Fall 1988

A Structured Light Method for Sensing Alignment During Automated Truss Assembly

Jayesh K. Champaneri
Old Dominion University

A STRUCTURED LIGHT METHOD FOR SENSING ALIGNMENT DURING AUTOMATED TRUSS ASSEMBLY

by

Jayesh K. Champaneri
B.E. May 1984, University of Poona

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment
of the Requirements for the Degree of

MASTER OF ENGINEERING

ELECTRICAL ENGINEERING

OLD DOMINION UNIVERSITY
November, 1988

Approved by:

David L. Livingston (Director)

Karin Cornils

Nicolas Alvertos

ABSTRACT

A STRUCTURED LIGHT METHOD FOR CYLINDRICAL BEAM ALIGNMENT

Jayesh K. Champaneri

Old Dominion University, 1988

Director: Dr. David L. Livingston

A new method using structured light is proposed to obtain visual feedback information for aligning two cylindrical beams. For a robotic system employed to perform alignment operations in real time it is essential that the accompanying vision system does not pose a heavy burden on the computing machinery and reduce the overall speed of operation. The method proposed here involves two stripes of light projected on each cylinder. One picture frame is sufficient to completely determine the position of the cylinder in space. An experiment was conducted to demonstrate the principle of this method. Results showed that the errors involved were within the practical limitations of the components.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor, Dr. David L. Livingston, for his guidance, encouragement and persistence throughout the course of this research, and Mr. Plesent W. Goode, for his guidance on the practical aspects of the project.

I would like to thank Ms. Karin Cornils for her time and perserverance in introducing me to the image processing unit and its software, and Dr. Nicolas Alvertos, for his time and effort in many stimulating discussions.

I would also like to acknowledge the financial support by the National Aeronautics and Space Administration under contract NAS1-17993-71.

TABLE OF CONTENTS

		PAGE
LIST OF TABLES		v
LIST OF FIGURES		vi
LIST OF PLATES		vii
CHAPTER		
I	INTRODUCTION	1
	Objective	3
	Thesis Structure	3
II	BACKGROUND	5
	A General Approach	5
	Previous Work	6
	Proposed Methods	8
III	THEORETICAL DEVELOPMENT	10
	System Configuration	10
	Geometrical Analysis	18
IV	PRACTICAL IMPLEMENTATION	28
	Empirical Evidence	28
	Approximation Error	32
	Practical Considerations	36
	Experimental Set-Up	41
	Results	44
V	CONCLUSIONS AND FUTURE RESEARCH	46
	Remarks	46
	Future Research	48

PAGE

LIST OF REFERENCES. 50

APPENDIX A 57

Program Listings 58

LIST OF TABLES

TABLE		PAGE
1.	Results	45

LIST OF FIGURES

FIGURE		PAGE
1.1.	Geometrical configuration of cylinders K and L with respect to the cartesian coordinate system.	2
3.1.	Initial cylinder position in space	11
3.2.	Schematic top view of cylinder, projectors and camera.	13
3.3.	Height comparison of two cylinders	14
3.4.	Depth comparison of two cylinders.	16
3.5.	Schematic of three-dimensional set-up	17
3.6.	Planes of light casting light stripes	19
3.7.	Image stripes on the focal plane	20
3.8.	G--the axis of rotation	23
3.9.	H--the final alignment point	25
4.1a.	Image stripe on a planar surface	29
4.1b.	Image stripe on cylinder as seen by the camera.	29
4.2.	Image stripe comparison on curved and planar surface	31
4.3.	Representation of the corners of the image stripe.	33
4.4.	Projection of corner points.	34
4.5.1.	Graph comparison for corner detection	39
4.5.1.	Graph comparison for corner detection	40
4.5.1.	Graph comparison for corner detection	42

LIST OF PLATES

PAGE

PLATE

1. Captured image at sixteen levels of gray . . 52

2. Binary thresholded image. 53

3. Image after edge detection 54

4. Filtered image 55

5. Image showing corner and center points . . . 56

CHAPTER I

INTRODUCTION

Vision as a means of feedback in robotic manipulator assisted operations is gaining widespread attention. Various techniques have been explored in this area depending upon the nature of information to be extracted. For example, edge detection techniques are being particularly employed for object identification purposes. For more specific applications where sufficiently accurate information about the object's surface features, position, and orientation is required, structured light methods are often used [5], [6], [7].

In a structured light method, the object whose features or properties are to be determined is illuminated either by stripes of light or a grid pattern. The stripes of light are commonly referred to as planes of light since they are formed due to light emanating in a single plane from the projector for each stripe. The grid pattern consists of two sets of planes of light where one set has the planes of light perpendicular to those in the other set. The projection pattern used in this research consists of four planes of light forming two stripes on each of the two cylinders. The position of two cylinders with respect to the coordinate system is shown in Figure 1.1. The axes of the two cylinders are colinear and parallel to the y-axis when they are aligned.

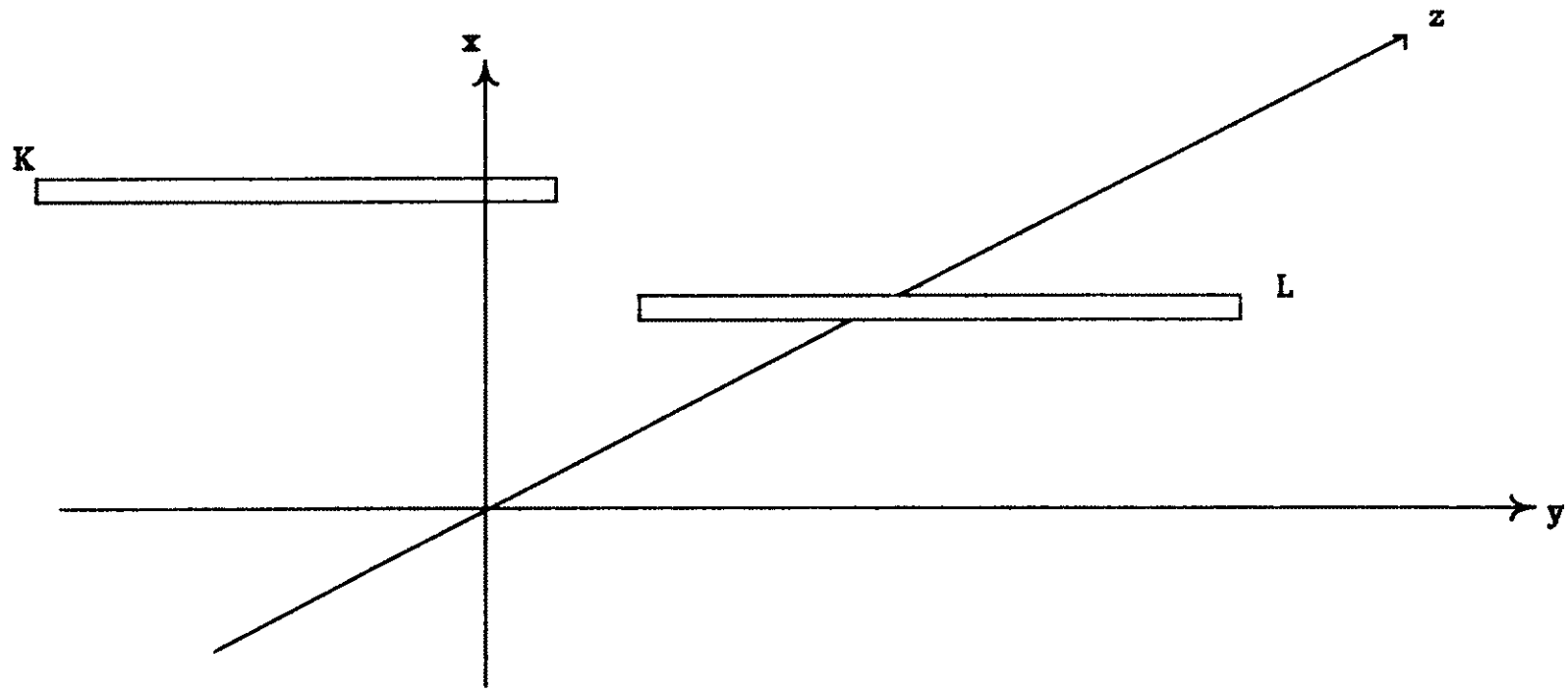


Figure 1.1. Geometrical configuration of cylinders K and L with respect to the cartesian coordinate system.

OBJECTIVE

The objective of this thesis is to investigate the effectiveness of a visual feedback mechanism using structured light for a robotic system which performs assembly operations of a structure consisting of beams and nodes [16]. This assembly operation can be carried out either in the fully automatic mode or the teleoperated mode. At least two robotic arms are required to manipulate the cylinders, one for each cylinder. Before the two cylinders can be assembled, it is necessary that they are axially aligned. In the teleoperated mode, an operator is given feedback about the degree of alignment. Information about the direction and the amount required to move the cylinders to achieve alignment is then specified. In the fully automatic mode, the process of alignment of the cylinders is entirely performed by the robotic system. Direct commands for manipulating the cylindrical beams are presented and are decomposed and interpreted by the system as primitive commands for the arms handling each beam. It is here that primitive commands are defined to translate or rotate the cylindrical beam about a given axis by a certain amount.

THESIS STRUCTURE

This thesis describes how a structured light pattern consisting of stripes of light can be applied to the task of performing cylindrical beam alignment in real time with little computational overhead. The sequence of chapters closely matches the evolution of ideas, considerations and modifications that were required in its implementation. The image of the stripes of light projected on the cylinder is shown in Plate 1. The thresholded image as well as further processed images are shown in plates 2, 3, 4, and 5. Information about the position of the axis of the cylinder is obtained from the image shown in Plate 5.

Chapter II discusses the development of a general approach from which the final idea that was implemented was evolved. A survey of past approaches to robotic vision problems is also presented in this chapter. The need for the method presented in this thesis is also addressed.

The geometrical configuration for the sake of analysis of the proposed scheme is introduced in Chapter III. This chapter explains the coordinate system used and the relative positions of the projectors, camera, and cylinders. An empirical result that relates the position of a point on the axis of the cylinder with that of the image as seen by the camera is formulated in Chapter IV. The actual experimental set-up and the practical difficulties which necessitated slight modifications are also presented in this chapter.

Chapter V summarizes the results and presents the conclusions that are drawn from this research. Research topics for further exploration are also suggested in this chapter.

CHAPTER 2

BACKGROUND

A GENERAL APPROACH

The proposed method to perform the task of assembling trusses and nodes for the construction of large space structures, such as the space station, involves robotic manipulators aided with vision capabilities for feedback [16]. The feedback information is to be such that it can be interpreted by a human controlling the robotic arms in a teleoperated mode, or by the robotic control system in an automatic mode. This feedback information consists of the amount and direction in which the cylindrical beams are to be rotated and then displaced to achieve alignment.

In using vision as a feedback aid to sense the position of objects, various techniques are generally employed depending upon the nature of the problem. The object may be viewed in ambient light and edge detection may be performed on its image to identify the object. Other features may be detected by carrying out specific processing techniques, for example, using reflectance properties and light scattering effects to determine surface texture, or using a grid pattern of light to determine object curvature. Most of the time positional information about an object, whether known or unknown, is determined using structured light patterns. The commonly used light patterns are the mesh projection which consists of horizontal and vertical stripes of light, and the grid projection which consists of evenly spaced points in a horizontal and vertical direction [5]. Some structured

light methods use either horizontal or vertical stripes [6]. The choice of a particular type of projection system depends on the system requirements.

The structured light pattern chosen here consists of four stripes of light. Two light stripes project on each cylindrical beam. It is assumed that we know that the objects whose positions are to be determined are cylinders, but their diameters are unknown. As will be apparent from the discussion in forthcoming chapters, it is not necessary to determine the cylinder diameters. The light stripes are projected such that the vertical stripes are perpendicular to the cylinders' axes when they are aligned.

PREVIOUS WORK

The goal of the present research is to investigate a structured light robot vision system suitable for visual servoing in real time. This implies the capability to completely specify the position and orientation of an object in space relative to the robot's coordinate system (refer to Chapter III) in real time.

Different methods and their capabilities using structured light are now discussed. The definition of the position and orientation in space of an object relative to the robot's coordinate system has six degrees of freedom as explained by J. Albus, et al. [1]. Since the object used here is a cylinder, one of the degrees of freedom is about its axis. In this study, this degree of freedom is immaterial. This leaves five degrees of freedom to be determined. The use of structured light to extract three-dimensional shape and position information is a well known technique. A plane of light has been demonstrated to be a practical solution for feedback in robot arc welding [2], [3]. Discontinuities in the image of the plane of light perceived by a camera are analyzed and feedback information is given to

the robot controller. Where determining position of objects is concerned, this method does not generate sufficient information.

