate and associate y-coordinate
10. max x-coordinate and associated y-coordinate
11. min y-coordinate and associated x-coordinate
12. max y-coordinate and associated x-coordinate
13. area of largest hole
14. centroid coordinates of largest hole

Again, the binary image is subjected to connectivity analysis routines and the above features identify individual objects.

System speed limitations are reported in [66] to be imposed only by the cycle time of the associated robot arm.

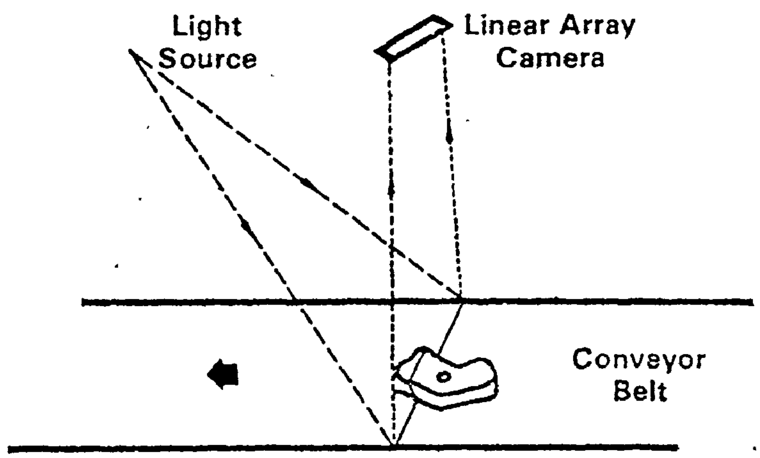


(a) Hardware Schematic

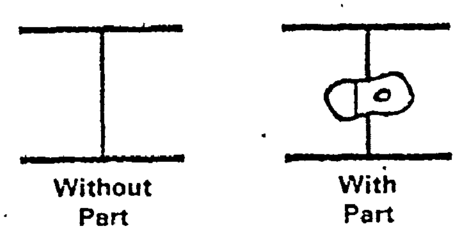


(b) Coordinate Systems

Figure 4.3 CONSIGHT-I System Configuration
(reproduced from [66])



(a) Basic Lighting Principle



(b) Computer's View of Parts

Figure 4.4 CONSIGHT-I Lighting Arrangement
(reproduced from [66])

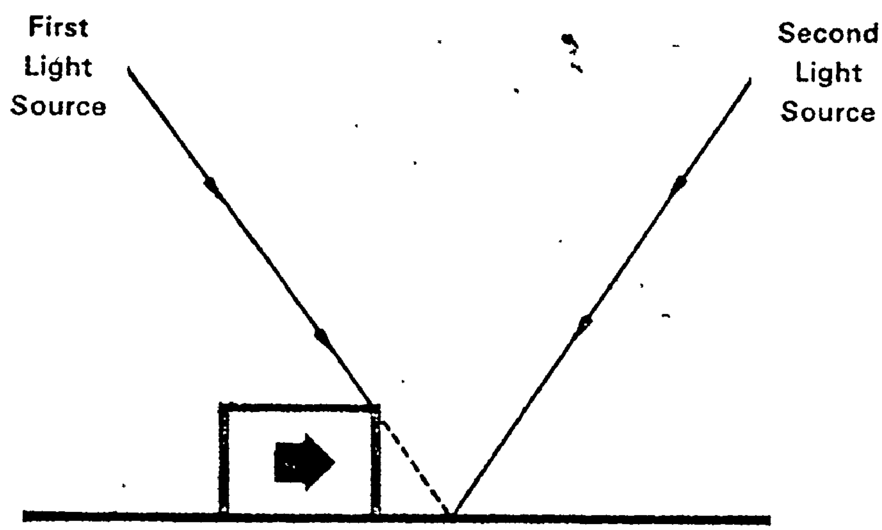


Figure 4.5 Improved Lighting Arrangement
(reproduced from [66])

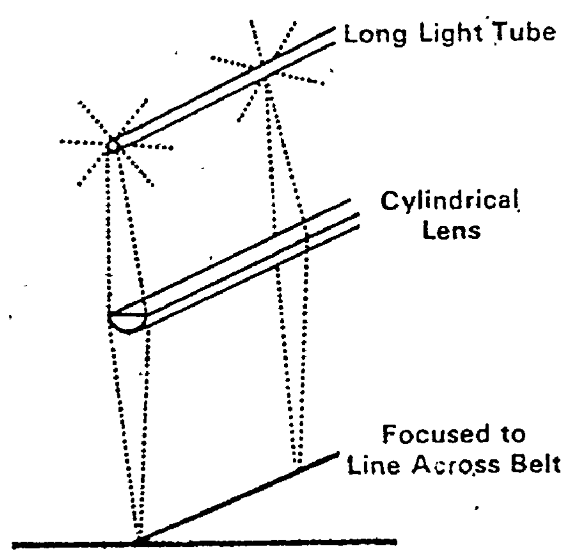


Figure 4.6 Line of Light Generation
(reproduced from [66])

4.5 NBS Vision System

A vision-based control system for a robot has been developed by the National Bureau of Standards (NBS) and is described in [67]. Other visual-servoing work is reported in [42] and [44]. The system consists of an 8-bit microcomputer, a solid state TV camera (resolution 128 x 128) and a strobographic light source which projects a plane of light. Both camera and strobe are mounted at the wrist of a robot arm.

The camera and strobe are aligned as shown in figure 4.7. If the plane of light strikes any object within a distance of 1 meter of the camera, a line of light is reflected as shown in the examples of figure 4.8. Figure 4.9 is the calibration chart used to determine the distance to the object. The reflected line becomes longer and appears nearer to the bottom of the 128 x 128 matrix as the camera moves closer to the object. The resolution of the chart is coarse at greater distances and improves with the proximity of the object.

With the strobe duration controlled by the computer, thresholding of the image data to form silhouettes can be optimized for various ambient lighting conditions. In addition, the camera iris setting is automatically adjusted. Run-length codes are generated in the interval between line scans.

The NBS vision technique is used to enable a robot to "zero in" and grasp an object placed randomly on a table. The first step is to scan the table in a plane parallel to the table surface. From this data, a coarse determination of an object's location is calculated. This estimate allows the robot to approach for a second, more accurate view. This more precise measurement allows the arm to position over the object and to take a third view before the actual acquisition.

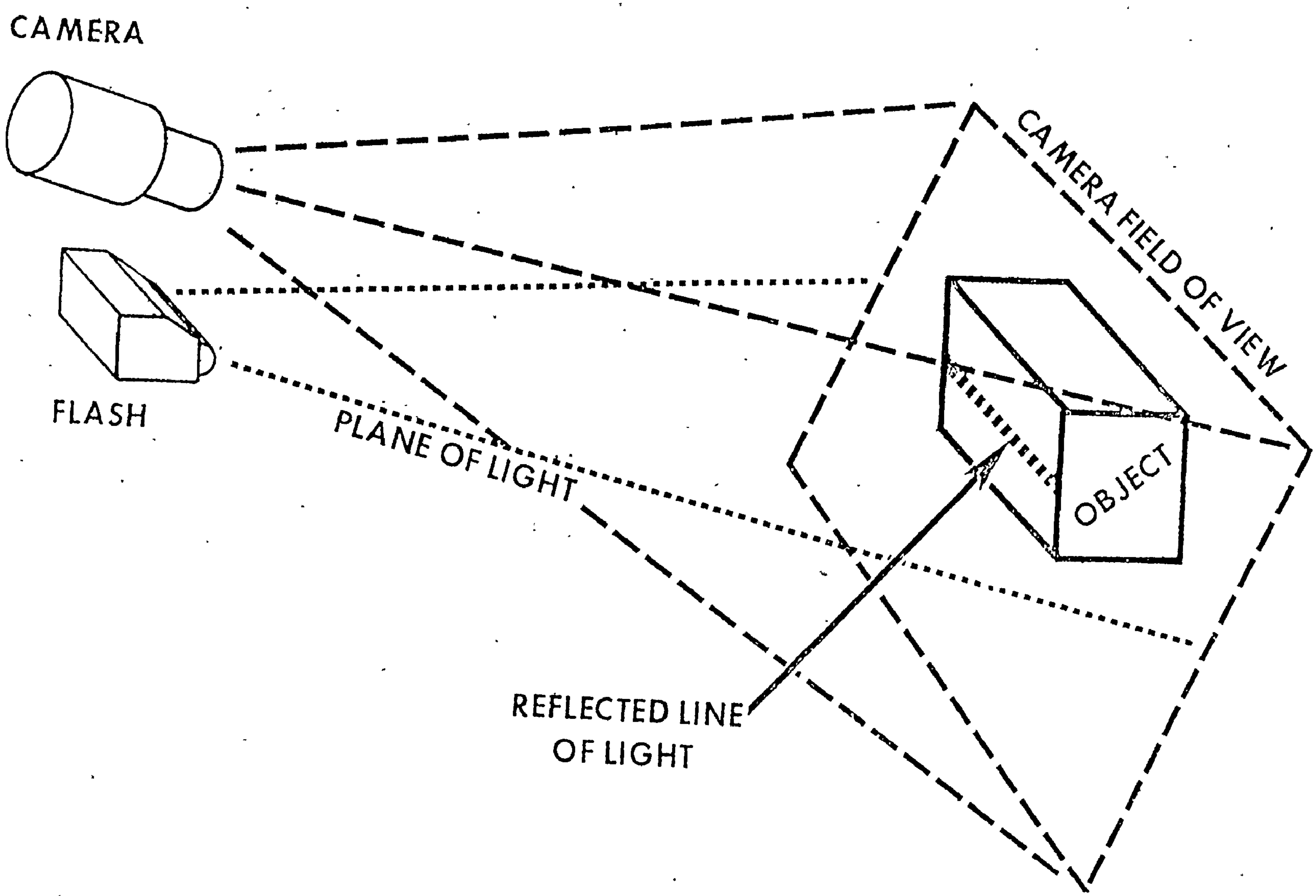


Figure 4.7 Camera and Strobe Arrangement for NBS Vision System
(reproduced from [67])

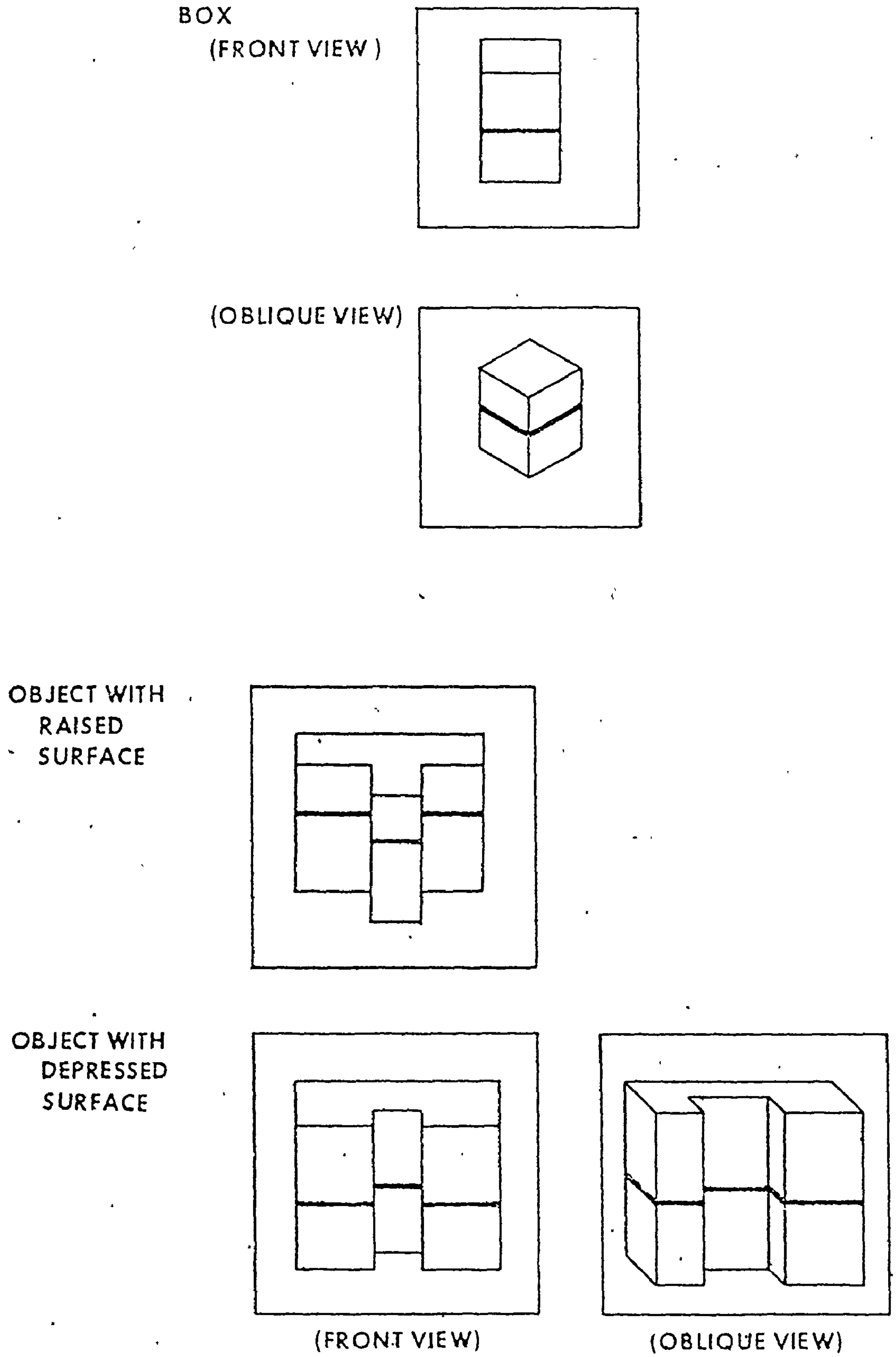


Figure 4.8 Example Objects and the Line Segment Patterns Formed by the Plane of Light as Seen by the Camera (reproduced from [67])

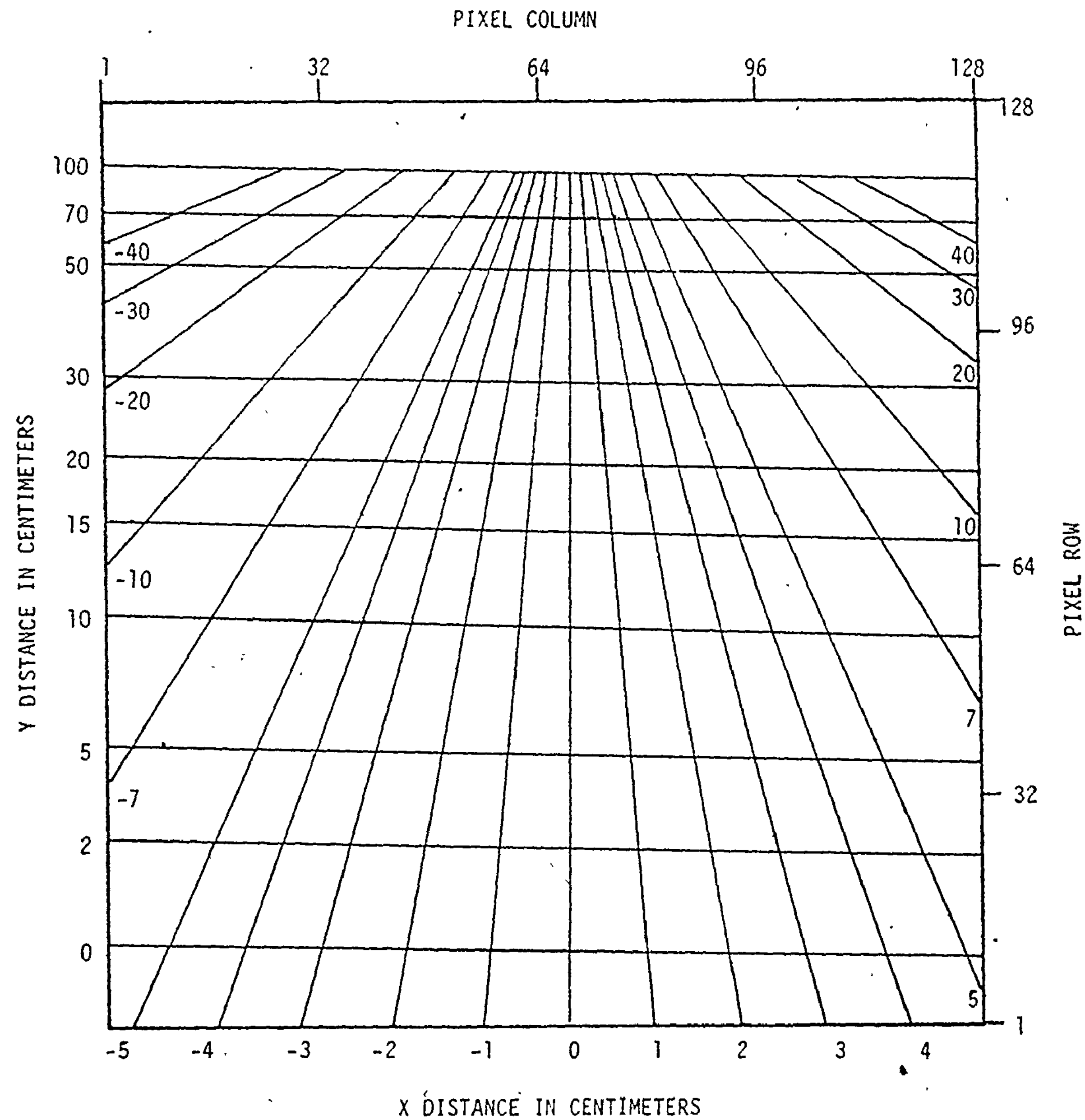


Figure 4.9 The Calibration Chart for the NBS Vision System (reproduced from [67])

CHAPTER 5

AN IMPLEMENTATION

5.1 Introduction

Hardware and software for a basic silhouette processing system have been developed. The hardware is designed around a solid state matrix television camera and a 16-bit microcomputer. Software is written in a high level, structured microprocessor language. A photograph of the system is shown in figure 5.1.

In its present form, the system identifies objects placed individually within the camera's view on the basis of their area. Also, the centroid coordinates within the frame are determined. There are 4 modes of operation, two for the actual recognition process and two which are development aids for the programmer.

Recognition begins in the "Teach" mode in which various objects are shown to the machine, their areas calculated, and an identifier assigned to each shape. Later, in the "Run" mode, any of the learned objects can be reshown to the system at which time the centroid and appropriate identifier are displayed on the CRT.

A "Monitor" mode is available for overseeing the operation of the microcomputer and a "Graphics" mode for displaying the binary image of each object as it resides in the memory.

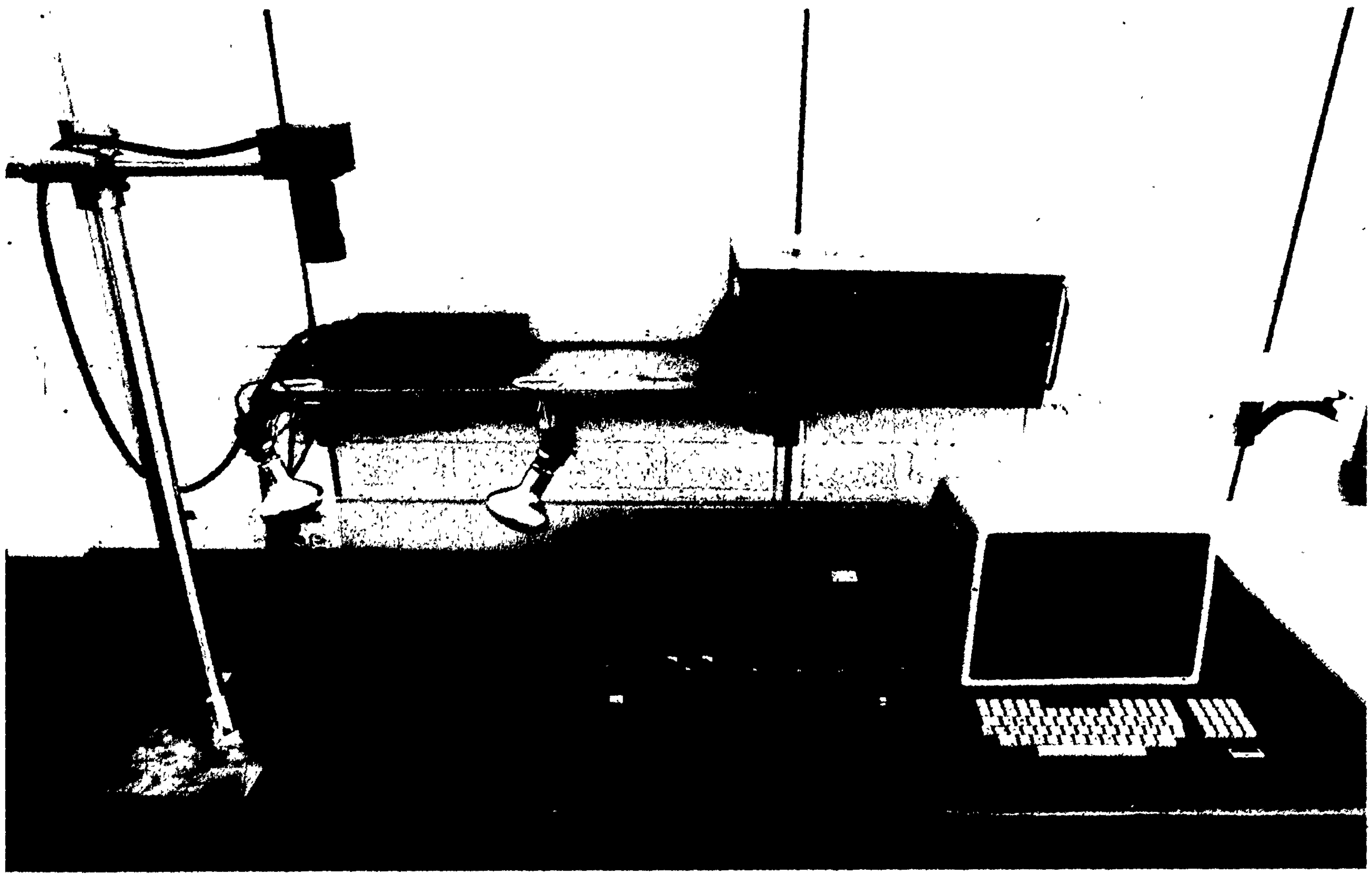


Figure 5.1 Binary Image Processing System

5.2 System Hardware

A computer vision system consists of four modules assembled as shown in figure 5.2. The processing is performed with an Intel single board computer based on an 8086 CPU. Images, stored in the on board RAM, are acquired from a General Electric TN2500 solid state matrix camera and transferred via the Intel SBC 501 DMA controller. A fourth component is a custom built interface between the camera and the DMA board.

Communication between the CPU board and the DMA controller is accomplished by way of the Multibus. Figure 5.3 gives the priority resolution scheme between these two modules. Each is housed in an iSBC cardcage along with the interface module. The BPRN/ pin of the DMA controller is tied to ground so that this board always has the highest priority in accessing the Multibus. Only when a DMA operation is not in progress does the BPRO/ pass on Multibus availability to the CPU board. The custom interface module does not access the Multibus and derives only its power supplies from it.

A brief description of each of the 4 major components follows. Further details of the operation of the camera, DMA controller and CPU board can be found in [68], [69] and [70] respectively.

Solid State Digital T.V. Camera

Increasingly, tasks traditionally relegated to analog devices are being replaced with new digital technology. Such is the case with the General Electric TN2500 sol~~id state dig~~ital T.V. Camera.

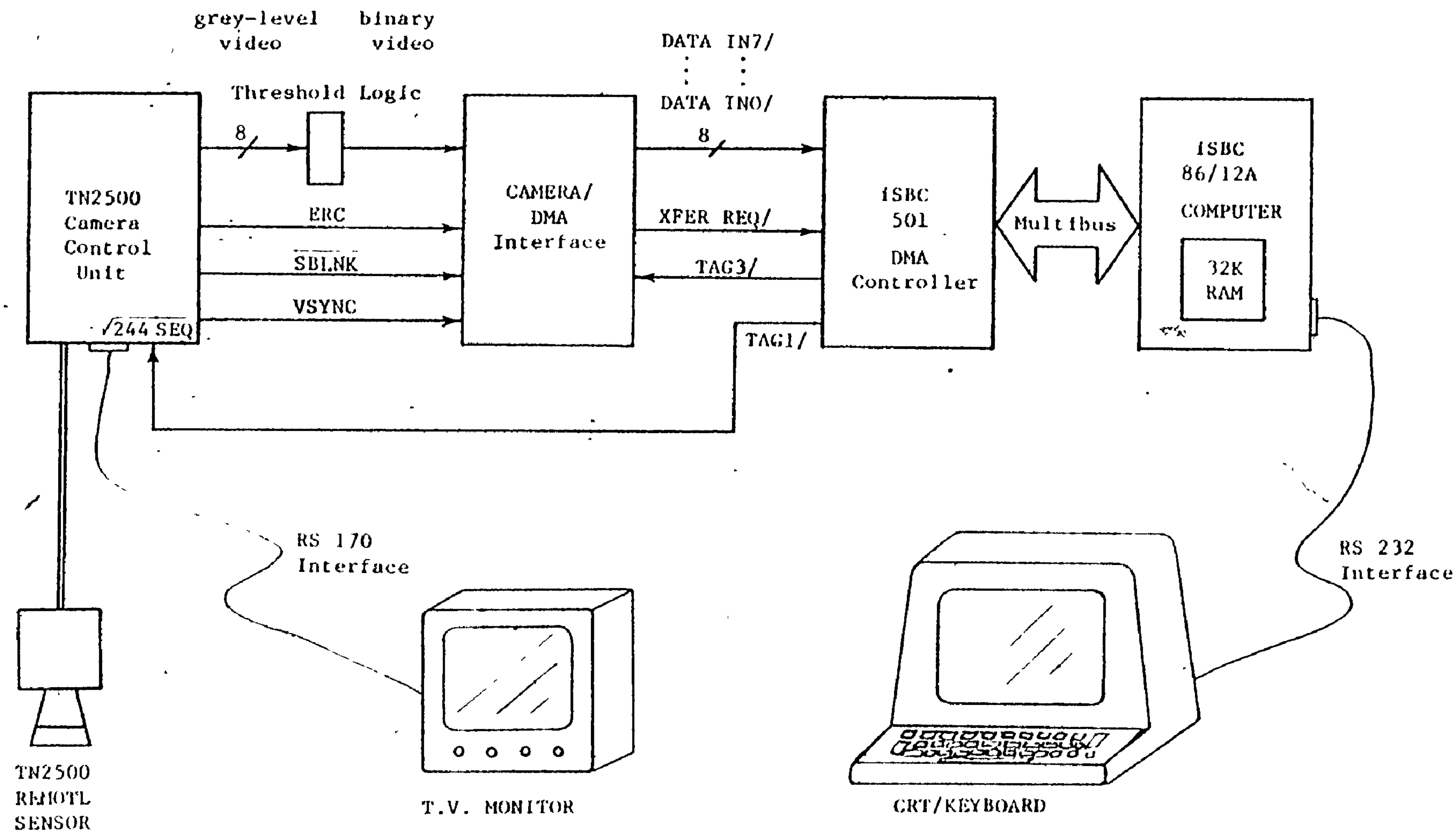


Figure 5.2 System Hardware Configuration

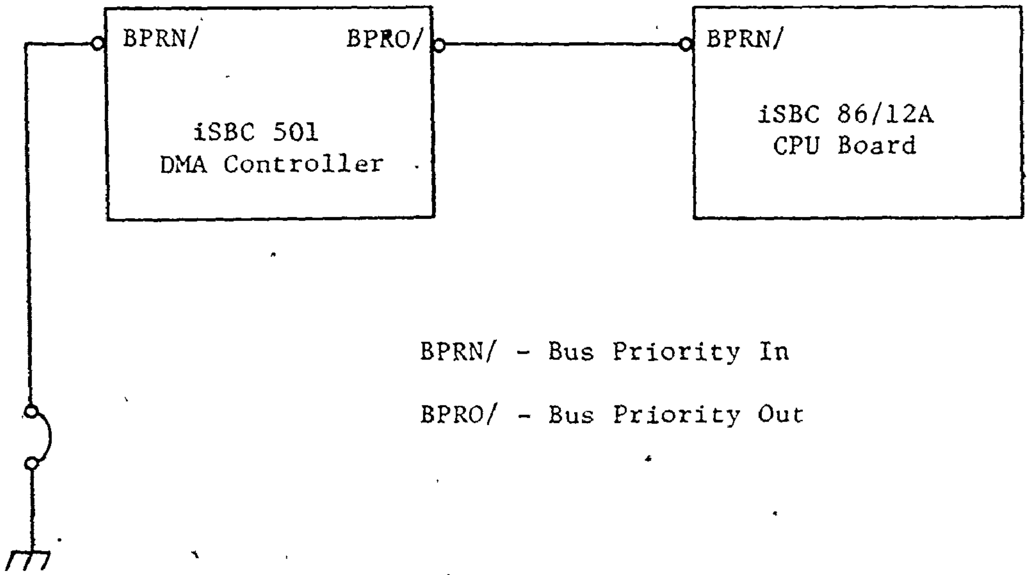


Figure 5.3 Priority Resolution Between Modules

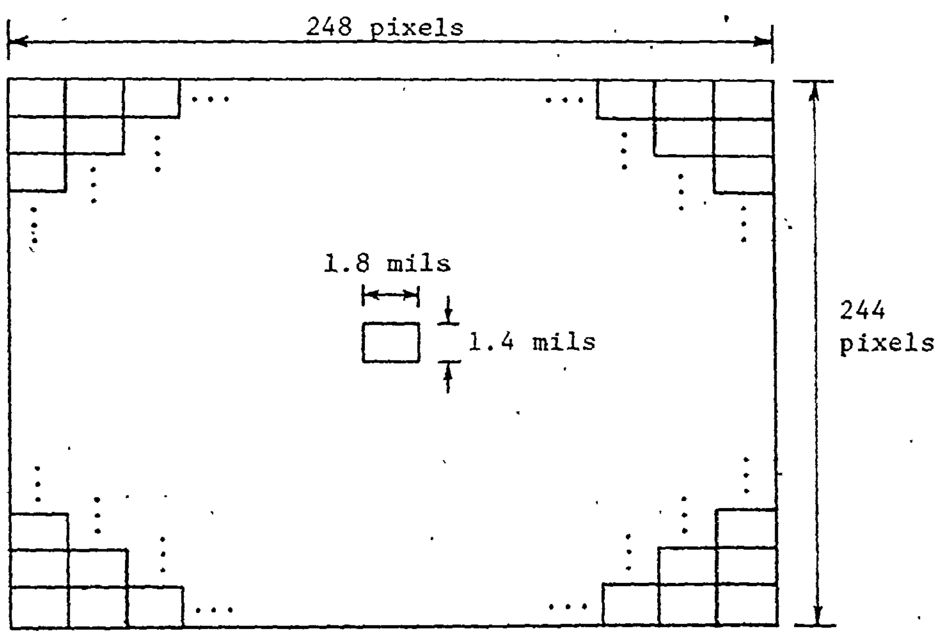


Figure 5.4 CID Sensor Geometry

The heart of the camera is the CID (charge injection device) matrix sensor with more than 60 000 discrete picture elements (pixels) integrated on one chip. These are arranged as 248 rows of 244 elements, each of which is electronically scanned to give an 8-bit representation of the grey-level at that particular point. Figure 5.4 gives the format of the CID sensor.

In addition to the digital output of the camera, a composite video signal is also produced for connection to a television monitor (RS170 scan). Frame rates of 60 Hz (standard TV interlaced format) and 30 Hz (sequential operation) are available.

The TN2500 consists of two units; a remote head which contains the actual sensor and scanning circuitry, and the camera control unit (C.C.U.) which houses timing and processing functions. Connection to the camera is made through the C.C.U. with a 25 pin plug. All signals produced by the C.C.U. at this interface are TTL compatible.

Scanning of the CID imager array is done in a raster fashion and in addition to the 8 video bits, several other signals are produced and made available at the C.C.U. output. Specifically, those utilized in this application include the "element rate clock" (ERC), "synchronized blanking" ($\overline{\text{SBLNK}}$), "vertical sync" (VSYNC), and "interlace $\sqrt{\text{244 sequential}}$"(I $\sqrt{\text{244 seq}}$).

The element rate clock denotes the presentation of the digital output for each pixel being scanned. Synchronized blanking signals the horizontal and vertical retrace intervals during the scan. Vertical sync indicates the beginning of a new scan and interlace $\sqrt{\text{244 seq-}}$

uential changes the mode of the scan from interlaced to sequential. When this pin is taken "LO" the scan of the pixels reverts to a sequential output of every line of the frame from top to bottom. In this mode, the composite video cannot be displayed on the monitor since it expects the interlaced format. In addition, two pins act as 'enables' for the ERC and video data bits.

Of the 248 horizontal pixels only 237 are actually made available during each line scan. This is necessary to keep the line interval compatible with standard television speeds (i.e. 52.614 μs line time and 10.88 μs horizontal retrace). Similarly, only 241½ of the 244 lines are actually available. Timing diagrams for these intervals are given in [68].

A variety of lenses may be attached to the remote camera head. Instead of the 25 mm f1.4 lens which is supplied with the unit, this application makes use of a 75 mm f1.8 lens. At a height of approximately 1.25 m, the field of view with this lens is 23 cm by 17 cm.

DMA Controller

The Intel SBC 501 Direct Memory Access controller board provides direct memory access of 8 (or 16) bit data between a peripheral device (in this case the matrix camera) and the system memory. Up to 2^{16} bytes can be transferred independently of the CPU once it initiates the operation. For this particular system, use was made of the facility to interrupt the CPU and transfer each data byte into RAM and relegate control back to the CPU.

The SBC 501 includes 5 I/O ports through which the CPU can communicate with external devices. In this system the 4-bit tag register was used to control the camera and the camera/DMA interface board. Bit 1 of the tag register sets the camera mode to either interlaced or sequential and bit 3 issues the reset signal to the interface board.

The CPU initiates the transfer by programming a series of registers on the DMA board. The DMA board is assigned a switch selectable base address (chosen as F0H) and output to these registers is relative to this base. The number of bytes to be transferred is first fed to the 16-bit length register (Base + C and Base + D). For this case the number of bytes will be:

$$\frac{232 \times 240}{8} + (29 \times 4) = 7076_{10} = 1BA4_H$$

Of the 237 scanned pixels in each row of the matrix sensor the camera/DMA interface board accepts only the first 232. Similarly, only the first 240 lines are accepted. Division by 8 gives the number of bytes. In addition, there are 4 lines (29 bytes/line) of invalid image data at the beginning.

The control register (Base + A) is set as shown:

0	0	0	0	0	1	1	1

Control Reg ← 07H

Bit 0 - write from device to memory
1 - 8 bit transfer
2 - DMA busy
3 - inhibit transfer
4 - enable interrupts
5 - override CPU access of multibus
6 - } not used
7 - }

The address register (Base + E and Base + F) is set to A000H. Details of this figure are described in the section concerning the CPU board.

Figure 5.5 shows the software for the DMA controller. Following the output of the control registers, bit 1 of the tag register (Base + B) is set low, placing the camera in sequential mode. Next, bit 3 is set low which issues the reset signal to the camera/DMA interface. During the actual transfer the CPU is simply idled by a time delay loop. (After each byte is transferred the length registered is automatically decremented, and the address register incremented). Finally, the tag register is reset to place the camera back into interlace mode and disable the interface board. Output to Base + 9 resets the DMA board.

```
DECLARE BASE LITERALLY '0F0H';

OUTPUT(BASE+0BH)=0;

    OUTPUT(BASE+0CH)=0A4H;
    OUTPUT(BASE+0DH)=1BH;     /*   LENGTH  REGISTER OF DMA BOARD  */

    OUTPUT(BASE+0AH)=07H;     /*   CONTROL REGISTER OF DMA BOARD  */

    OUTPUT(BASE+0EH)=0;
    OUTPUT(BASE+0FH)=0A0H;    /* FIRST ADDRESS REG. OF DMA BOARD  */

    OUTPUT(BASE+0BH)=02H;     /*  CAMERA -> SEQUENTIAL MODE   */

CALL TIME(250);
CALL TIME(250);
CALL TIME(250);

    OUTPUT(BASE+0BH)=0AH;     /*  RESET/GO   */

CALL TIME(250);
CALL TIME(250);
CALL TIME(250);

    OUTPUT(BASE+0BH)=0;        /*  CAMERA -> INTERLACED MODE   */
    OUTPUT(BASE+09H)=0;
```

Fig. 5.5 DMA Controller Software

Single Board Computer

The host processor for the vision system is an Intel SBC 86/12A single board computer. Centered on the 16 bit 8086 CPU, it also includes 32K RAM, 8K EPROM, an RS232 serial communications interface, 3 programmable I/O ports and a programmable interrupt controller and timer. Switches are configured so as to allow multiples of 8K of the on-board RAM to be accessed by other bus masters through the Multibus.

In this application, the top 16K of RAM is made available for image acquisition from the DMA controller. The starting address of the incoming data is set to A000H (see page 2.7 of [70]). Through the appropriate jumpers and switches, this address is mapped into the on-board RAM beginning at location 4000H of the 32K address space. A connection between posts 127 and 128 is the only modification to the factory default jumpers.

The format of each acquired image as it resides in the on-board RAM is shown in figure 5.6. The top of the image is stored at location 4000H with the first valid image data beginning at 4074H. The 240 lines are stored in consecutive locations with the last byte at 5BA3H.

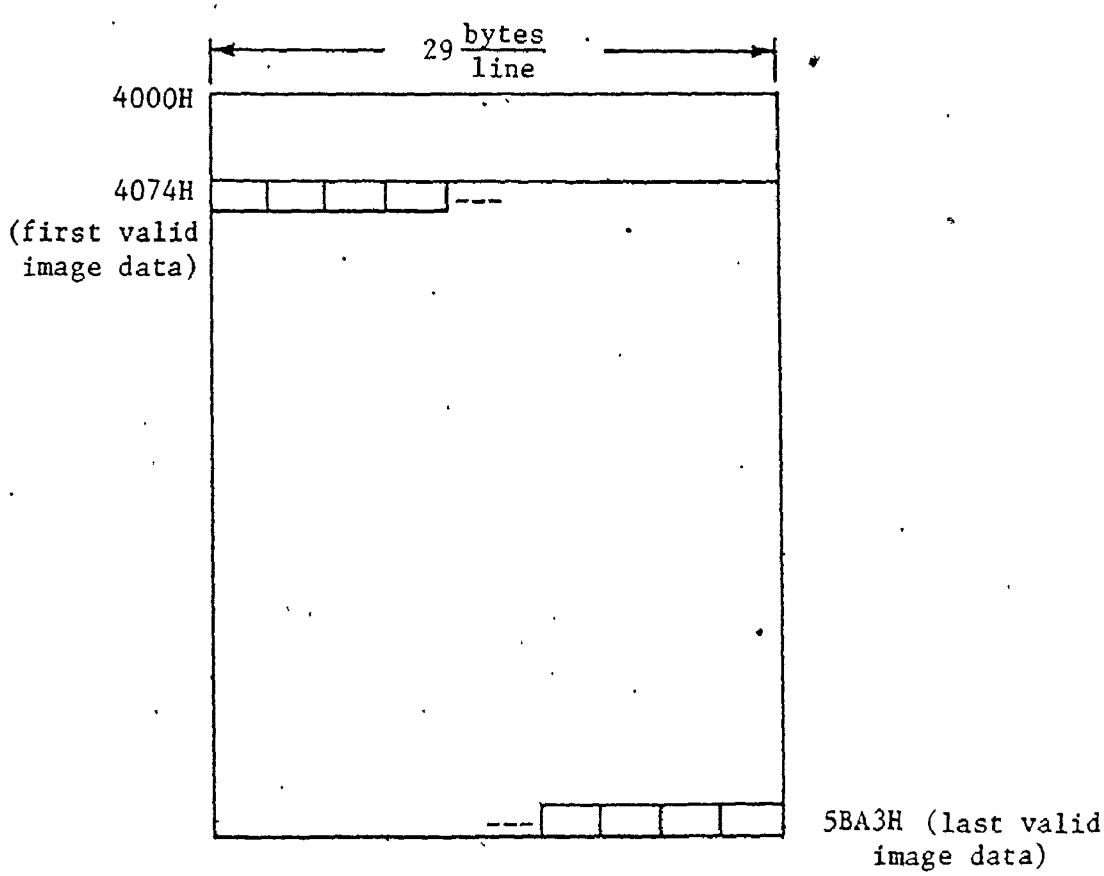


Figure 5.6 Image Data Format in RAM

Camera/DMA Controller Interface

Since the matrix camera produces image data at the rate of 4.5045 MHz and the SBC 501 DMA Controller has an upper limit of 1 MHz data transfer rate, it is necessary to include an interface module between these two units.

The implementation is restricted to dealing with binary images (silhouettes) and therefore one bit is sufficient for representing each pixel, although the camera presents 8 bits for each pixel. Assuming these 8 bits have been thresholded to give just one bit per pixel, we can take advantage of this fact to reduce the 4.5045 MHz rate by a factor of 8. Figure 5.7 illustrates the circuit that accomplishes this task without loss of information. An entire image is collected during one frame with the camera in sequential mode (1/30 sec.).

Before a frame grab is initiated, TAG3/ from the DMA controller is held HI, clearing flip flop A1. With the Q output of A1 LO, flip flop A2 is reset and its Q output disables the element rate clock(ERC). The Q output of A5 is LO, thereby holding XFER REQ/ HI. A XFER REQ/ must be issued to the DMA board each time a byte of data is to be transferred. When TAG3/ goes LO (i.e. a transfer operation is initiated by the CPU), flip flop A1 is enabled and the next occurrence of VSYNC from the camera sets the Q output. VSYNC signifies the beginning of a frame. When the Q output of A1 goes HI flip flop A2 is enabled and the next $\overline{\text{SBLNK}}$ negative transition sets A2.