Other techniques which recognize objects and determine their position have also been studied. Various kinds of pattern recognition and image processing techniques have been employed. Will and Pennington [5] have demonstrated the use of grid coding for curvature measurement. The object, which is to be identified and whose position is to be determined, is illuminated using either a one-dimensional or two-dimensional grating. Two-dimensional gratings provide an extra degree of freedom in the feature detection task. This technique is very useful in polyhedral object identification. Though this method is computationally taxing, it can extract range information, segment plane area, etc., from a scene. This technique applied to cylindrical beam alignment would generate redundant information.

Methods for making surface measurements using space encoded beams have been studied by Posdamer and Altschuler [6]. Ishii and Nagata have used a laser tracker to extract feature information from the object [7]. The laser tracks the edge of the object and determines the position of points on the edge using triangulation. This method is slower as compared with other methods discussed above, as well as the technique used in this research.

Curved object location has been studied and techniques including stereo measurements, material identification, and simulated imagery have been employed [8]. Surface recognition which includes determining its location and orientation, specifically as it applies to quadric surfaces, is described in detail in a paper by E.L. Hall, et al. [9].

R. M. Haralick has explained the interpretation of information of the three-dimensional world on a two-dimensional image [10]. This involves the

understanding of perspective transformations which enables a vision system to perform scene analysis. In studying perspective transfomations it is important to know the camera parameters; such as the focal length of its lens system, the position of the optical center and dimensions of the image plane. A rectangle of known size is taken as the object and the perspective transformations that occur on its image are analyzed [11].

The task discussed here for alignment of cylindrical beams in space can be used in the teleoperated mode. A recognition operator for telerobotic vision is discussed by P.W. Goode [12]. An intelligent system employed with robotic manipulators can enhance the capabilities of accomplishing this task in different kinds of environments [13], [14]. A high level overview of the requirements of object recognition have been discussed in a paper by P. Besl and R. Jain [15].

PROPOSED METHODS

Two methods are proposed here to perform the alignment task using four light stripe projectors. In the first method, two cameras are used and positioned between the two light stripe projectors for each cylinder. At this time no positional information about the cylinders is investigated; however, the necessary translational and rotational displacements that are to be initiated are determined. Such displacements are made until the two cylinders are axially aligned.

In the second method, the positions and orientations of the cylinders are determined relative to the camera and light stripe projectors. From this information, the amount and direction in which the cylinders are to be rotated and displaced is computed and necessary action is taken to perform alignment. Various techniques that generate visual feedback information in a robotic system

have been discussed here. The geometrical configuration of the camera, projectors, and cylindrical beams is discussed in the next chapter.

CHAPTER III

THEORETICAL DEVELOPMENT

In the previous chapter a brief introduction on potential methods to achieve alignment of two cylindrical beams was presented. In the following material two new methods are described in detail to sense the alignment of two cylinders. Both methods, as proposed, employ two light-stripe projectors for each cylinder. The first method consists of analyzing the properties of the relative position of the image stripes as seen by the camera; whereas the second method determines the equation of the cylinder in space with respect to the camera and light stripe projectors. The coordinate system is as explained in Chapter I. In both methods the two cylinders are independently rendered parallel to the y-axis before they are axially aligned.

SYSTEM CONFIGURATION

For method one, consider cylinder L as shown in Figure 3.1. Points A and B represent the center of the segments projected on the cylinder by stripes of light. The image points A and B on the camera focal plane will have their own set of pixel coordinates. Row pixels correspond to y coordinates while column pixels correspond to x coordinates. It is necessary to first align the cylinder axis parallel to the y-axis in the pixel coordinates of the camera focal plane.

Consider the configuration shown in Figure 3.2. The camera focal plane is parallel to the XY-plane. It intersects with the z-axis at z = -k, where "k" is the

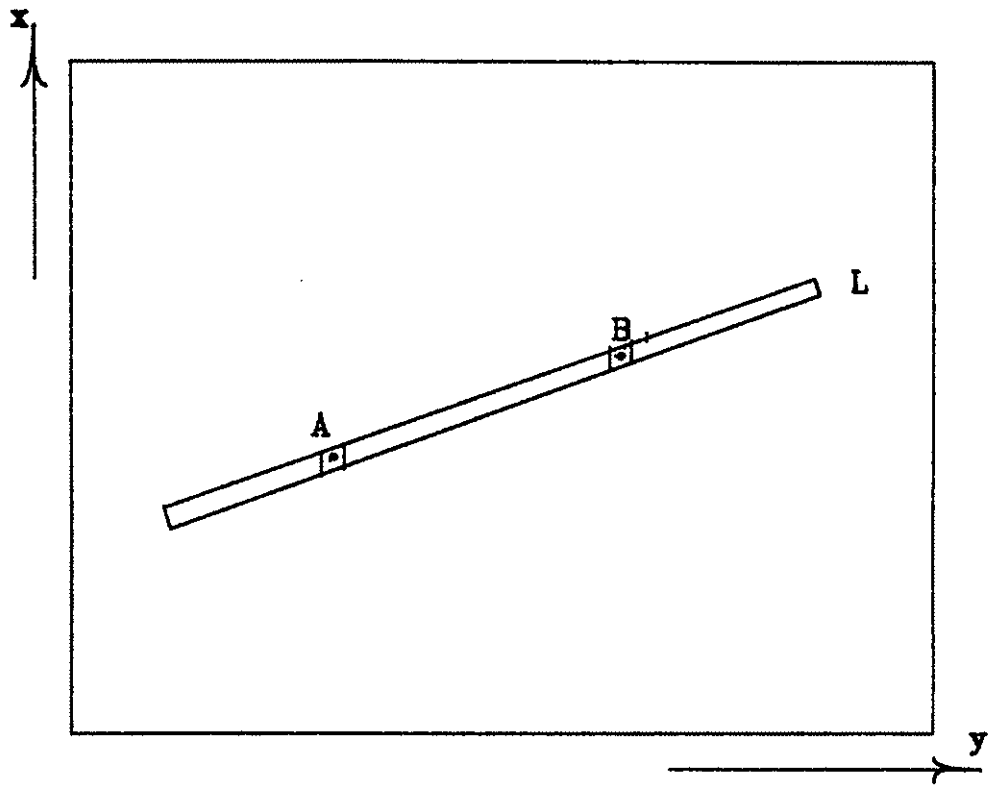


Figure 3.1. Initial cylinder position in space.

focal length of the camera lens. From this figure we also note that aligning the cylinder axis parallel to the y-axis in the pixel coordinates of the camera focal plane is the same as aligning the cylinder axis parallel to the YZ-plane. This is done by rotating the cylinder about an axis parallel to the z-axis passing through the midpoint of segment AB. The cylinder is rotated until the x-pixel coordinates of the image points A and B become equal. When this condition is achieved, the cylinder lies in a plane parallel to the YZ-plane.

Since method one compares the symmetry of the two image points, A and B, about a vertical bisector of the camera focal plane, it is necessary that the camera lies exactly at the midpoint of the segment connecting projectors P_1 and P_2. Hence, this arrangement requires two cameras, one for each cylinder.

If the cylinder is parallel to the y-axis, the points A and B lie symmetrically about a line which vertically bisects the focal plane. If the cylinder is not parallel to the y-axis, then due to perspective these points are not equidistant. From the geometry shown in Figure 3.2 it is clear that the further the point (A or B) is from the camera, the closer its image is to the vertical bisector in the focal plane. Thus the direction in which the cylinder is to be rotated, can be determined. The cylinder can then be rotated about an axis parallel to the x-axis passing through the midpoint of segment AB, until l_A equals l_B. When this condition is satisfied, the cylinder is parallel to the y-axis. A similar procedure is carried out for the other cylinder K.

The first step necessary to align the two cylinders is to bring them to the same horizontal level; that is, to make the x-coordinate value of points A and B corresponding to cylinder L and points C and D corresponding to cylinder K the same as shown in Figure 3.3.

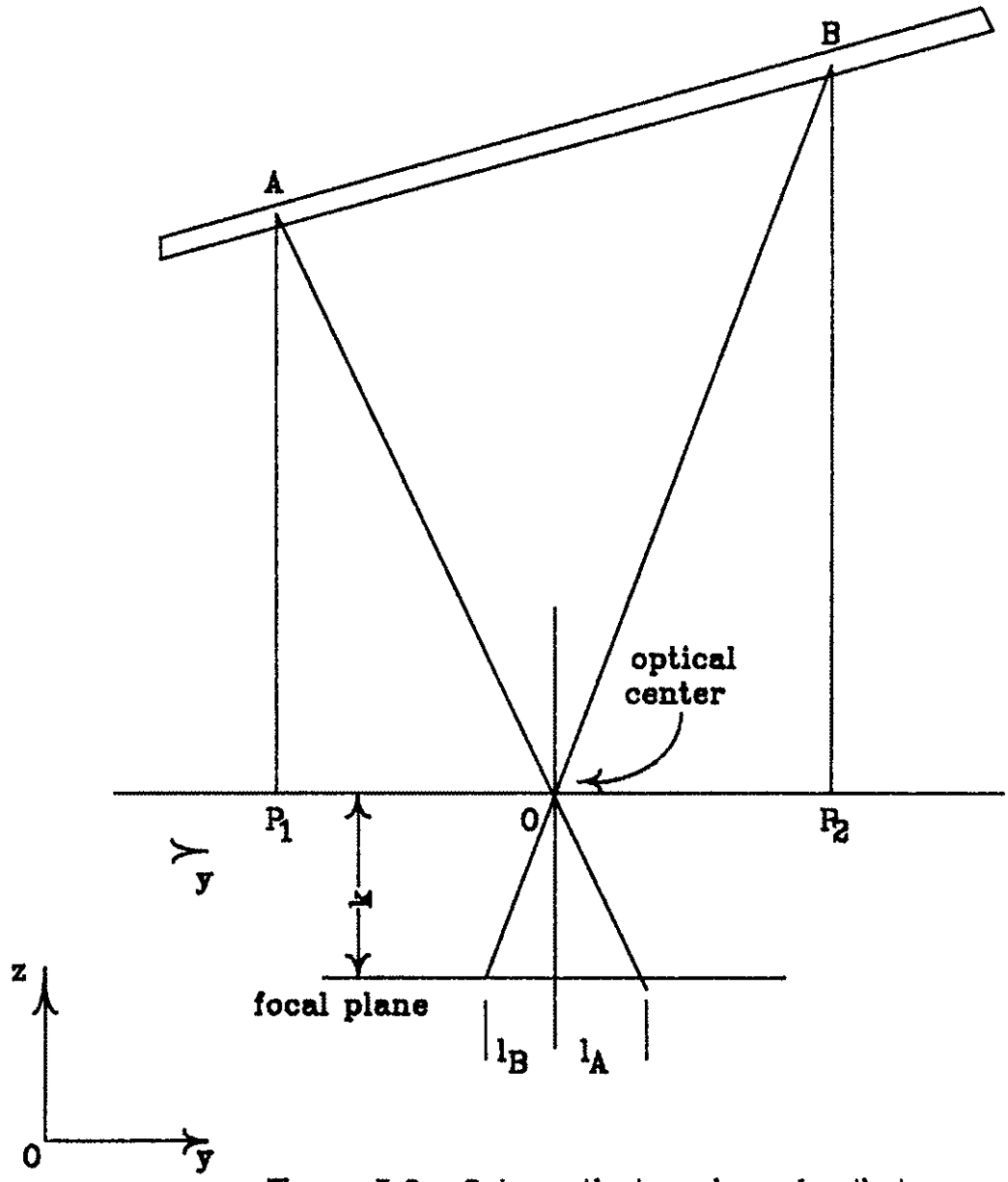


Figure 3.2. Schematic top view of cylinder, projectors and camera.