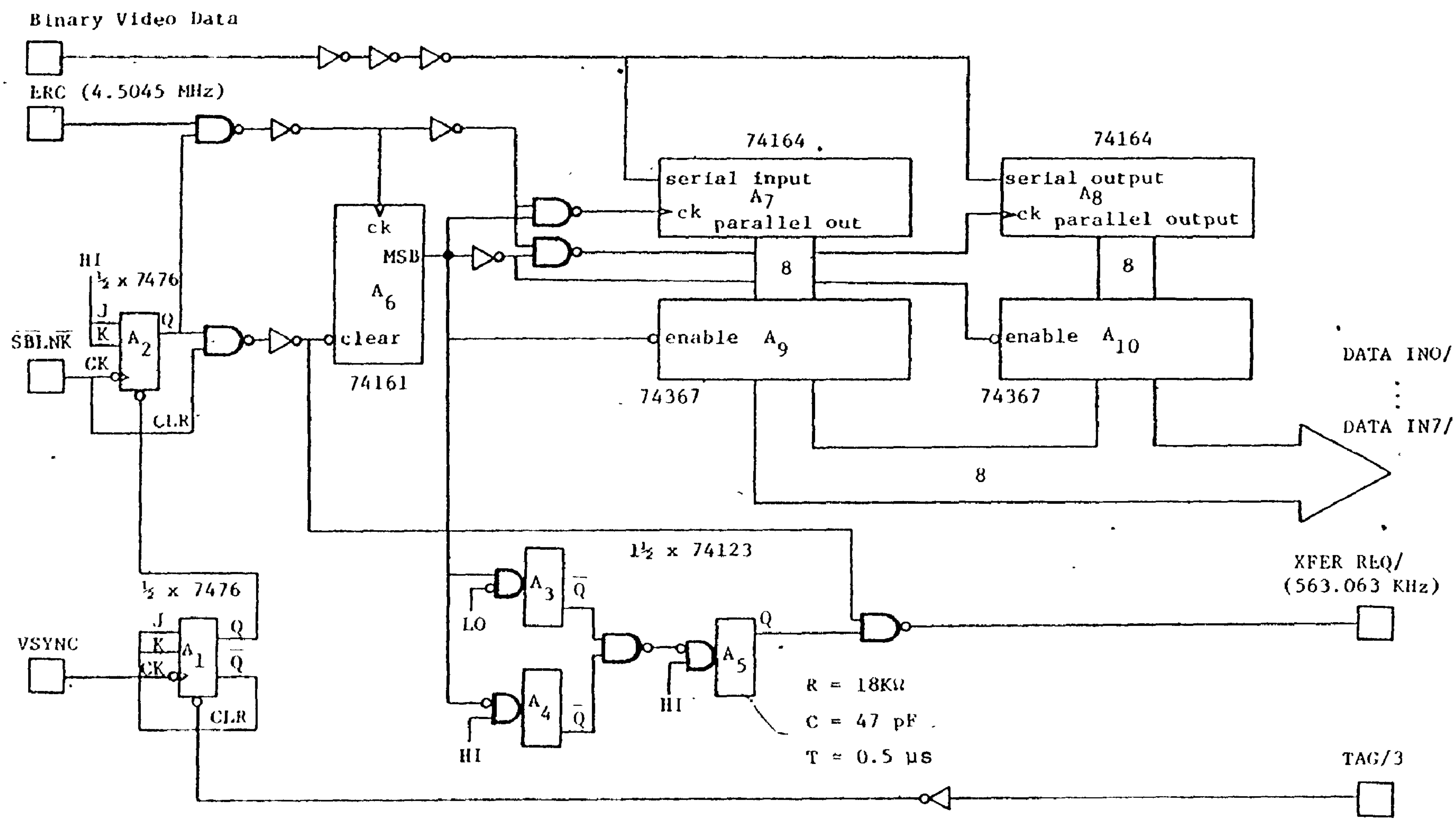


Figure 5.7 Camera/DMA Interface Module

Timing for VSYNC and $\overline{\text{SBLNK}}$ when the camera is operated in the sequential mode is shown in figure 5.8.

The group of monostables A3, A4, and A5 detect any transition of the MSB of counter A6. At each transition a delay of approximately 0.5 μs is initiated followed by a negative pulse of duration 0.5 μs. This pulse provides the XFER REQ/ signal to the DMA board.

A four bit binary counter A6, serial-in/parallel out shift registers A7 and A8 and tri-state buffers A9 and A10 provide the data rate reduction between the camera and the DMA controller. After A2 detects the first negative edge of $\overline{\text{SBLNK}}$, the counter will be cleared synchronously with $\overline{\text{SBLNK}}$. The MSB of the counter is used to gate ERC into one of the shift registers and enable one set of tri-state buffers. The first 8 ERC pulses shift the first 8 video bits into A8. At this time, the MSB of A6 changes and the next 8 video bits will shift into A7. During the time the second 8 bits are filling A7, A10 has been enabled allowing the previous 8 bits to appear in parallel on the data bus to the DMA controller. At the same time, a XFER REQ/ has been generated and the DMA board accepts 8 bits in parallel. This sequence is repeated as long as $\overline{\text{SBLNK}}$ remains HI. When $\overline{\text{SBLNK}}$ drops LO (corresponding to the end of a scan line), the counter is cleared and the process repeats until 1BA4H XFER REQ/'s have been generated to reduce the DMA controllers length register to zero. Since the camera outputs each line of video data twice (fig. 5.9), only alternate lines are collected by toggling A2.

The video data is fed through 3 invertors before entering the shift registers. These act as a buffer as well as a delay.

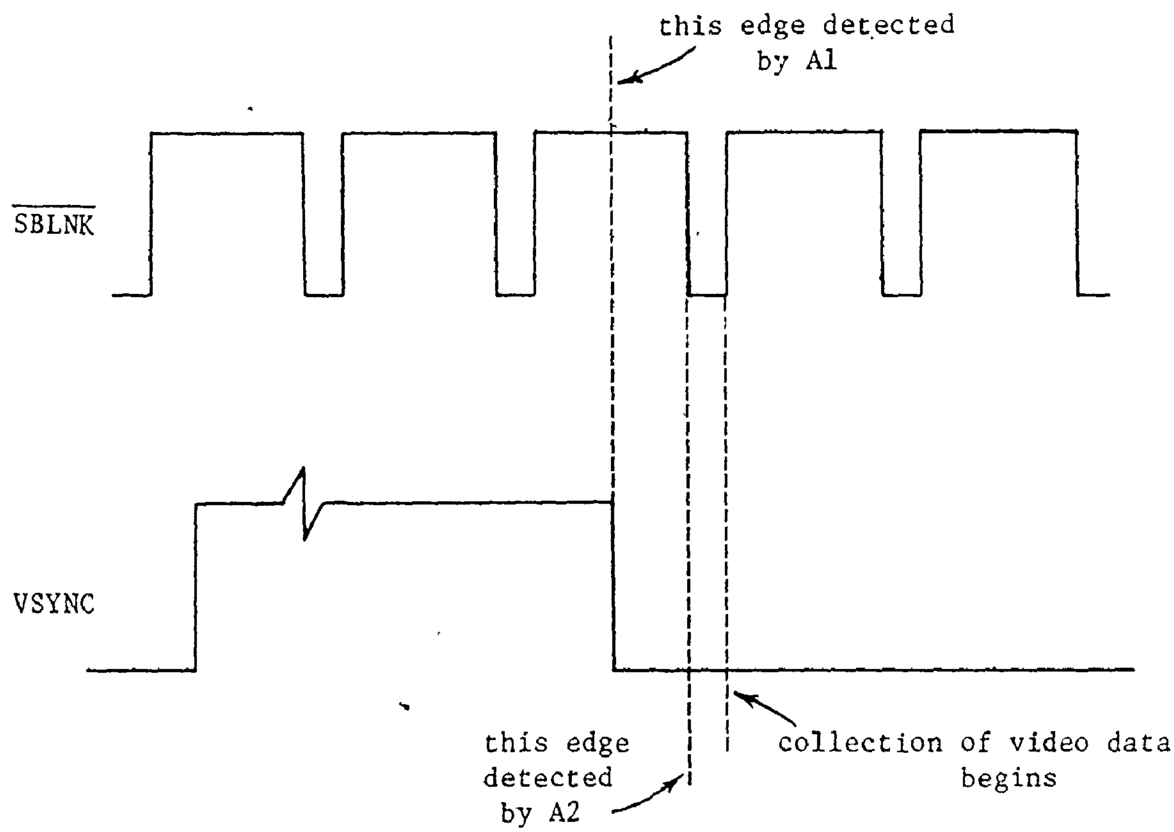


Figure 5.8 Relative Timing of VSYNC & $\overline{SBLNK}$

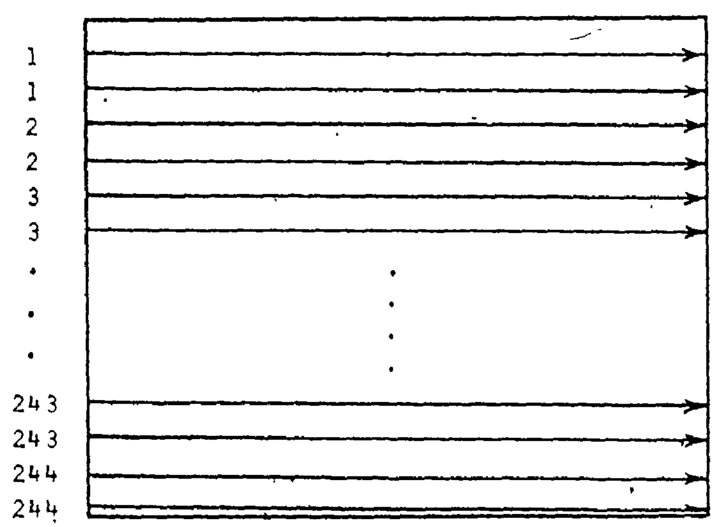


Figure 5.9 Sequential Mode Frame Presentation

The corresponding ERC pulses are subject to 4 gate delays before appearing at the shift registers, and the data must be kept in synchronization with the clocks.

Threshold Logic

The 8 bits which are presented from the camera represent the grey-level at each pixel. To convert to a binary value the threshold logic shown in fig. 5.10 was used. Only the 4 most significant bits from the camera are considered.

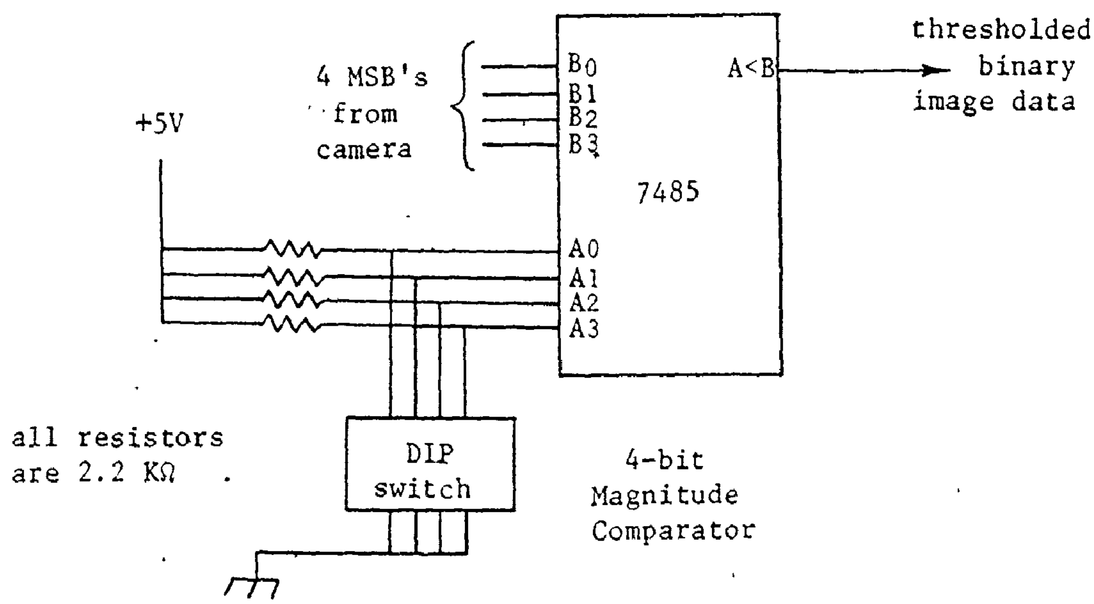


Figure 5.10 Threshold Logic

The DIP switch allows adjustment of the threshold value depending on lighting conditions. Presently, $A_3A_2A_1A_0$ are set to 0011 so that whenever the 4 most significant bits of video data ($B_3B_2B_1B_0$) are > 0011, the thresholded binary value is taken as 1. Otherwise, it is 0. The four least significant bits of video data from the camera were found to be too variable under the existing lighting arrangement.

Lighting

Lighting is a crucial factor in silhouette processing. In practical situations, proper lighting techniques must provide a high contrast, low noise silhouette of 3-dimensional objects. A popular method is to place the objects on a back-lit light table. However, for experimental purposes, some arbitrary shapes were cut from black paper and 2 75 watt spot lamps were used to illuminate them against a grey background. This was found to provide an adequate silhouette for system testing. A sample image is shown in the photograph of figure 5.11.

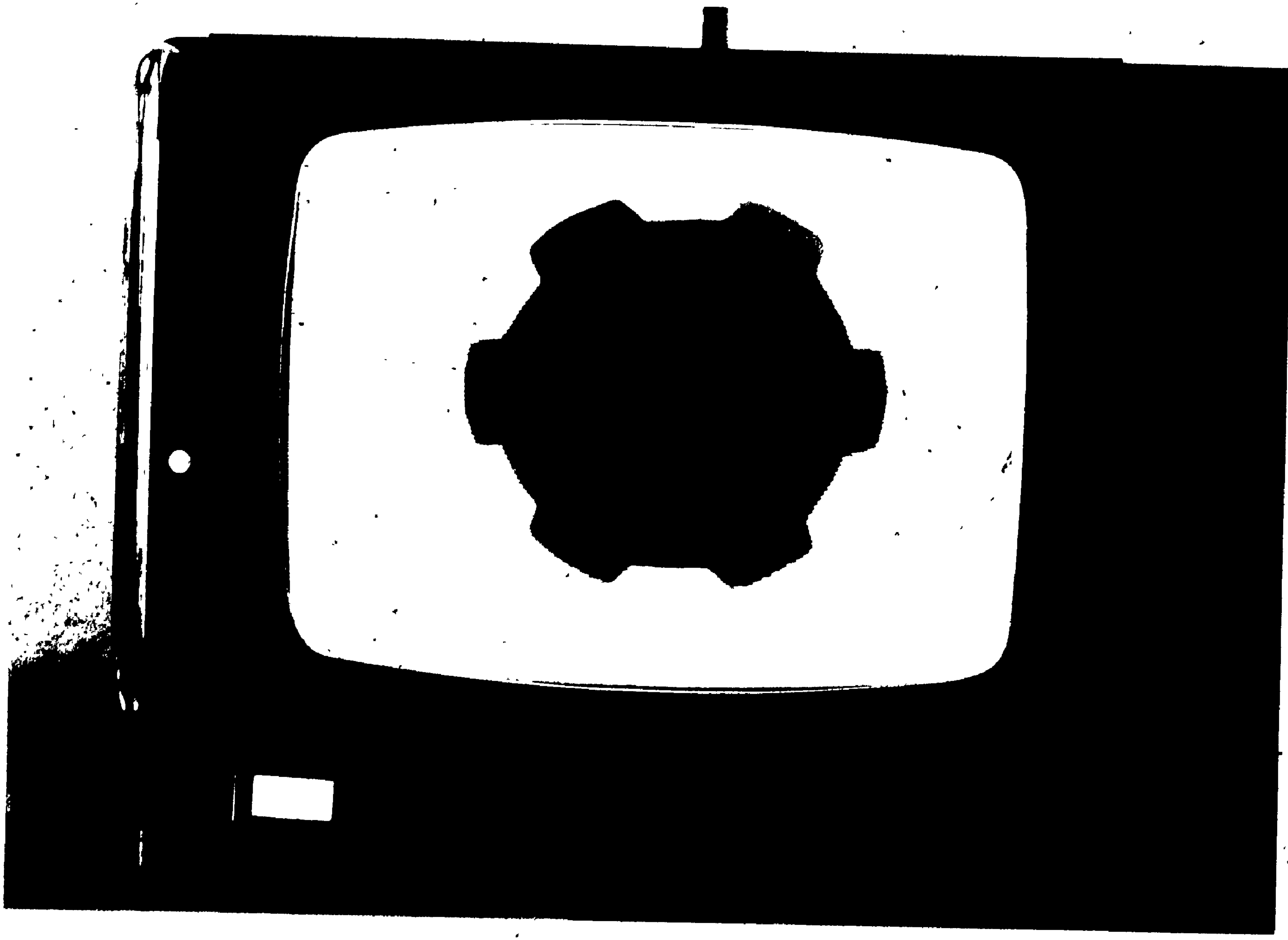


Figure 5.11 Silhouette Generated for Input to the System

5.3 System Software

Programming for the system was written with Intel's PL/M-86 language [71], and developed using Intel's MCS-86 software development utilities [72]. This software is run on an MDS-230 microprocessor development system.

The overall structure of the software is given in Figure 5.12. For convenience, each level of the structure chart is stored as a library and each module is individually contained in separate files. The service library (:FI: SERVIC. LIB) contains utilities for common tasks such as input/output to the CRT, displaying error messages, math functions, etc. These programs are available to all other modules in the system.

In keeping with modern programming practices, a top-down design was incorporated and care was taken in the selection of appropriate data structures. Wherever possible, global variables have been avoided and most routines are written so that they are application independent. Editing, compiled and list files are stored on floppy disks and the final hexadecimal 8086 code is resident on Intel 2716 EPROM.

Details of the Monitor mode are contained in Appendix "A". Other modes of operation are described in the following sections.

Graphics Mode

The Graphics Mode is accessed when the command 'G' is given from the system module. In this mode, locations 4074H to 5BA3H of the on-board RAM are output (in hexadecimal) 29 bytes per line. This

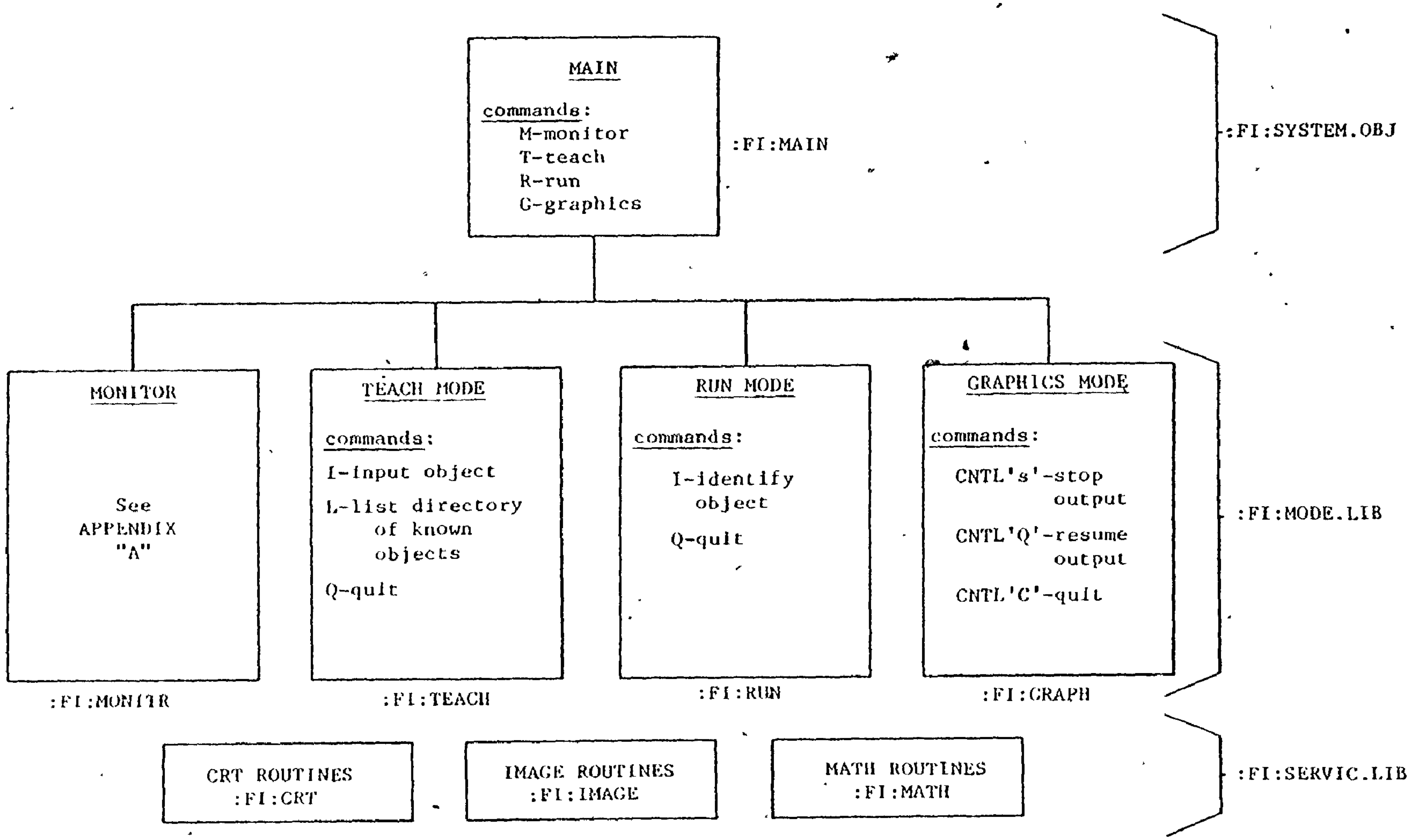


Figure 5.12 System Software Configuration

data represents the current image collected from the matrix camera.