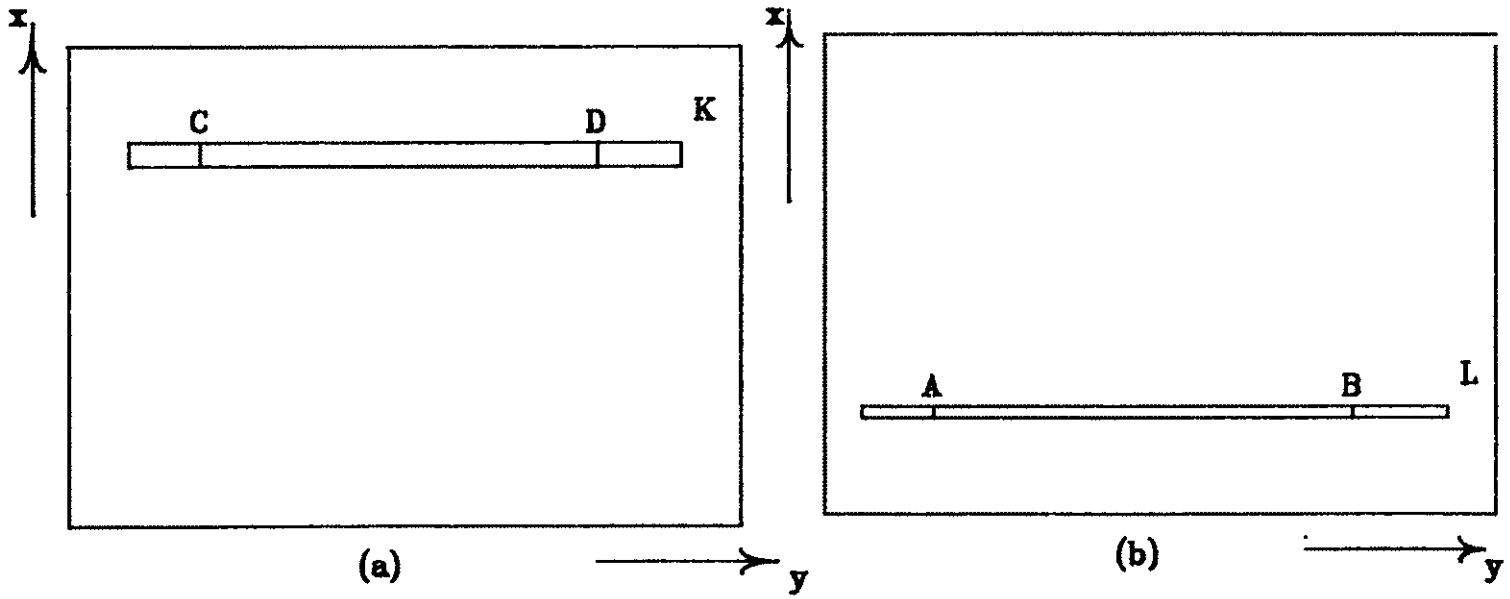


Figure 3.3. Height comparison of two cylinders

Here the x-coordinate of any point on cylinder K is larger than the x-coordinate of any point on cylinder L. Thus, cylinder K is translated in such a direction that its x-coordinate decreases and cylinder L is translated such that its x-coordinate value increases. This is done until the x-coordinates of both cylinders become equal. The distance between points A and B, and C and D are compared, as is represented in Figure 3.4. Since the two cylinders are not the same distance away from the camera, l_K and l_L are not equal; this signifies that the z-coordinates are unequal.

Due to perspective, the size of l_K or l_L is larger if the corresponding cylinder has a smaller z-coordinate value. In the example, l_L is larger than l_K since cylinder L is closer to the camera than cylinder K. From this relationship the direction in which the cylinders are to be moved along the z-direction is known. L is moved away from the camera while K is moved towards the camera until l_K equals l_L.

As has been demonstrated, this method tends to achieve alignment of the two cylinders by comparing the relative positions of the image stripes on the camera and deducing the direction of motion of the cylinders to achieve alignment. No attempt is made to determine the position of the cylinder with respect to the camera and light stripe projectors.

The second method proposed, which was finally used and is described extensively, uses only one camera which is placed exactly in between the two sets of two projectors. This method determines the equation of the axis of each cylinder and requires the use of algebraic and trigonometric techniques for triangulation.

Figure 3.5 shows the configuration for use of the second method; the XYZ coordinate system is as shown. The optical center of the camera lens is at the origin. Its optical axis coincides with the z-axis. The camera focal plane is

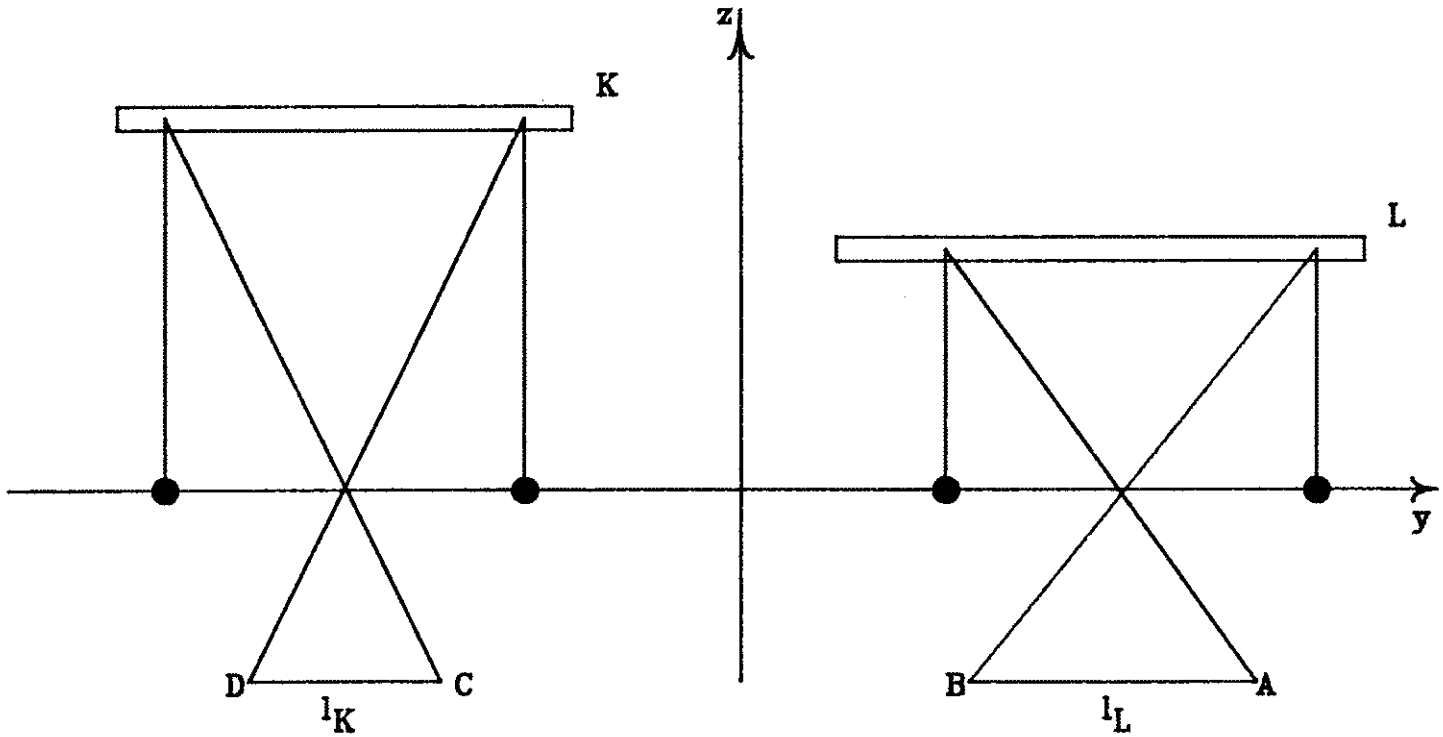


Figure 3.4. Depth comparison of two cylinders

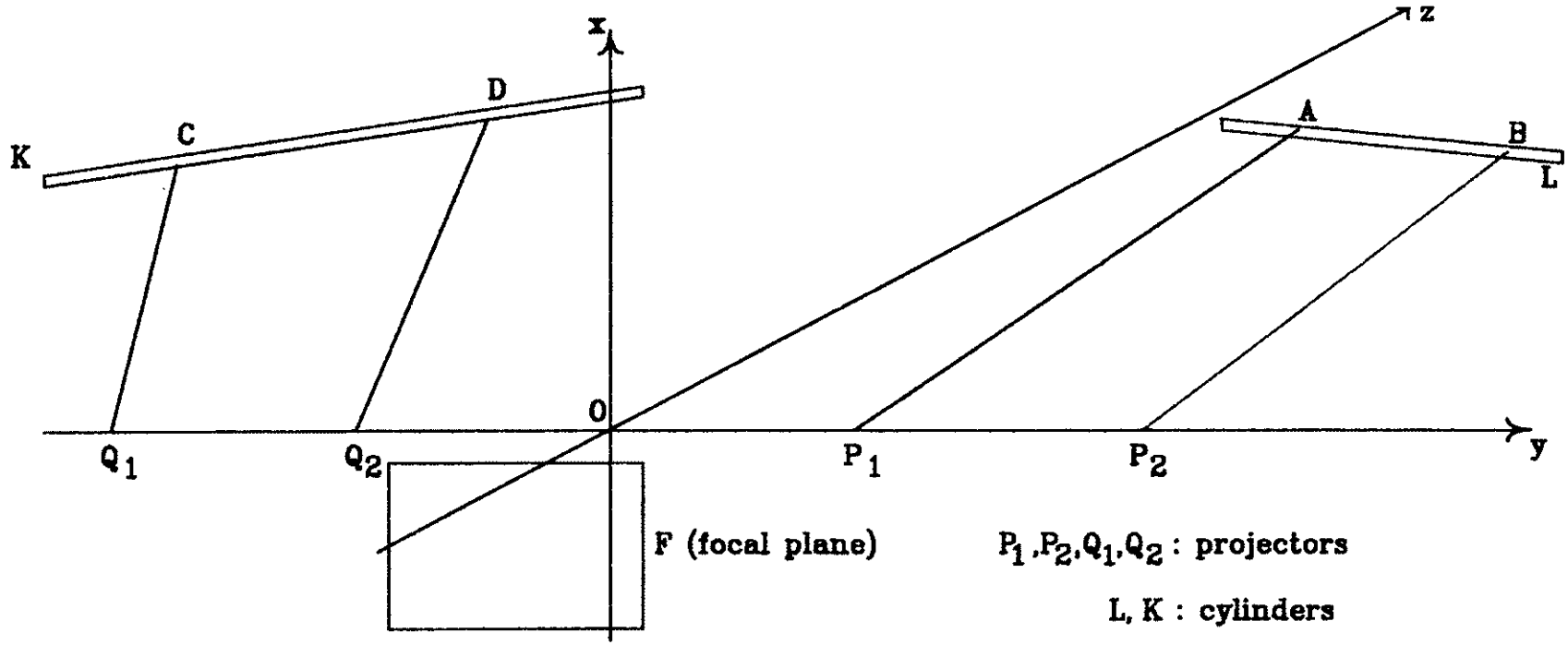


Figure 3.5. Schematic of three–dimensional set–up.

perpendicular to the z-axis and is at a distance "k" in the negative z-direction. Projectors P_1 and P_2 lie on the positive y-axis and cast stripes of light on cylinder L forming patterns A and B. Projectors Q_1 and Q_2 lie on the negative y-axis and cast stripes of light on cylinder K forming patterns C and D.

The primary requirement in the alignment task is to determine the equation of the axis of the cylinder in question. This requires that the coordinates of at least two points on that axis be known in the reference space. Two parallel stripes of light are projected on the cylinder. The stripes of light are assumed to be of zero thickness so that the image formed on the cylinder is either a straight or a curved line depending upon the orientation of the cylinder and the viewing position.

GEOMETRICAL ANALYSIS

As shown in Figure 3.6, projectors P_1 and P_2 project stripes of light on cylinder L which form two patterns A and B. For the sake of analysis, let the diameter of the cylinders L and K be zero; therefore, stripes A and B are reduced to points and from now on shall be referred to as points A and B. Thus A and B represent the points on the axis of the cylinder, that is, the points due to the intersection of the axis and the plane of the light stripes. In Figure 3.7, L is in the positive octant of the three-dimensional cartesian coordinate system and projectors P_1 and P_2 lie on the y-axis. P_1 is at a distance "m" from the origin and P_2 is at a distance "n" from the origin. The planes of these stripes of light are parallel to the XZ-plane. Rectangle F represents the focal plane or image plane of the camera, which consists of picture elements or pixels; thus rays from all patterns imaging on the focal plane pass through the origin. Focal plane F is