To suspend the output, the user types CNTL 'S'; to resume, CNTL 'Q'; and to terminate output and return to the system the command CNTL 'C' is issued. At the end of the display, control is automatically returned to the system.

Graphics mode is intended as a debugging tool for the programmer. With this aid, one can easily check the binary data as it is presented by the system hardware.

Teach Mode

The first step in the recognition process is the Teach Mode, accessed by typing 'T' on the keyboard. In this mode, numerous silhouettes can be shown to the system, identifiers assigned, and the new object added to the directory of learned shapes.

In its present configuration, Teach Mode also displays the centroid of each object.

To input a new shape, the command 'I' is issued. The system responds with the centroid and calculated area and requests input of an identifier (arbitrary name of the object). This name along with its corresponding 16-bit area is added to the list of known objects, (OBJECT$ID). The 'L' command can be used to display this list on the CRT.

At present, the software is designed to handle 20 objects although this is easily changed. Exit to the system module is achieved by typing 'Q'.

Run Mode

The Run Mode is the second stage of recognition. During the Teach Mode, an object identifier list is constructed. In Run Mode operation, any of the learned objects can now be shown again (in any orientation) and its area is determined. The routine 'SEARCH' scans the object list and returns the area which is closest to that of the object in question. The corresponding object identifier and new centroid are also displayed. This process is initiated with the keyboard command 'I', and can be repeated as desired (with the same or a different previously learned object). Return to the system module is accomplished with the 'Q' command.

Global Variables

The image is stored in the on-board RAM with valid data beginning in absolute location 4074H. For software reference, the matrix IMAGE$ROW (240).IMAGE$COL (29) is positioned at this address. This structure contains 240 rows of 29 8-bit bytes to give a binary matrix of 232 x 240 bits. Any position in the image can now be referenced, 8-bits at a time.

The variables TOPOFIMAGE, BOTTOMOFIMAGE, LEFTOFIMAGE and RIGHTOFIMAGE are stored at absolute locations 7004H to 7007H inclusively. These bytes store the coordinates of the extremities of the image within the above matrix. The intent is to reduce the area and centroid calculation times by skipping locations in the matrix which are known to contain no part of the object.

The area (number of bits which are HI) of the object is stored as the variable AREA at location 7002H (absolute). This value is needed for searching in the object identifier list.

Image Routines Operation

The module "FI:IMAGE" in the service library contains procedures for processing the binary image collected from the camera. Their functions are described below:

EDGE$CHECK - To prevent misidentification, each shape must be presented entirely within the field of view of the camera. This routine scans the four edges of each frame checking for non-zero bits. If any are found, an error message is displayed and processing is immediately terminated for that frame. The left edge of each frame contains 2 HI bits which are always transmitted by the camera. Therefore, on the left edge, the object is considered to be overlapping the edge of the frame if any of the third bits in are set. On the right, if any right most bits are set, the object is rejected. The top and bottom rows must be all zeros to clear this check.

FIND$AREA - The area of the object within the frame is the total number of non-zero bits. In this routine, each row of the image is scanned and the 8-bit value of each byte in the row is used as an index to a table. The table contains the number of bits in each byte and this value is added to the variable

AREA. Also, the appropriate summations for the centroid are carried out. At each pixel which is set, the row and column values within the matrix are accumulated in the 32-bit variables (SUMXHI, SUMXLO) and (SUMYHI, SUMYLO). Finally, the AREA is displayed, and the values:

$$\frac{\text{SUMXHI, SUMXLO}}{\text{AREA}} \quad \text{and} \quad \frac{\text{SUMYHI, SUMYLO}}{\text{AREA}}$$

are calculated and displayed as the centroid.

LOCATE$TOP, LOCATE$BOTTOM, LOCATE$LEFT, LOCATE$RIGHT - These procedures scan the image matrix and determine the extremities of the object within the frame. The appropriate variables are set to these values. (TOPOFIMAGE, BOTTOMOFIMAGE, LEFTOFIMAGE, RIGHTOFIMAGE).

FRAME$GRAB - This procedure programs the registers of the SBC 501 DMA controller, and controls the acquisition of data from the camera. After setting the camera to sequential mode, a time delay is executed while data is being transferred. In the present configuration (no attempt was made to optimize acquisition time) the delay is 150 msec. After data transfer, the camera is returned to interlaced operation to that its composite video signal can be displayed on the monitor.

ISIS-II PL/M-86 V2.0 COMPILATION OF MODULE MAIN
OBJECT MODULE PLACED IN :F1:MAIN.OBJ
COMPILER INVOKED BY: PLM86 :F1:MAIN.SRC LARGE OPTIMIZE(3) TITLE('   SYSTEM COMMAND DISPATCH')

```
 1          MAIN:   DO;

         /*******************************************************************
          *                                                                 *
          *        C O M P U T E R   V I S I O N   S Y S T E M              *
          *                                                                 *
          *           NOVEMBER 1980             JUNE 1980                   *
          *                                     AUTHOR: DAVID CARSON        *
          *                                                                 *
          *       THESE ROUTINES COMPRISE THE EPROM-BASED SOFTWARE          *
          *    FOR A COMPUTER VISION SYSTEM.  THE SYSTEM CONSISTS OF        *
          *    AND ISBC 86/12A MICROCOMPUTER, AN ISBC 501 DMA CON-          *
          *    TROLLER AND A GE TN2500 SOLID STATE DIGITAL TV CAMERA.       *
          *                                                                 *
          *       THIS MODULE IS THE MAIN COMMAND DISPATCH FOR THE 4        *
          *    MODES OF OPERATION, SELECTED WITH THE COMMANDS BELOW.        *
          *                                                                 *
          *       COMMANDS:                                                 *
          *       ---------                                                 *
          *           M - ISBC 86/12 VISION MONITOR                         *
          *           T - TEACH MODE                                        *
          *           R - RUN MODE                                          *
          *           G - GRAPHICS MODE                                     *
          *                                                                 *
          *******************************************************************/
```

```
              $EJECT

              /* -------------------------------------------------------------
                   E X T E R N A L     U S A G E     D E C L A R A T I O N S
                 -------------------------------------------------------------    */

  2   1       CRT$STRING$OUT: PROCEDURE(PTR) EXTERNAL;
  3   2           DECLARE PTR POINTER;
  4   2       END CRT$STRING$OUT;

  5   1       CRTIN: PROCEDURE BYTE EXTERNAL;
  6   2       END CRTIN;

  7   1       ERROR$MESSAGE: PROCEDURE(ERRORNUMBER) EXTERNAL;
  8   2           DECLARE ERRORNUMBER BYTE;
  9   2       END ERROR$MESSAGE;

 10   1       SHOWBYTE: PROCEDURE(BIT$PATTERN) EXTERNAL;
 11   2           DECLARE BIT$PATTERN BYTE;
 12   2       END SHOWBYTE;

 13   1       CHAR$READY: PROCEDURE BYTE EXTERNAL;
 14   2       END CHAR$READY;

 15   1       MONITOR: PROCEDURE EXTERNAL;
 16   2       END MONITOR;

 17   1       TEACH: PROCEDURE EXTERNAL;
 18   2       END TEACH;

 19   1       GRAPHICS: PROCEDURE EXTERNAL;
 20   2       END GRAPHICS;

 21   1       RUN: PROCEDURE EXTERNAL;
 22   2       END RUN;

 23   1           DECLARE
                      SIGNON(*) BYTE DATA (0CH,0DH,0AH,'                         COMPUTER VISION SYSTEM',0AH,0DH,03H),
                      PROMPT(*) BYTE DATA (0DH,0AH,'-',03H),
                      CMND BYTE,
                      C BYTE, BRF WORD,
                      COMMANDS(*) BYTE DATA ('TRMG'),
                      OBJECT$ID$PTR BYTE AT (7000H);
```

```
          $EJECT

          /*    I N I T I A L I Z E     U S A R T     AND     P I T        */

 24   1         DISABLE;                      /*    DISABLE INTERUPTS     */
 25   1         OUTPUT(0DAH)=0H;
 26   1         C=0H;
 27   1         OUTPUT(0DAH)=0H;
 28   1         C=0H;
 29   1         OUTPUT(0DAH)=0H;
 30   1         C=0H;
 31   1         OUTPUT(0DAH)=40H;             /*    USART RESET           */
 32   1         OUTPUT(0DAH)=4EH;             /*    USART MODE CONTROL    */
 33   1         OUTPUT(0D6H)=0B6H;            /*    PIT MODE CONTROL      */
 34   1         OUTPUT(0DAH)=37H;             /*                          */
 35   1         BRF=0008H;
 36   1         OUTPUT(0D4H)=LOW(BRF);        /*    SET BAUD RATE TO      */
 37   1         OUTPUT(0D4H)=HIGH(BRF);       /*         9600             */
 38   1         CALL TIME(100);

          /*      S Y S T E M   C O M M A N D   D I S P A T C H      */

 39   1         OBJECT$ID$PTR=0;             /*  POINTER TO LIST OF LEARNED OBJECTS  */

 40   1         CMND=0;
 41   1         DO WHILE 1;
 42   2              CALL CRT$STRING$OUT(@SIGNON);
 43   2           CALL CRT$STRING$OUT(@PROMPT);                 /*   PROMPT FOR COMMAND   */
 44   2           CMND=LOW(FINDB(@COMMANDS,CRTIN,4));
 45   2           IF CMND=0FFH
                         THEN CALL ERROR$MESSAGE(0);
                         ELSE
 47   2                       DO CASE CMND;
 48   3                            CALL TEACH;
 49   3                            CALL RUN;
 50   3                            CALL MONITOR;
 51   3                            CALL GRAPHICS;
 52   3                       END;
 53   2         END;

 54   1      END MAIN;
```

MODULE INFORMATION:

```
     CODE AREA SIZE     = 011AH    282D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0004H      4D
     MAXIMUM STACK SIZE = 000AH     10D
     122 LINES READ
     0 PROGRAM ERROR(S)
```

```
ISIS-II PL/M-86 V2.0 COMPILATION OF MODULE TEACHMODE
OBJECT MODULE PLACED IN :F1:TEACHS.OBJ
COMPILER INVOKED BY:  PLM86 :F1:TEACHS.SRC LARGE OPTIMIZE(3) TITLE('     TEACH MODE')



   1          TEACH$MODE: DO;

                 /*************************************************************
                  *                                                           *
                  *                    T E A C H   M O D E                    *
                  *                                                           *
                  *   iSBC 8612/A VISION SYSTEM              AUG. 1980        *
                  *                                          AUTHOR: D. CARSON *
                  *                                                           *
                  *    THIS MODE OF OPERATION ALLOWS THE SYSTEM TO BE         *
                  *   SHOWN VARIOUS SHAPES, ASSIGN IDENTIFIERS TO THEM        *
                  *   AND REVIEW THE LIST OF CURRENT MEMBERS IN THE DI-       *
                  *   RECTORY.                                                *
                  *                                                           *
                  *     COMMANDS:                                             *
                  *     ---------                                             *
                  *          I - INPUT NEW MEMBER INTO DIRECTORY              *
                  *          L - LIST MEMBERS OF DIRECTORY                    *
                  *          Q - QUIT                                         *
                  *                                                           *
                  *************************************************************/
```

```
          $EJECT

          /*   -----------------------------------------------------------
                E X T E R N A L    U S A G E    D E C L A R A T I O N S
               -----------------------------------------------------------   */

 2   1        CRT$STRING$OUT: PROCEDURE(PTR) EXTERNAL;
 3   2              DECLARE PTR POINTER;
 4   2        END CRT$STRING$OUT;

 5   1        CRTIN: PROCEDURE BYTE EXTERNAL;
 6   2        END CRTIN;

 7   1        ERROR$MESSAGE: PROCEDURE(ERRORNUMBER) EXTERNAL;
 8   2              DECLARE ERRORNUMBER BYTE;
 9   2        END ERROR$MESSAGE;

10   1        SHOWBYTE: PROCEDURE(BIT$PATTERN) EXTERNAL;
11   2              DECLARE BIT$PATTERN BYTE;
12   2        END SHOWBYTE;

13   1        EDGE$CHECK: PROCEDURE BYTE EXTERNAL;
14   2        END EDGE$CHECK;

15   1        FIND$AREA: PROCEDURE EXTERNAL;
16   2        END FIND$AREA;

17   1        FRAME$GRAB: PROCEDURE EXTERNAL;
18   2        END FRAME$GRAB;

19   1        LOCATE$TOP: PROCEDURE BYTE EXTERNAL;
20   2        END LOCATE$TOP;

21   1        LOCATE$BOTTOM: PROCEDURE BYTE EXTERNAL;
22   2        END LOCATE$BOTTOM;

23   1        LOCATE$LEFT: PROCEDURE(TOP,BOTTOM) BYTE EXTERNAL;
24   2                DECLARE (TOP,BOTTOM) BYTE;
25   2        END LOCATE$LEFT;

26   1        LOCATE$RIGHT: PROCEDURE(TOP,BOTTOM) BYTE EXTERNAL;
27   2                DECLARE (TOP,BOTTOM) BYTE;
28   2        END LOCATE$RIGHT;


29   1        DECLARE
                   AREA WORD AT (7002H),
                   TOP$OF$IMAGE BYTE AT (7004H),      /*  EXTREMITIES OF IMAGE */
                   BOTTOM$OF$IMAGE BYTE AT (7005H),   /*  WITHIN IMAGE MATRIX  */
                   LEFT$OF$IMAGE BYTE AT (7006H),
                   RIGHT$OF$IMAGE BYTE AT (7007H);
```

```
          $EJECT

 30   1   TEACH: PROCEDURE PUBLIC;

          /*    ----------------------------------------------------
                   G L O B A L    D E C L A R A T I O N S
                ----------------------------------------------------    */

 31   2      DECLARE
                TEACH$SIGNON(*) BYTE DATA (0CH,0DH,0AH,'                    TEACH MODE -  SEQUENTIAL OPERATION',
       -  0DH,0AH,03H),
                TEACH$PROMPT(*) BYTE DATA (0DH,0AH,'+',03H),
                TEACH$COMMANDS(*) BYTE DATA ('ILD'),
                CMND BYTE,
                IMAGE$ROW(240) STRUCTURE (IMAGE$COL(29) BYTE) AT (4074H),
                OBJECT$ID(20) STRUCTURE(AREA WORD, NAME(30) BYTE) AT (6000H),
                OBJECT$ID$PTR BYTE AT (7000H);
```

```
          $EJECT

          /*-----------------------------------------------
            -                                              -
            -                  A D D                       -
            -                                              -
            -      THIS ROUTINE ACCEPTS AS INPUT THE NAME  -
            -    OF A NEW OBJECT TO BE ADDED TO THE LIST OF -
            -    KNOWN OBJECTS.  AN ELEMENT OF THE STRUCTURE -
            -    OBJECT$ID IS ASSIGNED THE NAME AND         -
            -    OBJECT$ID$PTR IS INCREMENTED.              -
            -                                              -
            -----------------------------------------------   */

 32   2     ADD: PROCEDURE;

 33   3        DECLARE
                     LISTFULL(*) BYTE DATA(0DH,0AH,'    - OBJECT LIST FULL -',0DH,0AH,03H),
                     CNT BYTE,
                     ADD$PROMPT(*) BYTE DATA(0DH,0AH,'Enter OBJECT IDENTIFIER:  ',07H,03H);

 34   3        IF OBJECT$ID$PTR=21 THEN CALL CRT$STRING$OUT(@LISTFULL);
                                  ELSE
 36   3              DO;
 37   4                 OBJECT$ID(OBJECT$ID$PTR).AREA=AREA;
 38   4                 CALL CRT$STRING$OUT(@ADD$PROMPT);
 39   4                 CNT=0;
 40   4                 DO WHILE CNT<=26;
 41   5                   IF (OBJECT$ID(OBJECT$ID$PTR).NAME(CNT):=CRTIN)=0DH THEN
 42   5                                 DO;
 43   6                                  OBJECT$ID(OBJECT$ID$PTR).NAME(CNT+1)=0AH;
 44   6                                  OBJECT$ID(OBJECT$ID$PTR).NAME(CNT+2)=03H;
 45   6                                  OBJECT$ID$PTR=OBJECT$ID$PTR + 1;
 46   6                                  RETURN;
 47   6                                 END;
 48   5                   CNT=CNT+1;
 49   5                 END;

 50   4                 OBJECT$ID(OBJECT$ID$PTR).NAME(CNT)=0AH;
 51   4                 OBJECT$ID(OBJECT$ID$PTR).NAME(CNT+1)=0DH;
 52   4                 OBJECT$ID(OBJECT$ID$PTR).NAME(CNT+2)=03H;
 53   4                 OBJECT$ID$PTR=OBJECT$ID$PTR+1;
 54   4              END;

 55   3     END ADD;
```

```
              $EJECT

              /*    -----------------------------------------
                    -                                       -
                    -                 L I S T               -
                    -                                       -
                    -     THIS ROUTINE OUTPUTS THE LIST OF  -
                    -   CURRENTLY KNOWN OBJECTS TO THE CRT  -
                    -   ALONG WITH THEIR CORRESPONDING AREAS. -
                    -                                       -
                    -----------------------------------------     */


 56   2       LIST: PROCEDURE;

 57   3   *      DECLARE
                       HEADING(*) BYTE DATA(0DH,0AH,0AH,'AREA       IDENTIFIERS',0AH,0DH,03H),
                       CNT BYTE,
                       BLANKS(*) BYTE DATA('           ',03H),
                       CRLF(*) BYTE DATA(0AH,0DH,03H),
                       EMPTY(*) BYTE DATA(0AH,0DH,'    -NO MEMBERS IN LIST-',0DH,0AH,03H);

 58   3          CALL CRT$STRING$OUT(@HEADING);
 59   3          CALL CRT$STRING$OUT(@CRLF);

 60   3          IF OBJECT$ID$PTR=0 THEN CALL CRT$STRING$OUT(@EMPTY);
                               ELSE
 62   3          DO CNT=0 TO OBJECT$ID$PTR-1;
 63   4                 CALL SHOWBYTE(HIGH(OBJECT$ID(CNT).AREA));
 64   4                 CALL SHOWBYTE(LOW(OBJECT$ID(CNT).AREA));
 65   4                 CALL CRT$STRING$OUT(@BLANKS);
 66   4                 CALL CRT$STRING$OUT(@OBJECT$ID(CNT).NAME);
 67   4          END;

 68   3       END LIST;
```

```
            $EJECT

            /*          C O M M A N D    D I S P A T C H            */


 69   2         CALL CRT$STRING$OUT(@TEACH$SIGNON);

 70   2         DO WHILE 1;
 71   3                 CALL CRT$STRING$OUT(@TEACH$PROMPT);
 72   3                 CMND=LOW(FINDB(@TEACH$COMMANDS,CRTIN,3));
 73   3                 IF CMND=0FFH THEN
 74   3                             CALL ERROR$MESSAGE(0);
                               ELSE
 75   3                             DO CASE CMND;
 76   4                              DO;
 77   5                                CALL FRAME$GRAB;
 78   5                                IF EDGE$CHECK THEN CALL ERROR$MESSAGE(7);
                                                   ELSE
 80   5                                DO;
 81   6                                 IF (TOP$OF$IMAGE:=LOCATE$TOP)=0FFH THEN CALL ERROR$MESSAGE(8);
                                                                       ELSE
 83   6                                              DO;
 84   7                                                BOTTOM$OF$IMAGE=LOCATE$BOTTOM;
 85   7                                                LEFT$OF$IMAGE=LOCATE$LEFT(TOP$OF$IMAGE,BOTTOM$OF$IMAGE);
 86   7                                                RIGHT$OF$IMAGE=LOCATE$RIGHT(TOP$OF$IMAGE,BOTTOM$OF$IMAGE);
 87   7                                                CALL FIND$AREA;
 88   7                                                CALL ADD;
 89   7                                              END;
 90   6                               END;
 91   5                             END;
 92   4                             CALL LIST;
 93   4                             RETURN;
 94   4                             END;
 95   3          END;


 96   2      END TEACH;

 97   1      END TEACH$MODE;


MODULE INFORMATION:

     CODE AREA SIZE     = 0363H    867D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = [illegible]
     MAXIMUM STACK SIZE = 0010H     16D
     233 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```

```
ISIS-II PL/M-86 V2.0 COMPILATION OF MODULE RUNMODE
OBJECT MODULE PLACED IN :F1:RUN.OBJ
COMPILER INVOKED BY:  PLM86 :F1:RUN.SRC LARGE OPTIMIZE(3) TITLE('      RUN MODE')


   1          RUN$MODE:   DO;


                /**************************************************************
                 *                                                            *
                 *                    R U N    M O D E                        *
                 *                                                            *
                 *   iSBC 86/12A VISION SYSTEM            Aug. 1980           *
                 *                                        Author: D. Capson   *
                 *                                                            *
                 *     This mode of operation identifies an image by          *
                 *    finding the nearest match to a given area with          *
                 *    those members currently in the directory.  The          *
                 *    corresponding identifier is then output.                *
                 *                                                            *
                 *     Commands:                                              *
                 *     ---------                                              *
                 *             I - Identify given object                      *
                 *             Q - Quit                                       *
                 *                                                            *
                 **************************************************************/
```

```
            $EJECT

            /*   -------------------------------------------------------------
                   E X T E R N A L   U S A G E   D E C L A R A T I O N S
                 -------------------------------------------------------------   */

 2   1        CRT$STRING$OUT: PROCEDURE(PTR) EXTERNAL;
 3   2            DECLARE PTR POINTER;
 4   2        END CRT$STRING$OUT;

 5   1        CRTIN: PROCEDURE BYTE EXTERNAL;
 6   2        END CRTIN;

 7   1        ERROR$MESSAGE: PROCEDURE(ERRORNUMBER) EXTERNAL;
 8   2            DECLARE ERRORNUMBER BYTE;
 9   2        END ERROR$MESSAGE;

10   1        SHOWBYTE: PROCEDURE(BIT$PATTERN) EXTERNAL;
11   2             DECLARE BIT$PATTERN BYTE;
12   2        END SHOWBYTE;

13   1        EDGE$CHECK: PROCEDURE BYTE EXTERNAL;
14   2        END EDGE$CHECK;

15   1        FIND$AREA: PROCEDURE EXTERNAL;
16   2        END FIND$AREA;

17   1        FRAME$GRAB: PROCEDURE EXTERNAL;
18   2        END FRAME$GRAB;

19   1        LOCATE$TOP: PROCEDURE BYTE EXTERNAL;
20   2        END LOCATE$TOP;

21   1        LOCATE$BOTTOM: PROCEDURE BYTE EXTERNAL;
22   2        END LOCATE$BOTTOM;

23   1        LOCATE$LEFT: PROCEDURE(TOP,BOTTOM) BYTE EXTERNAL;
24   2                DECLARE (TOP,BOTTOM) BYTE;
25   2        END LOCATE$LEFT;

26   1        LOCATE$RIGHT: PROCEDURE(TOP,BOTTOM) BYTE EXTERNAL;
27   2                DECLARE (TOP,BOTTOM) BYTE;
28   2        END LOCATE$RIGHT;


29   1        DECLARE
                    AREA WORD AT (7002H),
                    TOP$OF$IMAGE BYTE AT (7004H),
                    BOTTOM$OF$IMAGE BYTE AT (7005H),
                    LEFT$OF$IMAGE BYTE AT (7006H),
                    RIGHT$OF$IMAGE BYTE AT (7007H);
```

```
          $EJECT

 30   1   RUN: PROCEDURE PUBLIC;

          /*   ----------------------------------------------------
                  G L O B A L    D E C L A R A T I O N S
               ----------------------------------------------------   */

 31   2      DECLARE
                RUN$SIGNON(*) BYTE DATA (0CH,0DH,0AH,'             RUN MODE -  SEQUENTIAL OPERATION',0DH,0
        -   AH,03H),
                RUN$PROMPT(*) BYTE DATA (0DH,0AH,')',03H),
                RUN$COMMANDS(*) BYTE DATA ('IQ'),
                CMND BYTE,
                IMAGE$ROW(240) STRUCTURE (IMAGE$COL(29) BYTE) AT (4074H),
                OBJECT$ID(20) STRUCTURE(AREA WORD, NAME(30) BYTE) AT (6000H),
                OBJECT$ID$PTR BYTE AT (7000H);
```

```
            $EJECT

            /*     ---------------------------------------------
                   -             S E A R C H                   -
                   -                                           -
                   -   THIS ROUTINE STEPS THROUGH THE LIST     -
                   -  OF OBJECT IDENTIFIERS AND OUTPUTS THE    -
                   -  NAME OF THE OBJECT WHOSE AREA IS CLOS-   -
                   -  EST TO THAT OF THE GIVEN IMAGE.          -
                   -                                           -
                   ---------------------------------------------        */

  32   2    SEARCH: PROCEDURE;

  33   3       DECLARE
                        CNT BYTE,
                        DIFF WORD,
                        MINDIFF WORD,
                        MINDIFF$CNT BYTE,
                        CRLF(*) BYTE DATA(0DH,0AH,0AH,03H),
                        EMPTY(*) BYTE DATA(0DH,0AH,'    -NO MEMBERS IN LIST-',0AH,0DH,03H);

  34   3       MINDIFF=0FFFFH;

  35   3       IF OBJECT$ID$PTR=0 THEN CALL CRT$STRING$OUT(@EMPTY);
                                    ELSE
  37   3       DO CNT=0 TO OBJECT$ID$PTR-1;
  38   4            IF AREA>=OBJECT$ID(CNT).AREA THEN DIFF=AREA-OBJECT$ID(CNT).AREA;
  40   4                                      ELSE DIFF=OBJECT$ID(CNT).AREA-AREA;
  41   4            IF DIFF< MINDIFF THEN
  42   4                              DO;
  43   5                               MINDIFF=DIFF;
  44   5                               MINDIFF$CNT=CNT;
  45   5                              END;
  46   4       END;

  47   3       CALL CRT$STRING$OUT(@CRLF);

  48   3           CALL CRT$STRING$OUT(@OBJECT$ID(MINDIFF$CNT).NAME);

  49   3    END SEARCH;
```

```
              $EJECT

              /*           C O M M A N D    D I S P A T C H            */


 50   2        CALL CRT$STRING$OUT(@RUN$SIGNON);

 51   2        DO WHILE 1;
 52   3             CALL CRT$STRING$OUT(@RUN$PROMPT);
 53   3             CMND=LOW(FINDB(@RUN$COMMANDS,CRTIN,2));
 54   3             IF CMND=0FFH THEN
 55   3                     CALL ERROR$MESSAGE(0);
                           ELSE
 56   3                     DO CASE CMND;
 57   4                       DO;
 58   5                        CALL FRAME$GRAB;
 59   5                        IF EDGE$CHECK THEN CALL ERROR$MESSAGE(7);
                                       ELSE
 61   5                         DO;
 62   6                         IF (TOP$OF$IMAGE:=LOCATE$TOP)=0FFH THEN CALL ERROR$MESSAGE(8);
                                                    ELSE
 64   6                                  DO;
 65   7                                   BOTTOM$OF$IMAGE=LOCATE$BOTTOM;
 66   7                                   LEFT$OF$IMAGE=LOCATE$LEFT(TOP$OF$IMAGE,BOTTOM$OF$IMAGE);
 67   7                                   RIGHT$OF$IMAGE=LOCATE$RIGHT(TOP$OF$IMAGE,BOTTOM$OF$IMAGE
      );
 68   7                                   CALL FIND$AREA;
 69   7                                   CALL SEARCH;
 70   7                                  END;
 71   6                         END;
 72   5                       END;
 73   4                       RETURN;
 74   4                     END;
 75   3        END;


 76   2      END RUN;

 77   1      END RUN$MODE;


MODULE INFORMATION:

     CODE AREA SIZE     = 020FH    527D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0007H      7D
     MAXIMUM STACK SIZE = 0010H     16D
     189 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```

```
ISIS-II PL/M-86 V2.0 COMPILATION OF MODULE GRAPHICSMODE
OBJECT MODULE PLACED IN :F1:GRAPH.OBJ
COMPILER INVOKED BY:  PLM86 :F1:GRAPH.SRC LARGE OPTIMIZE(3) TITLE('       GRAPHICS')


   1          GRAPHICS$MODE: DO;


                    /************************************************************
                     *                                                          *
                     *                      G R A P H I C S                     *
                     *                                                          *
                     *      THIS ROUTINE OUTPUTS TO THE CRT, THE HEX CONTENTS    *
                     *     OF THE MEMORY CONTAINING THE ACQUIRED IMAGE.          *
                     *     IMAGE SPATIAL RELATIONSHIPS ARE PRESERVED BY OUT-     *
                     *     PUTTING 29 BYTES PER LINE.                            *
                     *                                                          *
                     ************************************************************/


              /*    ------------------------------------------------------------
                       E X T E R N A L   U S A G E   D E C L A R A T I O N S
                    ------------------------------------------------------------   */

   2   1      CRT$STRING$OUT: PROCEDURE(PTR) EXTERNAL;
   3   2          DECLARE PTR POINTER;
   4   2      END CRT$STRING$OUT;

   5   1      SHOWBYTE: PROCEDURE(BIT$PATTERN) EXTERNAL;
   6   2          DECLARE BIT$PATTERN BYTE;
   7   2      END SHOWBYTE;

   8   1      CHAR$READY: PROCEDURE BYTE EXTERNAL;
   9   2      END CHAR$READY;

  10   1      CRTIN: PROCEDURE BYTE EXTERNAL;
  11   2      END CRTIN;
```

```
          $EJECT


 12   1      GRAPHICS: PROCEDURE PUBLIC;

 13   2         DECLARE
                   CRLF(*) BYTE DATA(0DH,0AH,03H),
                   IR(240) STRUCTURE(IC(29) BYTE) AT (4074H),
                   X BYTE,
                   Y BYTE,
                   CHAR BYTE;

 14   2         DO Y=0 TO 239;
 15   3            CALL CRT$STRING$OUT(@CRLF);
 16   3            IF CHAR$READY THEN
 17   3               DO;
 18   4               IF ((CHAR:=CRTIN)=13H) THEN          /*   CHECK FOR CNTRL'S'  */
 19   4                      DO WHILE CHAR<>11H;           /*    ''    '' CNTRL'Q'  */
 20   5                         IF CHAR$READY THEN         /*    ''    '' CNTRL'C'  */
 21   5                         IF((CHAR:=CRTIN)=03H) THEN
 22   5                                                    RETURN;
 23   5                      END;
 24   4               END;
 25   3         DO X=0 TO 28;
 26   4         CALL SHOW$BYTE(IR(Y).IC(X));
 27   4         END;
 28   3         END;

 29   2      END GRAPHICS;

 30   1      END GRAPHICS$MODE;


MODULE INFORMATION:

     CODE AREA SIZE     = 0096H    150D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0003H      3D
     MAXIMUM STACK SIZE = 000CH     12D
     67 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```

```
ISIS-II PL/M-86 V2.0 COMPILATION OF MODULE IMAGEROUTINES
OBJECT MODULE PLACED IN :F1:IMAGE.OBJ
COMPILER INVOKED BY:  PLM86 :F1:IMAGE.SRC LARGE OPTIMIZE(3) TITLE('     IMAGE ROUTINES')



   1          IMAGE$ROUTINES: DO;


              /**********************************************************
               *                                                        *
               *         I M A G E    R O U T I N E S                   *
               *                                                        *
               *   iSBC 86/12 VISION SYSTEM          AUGUST 1980        *
               *                                     AUTHOR: D. CAPSON  *
               *                                                        *
               *      THIS MODULE CONTAINS THE NECESSARY SOFTWARE       *
               *   FOR ACQUISITION AND PROCESSING OF DIGITAL IMAGE      *
               *   DATA FROM THE CAMERA.                                *
               *                                                        *
               *             ROUTINES INCLUDED:                         *
               *            --------------------                        *
               *                 EDGE$CHECK                             *
               *                 FIND$AREA                              *
               *                 LOCATE$TOP                             *
               *                 LOCATE$BOTTOM                          *
               *                 LOCATE$LEFT                            *
               *                 LOCATE$RIGHT                           *
               *                 FRAME$GRAB                             *
               *                                                        *
               **********************************************************/
```

```
             $EJECT


             /*   --------------------------------------------------------------
                    E X T E R N A L    U S A G E    D E C L A R A T I O N S
                  --------------------------------------------------------------    */


  2   1         CRT$STRING$OUT: PROCEDURE(PTR) EXTERNAL;
  3   2              DECLARE PTR POINTER;
  4   2         END CRT$STRING$OUT;

  5   1         CRTIN: PROCEDURE BYTE EXTERNAL;
  6   2         END CRTIN;

  7   1         ERROR$MESSAGE: PROCEDURE(ERRORNUMBER) EXTERNAL;
  8   2              DECLARE ERRORNUMBER BYTE;
  9   2         END ERROR$MESSAGE;

 10   1         SHOWBYTE: PROCEDURE(BIT$PATTERN) EXTERNAL;
 11   2               DECLARE BIT$PATTERN BYTE;
 12   2         END SHOWBYTE;

 13   1         CRT$BYTE$OUT: PROCEDURE(CHAR) EXTERNAL;
 14   2               DECLARE CHAR BYTE;
 15   2         END CRT$BYTE$OUT;

 16   1         DIVIDE$32$BITS: PROCEDURE(DIVIDENDHI,DIVIDENDLO,DIVISORHI,DIVISORLO) WORD EXTERNAL;
 17   2               DECLARE
                        (DIVIDENDHI,DIVIDENDLO,DIVISORHI,DIVISORLO) WORD;
 18   2         END DIVIDE$32$BITS;

 19   1         OUTPUT$DECIMAL: PROCEDURE(BINARY$BYTE) EXTERNAL;
 20   2               DECLARE  BINARY$BYTE BYTE;
 21   2         END OUTPUT$DECIMAL;


 22   1         DECLARE
                      IMAGE$ROW(240) STRUCTURE (IMAGE$COL(29) BYTE) AT (4074H),
                      TOP$OF$IMAGE BYTE AT (7004H),
                      BOTTOM$OF$IMAGE BYTE AT (7005H),
                      LEFT$OF$IMAGE BYTE AT (7006H),
                      RIGHT$OF$IMAGE BYTE AT (7007H),
                      AREA WORD AT (7002H);
```

```
              $EJECT
              /*       ----------------------------------------------------------
                       -                                                        -
                       -                  E D G E     C H E C K                 -
                       -                                                        -
                       -   CHECKS FOR OBJECT INTERSECTION OF EDGES OF FRAME.    -
                       -                                                        -
                       ----------------------------------------------------------     */

 23   1       EDGE$CHECK: PROCEDURE BYTE PUBLIC;

 24   2          DECLARE
                       ROWCNT WORD,
                       COLCNT BYTE,
                       TOP BYTE,
                       BOTTOM BYTE,
                       LEFT BYTE,
                       RIGHT BYTE;

 25   2          LEFT,RIGHT,BOTTOM,TOP=0;

              /*         C H E C K    S I D E S                    */

 26   2          DO ROWCNT=0 TO 239;
 27   3             IF IMAGE$ROW(ROWCNT).IMAGE$COL(0)<>0F0H THEN      /*  LEFT EDGE     */
 28   3                                                LEFT=0FFH;
 29   3             IF IMAGE$ROW(ROWCNT).IMAGE$COL(28)<>0F0H THEN     /*  RIGHT  EDGE   */
 30   3                                                 RIGHT=0FFH;
 31   3             IMAGE$ROW(ROWCNT).IMAGE$COL(0)=IMAGE$ROW(ROWCNT).IMAGE$COL(0) AND 0FH;
 32   3          END;

              /*          C H E C K     T O P    A N D    B O T T O M          */

 33   2          DO COLCNT=1 TO 27;
 34   3             IF IMAGE$ROW(0).IMAGE$COL(COLCNT)<>0 THEN       /*   TOP EDGE       */
 35   3                                              TOP=0FFH;
 36   3             IF IMAGE$ROW(239).IMAGE$COL(COLCNT)<> 0 THEN    /*   BOTTOM EDGE    */
 37   3                                              BOTTOM=0FFH;
 38   3          END;

 39   2          IF LEFT THEN CALL ERROR$MESSAGE(3);
 41   2          IF RIGHT THEN CALL ERROR$MESSAGE(4);
 43   2          IF TOP THEN CALL ERROR$MESSAGE(5);
 45   2          IF BOTTOM THEN CALL ERROR$MESSAGE(6);

 47   2          IF TOP OR BOTTOM OR LEFT OR RIGHT THEN RETURN 0FFH;
 49   2          RETURN 0;

 50   2       END EDGE$CHECK;
```

```
                $EJECT

                /*      -------------------------------------------------------
                        -                                                     -
                        -    A R E A   &   C E N T E R - O F - A R E A         -
                        -                                                     -
                        -   CALCULATES AREA AND CENTROID OF OBJECT IN IMAGE   -
                        -   MATRIX.                                           -
                        -                                                     -
                        -------------------------------------------------------   */

 51   1         FINDAREA: PROCEDURE PUBLIC;

 52   2            DECLARE
                        AREALABEL(*) BYTE DATA(0DH,0AH,0AH,'AREA: ',03H),
                        C$LABEL(*) BYTE DATA(0DH,0AH,'CENTROID: ',0DH,0AH,03H),
                        XLABEL(*) BYTE DATA('    Xc = ',03H),
                        YLABEL(*) BYTE DATA(0AH,0DH,'    Yc = ',03H),
                        AREAUNITS(*) BYTE DATA(' BITS SET.',03H),
                        (SUMXLO,SUMXHI,SUMYLO,SUMYHI) WORD,
                        ROW BYTE,
                        TEMP BYTE;
```

```
              $EJECT
              /*    ---------------------------------------
                    -                                     -
                    -         L I N E     S C A N         -
                    -                                     -
                    -      INCREMENTS AREA TOTALS AND     -
                    -    SUMS X AND Y CO-ORDINATES DURING -
                    -    SCAN OF ONE ROW OF IMAGE MATRIX. -
                    -                                     -
                    ---------------------------------------     */

 53   2       LINE$SCAN: PROCEDURE(LINE$NUMBER);

 54   3         DECLARE NUMBER$OF$BITS(*) BYTE DATA    (0,1,1,2,1,2,2,3,1,2,2,3,2,3,3,4,
                                                        1,2,2,3,2,3,3,4,2,3,3,4,3,4,4,5,
                                                        1,2,2,3,2,3,3,4,2,3,3,4,3,4,4,5,
                                                        2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,
                                                        1,2,2,3,2,3,3,4,2,3,3,4,3,4,4,5,
                                                        2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,
                                                        2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,
                                                        3,4,4,5,4,5,5,6,4,5,5,6,5,6,6,7,
                                                        1,2,2,3,2,3,3,4,2,3,3,4,3,4,4,5,
                                                        2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,
                                                        2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,
                                                        3,4,4,5,4,5,5,6,4,5,5,6,5,6,6,7,
                                                        2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,
                                                        3,4,4,5,4,5,5,6,4,5,5,6,5,6,6,7,
                                                        3,4,4,5,4,5,5,6,4,5,5,6,5,6,6,7,
                                                        4,5,5,6,5,6,6,7,5,6,6,7,6,7,7,8);
 55   3         DECLARE
                     LINE$NUMBER BYTE,
                     (ROWCNT,SHIFTCNT) BYTE,
                     TEMP BYTE,
                     DELTAX BYTE;

 56   3         DO ROWCNT= LEFT$OF$IMAGE TO RIGHT$OF$IMAGE;
 57   4              IF  (TEMP:=IMAGE$ROW(LINE$NUMBER).IMAGE$COL(ROWCNT))  <> 0 THEN
 58   4                  DO;
 59   5                   AREA=NUMBER$OF$BITS(TEMP) + AREA;

 60   5                   DO SHIFTCNT=0 TO 7;
 61   6                      TEMP=SHL(TEMP,1);
 62   6                        IF CARRY THEN
 63   6                            DO;
 64   7                             DELTAX=SHIFTCNT + SHL(ROWCNT,3);
 65   7                             IF (0FFFFH-DELTAX)<=SUMXLO THEN SUMXHI=SUMXHI + 1;
 67   7                             SUMXLO=SUMXLO + DELTAX;

 68   7                             IF (0FFFFH-LINE$NUMBER)<=SUMYLO THEN SUMYHI=SUMYHI + 1;
 70   7                             SUMYLO=SUMYLO + LINE$NUMBER;
 71   7                            END;
 72   6                    END;
 73   5                   END;
 74   4         END;

 75   3       END LINE$SCAN;
```

```
               $EJECT


 76   2           AREA,SUMYLO,SUMYHI,SUMXLO,SUMXHI=0;

 77   2           DO ROW= TOP$OF$IMAGE TO BOTTOM$OF$IMAGE;
 78   3                   CALL LINE$SCAN(ROW);
 79   3           END;


 80   2           CALL CRT$STRING$OUT(@AREALABEL);
 81   2           CALL SHOWBYTE(HIGH(AREA));
 82   2           CALL SHOWBYTE(LOW(AREA));
 83   2           CALL CRT$STRING$OUT(@AREAUNITS);

 84   2           CALL CRT$STRING$OUT(@CLABEL);