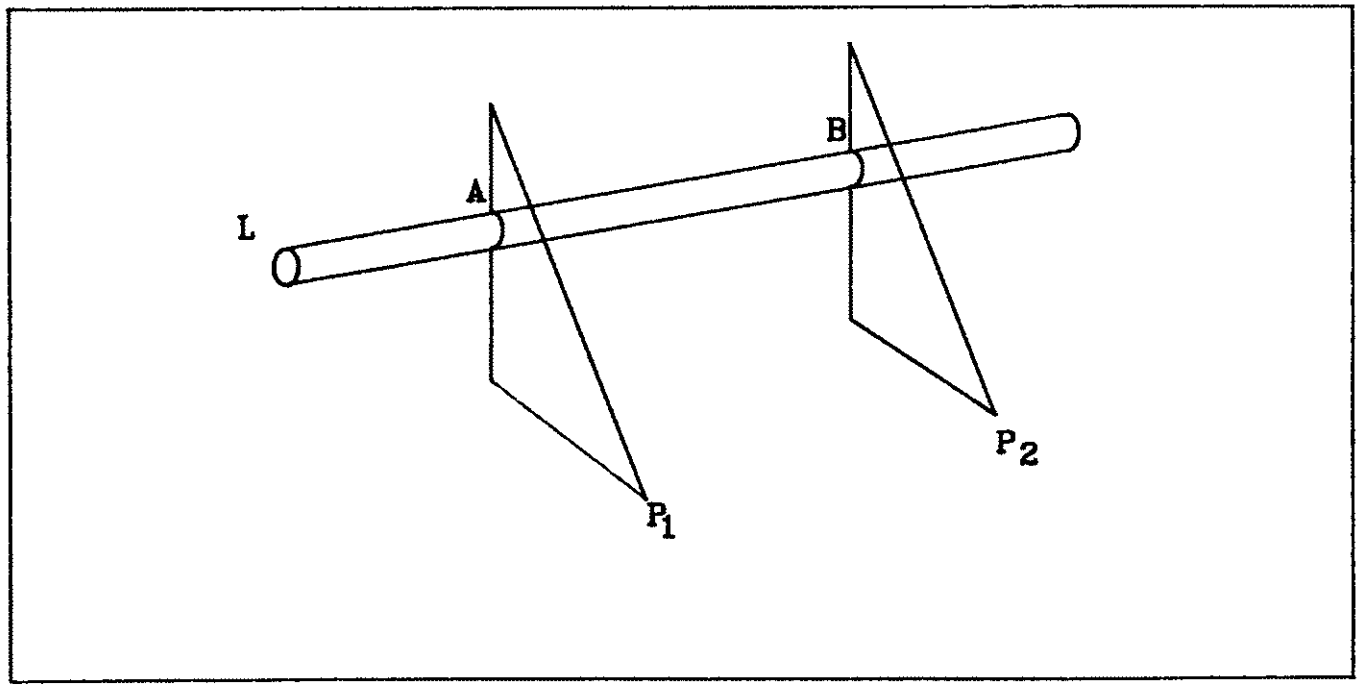


Figure 3.6 Planes of light casting light stripes.

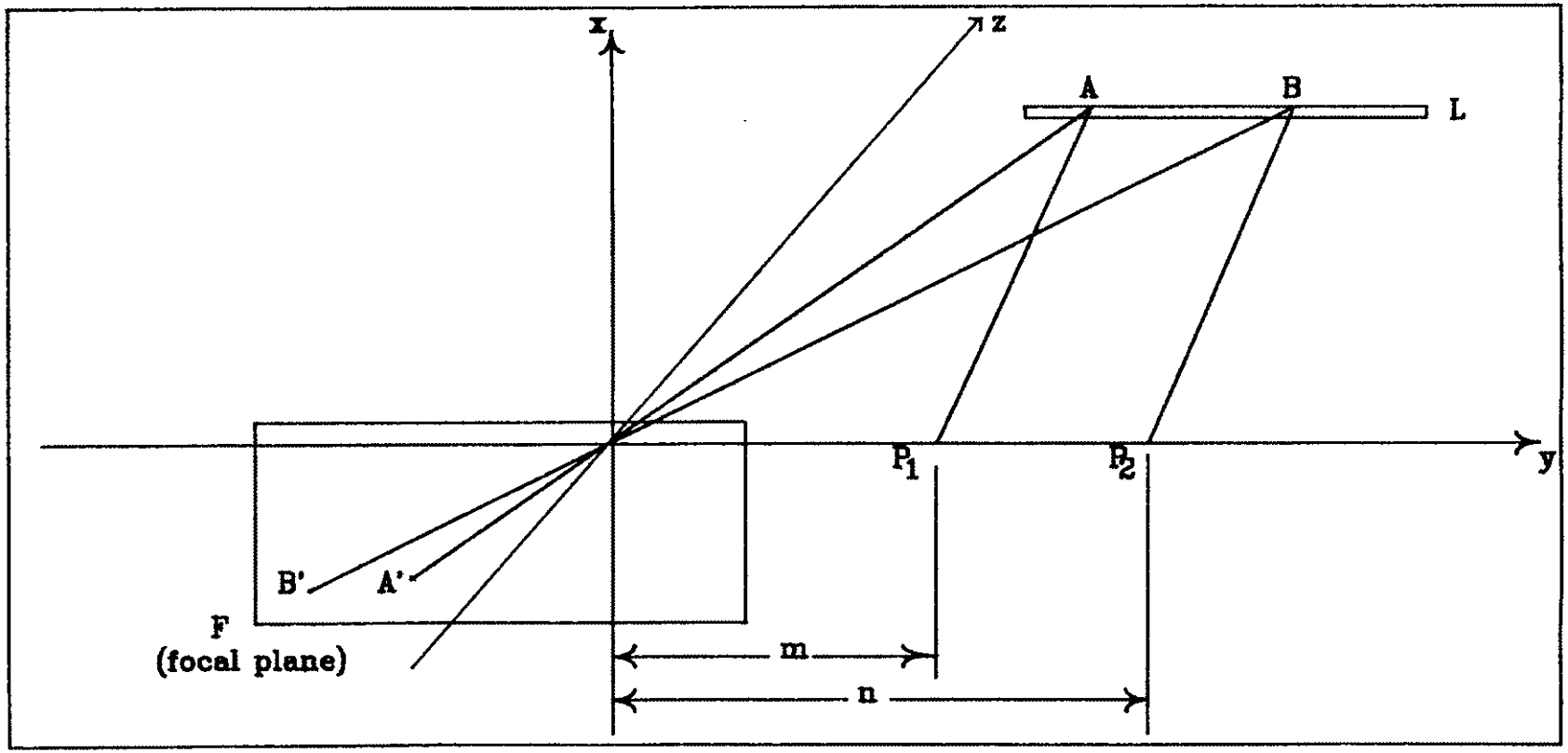


Figure 3.7. Image stripes on the focal plane.

parallel to the XY-plane and lies in the negative z-direction at a distance "k" from the origin. The z-axis passes through the center of F.

A' and B' are the image points on the camera focal plane corresponding to points A and B, respectively. The lines connecting A to A' and B to B' pass through the origin. The positions of points A' and B' with reference to F is known since the pixels to which they correspond can be easily determined. Since the camera position and orientation in space are known, the position of the focal plane F is also known. Hence the positions of points A' and B' can be determined in space.

To find the positions of points A and B in space, we need to solve the equations of the lines AA' and BB' with the equations of the planes due to the stripes of light formed by projectors P_1 and P_2 respectively.

Let the position of point A' be (x_1,y_1,z_1) and that of point B' be (x_2,y_2,z_2). Then the equation of line AA' is

$$x/x_1 = y/y_1 = z/z_1 \qquad (1)$$

the equation of line BB' is

$$x/x_2 = y/y_2 = z/z_2 \qquad (2)$$

From Figure 3.7, it is apparent that the equation of the plane of light due to projector P_1 is

$$y = m \text{ and}$$

that due to projector P_2 is

$$y = n .$$

Therefore, by substituting $y = m$ in (1) we obtain

$$x = (x_1/y_1)m ,$$

$$y = m , \text{and}$$

$$z = (z_1/y_1)m .$$

By substituting $y = n$ in (2) we obtain

$$x = (x_2/y_2)n\,,$$

$$y = n\,, \text{ and}$$

$$z = (z_2/y_2)n\,.$$

Thus the coordinates of point A are

$$\{(y_1/x_1)m,\ m,\ (y_1/z_1)m\}$$

and of point B are

$$\{(y_2/x_2)n,\ n,\ (y_2/z_2)n\}$$

In terms of the coordinates of the points A and B, the equation of the axis of the cylinder is

$$(x - x_A)/(x_A - x_B) = (y - y_A)/(y_A - y_B) = (z - z_A)/(z_A - z_B)\,,$$

where (x_A,y_A,z_A) and (x_A,x_B,x_C) are the space coordinates of points A and B respectively. As illustrated in Figure 3.8, M lies exactly in the center between points A and B. We can now determine the amount and direction through which the cylinder is to be rotated.

Consider an axis G through point M which is perpendicular to the plane defined by the cylinder axis and a line passing through M and parallel to the y-axis. The cylinder is to be rotated about G. The direction numbers [17] of axis F are the same as the direction numbers of the normal to the above mentioned plane. The direction numbers of any line parallel to the y-axis are (0,1,0). Let the direction numbers of the axis of the cylinder L be (d_x,d_y,d_z) where

$$d_x = x_A - x_B,$$

$$d_y = y_A - y_B \text{ and}$$

$$d_z = z_A - z_B\,.$$

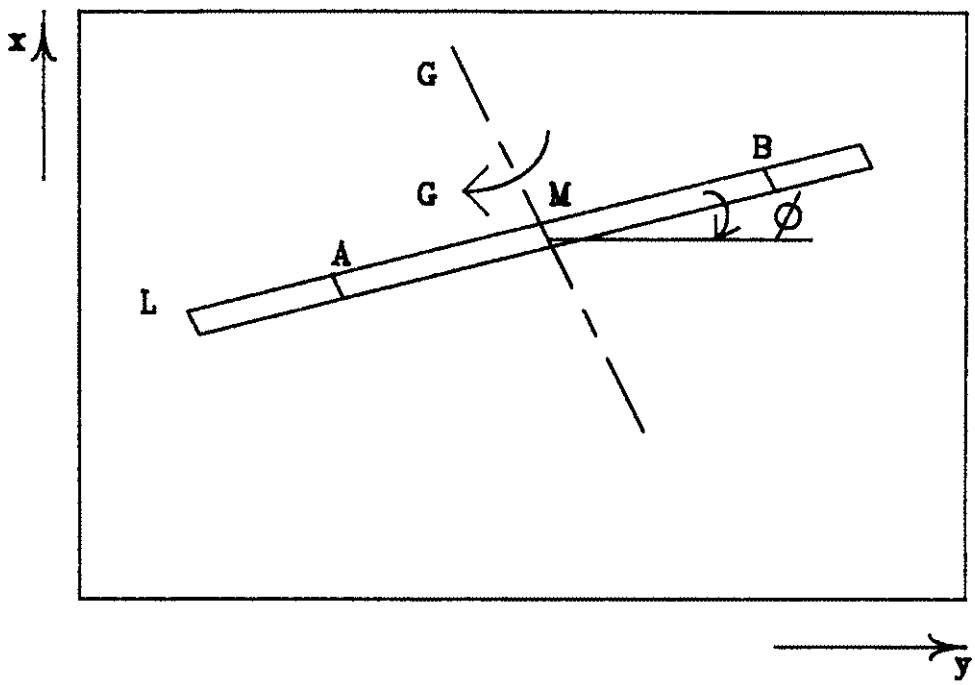


Figure 3.8. G—the axis of rotation.

Let the direction numbers of G be (n_x,n_y,n_z). Thus

$$n_x d_x + n_y d_y + n_z d_z = 0 \text{ and}$$

$$n_x(0) + n_y(1) + n_z(0) = 0,$$

which gives

$$d_x = 0,$$

$$d_y = 1 \text{ and}$$

$$d_z = 0.$$

Therefore, the direction numbers of axis G are $(-d_z, 0, d_x)$. This demonstrates that G always lies in a plane parallel to the XZ-plane. The equations of the axis of G are

$$n_x = -d_z,$$

$$n_y = 0 \text{ and}$$

$$n_z = d_x.$$

Consider the direction numbers $(d_z, 0, -d_x)$. These will also give the same equation for axis G, but will signify the opposite direction. Maintaining consistency in the calculation of d_x, d_y, and d_x gives the proper direction for the axis of cylinder L and axis G. The direction in which L is to be rotated is determined by the Left-Hand Rule. Let the amount of rotation in this direction be ø. From the geometry of Figure 3.8, ø is given by

$$ø = \tan^{-1} \{[(x_B-x_M)^2 + (z_B-z_M)^2]^{½}/(y_B-y_M)\}$$

Performing similar calculations for the second cylinder will give the amount and the direction through which it is to be rotated to render it horizontal.

In Figure 3.9, both the cylinders L and K are horizontal but are not axially aligned. This is the general case. Let C and D be the corresponding points for cylinder K due to the projected light stripes, as A and B are to cylinder L. Now the equations of the axis of L are

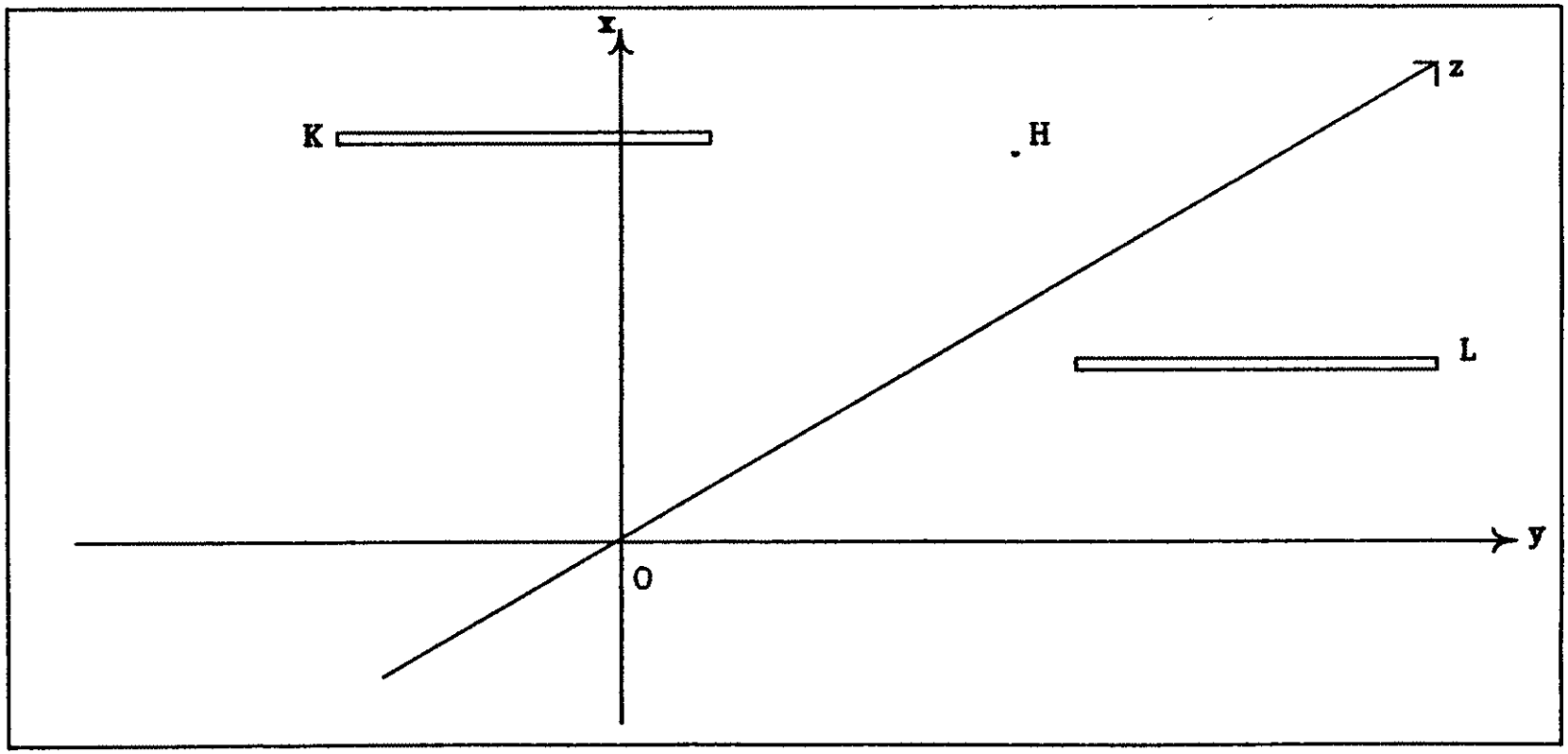


Figure 3.9. H—the final alignment point.

$$x - x_A = 0 \text{ and}$$

$$z - z_A = 0 .$$

That of the axis of K are

$$x - x_C = 0 \text{ and}$$

$$z - z_C = 0 .$$

We need consider only one point on each cylinder; for example, point A on cylinder L and point C on cylinder K. These two cylinders can be aligned at any position; that is, at any desired x and z coordinate values.

Such a point, H, is shown in Figure 3.9. Point H is in the XOZ-plane with coordinates $(x_H,0,z_H)$ such that its x and z coordinates correspond to the x and z coordinate values of the desired position of the cylinders. Then the direction in which L is translated is along the vector

$$(x_H - x_A, 0, z_H - z_A) \text{ and}$$

the direction in which K is translated is along the vector

$$(x_H - x_C, 0, z_H - z_C) .$$

L is translated by an amount

$$l = [(x_H - x_A)^2 + (z_H - z_A)^2]^{1/2},$$

where l is the Euclidean distance between point H and point A. K is moved by an amount

$$k = [(x_H - x_C)^2 + (z_H - z_C)^2]^{1/2},$$

where k is the Euclidean distance between point H and point C. After moving to this position, the equation of the axis of L is

$$x - x_A - (x_H - x_A) = 0 \text{ and}$$

$$z - z_A - (z_H - z_A) = 0.$$

That is,

$$x - x_H = 0 \text{ and}$$

$$z - z_H = 0.$$

Similarly, the equation of K is

$$x - x_C - (x_H - x_C) = 0 \text{ and}$$

$$z - z_C - (z_H - z_C) = 0.$$

That is,

$$x - x_H = 0 \text{ and}$$

$$z - z_H = 0.$$

Here we see that the axes of cylinder L and K are the same, that is, their axes are colinear.

In this chapter two possible methods for aligning cylindrical beams were discussed. The first method does not require that the positions of the cameras and projectors be known. The algorithm works towards achieving an image, as seen by the camera, that corresponds to alignment. In the second method, the positions of the cylinders are determined with respect to the camera and projector positions. The cylinder displacements are then computed to achieve alignment. Even though both methods would render the two cylinders aligned, the second method has obvious advantages over the first. Since the position of the cylinders can be determined in the second method, they can be moved to any other position relative to the camera and projectors if necessary. This feature would be impossible with the first method. In the next chapter we discuss the practical problems that were encountered in analyzing the image, especially that the cylinder has a finite diameter.

CHAPTER IV

PRACTICAL IMPLEMENTATION

From the discussion in Chapter III it is evident that two points on the axis of the cylinder are required to determine its position. In this chapter, practical problems that arise in determining the points on the axis of the cylinder from the patterns cast by the light stripe projectors are discussed.

EMPIRICAL EVIDENCE

Ideally, the stripe of light emanates along a plane of zero thickness. However, under practical situations a line having finite width is projected. Since the cylinder has a finite diameter, the image of the light stripe as projected on the cylinder may appear curved or as a rectangular area in the camera, depending upon the relative positions of the cylinder and the projectors with respect to the camera.

Figure 4.1a shows the image of a single stripe as a straight segment. The center point C of this stripe can be found easily since it lies midway along the length and width. Now consider the case when the image stripe is a curved segment as illustrated in Figure 4.1b. Again, point C can be found in the same manner as explained above. Alternatively, the point C for both cases may be found by calculating the centroid of the segment. For the first case the centerpoint and centroid are identical points, but for the second case it is displaced slightly to the side depending upon the amount of curvature. The

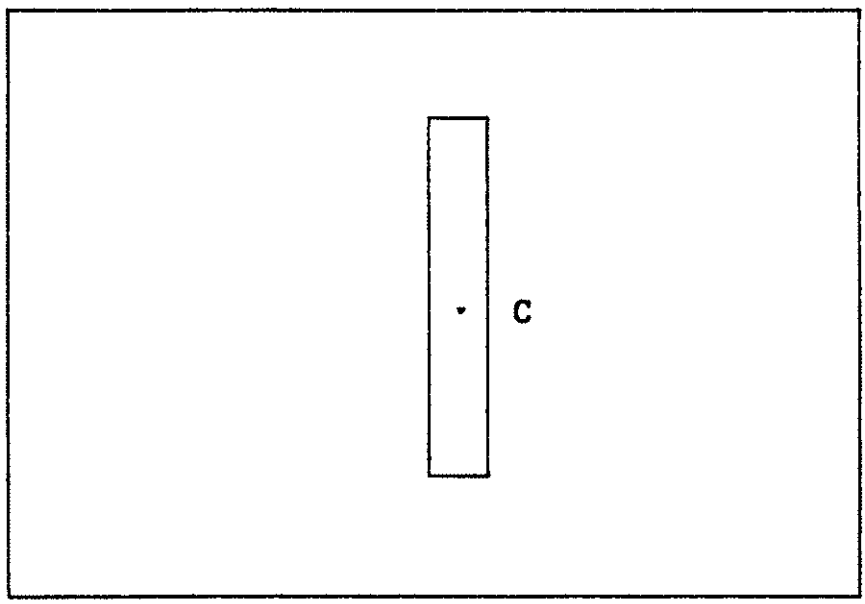


Figure 4.1a. Image stripe on a planar surface

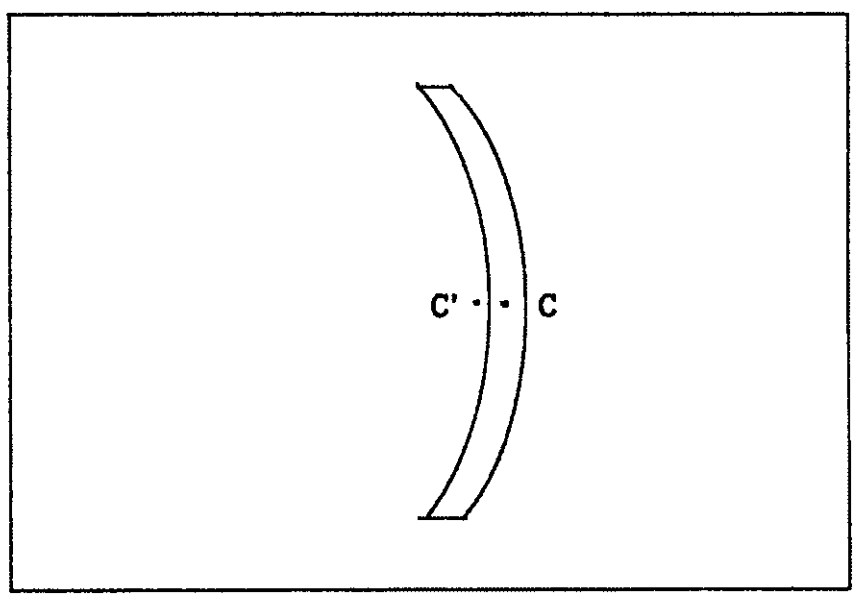


Figure 4.1b. Image stripe on cylinder as seen by the camera.

centroid in the second case is represented by the point C'. The position of the point C or C', which lies on the focal plane, can be determined in space since the camera position is known, and its corresponding point on the cylinder found as explained in the previous theory. Here it is evident that this point would not correspond to a point on the axis of the cylinder. Point C would correspond to a point on the surface of the cylinder; whereas, point C', the centroid, would correspond to a point within the cylinder but not on the axis.

Figure 4.2 represents a stripe of light projected on the cylinder L. The image stripe on L has corners designated C_1, C_2, C_3, and C_4. If L is replaced by a "half-cylinder" with a semi-circular cross-section, as shown in Figure 4.2b, we obtain the image with corresponding corner points D_1, D_2, D_3, and D_4. The planar area of this half-cylinder is perpendicular to the YZ-plane. The only difference between the image in Figure 4.2a and 4.2b is that in the second case the closed curve formed with corner points D_1, D_2, D_3, and D_4 is a quadrilateral. From the geometry of the quadrilateral, the central point C can be computed. It will be shown how closely point C in the image plane corresponds to the point on the axis of the cylinder, since it is theoretically impossible to determine a point on the axis of the cylinder from the four corner points D_1, D_2, D_3, and D_4. In the theoretical development in Chapter 3, it was assumed that the plane of light has zero thickness. This is practically not possible. Instead, a volume of light is emanated from the light stripe projector. Points D_1, D_4 and D_2, D_3 are formed due to the intersection of the cylinder with the two boundary planes, respectively. The ideal plane of light can be represented by a plane lying exactly between the two boundary planes, and it vertically bisects the projected stripe of light. It is possible to determine a point on the axis of the cylinder only if we know the