 85   2           CALL CRT$STRING$OUT(@XLABEL);
 86   2           CALL OUTPUT$DECIMAL(LOW(DIVIDE$32$BITS(SUMXHI,SUMXLO,0,AREA)));

 87   2           CALL CRT$STRING$OUT(@YLABEL);
 88   2           CALL OUTPUT$DECIMAL(LOW(DIVIDE$32$BITS(SUMYHI,SUMYLO,0,AREA)));

 89   2       END FINDAREA;
```

```
             $EJECT
             /*     --------------------------------------------------
                    -                                                -
                    -              W I N D O W   S E T S             -
                    -                                                -
                    -   LOCATES TOP, BOTTOM, RIGHT AND LEFT          -
                    -      EXTREMITIES OF IMAGE WITHIN FRAME.        -
                    -                                                -
                    --------------------------------------------------     */

 90   1      LOCATE$TOP: PROCEDURE BYTE PUBLIC;

 91   2         DECLARE
                       ROW BYTE,
                       COL BYTE;

 92   2         DO ROW= 1 TO 238;
 93   3                DO COL=0 TO 29;
 94   4                   IF IMAGE$ROW(ROW).IMAGE$COL(COL)<>0 THEN RETURN ROW;
 96   4                END;
 97   3         END;

 98   2         RETURN 0FFH;

 99   2      END LOCATE$TOP;


100   1      LOCATE$BOTTOM: PROCEDURE BYTE PUBLIC;

101   2         DECLARE
                       ROW BYTE,
                       COL BYTE;

102   2         DO ROW = 1 TO 238;
103   3                DO COL=0 TO 29;
104   4                   IF IMAGE$ROW(239-ROW).IMAGE$COL(COL)<>0 THEN RETURN (239-ROW);
106   4                END;
107   3         END;

108   2      END LOCATE$BOTTOM;
```

```
            $EJECT

 109   1      LOCATE$LEFT: PROCEDURE(TOP,BOTTOM) BYTE PUBLIC;

 110   2          DECLARE
                       TOP BYTE,
                          BOTTOM BYTE,
                          ROW BYTE,
                          COL BYTE;

 111   2          DO COL=0 TO 28;
 112   3             DO ROW=TOP TO BOTTOM;
 113   4                IF IMAGE$ROW(ROW).IMAGE$COL(COL) <> 0 THEN RETURN COL;
 115   4             END;
 116   3          END;

 117   2      END LOCATE$LEFT;


 118   1      LOCATE$RIGHT: PROCEDURE(TOP,BOTTOM) BYTE PUBLIC;

 119   2          DECLARE
                       TOP BYTE,
                       BOTTOM BYTE,
                       ROW BYTE,
                       COL BYTE;

 120   2          DO COL=0 TO 28;
 121   3              DO ROW=TOP TO BOTTOM;
 122   4                IF IMAGE$ROW(ROW).IMAGE$COL(28-COL) <> 0 THEN RETURN (28-COL);
 124   4              END;
 125   3          END;

 126   2      END LOCATE$RIGHT;
```

```
          $EJECT


          /*   -----------------------------------------------------
               -                                                   -
               -               F R A M E    G R A B                -
               -                                                   -
               -   GRABS ONE FRAME OF IMAGE DATA FROM CAMERA. DMA  -
               -   CONTROLLER IS PROGRAMMED FOR INTERRUPTED        -
               -   TRANSFER.                                       -
               -                                                   -
               -----------------------------------------------------      */

 127   1   FRAME$GRAB: PROCEDURE PUBLIC;

 128   2      DECLARE BASE LITERALLY '0F0H';

 129   2      OUTPUT(BASE+0BH)=0;

 130   2          OUTPUT(BASE+0CH)=0A4H;
 131   2          OUTPUT(BASE+0DH)=1BH;     /*   LENGTH  REGISTER OF DMA BOARD  */

 132   2          OUTPUT(BASE+0AH)=07H;     /*   CONTROL REGISTER OF DMA BOARD   */

 133   2          OUTPUT(BASE+0EH)=0;
 134   2          OUTPUT(BASE+0FH)=0A0H;   /* FIRST ADDRESS REG. OF DMA BOARD   */

 135   2          OUTPUT(BASE+0BH)=02H;    /*  CAMERA -> SEQUENTIAL MODE    */

 136   2      CALL TIME(250);
 137   2      CALL TIME(250);
 138   2      CALL TIME(250);

 139   2          OUTPUT(BASE+0BH)=0AH;    /*  RESET/GO   */

 140   2      CALL TIME(250);
 141   2      CALL TIME(250);
 142   2      CALL TIME(250);

 143   2          OUTPUT(BASE+0BH)=0;        /*  CAMERA -> INTERLACED MODE   */
 144   2          OUTPUT(BASE+09H)=0;


 145   2   END FRAME$GRAB;


 146   1   END IMAGE$ROUTINES;


MODULE INFORMATION:

     CODE AREA SIZE     = 05CEH   1486D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0C1DH     29D
```

```
MAXIMUM STACK SIZE = 0010H     16D
372 LINES READ
0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```

```
ISIS-II PL/M-86 V2.0 COMPILATION OF MODULE MATHROUTINES
OBJECT MODULE PLACED IN :F1:MATH.OBJ
COMPILER INVOKED BY:  PLM86 :F1:MATH.SRC LARGE OPTIMIZE(3) TITLE('     MATH ROUTINES')



   1          MATH$ROUTINES:  DO;

              /*********************************************************
               *                                                       *
               *             M A T H    R O U T I N E S                *
               *                                                       *
               *   iSBC 86/12 VISION SYSTEM          NOVEMBER 1980     *
               *                                     AUTHOR: D. CARSON *
               *                                                       *
               *      THIS MODULE CONTAINS PROCEDURES FOR 32, 16 AND   *
               *     8 BIT ARITHMETIC FUNCTIONS.                       *
               *                                                       *
               *              ROUTINES INCLUDED:                       *
               *             -------------------                       *
               *               DIVIDE$32$BITS                          *
               *               OUTPUT$DECIMAL                          *
               *                                                       *
               *********************************************************/

              /*    ----------------------------------------------------------
                     E X T E R N A L    U S A G E    D E C L A R A T I O N S
                    ----------------------------------------------------------    */


   2   1         CRT$STRING$OUT: PROCEDURE(PTR) EXTERNAL;
   3   2              DECLARE PTR POINTER;
   4   2         END CRT$STRING$OUT;

   5   1         CRTIN: PROCEDURE BYTE EXTERNAL;
   6   2         END CRTIN;

   7   1         ERROR$MESSAGE: PROCEDURE(ERRORNUMBER) EXTERNAL;
   8   2              DECLARE ERRORNUMBER BYTE;
   9   2         END ERROR$MESSAGE;

  10   1         SHOWBYTE: PROCEDURE(BIT$PATTERN) EXTERNAL;
  11   2              DECLARE BIT$PATTERN BYTE;
  12   2         END SHOWBYTE;

  13   1         CRT$BYTE$OUT: PROCEDURE(CHAR) EXTERNAL;
  14   2              DECLARE CHAR BYTE;
  15   2         END CRT$BYTE$OUT;
```

```
          $EJECT

          /*      -------------------------------------------
                  -                                         -
                  -     D I V I D E     R O U T I N E       -
                  -                                         -
                  -   THIS PROCEDURE RETURNS THE 16 BIT     -
                  -   QUOTIENT FROM THE DIVISION OF TWO 32  -
                  -   BIT BINARY NUMBERS.                   -
                  -                                         -
                  -------------------------------------------      */

 16   1       DIVIDE$32$BITS: PROCEDURE(DIVIDENDHI,DIVIDENDLO,DIVISORHI,DIVISORLO) WORD PUBLIC;

 17   2          DECLARE
                      QUOTIENT WORD,
                      (DIVIDENDHI,DIVIDENDLO) WORD,
                      (DIVISORHI,DIVISORLO) WORD,
                      N BYTE,
                      CNT BYTE;
```

```
        $EJECT

18  2       SHIFT$DIVISOR$LEFT: PROCEDURE;
                   /* SHIFTS (DIVISORHI,DIVISORLO) 1 BIT LEFT AS A 32 BIT WORD  */
19  3               DIVISORHI=SHL(DIVISORHI,1);
20  3               IF (DIVISORLO AND 8000H)<>0 THEN
21  3                                      DIVISORHI=DIVISORHI OR 0001H;
22  3               DIVISORLO=SHL(DIVISORLO,1);
23  3       END SHIFT$DIVISOR$LEFT;


24  2       SHIFT$DIVISOR$RIGHT: PROCEDURE;
                   /* SHIFTS (DIVISORHI,DIVISORLO) 1 BIT RIGHT AS A 32 BIT WORD  */
25  3              DIVISORLO=SHR(DIVISORLO,1);
26  3              IF (DIVISORHI AND 0001H)<>0 THEN
27  3                                      DIVISORLO=DIVISORLO OR 8000H;
28  3              DIVISORHI=SHR(DIVISORHI,1);
29  3       END SHIFT$DIVISOR$RIGHT;


30  2       LESS$THAN: PROCEDURE(AHI,ALO,BHI,BLO) BYTE;

                   /*  IF  (AHI,ALO) < (BHI,BLO)   THEN RETURNS TRUE
                                                   ELSE RETURNS FALSE     */

31  3              DECLARE
                      (AHI,ALO,BHI,BLO) WORD;
32  3              IF AHI<BHI THEN RETURN 0FFH;
34  3              IF AHI>BHI THEN RETURN 0H;
36  3              IF ALO<BLO THEN RETURN 0FFH;
38  3              RETURN 0H;
39  3       END LESS$THAN;


40  2       SUBTRACT: PROCEDURE;
                   /* (DIVIDENDHI,DIVIDENDLO) <= (DIVIDENDHI,DIVIDENDLO) - (DIVISORHI,DIVISORLO)   */
41  3             IF DIVISORLO>DIVIDENDLO
                           THEN DIVIDENDHI=DIVIDENDHI-DIVISORHI-1;
43  3                      ELSE DIVIDENDHI=DIVIDENDHI-DIVISORHI;
44  3             DIVIDENDLO=DIVIDENDLO-DIVISORLO;
45  3       END SUBTRACT;
```

```
                 $EJECT

 46   2          QUOTIENT=0;
 47   2          N=0;

 48   2          DO WHILE LESS$THAN(DIVISORHI,DIVISORLO,DIVIDENDHI,DIVIDENDLO);
 49   3                  CALL SHIFT$DIVISOR$LEFT;
 50   3                  N=N+1;
 51   3          END;
 52   2          CALL SHIFT$DIVISOR$RIGHT;
 53   2          N=N-1;

 54   2          DO CNT=0 TO 6;
 55   3               IF NOT(LESS$THAN(DIVIDENDHI,DIVIDENDLO,DIVISORHI,DIVISORLO))
                         THEN DO;
 57   4                          CALL SUBTRACT;
 58   4                          QUOTIENT=QUOTIENT + 1;
 59   4                         END;
 60   3               QUOTIENT=SHL(QUOTIENT,1);
 61   3               CALL SHIFT$DIVISOR$RIGHT;
 62   3          END;

 63   2          QUOTIENT=SHR(QUOTIENT,7-N);

 64   2          RETURN QUOTIENT;

 65   2       END DIVIDE$32$BITS;
```

```
             $EJECT

             /*    ----------------------------------------------
                   -                                            -
                   -        O U T P U T   D E C I M A L         -
                   -                                            -
                   -    THIS PROCEDURE OUTPUTS THE DECIMAL      -
                   -    EQUIVALENT OF THE GIVEN 8-BIT BINARY    -
                   -    VALUE.                                  -
                   -                                            -
                   ----------------------------------------------     */

 66   1      OUTPUT$DECIMAL: PROCEDURE(BINARY$BYTE) PUBLIC;

 67   2          DECLARE
                        BINARY$BYTE BYTE,
                        CNT BYTE,
                        DECIMALLO BYTE;

 68   2          DECIMALLO=0;

 69   2          IF (BINARY$BYTE)=100) AND (BINARY$BYTE<=199)
                               THEN CALL CRT$BYTE$OUT('1');
 71   2                        ELSE IF (BINARY$BYTE)=200) THEN CALL CRT$BYTE$OUT('2');

             DO CNT=0 TO 7;

 74   3             DECIMALLO=DECIMALLO + DECIMALLO;
 75   3             DECIMALLO=DEC(DECIMALLO);

 76   3             IF  (BINARY$BYTE AND 80H)()0
                             THEN DO;
 78   4                            DECIMALLO=DECIMALLO + 1;
 79   4                            DECIMALLO=DEC(DECIMALLO);
 80   4                          END;

 81   3             BINARY$BYTE=SHL(BINARY$BYTE,1);

 82   3          END;


 83   2          CALL SHOW$BYTE(DECIMALLO);

 84   2      END OUTPUT$DECIMAL;


 85   1      END MATH$ROUTINES;


MODULE INFORMATION:

     CODE AREA SIZE     = 0193H    403D
     CONSTANT AREA SIZE = 0000H      0D
```

```
     VARIABLE AREA SIZE = 000EH     14D
     MAXIMUM STACK SIZE = 000EH     14D
     196 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```

```
ISIS-II PL/M-86 V2.0 COMPILATION OF MODULE CRTROUTINES
OBJECT MODULE PLACED IN :F1:CRT.OBJ
COMPILER INVOKED BY:  PLM86 :F1:CRT.SRC LARGE OPTIMIZE(3) TITLE('    CRT ROUTINES')


   1          CRT$ROUTINES: DO;



              /******************************************************
               *                                                    *
               *          C R T    R O U T I N E S                  *
               *                                                    *
               *   THIS MODULE CONTAINS A COLLECTION OF GENERAL     *
               *  I/O ROUTINES FOR COMMUNICATION WITH THE CRT       *
               *  TERMINAL.,                                        *
               *                                                    *
               *              ROUTINES INCLUDED:                    *
               *              ------------------                    *
               *                CRT$STRING$OUT                      *
               *                CRTIN                               *
               *                ERROR$MESSAGE                       *
               *  S             SHOW$BYTE                           *
               *                CHAR$READY                          *
               *                CRT$BYTE$OUT                        *
               *                                                    *
               ******************************************************/
```

```
             $EJECT

             /* ---------------------------------------------------------------
                -                                                             -
                -              C R T   S T R I N G   O U T                    -
                -                                                             -
                -    OUTPUTS ASCII CHARACTERS POINTED TO BY 'PTR' UNTIL       -
                -           END-OF-TEXT (03H) IS ENCOUNTERED.                 -
                -                                                             -
                ---------------------------------------------------------------    */

  2   1      CRT$STRING$OUT: PROCEDURE(PTR) PUBLIC;

  3   2         DECLARE
                       PTR POINTER,STRING BASED PTR(1) BYTE,
                       I BYTE;

  4   2         I=0;
  5   2         DO WHILE STRING(I)<>03H;
  6   3            DO WHILE (INPUT(0DAH) AND 01H)=0;     /* WAIT FOR TRANSMIT READY */
  7   4            END;
  8   3            OUTPUT(0D8H)=STRING(I);               /* OUTPUT NEXT ELEMENT     */
  9   3            I=I+1;
 10   3         END;

 11   2      END CRT$STRING$OUT;
```

```
              $EJECT


              /* ---------------------------------------------------------------
                 -                                                             -
                 -                        C R T   I N                          -
                 -                                                             -
                 -  ECHOS AND RETURNS ONE ASCII CHARACTER FROM THE CRT CONSOLE. -
                 -                                                             -
                 ---------------------------------------------------------------   */


 12   1       CRTIN: PROCEDURE BYTE PUBLIC;

 13   2          DECLARE CHAR BYTE;

 14   2          DO WHILE (INPUT(0DAH) AND 02H)=0;     /*  WAIT FOR RECEIVE READY */
 15   3          END;
 16   2          CHAR=INPUT(0D8H) AND 7FH;             /*     INPUT CHARACTER     */
 17   2          DO WHILE (INPUT(0DAH) AND 01H)=0;     /* WAIT FOR TRANSMIT READY */
 18   3          END;
 19   2          OUTPUT(0D8H)=CHAR;                    /*     ECHO CHARACTER      */
 20   2          RETURN CHAR;

 21   2       END CRTIN;
```

```
            $ EJECT

            /* -------------------------------------------------------------------
               -                                                                 -
               -                      E R R O R   M E S S A G E S                -
               -                                                                 -
               -        TAKES APPROPRIATE ACTION ON CRT FOR GIVEN ERROR CONDITION. -
               -                                                                 -
               -------------------------------------------------------------------  */

 22   1     ERROR$MESSAGE: PROCEDURE (ERRORNUMBER) PUBLIC;

 23   2        DECLARE
                    ERROR0(*) BYTE DATA (0DH,0AH,'ILLEGAL COMMAND',0DH,0AH,03H),
                    ERROR1(*) BYTE DATA (0DH,0AH,'SYNTAX ERROR',0AH,0DH,03H),
                    ERROR2(*) BYTE DATA (0AH,08H,'^',0AH,08H,03H),
                    ERROR3(*) BYTE DATA (0DH,0AH,'LEFT EDGE INTERSECTION',03H),
                    ERROR4(*) BYTE DATA (0DH,0AH,'RIGHT EDGE INTERSECTION',03H),
                    ERROR5(*) BYTE DATA (0DH,0AH,'TOP EDGE INTERSECTION',03H),
                    ERROR6(*) BYTE DATA (0DH,0AH,'BOTTOM EDGE INTERSECTION',03H),
                    ERROR7(*) BYTE DATA (0DH,0AH,0AH,'  - OBJECT REJECTED -',03H),
                    ERROR8(*) BYTE DATA (0DH,0AH,'ANALYSIS SUSPENDED - NO OBJECT FOUND',0DH,0AH,03H),
                    ERRORNUMBER BYTE;

 24   2        DO CASE ERRORNUMBER;
 25   3                  CALL CRT$STRING$OUT(@ERROR0);
 26   3                  CALL CRT$STRING$OUT(@ERROR1);
 27   3                  CALL CRT$STRING$OUT(@ERROR2);
 28   3                  CALL CRT$STRING$OUT(@ERROR3);
 29   3                  CALL CRT$STRING$OUT(@ERROR4);
 30   3                  CALL CRT$STRING$OUT(@ERROR5);
 31   3                  CALL CRT$STRING$OUT(@ERROR6);
 32   3                  CALL CRT$STRING$OUT(@ERROR7);
 33   3                  CALL CRT$STRING$OUT(@ERROR8);
 34   3        END;

 35   2     END ERROR$MESSAGE;
```

```
                $EJECT

                /* --------------------------------------------------------------
                 -                                                              -
                 -                      S H O W   B Y T E                       -
                 -                                                              -
                 -      DISPLAYS BIT PATTERN OF A BYTE ON CRT IN HEX FORMAT.    -
                 -                                                              -
                 --------------------------------------------------------------   */

 36   1         SHOWBYTE: PROCEDURE (BIT$PATTERN) PUBLIC;

 37   2           DECLARE
                      HEX(*) BYTE DATA ('0123456789ABCDEF'),
                      BIT$PATTERN BYTE, HEXDIGIT BYTE;

 38   2           HEXDIGIT=BIT$PATTERN AND 0F0H;      /*  MASK HIGH ORDER BITS   */
 39   2           HEXDIGIT=SHR(HEXDIGIT,4);
 40   2           DO WHILE (INPUT(0DAH) AND 01H)=0;
 41   3           END;
 42   2           OUTPUT(0D8H)=HEX(HEXDIGIT);        /*  OUTPUT HIGH ORDER BITS */

 43   2           HEXDIGIT=BIT$PATTERN AND 0FH;       /*  MASK LOW ORDER BITS    */
 44   2           DO WHILE (INPUT(0DAH) AND 01H)=0;
 45   3           END;
 46   2           OUTPUT(0D8H)=HEX(HEXDIGIT);        /*  OUTPUT LOW ORDER BITS  */

 47   2         END SHOWBYTE;
```

```
            $ EJECT


            /* -----------------------------------------------------------
               -                                                         -
               -          C H A R A C T E R    R E A D Y                 -
               -                                                         -
               -    IF A CHARACTER HAS BEEN INPUT FROM THE CRT, THEN     -
               -    THE VALUE TRUE IS RETURNED.  OTHERWISE, FALSE.       -
               -                                                         -
               -----------------------------------------------------------   */

  48   1    CHAR$READY: PROCEDURE BYTE PUBLIC;

  49   2        IF(INPUT(0DAH) AND 02H)=0
                            THEN RETURN 0H;
  51   2                    ELSE RETURN 0FFH;
  52   2        END CHAR$READY;


            /* -----------------------------------------------------------
               -                                                         -
               -          C R T    B Y T E    O U T                      -
               -                                                         -
               -      OUTPUT GIVEN ASCII CHARACTER TO CRT.               -
               -                                                         -
               -----------------------------------------------------------   */

  53   1    CRT$BYTE$OUT: PROCEDURE(CHAR) PUBLIC;

  54   2        DECLARE CHAR BYTE;

  55   2        DO WHILE (INPUT(0DAH) AND 01H)=0;
  56   3        END;
  57   2        OUTPUT(0D8H)=CHAR;

  58   2    END CRT$BYTE$OUT;



  59   1    END CRT$ROUTINES;


MODULE INFORMATION:

     CODE AREA SIZE     = 022CH    556D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0000H      0D
     MAXIMUM STACK SIZE = 0010H     16D
     201 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```

CHAPTER 6

DISCUSSION

6.1 Conclusions

Image processing problems are, in general, a problem of data reduction. Often images are represented by large amounts of data from which useful information or features must be extracted.