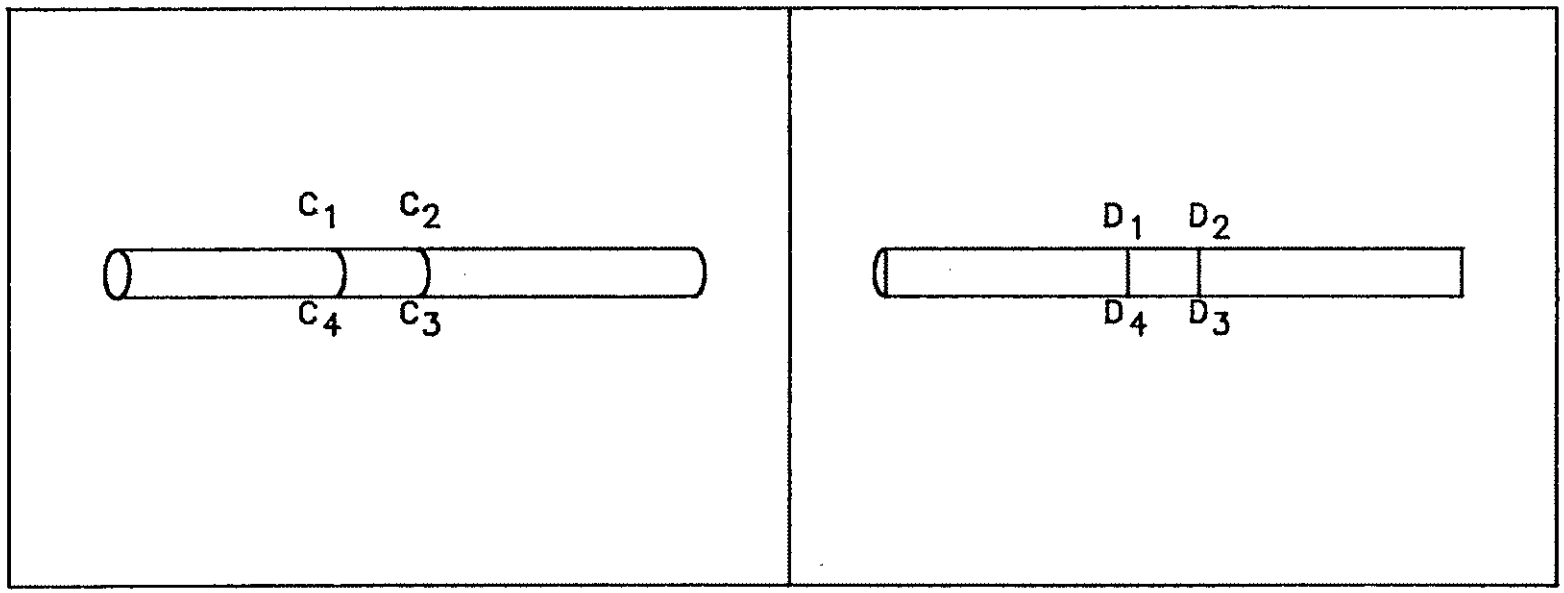


Figure 4.2. Image stripe comparison on curved and planar surface.

equations of the two boundary planes. Here it is assumed that the equations of the boundary planes are unknown but that of the ideal plane is known.

APPROXIMATION ERROR

Since it is theoretically impossible to determine a point on the axis of the cylinder unless the boundary planes are known, the amount of error incurred in determining a point on the axis of the cylinder for the worst case configuration is found. This is shown to be within the resolution of the image processing unit and therefore is acceptable.

In Figure 4.3, points A,D and B,C are the projection of points on the image plane due to the intersection of the two boundary planes with the cylinder; while, points E,F are due to the projection of points on the image plane due to the intersection of the ideal plane with the cylinder. Let points S,T on the cylinder correspond to points A,D on the image plane as shown in Figure 4.4. V is the midpoint of segment ST. Let the projected point on the image plane due to V be V'. The error in determining the position of V' on the image plane is found. In Figure 4.4, segment ST corresponds to the top portion of the stripe and coincides with the axis of the cylinder. The cylinder axis is inclined at 30 degrees to the y-axis. This is the worst case. In the actual experiment it was not possible to incline the cylinder by more than 20 degrees to the y-axis because of the experiment's configuration constraints. Segment ST is considered to be formed due to projector P_2, instead of P_1, since the farther the projector is from the camera the larger the error in determining the position V'. Projector P_2 is at a distance of 637mm from the optical center of the camera lens; point V is at a distance of 937mm; and the image plane of the camera is at a distance of 25.4mm in the negative z-direction.

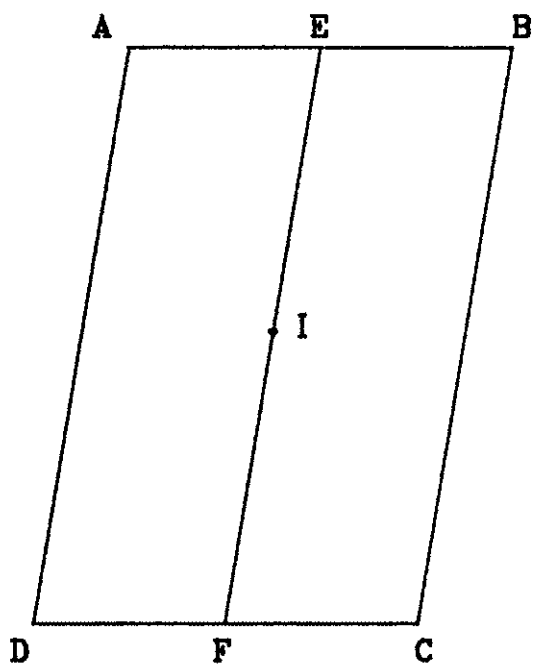


Figure 4.3. Representation of the corners of the image stripe.

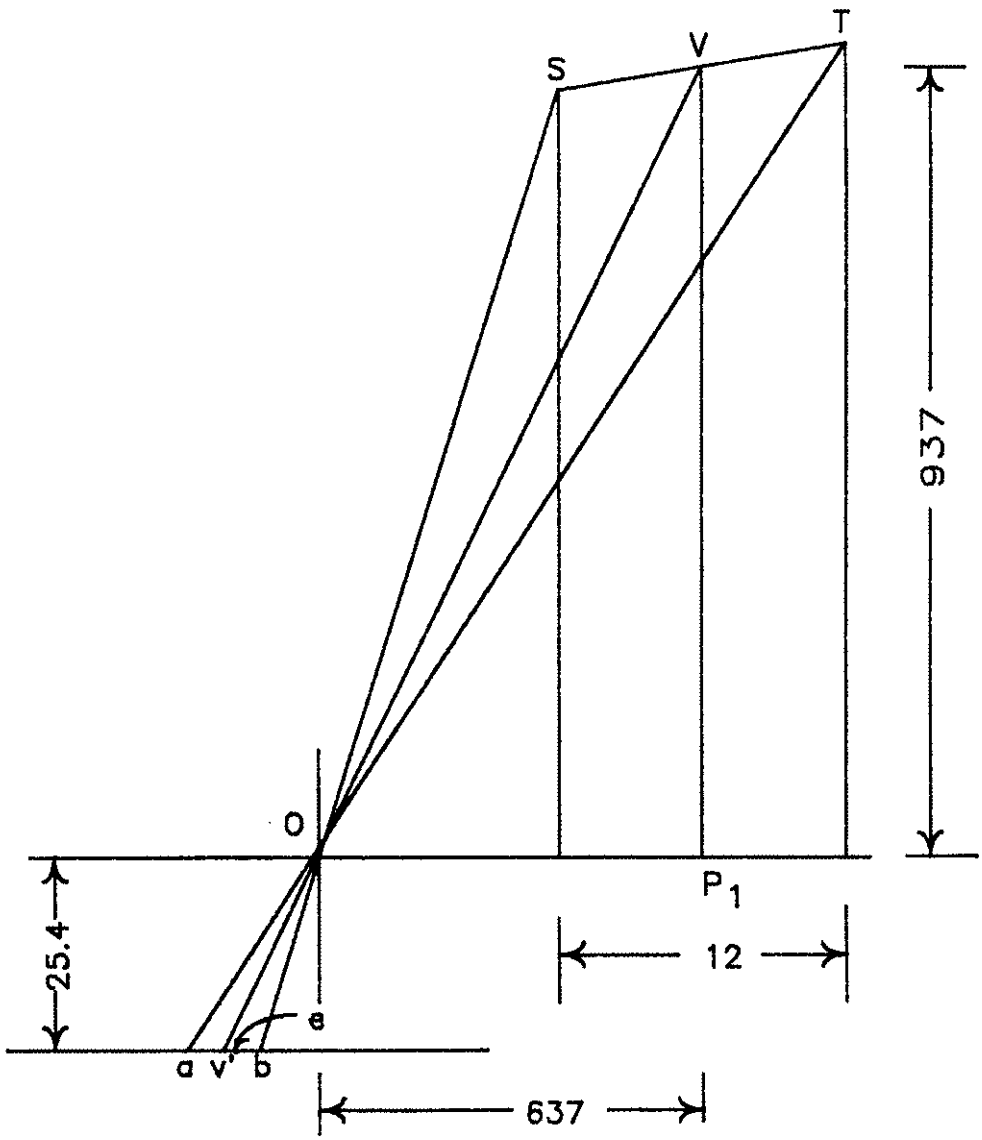


Figure 4.4. Projection of corner points.

The points A, B and V' lie at distances

$$a = 17.1765\text{mm},$$

$$b = 17.3746\text{mm and}$$

$$v' = 17.2750\text{mm},$$

respectively, from the XZ-plane.

Any point on segment ST maps on to the segment AB in the image plane. E is the midpoint of segment AB and is given by

$$e = (a + b)/2 .$$

Thus, e = 17.2755mm. This gives an error of 0.0005mm, which is less than the resolution of the image processing unit. The resolution of the image processing unit is 0.0275mm. Point E, when projected on the cylinder axis deviates from the point V by an amount which is negligible for practical purposes.

Another error that gets introduced is that the rays going into the camera enter through a single point. The tangent rays that define the points C_1, C_2, C_3, and C_4 in Figure 4.2a are unable to define points such that they lie in a plane passing through the axis of the cylinder. The distance between this plane and the axis of the cylinder as calculated for a cylinder of 50mm diameter is 0.66mm. Consequently, it has not been considered in this experiment since it does not contribute to the information regarding the amount the cylindrical beam is to be rotated to render it horizontal.

Referring to Figure 4.3, which represents the quadrilateral due to points D_1, D_2, D_3, and D_4 in Figure 4.2b, the central point I can be determined as follows. First points E and F are determined which are the midpoints of segments AB and CD, respectively. Then I, the midpoint of segment EF, is found. Therefore, by finding these corner points the parallelogram can be constructed and the center point which now corresponds closely to a point on the axis of the cylinder can be

found. Broad stripes of light were considered here so that the corners could be relatively well defined.

PRACTICAL CONSIDERATIONS

The approach chosen here for the alignment process is to first manipulate the cylindrical beams so that they are horizontal. Manipulation of both the cylindrical beams is performed through an identical sequence of steps. The experimental configuration and design was carried out for one cylinder only for the sake of simplicity. The method of obtaining information about the initial position of the cylinder was described. With this information, the amount the cylinder is to be rotated to render it horizontal can be computed.

From the preceding argument it is evident that this technique primarily involves finding the points of intersection of the lines formed due to image stripes on the camera focal plane passing through the lens center and the plane which projects the light stripes. These quantities can be determined with the projectors and camera in any other orientation. The optical center of the camera and the projectors, must always be colinear, with the exception that the camera lens center should not lie in any one of the planes of the light stripe projectors.

In the actual experimental set-up the optical axis of the camera, instead of being coincident with the z-axis, was inclined towards the stripe patterns on the cylinder by an angle μ. This does not change the x-coordinates of the image points formed on the camera focal plane. The new image y and z coordinates are given by

$$y = (k^2 + y_p^2)^{1/2}\sin[\tan^{-1}(k/y_p) + \mu] \text{ and}$$

$$z = (k^2 + y_p^2)^{1/2}\cos[\tan^{-1}(k/y_p) - \mu],$$

where μ is as given above, k is the distance of the focal plane from the lens center and y_p is the distance along the row in the focal plane from the row center.

The image obtained from the camera was captured by an image processor having a resolution of 320 x 240 pixels. The image was stored in a file consisting of 240 lines corresponding to the 240 scan lines of the camera. Each line consisted of 160 words each of which contained coded information representing the intensity of two horizontally adjacent pixels which were digitized into sixteen gray levels. Two neighboring pixels which were binary coded using eight bits were concatenated to form a single sixteen bit word. The even-column pixels, starting with zero, were assigned the least significant eight bits of the sixteen bit word, while the odd-column pixels were assigned the most significant eight bits.

Reflective properties of the cylinder had to be modified by painting its surface with white matte paint. Such a painting reduces the reflectivity of the surface to a great extent while increasing the scattering effect. This resulted in a uniform brightness over the entire segment. The image stored in the computer is at a resolution of sixteen levels of gray (0 - 15) as is seen in Plate 1. Since we are interested only in the two image stripes, which are the brightest parts in the picture with an intensity level ranging from thirteen to fifteen, a threshold value of twelve was chosen. Thus, all brightness values below twelve were converted to zero and values at twelve or above were converted to fifteen. The thresholded image is shown in Plate 2.

This binary image file was then processed by an edge finding algorithm. This program looks for the total number of connected white areas present in the image file which is two in our example. It gives the total number of boundary points and their pixel coordinate values. Because of the nature of the edge finding algorithm, some pixel values listed as edge pixels get repeated. Also

certain stray or unnecessary pixel values get listed as border pixels. The repeating pixels and the stray pixels are removed by passing this edge pixel list file through another algorithm which filters out unwanted pixel coordinates. The difference in the two edge images before filtering and after filtering can be seen by comparing Plates 3 and 4. This filter is essential so that the corner finding algorithms can work effectively.

There are different ways to find the corners in a given closed curve. One way is to find the rate of change in the slope of a tangent as we move along the curve. Another way is to find the curvature at each point and plot the curvature against the distance traveled along the curve. We obtain a graph as shown in Figure 4.5.1. This graph shows four peaks, one for each corner. Assuming there are four corners in the closed curve, we demonstrate two methods for finding the corners of the closed curve.

In the first method we consider a pixel on the curve. Let this pixel be called the vertex pixel. Two more pixels on either side of this pixel at a distance, e.g., ten pixels, are chosen. Both of these pixels are connected to the vertex pixel by two segments. The angle of the segments at the vertex pixel within the closed curve is then computed; therefore this process is repeated for all the pixels. If we plot a graph of the angle at each pixel against the pixel number, we obtain a graph similar to the one shown in Figure 4.5.1, except that it is inverted. See Figure 4.5.2.

In the second method, instead of finding the angle within the curve at each point, we find the perpendicular distance from the vertex point and the line joining the two pixels on either side of the vertex pixel, as mentioned before. A graph showing this perpendicular distance for each pixel against the pixel number

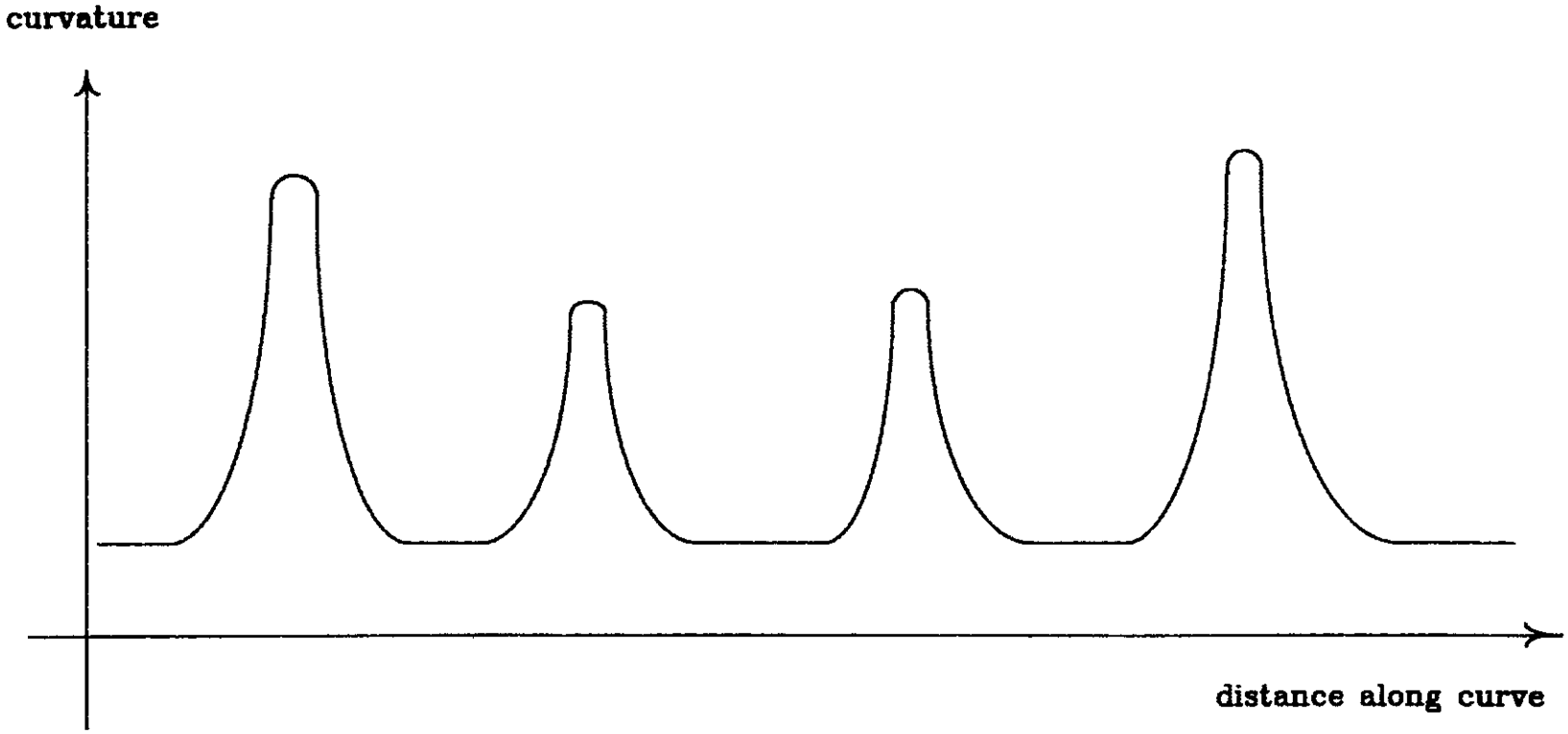


Figure 4.5.1. Graph comparison for corner detection.

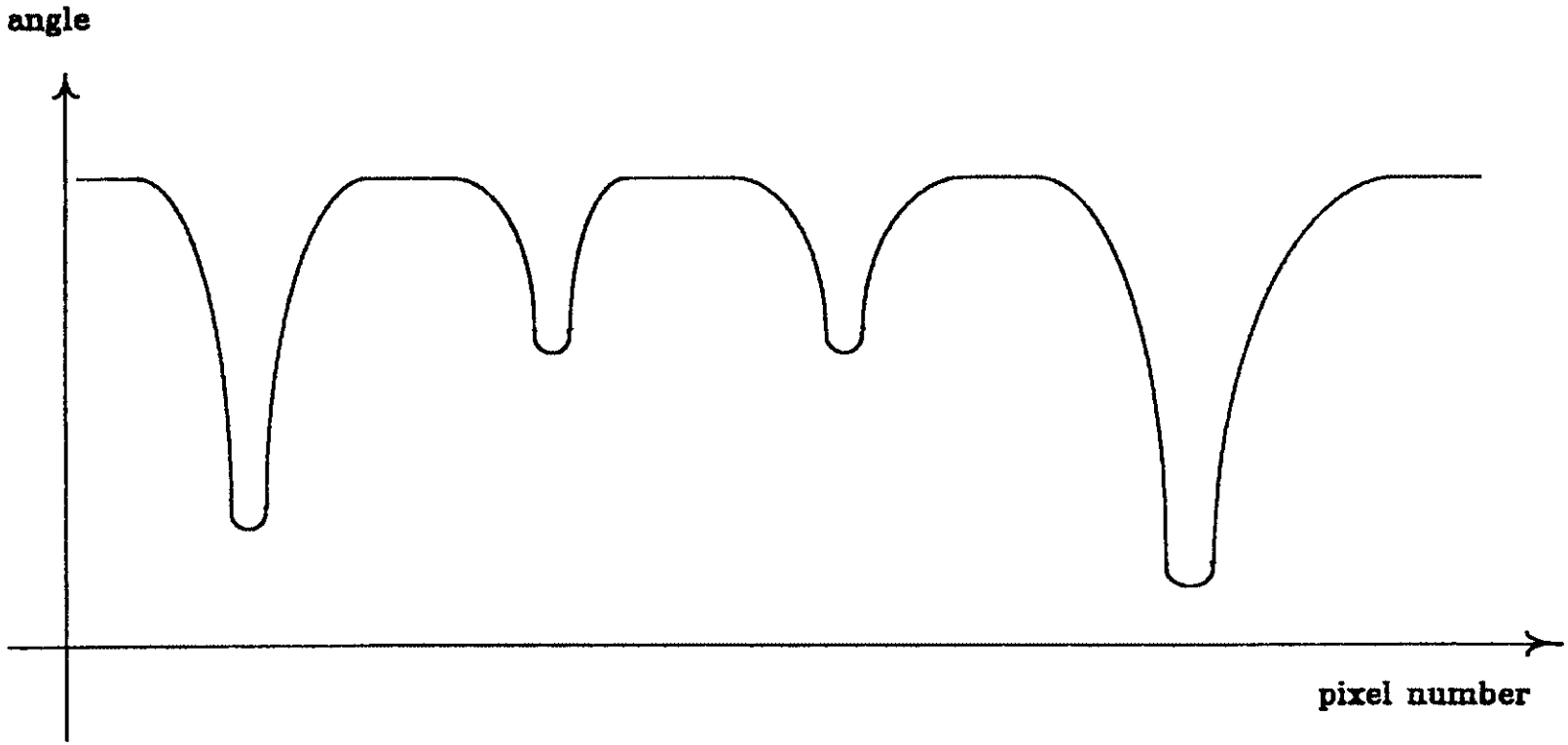


Figure 4.5.2. Graph comparison for corner detection.

is plotted. This graph is similar to the first graph of the curvature, as shown in Figure 4.5.3.

The second method is preferred since the algorithm is more simple than the algorithm used in the first method. Both the methods gave identical results on test samples. The peaks of the resulting graph represent the corners of the closed curve. The procedure used to detect these corners is as follows.

The highest value of the curve on the graph is chosen. This point represents any one of the corners. About twenty points on either side of this point are taken and their value is made zero: see Figure 4.5.3. We now have three peaks. The above process is repeated so that the next peak represents another corner and the number of peaks is reduced to two. This process is repeated two more times to obtain the two remaining corners. The corners are obtained in random order; that is, if we move along the curve the corners are not necessarily in a sequential order. This ordering is important in order to restore the correct relative sequence of the points. The two points which are closest to one another are found and a midpoint between them is then computed. The midpoint between the other two points is also computed. The midpoint between the two previously found midpoints is then computed. This is the point which is the intersection of the axis of the cylinder with the plane of light. In order to represent this point on a monitor, it is rounded-off to the nearest pixel coordinates. This point and the four corner points are shown in Plate 5.

EXPERIMENTAL SET-UP

As explained earlier, the alignment task consists of two steps. First, the two cylinders are made horizontal, and then are axially aligned. Since the experiment was carried out on only one cylinder, the process of making it horizontal was

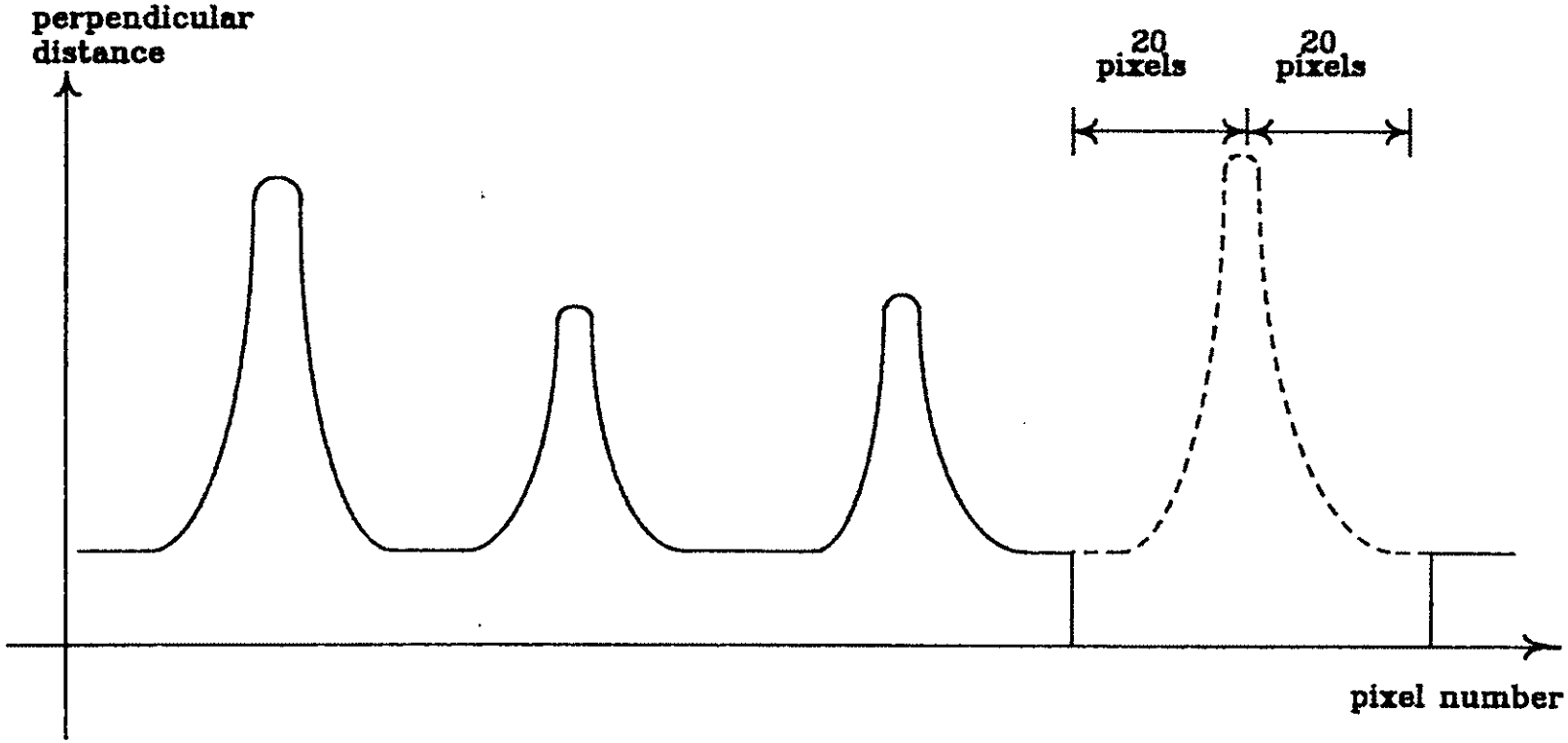


Figure 4.5.3. Graph comparison for corner detection.

tested for different orientations in space. The cylinder was clamped onto a device which enabled it to be rotated in two degrees of freedom. If the cylinder is clamped and calibrated to be in a horizontal position initially, then one degree of freedom has a vertical axis of rotation, while the other has a horizontal axis of rotation, but perpendicular to the cylinder axis.