Of primary contention among computer vision designers is the selection of binary versus grey-level images. Grey-level systems are capable of processing complex scenes in which lighting may be uncontrollable, but they require large buffer memories and extensive computing resources. Binary (silhouette) processing involves smaller image buffers than do equal resolution grey-level systems, processing is generally faster, but flexibility is lost. However, with carefully engineered lighting and thresholding techniques, silhouette vision is capable of handling a wide range of tasks.

The fundamental information contained in a silhouette is shape. Shape can be quantitatively examined using two basic methodologies: information preserving and information non-preserving descriptors. The former techniques allow a shape to be reconstructed from its mathematical description to a pre-selected resolution. The latter descriptors are indications of shape characteristics such as circularity, elongation, etc. and many widely different shapes can exhibit identical traits of this nature. If an information non-preserving descriptor is related to a reference shape, then it is termed a shape factor. A combination of information non-preserving descriptors for a shape will yield a more accurate representation.

Computer vision (both grey-level and binary systems) are finding application in industrial environments. It is convenient to divide application areas into two categories: inspection, and sensor-controlled manipulation. In particular, binary systems are adequate for such roles as sorting workpieces, identifying or measuring shape, parts fitting or mating, measurement of critical dimensions and tool wear, and checking for integrity or completeness. Other inherently binary images include text, handwriting, and various types of line drawings. In addition, a binary computer vision system can be used as visual feedback for servoing applications.

A binary image has been successfully acquired from a GE TN2500 digital matrix camera during a single television frame (33 ms). Using an Intel iSBC 86/12A microcomputer, the area and centroid coordinates have been determined with software. Calculation times increase with the area of the object being observed, with a typical cycle time of 2 seconds for a shape occupying approximately 30% of the frame area. A hardware implementation would drastically reduce this time by evaluating the area and coordinate summations as the data is dumped from the camera. Identification of various shapes has been realized based on their area. Experimental measurement has shown that objects which differ in area by approximately 2% can be discerned with high accuracy.

6.2 Future Work

Further development of the silhouette vision system can progress in several directions. An implementation of the various shape descriptors will allow identification of a wide range of shapes which may have nearly equal area. The use of run-length encoding will speed up the processing as well as the exploitation of the 8086 assembly language software. Connectivity analysis would permit several objects to be examined in the same frame. For utilization in a manipulator system for example, orientation should also be evaluated.

If these functions are designed in hardware, a substantial speed increase can be achieved. The addition of a specialized numerical processor can enhance the performance of the 8086 microcomputer.

A variety of techniques can improve binary vision, including the use of dynamically controlled threshold settings for adaptation to different lighting conditions. Noise pixel elimination is also desirable for less than ideal lighting situations.

APPENDIX "A"

iSBC 86/12 VISION MONITOR

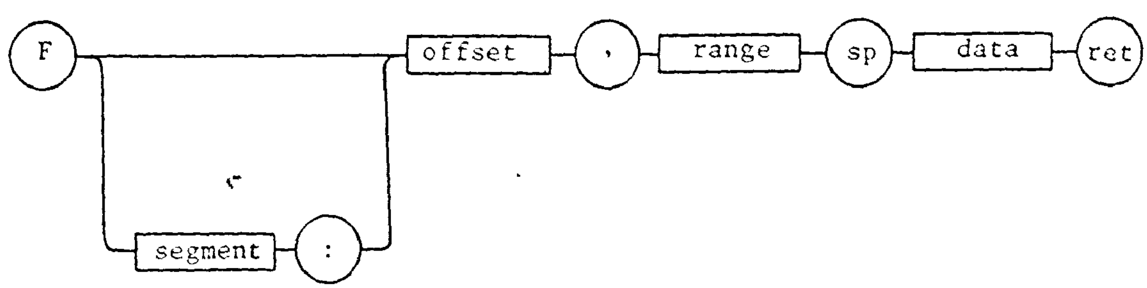
A system command of "M" typed on the keyboard gives access to the iSBC 86/12 VISION MONITOR. In this mode various facilities exist to examine and manipulate memory, read or write to I/0 ports, and branch to begin execution at any location.

Whenever addresses or data are requested, only the last 4 hexidecimal digits are accepted for word values and only the final 2 in the case of bytes values. For entry of less than the required number of digits, leading zeros are appended to bring the total to either 4 or 2 as the case may be.

Errors are flagged with a caret beneath the offending character (which is ignored) and input continues normally.

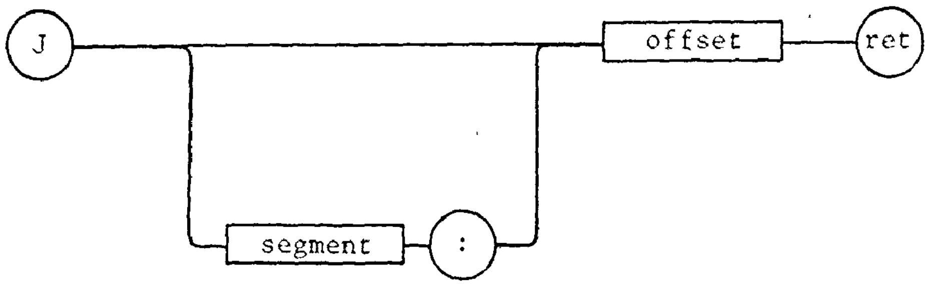
The commands and correct syntax are summarized below.

F - Fill Memory Locations with Data



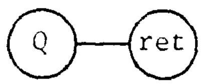
The memory locations "segment:offset" to "segment:range" are set to the value of "data". If segment is omitted, it is assumed to be zero.

J - Jump to Address and Begin Execution



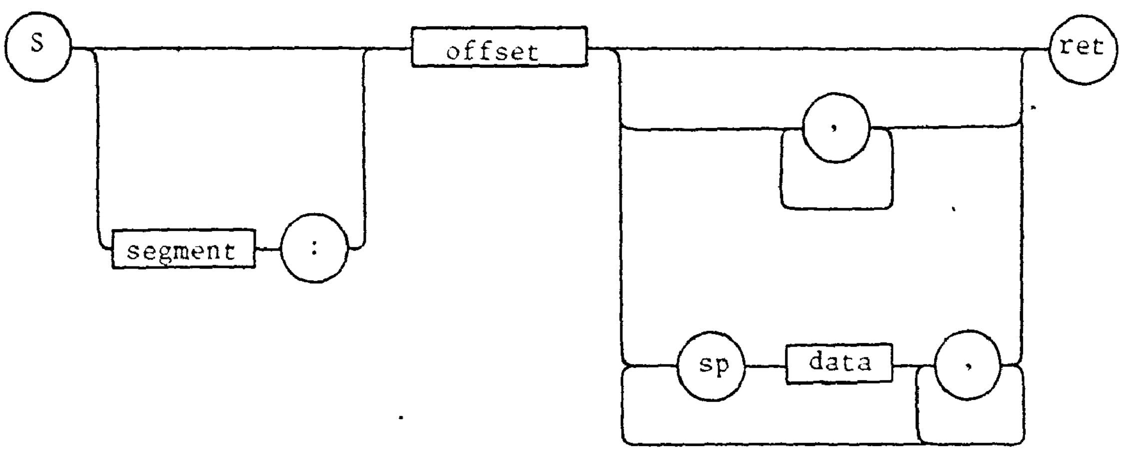
Execution branches to "segment:offset". Segment is assumed zero if omitted.

Q - Return to System



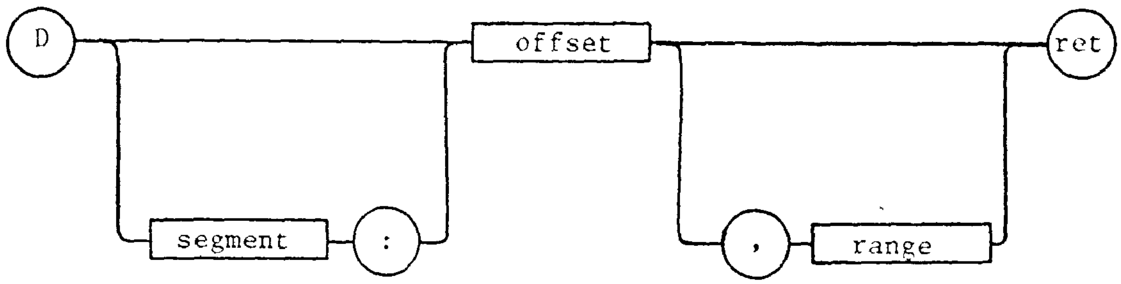
Control is returned to the system.

S - Substitute Memory



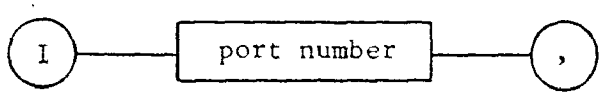
The substitute command displays the current contents of "segment: offset (again segment is zero if omitted) and allows new data to be substituted following the space character. Entering a comma advances the display to the next location until terminated with a "return".

D - Dump Memory Locations



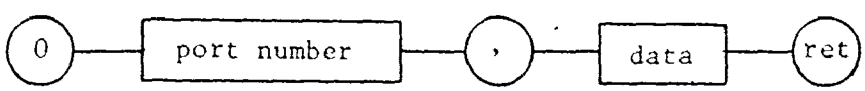
The memory locations "segment:offset" to "segment:range" are displayed on the CRT. Each offset value is displayed in the left margin before 16 bytes of memory contents. If omitted, "segment" is assumed zero. If the range is omitted, only one memory location is displayed.

I - Display Input Port Data



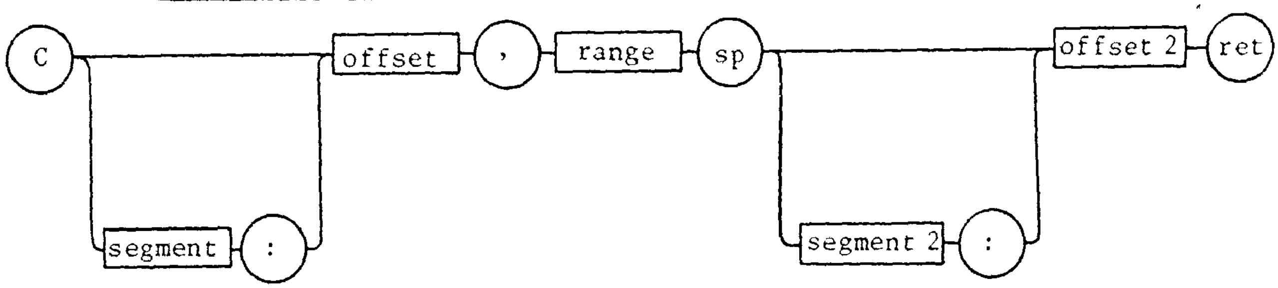
The data on input port "port number" is displayed on the CRT.

O - Write Data to Output Port



Data is output to the specified output port.

C - Copy a Block of Memory to a New Location



The block of memory specified by "segment:offset" to "segment:range: is copied beginning at "segment2:offset2". Omission of either "segment" or "segment2" defaults them to zero.

ISIS-II PL/M-86 V2.0 COMPILATION OF MODULE SBC8612MONITOR
OBJECT MODULE PLACED IN :F1:MONITR.OBJ
COMPILER INVOKED BY: PLM86 :F1:MONITR.SRC LARGE OPTIMIZE(3) TITLE(' VISION MONITOR')

```
1         SBC8612$MONITOR: DO;


     /*********************************************************************
      *                                                                   *
      *                        M O N I T O R                              *
      *                                                                   *
      *     iSBC 86/12   VISION  SYSTEM            JUNE 1980              *
      *                                            AUTHOR: DAVID CARSON   *
      *                                                                   *
      *       THESE ROUTINES FORM A SMALL (APPROX. 1K BYTES) MONITOR      *
      *     FOR MANIPULATING THE MEMORY AND I/O FUNCTIONS OF THE          *
      *     iSBC 86/12A MICROCOMPUTER.                                    *
      *                                                                   *
      *      COMMANDS:                                                    *
      *      ---------                                                    *
      *            F - FILL ADDRESS RANGE WITH GIVEN DATA                 *
      *            D - DUMP ADDRESS RANGE MEMORY CONTENTS                 *
      *            I - DISPLAY INPUT PORT BYTE                            *
      *            O - OUTPUT BYTE TO OUTPUT PORT                         *
      *            J - START EXECUTION AT GIVEN ADDRESS                   *
      *            R - RETURN TO SYSTEM                                   *
      *            S - SUBSTITUTE MEMORY                                  *
      *            C - COPY A BLOCK OF MEMORY                             *
      *                                                                   *
      *********************************************************************/
```

```
            $EJECT

            /*  ------------------------------------------------------------------------
                        E X T E R N A L   U S A G E   D E C L A R A T I O N S
                ------------------------------------------------------------------------  */
  2   1       CRT$STRING$OUT: PROCEDURE(PTR) EXTERNAL;
  3   2               DECLARE PTR POINTER;
  4   2       END CRT$STRING$OUT;

  5   1       CRTIN: PROCEDURE BYTE EXTERNAL;
  6   2       END CRTIN;

  7   1       ERROR$MESSAGE: PROCEDURE(ERRORNUMBER) EXTERNAL;
  8   2                       DECLARE ERRORNUMBER BYTE;
  9   2       END ERROR$MESSAGE;

 10   1       SHOW$BYTE: PROCEDURE(BIT$PATTERN) EXTERNAL;
 11   2                   DECLARE BIT$PATTERN BYTE;
 12   2       END SHOW$BYTE;

 13   1       CHAR$READY: PROCEDURE BYTE EXTERNAL;
 14   2       END CHAR$READY;

 15   1       CRT$BYTE$OUT: PROCEDURE(CHAR) EXTERNAL;
 16   2                   DECLARE CHAR BYTE;
 17   2       END CRT$BYTE$OUT;


 18   1     MONITOR: PROCEDURE PUBLIC;


            /*  ------------------------------------------------------------------
                            G L O B A L   D E C L A R A T I O N S
                ------------------------------------------------------------------  */
 19   2       DECLARE
                  LOCATION STRUCTURE (OFFSET WORD, SEGMENT WORD),  /* SEE PAGE 8.1 OF PL/M 86 COMPILER */
                  MEM$PTR POINTER AT (@LOCATION),                  /* OPERATORS MANUAL FOR FORMAT OF   */
                  CONTENTS BASED MEM$PTR BYTE,                     /*  POINTER VALUES.                 */
                  CMND BYTE,
                  HIGHLIMIT WORD,
                  MONCMND(*) BYTE DATA ('FDGIOJSC'),
                  MONSIGNON(*) BYTE DATA (0CH,0DH,0AH,'                   iSBC 86/12 VISION MONITOR',03H),
                  MONPROMPT(*) BYTE DATA (0DH,0AH,'.',03H),
                  CRLF(*) BYTE DATA (0DH,0AH,03H);
```

```
              $ EJECT

              /* -------------------------------------------------------------
                 -                                                           -
                 -                       G E T   W O R D                     -
                 -                                                           -
                 -   INPUTS ASCII CHARACTERS FROM THE CRT UNTIL ONE OF THE   -
                 -   DELIMITERS IS ENCOUNTERED. THE HEX BIT PATTERNS OF THE  -
                 -   LAST FOUR CHARACTERS INPUT ARE PACKED INTO THE          -
                 -   RETURNED WORD.                                          -
                 -                                                           -
                 -------------------------------------------------------------  */

 20   2       GET$WORD: PROCEDURE(DELIMITER1,DELIMITER2) WORD;

 21   3          DECLARE
                        HEX(*) BYTE DATA ('0123456789ABCDEF'),
                        ACC WORD, CHAR BYTE,
                        (DELIMITER1,DELIMITER2) BYTE;

 22   3          ACC=0;
 23   3          DO WHILE ((CHAR:=CRTIN)<>DELIMITER1) AND (CHAR<>DELIMITER2);
 24   4                   CHAR=LOW(FINDB(@HEX,CHAR,16));
 25   4                   IF CHAR=0FFH THEN CALL ERROR$MESSAGE(2);
 27   4                                ELSE DO;
 28   5                                       ACC=SHL(ACC,4);
 29   5                                       ACC=ACC OR CHAR;
 30   5                                      END;
 31   4          END;
 32   3          RETURN ACC;

 33   3       END GET$WORD;
```

```
          $ EJECT

          /*  ----------------------------------------------------------------
              -                                                              -
              -                 G E T   A D D R E S S                        -
              -                                                              -
              -   INPUTS AN ADDRESS WITH AN OPTIONAL SEGMENT FROM THE CRT.   -
              -   ANY LOCATION IN THE 1 MEGABYTE RANGE MAY BE ACCESSED.  THE -
              -   SEGMENT, IF INCLUDED, IS FOLLOWED BY ':'.  ALSO, A RANGE   -
              -   OF UP TO 64K (0FFFFH) MAY FOLLOW THE OFFSET SEPARATED BY   -
              -   A COMMA IF THE HIGH$LIMIT$REQUEST PARAMETER IS TRUE.       -
              -                                                              -
              ----------------------------------------------------------------  */
 34   2   GET$ADDRESS: PROCEDURE(HIGH$LIMIT$REQUEST);

 35   3       DECLARE
                  BITSIN BYTE, CHAR BYTE,
                  HEX(*) BYTE DATA('0123456789ABCDEF'),
                  HIGH$LIMIT$REQUEST BYTE;

 36   3       LOCATION.SEGMENT=0;                                    /*   DEFAULT SEGMENT    */
 37   3       LOCATION.OFFSET=0;                                     /*   INITIALIZE OFFSET  */
 38   3   DO WHILE 1;
 39   4       DO WHILE (BITSIN:=LOW(FINDB(@HEX,(CHAR:=CRTIN),16)))<>0FFH;  /* INPUT ASCII CHARACTERS FROM */
 40   5              LOCATION.OFFSET=SHL(LOCATION.OFFSET,4);          /* CRT TO HEX FORMAT IN OFFSET */
 41   5              LOCATION.OFFSET=LOCATION.OFFSET OR BITSIN;       /* WORD                        */
 42   5       END;

 43   4       IF (CHAR=':') THEN DO;
 45   5                              LOCATION.SEGMENT=LOCATION.OFFSET;
 46   5                              LOCATION.OFFSET=GET$WORD(',',' ');
 47   5                              IF HIGH$LIMIT$REQUEST THEN HIGHLIMIT=GET$WORD(0DH,' ');
 49   5                              RETURN;
 50   5                             END;
                                ELSE
 51   4       IF (CHAR=',') THEN DO;
 53   5                              IF HIGH$LIMIT$REQUEST THEN HIGHLIMIT=GET$WORD(0DH,' ');
 55   5                              RETURN;
 56   5                             END;
                                ELSE
 57   4       IF (CHAR=0DH) OR (CHAR=' ') THEN DO;
 59   5                              HIGHLIMIT=LOCATION.OFFSET;
 60   5                              RETURN;
 61   5                             END;
                                ELSE
 62   4          CALL ERROR$MESSAGE(2);
 63   4   END;

 64   3   END GET$ADDRESS;
```

```
              $ EJECT

              /* -----------------------------------------------------------
                 -                                                         -
                 -                         D U M P                         -
                 -                                                         -
                 -   OUTPUTS CONTENTS OF MEMORY POINTED TO BY THE WORDS    -
                 -   LOCATION.OFFSET AND LOCATION.SEGMENT.  THE RANGE OF   -
                 -   LOCATIONS IS OBTAINED FROM THE 'GETADDRESS' PROCE-    -
                 -   DURE.  HEX PATTERNS ARE OUTPUT, 16 TO A LINE AND      -
                 -   CAN BE SUSPENDED BY KEYING CNTRL 'S', RESUMED WITH    -
                 -   CNTRL 'Q' AND TERMINATED BY ENTERING CNTRL 'C'.       -
                 -                                                         -
                 -----------------------------------------------------------     */
 65   2      DUMP: PROCEDURE;

 66   3         DECLARE
                        COUNTER BYTE,
                        CHAR BYTE;

 67   3         CALL GETSADDRESS(0FFH);                          /* OBTAIN OUTPUT RANGE */
 68   3         CALL CRT$STRING$OUT(@CRLF);
 69   3         DO WHILE HIGHLIMIT>=LOCATION.OFFSET;
 70   4                 CALL SHOWBYTE(HIGH(LOCATION.OFFSET));     /* OUTPUT LEFT MARGIN  */
 71   4                 CALL SHOWBYTE(LOW(LOCATION.OFFSET));      /*     LABELS          */
 72   4                 CALL CRT$BYTE$OUT(' ');
 73   4                 COUNTER=0;
 74   4                 IF CHAR$READY THEN
 75   4                            DO;
 76   5                             IF ((CHAR:=CRTIN)=13H) THEN        /* CHECK FOR CNTRL 'S' */
 77   5                                        DO WHILE CHAR<>11H;     /* CNTRL 'Q'  */
 78   6                                            IF CHAR$READY THEN  /* CNTRL 'C' */
 79   6                                                IF ((CHAR:=CRTIN)=03H) THEN RETURN;
 81   6                                        END;
 82   5                            END;
 83   4                 DO WHILE (COUNTER<=15) AND (HIGHLIMIT>=LOCATION.OFFSET);
 84   5                      CALL CRT$BYTE$OUT(' ');
 85   5                      COUNTER=COUNTER+1;
 86   5                      CALL SHOWBYTE(CONTENTS);
 87   5                      LOCATION.OFFSET=LOCATION.OFFSET + 1;
 88   5                 END;
 89   4                 CALL CRT$STRING$OUT(@CRLF);
 90   4         END;

 91   3      END DUMP;
```

```
          $ EJECT

          /*   ----------------------------------------------------
               -                                                  -
               -               F I L L                            -
               -                                                  -
               -    FILLS THE MEMORY RANGE OBTAINED FROM THE PRO- -
               -    CEDURE 'GETADDRESS' WITH ANY BIT PATTERN INPUT -
               -    AFTER THE ADDRESS FOLLOWED BY A SPACE.        -
               -                                                  -
               ----------------------------------------------------  */
 92   2   FILL: PROCEDURE;

 93   3       DECLARE
                  FILL$DATA BYTE,
                  FILLPROMPT(*) BYTE DATA(' -',03H);

 94   3       CALL GET$ADDRESS(0FFH);
 95   3       CALL CRT$STRING$OUT(@FILLPROMPT);
 96   3       FILL$DATA=LOW(GET$WORD(0DH,',',));

 97   3       DO WHILE HIGHLIMIT>=LOCATION.OFFSET;
 98   4              CONTENTS=FILL$DATA;
 99   4              LOCATION.OFFSET=LOCATION.OFFSET + 1;
100   4       END;

101   3   END FILL;
```

```
              $ EJECT

              /*  -----------------------------------------------------------
                  -                                                         -
                  -               I N P U T     P O R T                     -
                  -                                                         -
                  -  INPUTS THE VALUE OF AN INPUT PORT AND DISPLAYS IT ON   -
                  -  THE CRT.                                               -
                  -                                                         -
                  -----------------------------------------------------------    */
 102   2      INPUT$PORT: PROCEDURE;

 103   3         DECLARE
                       PORTNUM WORD,
                       BLANK(*) BYTE DATA(' -',03H);

 104   3         PORTNUM=GET$WORD(' ',',',' ');
 105   3         CALL CRT$STRING$OUT(@BLANK);
 106   3         CALL SHOWBYTE(INPUT(LOW(PORTNUM)));

 107   3      END INPUT$PORT;


              /*  -----------------------------------------------------------
                  -                                                         -
                  -               O U T P U T     P O R T                   -
                  -                                                         -
                  -    OUTPUTS A BYTE OF DATA TO ANY OUTPUT PORT            -
                  -                                                         -
                  -----------------------------------------------------------    */
 108   2      OUTPUT$PORT: PROCEDURE;

 109   3         DECLARE
                       PORTNUM WORD,
                       BLANK(*) BYTE DATA(' -',03H);

 110   3         PORTNUM=GET$WORD(',',',' ');
 111   3         CALL CRT$STRING$OUT(@BLANK);
 112   3         OUTPUT(LOW(PORTNUM))=LOW(GET$WORD(0DH,','));

 113   3      END OUTPUT$PORT;
```

```
              $ EJECT

              /*  -----------------------------------------------------------
                  -                                                         -
                  -                        J U M P                          -
                  -                                                         -
                  -        TRANSFERS CONTROL TO SPECIFIED LOCATION .        -
                  -                                                         -
                  -----------------------------------------------------------   */
 114   2      JUMP: PROCEDURE;

 115   3         DECLARE (CS,IP) WORD;
 116   3         DECLARE EXECUTE$CODE(*) BYTE DATA
                       (44H,       /* INC SP */
                        44H,       /* INC SP */
                        44H,       /* INC SP */
                        44H,       /* INC SP */
                        0CBH);     /* RET    */
 117   3         DECLARE EXECUTE$CODE$PTR POINTER DATA (@EXECUTE$CODE);

 118   3         CALL GET$ADDRESS(0);
 119   3         CS=LOCATION.SEGMENT;
 120   3         IP=LOCATION.OFFSET;

 121   3         CALL EXECUTE$CODE$PTR(CS,IP);

 122   3      END JUMP;
```

```
             $ EJECT

             /* ----------------------------------------------------------------------
                -                                                                    -
                -                            C O P Y                                 -
                -                                                                    -
                - COPIES A BLOCK OF MEMORY BEGINNING AT THE GIVEN NEW MEMORY LOCA-  -
                -  TION. FORMAT IS:  CSEG:OFF,RANGE-NEWSEG:NEWOFF                    -
                -                                                                    -
                ----------------------------------------------------------------------   */

 123   2     COPY: PROCEDURE;

 124   3        DECLARE
                      LOCATION2 STRUCTURE(OFFSET WORD, SEGMENT WORD),
                      MEMPTR2 POINTER AT (@LOCATION2),
                      CONTENTS2 BASED MEMPTR2 BYTE,
                      HIGHLIMIT2 WORD;

 125   3        CALL GET$ADDRESS(0FFH);               /* HIGH LIMIT REQUEST */
 126   3        LOCATION2.OFFSET=LOCATION.OFFSET;
 127   3        LOCATION2.SEGMENT=LOCATION.SEGMENT;
 128   3        HIGHLIMIT2=HIGHLIMIT;

 129   3        CALL CRT$BYTE$OUT('-');
 130   3        CALL GET$ADDRESS(0);                  /* NO HIGH LIMIT REQUEST */

 131   3        DO WHILE HIGHLIMIT2>=LOCATION2.OFFSET;
 132   4              CONTENTS=CONTENTS2;
 133   4              LOCATION.OFFSET=LOCATION.OFFSET+1;
 134   4              LOCATION2.OFFSET=LOCATION2.OFFSET+1;
 135   4        END;

 136   3     END COPY;
```

```
            $ EJECT

            /* -----------------------------------------------------------------
               -                                                               -
               -                       S U B S T I T U T E                     -
               -                                                               -
               -   DISPLAYS SPECIFIED MEMORY CONTENTS AND ALLOWS NEW CONTENTS  -
               -     TO BE ENTERED.                                            -
               -                                                               -
               -----------------------------------------------------------------   */

 137   2    SUBSTITUTE: PROCEDURE;

 138   3       DECLARE CHAR BYTE;

 139   3       CALL GET$ADDRESS(0);                    /*  NO HIGH LIMIT REQUEST  */

 140   3       DO WHILE 1;
 141   4             CALL SHOW$BYTE(CONTENTS);
 142   4             IF (CHAR:=CRTIN)=',' OR CHAR=' ' THEN
 143   4                     DO;
 144   5                      IF CHAR=' ' THEN DO;
 146   6                                     CALL CRT$BYTE$OUT('-');
 147   6                                     CONTENTS=LOW(GETWORD(',',',',','));
 148   6                                    END;
 149   5                      CALL CRT$STRING$OUT(@CRLF);
 150   5                      LOCATION.OFFSET=LOCATION.OFFSET + 1;
 151   5                      CALL SHOW$BYTE(HIGH(LOCATION.OFFSET));
 152   5                      CALL SHOW$BYTE(LOW(LOCATION.OFFSET));
 153   5                      CALL CRT$BYTE$OUT(' ');
 154   5                     END;
                                                ELSE
 155   4             RETURN;
 156   4       END;

 157   3    END SUBSTITUTE;
```

```
           $ EJECT

           /*          C O M M A N D   D I S P A T C H          */

 158  2          CALL CRT$STRING$OUT(@MONSIGNON);
 159  2          CMND=0;
 160  2          DO WHILE 1;                                  /* DO FOREVER */
 161  3                   CALL CRT$STRING$OUT(@MONPROMPT);
 162  3                   CMND=LOW(FINDB(@MONCMND,CRTIN,8));
 163  3                   IF CMND=0FFH THEN
 164  3                            CALL ERROR$MESSAGE(0);
                                  ELSE
 165  3                            DO CASE CMND;
 166  4                               CALL FILL;
 167  4                               CALL DUMP;
 168  4                               RETURN;
 169  4                               CALL INPUT$PORT;
 170  4                               CALL OUTPUT$PORT;
 171  4                               CALL JUMP;
 172  4                               CALL SUBSTITUTE;
 173  4                               CALL COPY;
 174  4                            END;
 175  3          END;
 176  2       END MONITOR;
 177  1       END SBC8612$MONITOR;
```

```
MODULE INFORMATION:

     CODE AREA SIZE     = 0460H   1120D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 001EH     30D
     MAXIMUM STACK SIZE = 001CH     28D
     393 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```

SPECIFICATIONS
Model VS-100 Vision System

PREVIOUSLY COPYRIGHTED MATERIAL
IN APPENDIX B, LEAF 150

NOT MICROFILMED

MAY BE OBTAINED

Machme Intelligence Corporation
1120 San Antonio Road
Palo Alto, California
94303

PREVIOUSLY COPYRIGHTED MATERIAL
IN APPENDIX C, LEAVES 151-154.

NOT MICROFILMED

OSO Optoelectronic Systems Operation	TN2500 Interface Specification

Maybe Obtained

General Electric Company
Room 201, BLDG 3
Electronics Park
Syracuse, New York
13221

REFERENCES

1. Rosenfeld, A., "SURVEY, Picture Processing: 1979", Computer Graphics and Image Processing, 13, 1980, pp. 46-79.

2. Pavlidis, T., "A Review of Algorithms for Shape Analysis", Computer Graphics and Image Processing, vol. 7, 1978, pp. 243-258.

3. Attneave, F., Arnoult, M. D., "The Quantitative Study of Shape and Pattern Perception", Psychological Bulletin, 53, 1956, 452-471.

4. Pavlidis, T., "Algorithms for Shape Analysis", Proceedings of the 4th International Joint Conference on Pattern Recognition, 1978, pp. 70-85.

5. Underwood, E. E., Quantitative Stereology, Addison-Wesley, 1970, p. 228.

6. Bookstein, F. L., The Measurement of Biological Shape and Shape Change, Springer-Verlag, 1978, pp. 27-57.

7. Danielsson, P. E., "A New Shape Factor", Computer Graphics and Image Processing, vol. 7, 1978, pp. 292-299.

8. Bribiesca, E., Guzman, A., "How to describe pure form and how to measure differences in shape using shape numbers", Proceedings of the IEEE Pattern Recognition and Image Processing Conference, 1979, pp. 427-436.

9. Zusne, L., Visual Perception of Form, Academic Press, New York, 1970, pp. 215-223.

10. Dudani, S. A., Breeding, K. J., McGhee, R. B., "Aircraft Identification by Moment Invariants", IEEE Transactions on Computers, vol. C-26, 1977, pp. 39-46.

11. Badler, N. I., Dane, C., "The Medial Axis of a Coarse Binary Image Using Boundary Smoothing", Proceedings IEEE Pattern Recognition and Image Processing Conference, 1979, pp. 286-291.

12. Shapiro, B., Pisa, J., Sklansky, J., "Skeletons from Sequential Boundary Data", Proceedings IEEE Pattern Recognition and Image Processing Conference, 1979, pp. 265-270.

13. Blum, H., "A Transformation for Extracting Descriptors of Shape", Symposium on Models for the Perception of Speech and Visual Form, MIT Press, 1967, pp. 362-380.

14. Rosenfeld, A., Pfaltz, J. L., "Sequential Operations in Digital Picture Processing", Journal of Assoc. of Computing Machinery, 13, 1966, pp. 471-494.

15. Montanari, U., "Continuous Skeletons from Digitized Images", Journal of Assoc. of Computing Machinery, 16, 1969, pp. 536-549.

16. Fu, K. S., Syntactic Methods in Pattern Recognition, Academic Press, 1974.

17. Fu, K. S., Syntactic Pattern Recognition: Applications, Springer-Verlag, 1977.

18. Pavlidis, T., Structural Pattern Recognition, Springer-Verlag, 1977.

19. Zahn, C. T., Roskies, R. Z., "Fourier Descriptors for Plane Closed Curves", IEEE Transactions on Computers, vol. C-21, 3, 1972, pp. 269-281.

20. Persoon, E., Fu, K. S., "Shape Descrimination Using Fourier Descriptors", IEEE Transactions on Systems, Man and Cybernetics, vol. SMC-7,3, 1977, pp. 170-179.

21. Chien, R. T., "Hardware for Visual Image Processing", IEEE Transactions on Circuits and Systems, vol. CAS-22, 6, June, 1975, pp. 541-551.

22. Fry, P. W., "Introduction to Solid State Image Scanners", Proceedings of a Symposium on Industrial Applications of Solid State Image Scanners, Sira Institute, London, 1978, pp. 2-8.

23. Goldberg, M., Gougeon, F., "Digital Image Processing Hardware", Proceedings of the Canadian Communications and Power Conference, 1980, pp. 417-420.

24. Rosenfeld, A., Picture Processing By Computer, Academic Press, New York, 1969.

25. Levine, M. A., "Feature Extraction: A Survey", Proceedings of the IEEE, vol. 57, 8, Aug. 1969.

26. Perkins, W. A., "Simplified Model-Based Part Locator", Proceedings of the 5th International Conference on Pattern Recognition, 1980, pp. 260-263.

27. Rosenfeld, A., "Connectivity in Digital Pictures", Journal of the Association of Computing Machinery, vol. 17, 1, Jan. 1970, pp. 146-160.

28. Agin, G. J., "Image Processing Algorithms for Industrial Vision", written correspondence, unpublished.

29. Freeman, H., "On the Encoding of Arbitrary Geometric Configurations", IRE Transactions on Electronic Computers, 10, (2), June, 1961, pp. 260-268.

30. Freeman, H., Shapiro, R., "Determining the Minimum-Area Encasing Rectangle for an Arbitrary Closed Curve", Communications of the Association for Computing Machinery, 18, 7, 1975, pp. 409-413.

31. Freeman, H., "Computer Processing of Line-Drawing Images", Computing Surveys, 6, 1, 1974, pp. 57-97.

32. Cederberg, R., "Chain-Link Coding and Segmentation for Raster Scan Devices", Computer Graphics and Image Processing, 10, 1979, pp. 224-234.

33. Sobel, I., "Neighborhood Coding of Binary Images for Fast Contour Following and General Binary Array Processing", Computer Graphics and Image Processing, 8, 1978, pp. 127-135.

34. Martini, P., Nehr, G., "Recognition of Angular Orientation of Objects with the Help of Optical Sensors", The Industrial Robot, June, 1979, pp. 62-69.

35. deCoulon, F., Kammenos, P., "Polar Coding of Planar Objects in Industrial Robot Vision", Nachricht Technik, 10, 1977, pp. 665-671.

36. Rosenfeld, A., "Compact Figures in Digital Pictures", IEEE Transactions on Systems, Man, and Cybernetics, SMC-4, 1974, pp. 221-223.

37. Kulpa, Z., "Area and Perimeter Measurement of Blobs in Discrete Binary Pictures", Computer Graphics and Image Processing, 6, 1977, pp. 434-451.

38. Sankar, P. V., Krishnamurthy, E. V., "On the Compactness of Subsets of Digital Pictures", Computer Graphics and Image Processing, 8, 1978, pp. 136-143.

39. Ellis, T. J., Proffitt, D., "Measurement of the Lengths of Digitized Curved Lines", Computer Graphics and Image Processing, 10, 1979, pp. 333-347.

40. Parks, J. R., "Image Processing - A New Tool for Quality Control", Proceedings of the International Seminar on Automation and Inspection Applications of Image Processing Techniques, Sept. 1977, pp. 2-7.

41. Agin, G. J., "An Experimental Vision System for Industrial Applications", Proceedings of the 5th International Symposium on Industrial Robots, Sept., 1975, pp. 135-148.

42. Agin, G. J., "Servoing with Visual Feedback", Proceedings of the 7th International Symposium on Industrial Robots, 1977, pp. 551-560.

43. Gleason, G. J., Agin, G. J., "A Modular Vision System for Sensor-Controlled Manipulation and Inspection", Proceedings of the 9th International Symposium on Industrial Robots, 1979, pp. 57-70.

44. Hill, J., Park, W. T., "Real Time Control of a Robot with a Mobile Camera", Proceedings of the 9th International Symposium on Industrial Robots, 1979, pp. 233-246.

45. Rosen, C. et. al., Machine Intelligence Research Applied to Industrial Automation, Eighth Report, Stanford Research Institute, Menlo Park, California, August, 1978.

46. Birk, J., et. al., "General Methods to Enable Robots with Vision to Acquire, Orient and Transport Workpieces", Fourth Report, University of Rhode Island, Kingston, Rhode Island, July 1978.

47. Seres, D. A., et. al., "Visual Robot Instruction", Proceedings of the 5th International Symposium on Industrial Robots, Sept. 1975, pp. 113-126.

48. Kelley, R., et. al., "Algorithms to Visually Acquire Workpieces", Proceedings of the 7th International Symposium on Industrial Robots, 1977, pp. 497-506.

49. Birk, J., et. al., "Acquiring Workpieces: Three Approaches Using Vision", Proceedings of the 8th International Symposium in Industrial Robots, 1978, pp. 724-733.

50. Kelly, R., et. al., "A Robot System Which Feeds Workpieces Directly from Bins into Machines", Proceedings of the 9th International Symposium on Industrial Robots, 1979, pp. 339-355.

51. Duncan, D., et. al., "Hardware Computation of Image Features Based on Local Gradient Direction Histograms", Proceedings of the 10th International Symposium on Industrial Robots, 1980, pp. 369-379.

52. Pugh, A., et. al., "Versatile Parts-Feeding Package Incorporating Sensory Feedback", Proceedings of the 8th International Symposium on Industrial Robots, 1978, pp. 206-217.

53. Perkins, W. A., "A Model-Based Vision System for Industrial Parts", IEEE Transactions on Computers, C-27, 2, Feb. 1978, pp. 126-143.

54. Baird, M. L., "Image Segmentation Technique for Locating Automotive Parts on Belt Conveyors", Proceedings of the 5th International Joint Conference on Artificial Intelligence, Aug. 1977, pp. 694-695.

55. Ward, M. R., et. al., "CONSIGHT: A Practical Vision-Based Robot Guidance System", Proceedings of the 9th International Symposium on Industrial Robots, 1979, pp. 195-211.

56. Perkins, W. A., "Area Segmentation of Images Using Edge Points", IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-2, 1, January 1980, pp. 8-15.

57. Ambler, A. P., et. al., "A Versatile Computer-Controlled Assembly System", Proceedings of the 3rd International Joint Conference on Artificial Intelligence, 1973, pp. 298-307.

58. Stewart, R. J. S., "The State-of-the-Art in Vision Systems", State-of-the-Art in Adaptable-Programmable Assembly Systems, International Fluidics Services Ltd., England, pp. 97-120.

59. Koskinen, K., "Object Recognition and Handling in an Industrial Robot System with Vision", Proceedings of the 8th International Symposium on Industrial Robots, 1978, pp. 744-755.

60. Pugh, A., et. al., "Research Experience in Visual and Tactile Feedback", Proceedings of the International Seminar on Automation and Inspection Applications of Image Processing Techniques, 1977, pp. 98-103.

61. Nitta, Y., "Visual Identification and Sorting with TV-camera Applied to Automated Inspection Apparatus", Proceedings of the 10th International Symposium on Industrial Robots, 1980, pp. 141-152.

62. Yachida, M., "A Machine Vision for Complex Industrial Parts with Learning Capability", Proceedings of the 4th International Joint Conference on Artificial Intelligence, 1975, pp. 819-826.

63. Rosen, C., "Machine Vision and Robotics: Industrial Requirements", Computer Vision and Sensor-Based Robots, Plenum, 1979, pp. 3-22.

64. Igarashi, K., et. al., "Fully Automated Integrated Circuit Wire Bonding System", Proceedings of the 9th International Symposium on Industrial Robots, 1979, pp. 87-97.

65. Batchelor, B. G., "Automatic Visual Inspection in Industry", The Industrial Robot, December 1978, pp. 174-176.

66. Holland, S. W., et. al., "CONSIGHT-I: A Vision-Controlled Robot System for Transferring Parts from Belt Conveyors", Computer Vision and Sensor-Based Robots, Plenum, 1979, pp. 81-100.

67. VanderBrug, G. J., et. al., "A Vision System for Real Time Control of Robots", Proceedings of the 9th International Symposium on Industrial Robots, 1979, pp. 213-231.

68. TN2500 Solid State Video/Digital Camera OPERATING MANUAL. April 1980, General Electric, Electronic Systems Division.

69. SBC 501 Direct Memory Access Controller Hardware Reference Manual 1976, Intel Corp.

70. iSBC 86/12 Single Board Computer Hardware Reference Manual, 1978, Intel Corp.

71. PL/M-86 Programming Manual, 1978, Intel Corp.

72. MCS-86 Software Development Utilities Operating Instructions for ISIS-II Users, 1978, Intel Corp.