The amount of movement about these two axes was initially set to any desired angle because of a calibrated scale present on the mounting device. Two slide projectors which served as the light-stripe projectors were at roughly the same height of the cylinder. The slides used in these projectors were blackened slides with a thin vertical slit cut along its center. The horizontal level at which the projectors were placed was not critical since it tended to displace the image-stripe on the cylinder in the vertical direction. This did not affect the position of the image stripe cast on the cylinder, but care had to be taken that the cylinder did not go out of the range of the stripes of light. It was also necessary that the stripes of light casted on the cylinder were vertical.

The camera was mounted on a tripod. Its optical axis was made horizontal to simplify the calculation process. In a general sense it could be oriented in any direction with the requirement that the optical center should lie on the y-axis and it should be able to capture the image of the cylinder. For simplicity again, the camera orientation about the optical axis was maintained such that the rows of pixels are horizontal and the columns of pixels are vertical. A spirit level was used to calibrate orientation of the cylinder, light stripe projector and camera. The cylinder was calibrated in a horizontal position, the light stripe projectors were calibrated to project a vertical stripe of light, and the camera was calibrated such that its row pixels were horizontal and column pixels were vertical. In the theoretical development it is assumed that the camera axis is coincident with the z-

axis. In the actual set-up this was not possible. Because of the close proximity of the components used in the experiment, the camera had to be turned about a vertical axis passing through its optical center towards the cylinder in order to capture its image. The optical center of the camera was the origin of the space coordinate system. The planes of the stripes of light were parallel to the xz-plane. With the initial position of the cylinder as horizontal, its axis was parallel to the y-axis.

Two monitors were used in the experiment; one was for viewing what was seen by the camera, and the other for the processed images. The image processor that was used has the capability to capture an image and display it on the monitor either directly or thresholded. The processor was not able to store a thresholded image which is necessary for further processing. The original captured image data is transferred to a computer on which all further processing work is executed because of its higher speed and storage capacity. The position of the cylinder was computed from the image data after a sequence of processing steps for various cylinder orientations. The results of these tests are discussed in the next section.

RESULTS

The positions of two points on the axis of the cylinder were computed from the equations of the planes of light and the equations of the lines passing through the optical center of the lens and the center of the image stripes on the focal plane. Let the angle through which the cylinder is rotated about a horizontal axis perpendicular to the cylinder be ALPHA and that about the vertical axis be BETA. Table 1 shows the set ALPHA and BETA angles and the corresponding computed angles. Appendix A contains programs that process the image and compute the angles ALPHA and BETA.

ALPHA (degrees)			BETA (degrees)		
Set	Computed	Error	Set	Computed	Error
0	1.387	1.387	0	0.313	0.313
0	0.753	0.753	-20	-20.287	0.287
0	1.420	1.420	15	16.357	1.357
-10	-9.180	0.82	0	0.757	0.757
5	6.6	1.6	0	0.312	0.312
5	6.44	1.44	-8	-7.357	0.643
-10	-9.120	0.88	-10	-9.423	0.577
-12	-10.6	1.4	15	14.86	0.140
5	6.87	1.87	17	17.725	0.725

Table 1. Results

CHAPTER V

CONCLUSIONS AND FUTURE RESEARCH

The type of structured light pattern to be used is determined by the kind of task in which it is employed. In this research, two planes of light were employed in a method to align two cylindrical beams or cylinders. The orientation of a single cylinder in three-dimensional space was determined. It is known a priori that the object was a cylinder. Hence, the positions of any two points on its axis were determined in three-dimensional space. From this information the cylinder orientation was computed.

REMARKS

The error observed in the measurement of the angles is due mainly to the finite resolution of the camera and image processing unit which is 320 x 240 pixels. From the geometry of the configuration it can be shown that the cylinder has to be rotated by approximately 2.8 degrees about a vertical axis, from a position parallel to the y-axis before a change in the coordinate of the excited pixel is detected. The results presented in Table 1 are well within the error limits. The largest error is 1.87 degrees, produced in measuring the angle ALPHA. There are two possible reasons for this: one, an image stripe was at the corner of the focal plane where the angular resolution is poor compared to that at the center; and two, images at the corner of the focal plane get highly distorted which makes it difficult

for the corner finding algorithm to exactly pinpoint the corner pixel. This finally results in an error in locating the position of the point on the cylinder axis.

Depending upon the kind of structured light pattern used, one can obtain different kinds of information about the properties of an object under study. Curvature measurement using grid coding has been studied by Will and Pennington [5] employing one and two dimensional gratings. In the single laser tracker [7] employed by Ishii and Nagata to extract feature information of three-dimensional objects, the process of digitization being point-by-point is slow, but has less stringent requirements on the memory capacity of the computing system. Since we are mainly interested in the positional information of cylinders, the above discussed methods, if applied to this task, would generate a significant amount of redundant information. This would involve additional computational requirements and slow the process of aligning the cylinders.

In the processing task of the image stripes we determine the position of a point close to the point on the axis of the cylinder. The "ideal" point on the axis of the cylinder is due to the intersection of the cylinder axis and the plane of light.

One of the most significant steps in this processing task is to find the four corners or vertices of the image stripe. Most of the commonly used methods employ the technique of finding the extremum values of the x and y coordinates, which would normally correspond to the four corners. Since the image stripes are from a curved surface, that is the cylinder, it is curved instead of being rectangular. Taking extremum values of the x and y coordinates would yield erroneous results; hence, a new technique has been employed here which detects the vertices for either a normal rectangular stripe or any closed curve with prominent vertices. The only constraint is that prior information about the number of corners or vertices should be known.

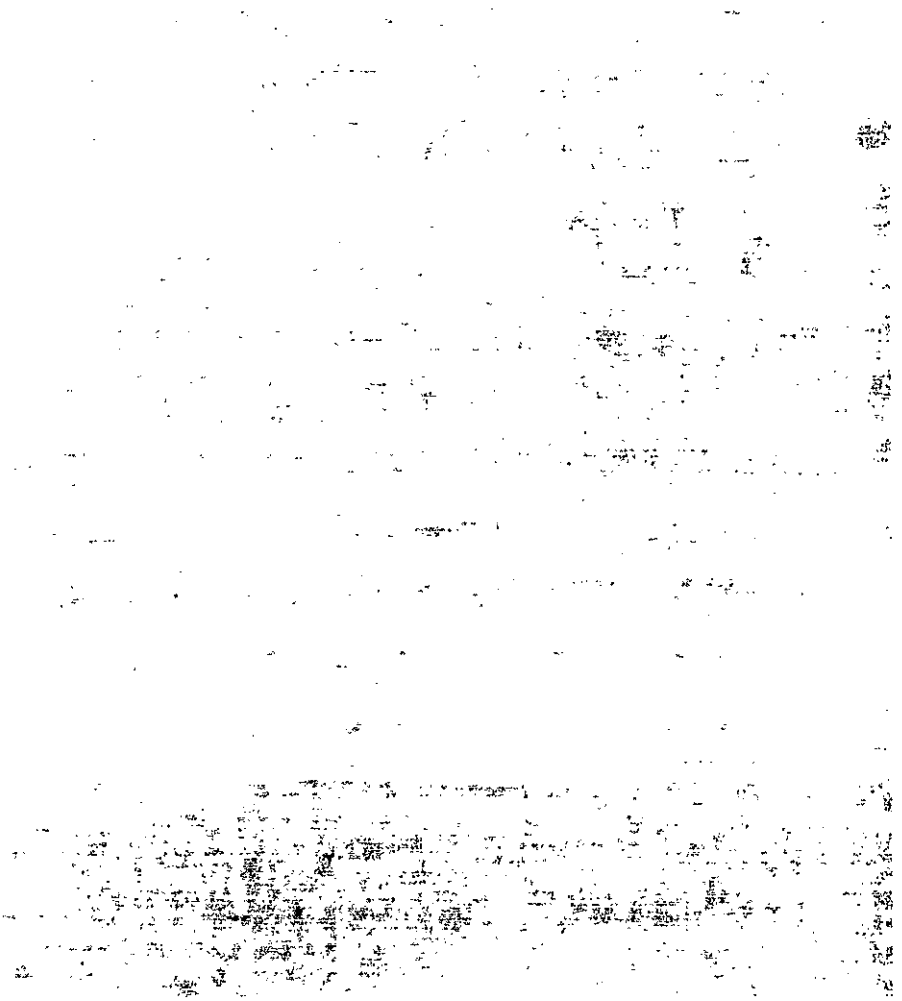
The method of using structured light described in this research requires one image frame to completely determine the position of the cylinders. Using a sufficiently strong source of light for the light stripe projection sytems, this method is less prone to errors that may occur due to stray light. Since the computations involve geometric and trigonometric identities, it is fast in determining the cylinders' positions, compared with other vision techniques where large amounts of image processing take up the bulk of the processing time.

FUTURE RESEARCH

Future work in this area could be done to integrate two robotic arms each carrying a cylinder with the vision system and implement the process of alignment of the cylinders in real time. Some of the current research shows that a single plane of light is used to extract feature and position information by rotating the plane of light in discrete steps so that it casts parallel adjacent stripes on the object. An enhancement to this research could be to use a single plane of light and use only a single line cast on the cylinder to obtain its orientation. Assuming that the cylinder dimensions are known, its generalized equation in three-dimensional form can be determined. Upon finding the position of any three points on the cylinder it is possible to determine these coefficients.

To conclude, the structured light method for beam alignment provides a computationally fast technique for aligning cylindrical beams in space which can be performed in real time. With sufficiently strong planes of light projecting on the cylinders, the system is less prone to errors due to stray light. The error in calculating the orientation of the cylinder is primarily dependent on the resolution of the camera. This thesis demonstrates the use of a structured light method for

determining the position of cylindrical beams. A similar approach can be used for locating or determining positions of objects in an industrial environment.



LIST OF REFERENCES

1. J. Albus, E. Kent, M. Nashman, P. Mansbach, L. Palombo, and M. Schneier, "Six-Dimensional Vision Systems," *Robot Vision*, *SPIE*, Vol. 336, 1982, pp. 142-153.

2. I. Masaki, M. J. Dunne, and H. Toda, "Vision Guided Robot System for Arc Welding," *Robot Vision* by Alan Pugh, Ed., 1983, pp. 179-185.

3. W. F. Clocksin, P. G. Davey, C. G. Morgan, and A.R. Vidler, "Progress in Visual Feedback for Robot Arc Welding of Thin Sheet Steel," *Robot Vision* by Alan Pugh, Ed., 1983, pp. 186-198.

4. B. K. P. Horn, *Robot Vision*, The MIT Press, McGraw Hill Book Company, 1986.

5. P. M. Will and K. S. Pennington, "Grid Coding: A Novel Technique for Image Processing," *Proceedings of the IEEE*, Vol. 60, No. 6, June 1972, pp. 669-680.

6. J. L. Posdamer and M. D. Altschuler, "Surface Measurement by Space-Encoded Projected Beam Systems," *Computer Graphics and Image Processing*, Vol. 18, 1982, pp. 1-17.

7. M. Ishii and T. Nagata, "Feature Extraction of Three-Dimensional Objects and Visual Processing in a Hand-Eye System using Laser Tracker," *Pattern Recognition*, Vol. 8, 1976, pp. 229-237.

8. E. L. Hall, J. B. K. Tio, C. A. McPherson, and J. J. Hwang, "Surface Location in Scene Content Analysis," *Proceedings of IEEE Southeastcon '82*, 1982

9. E. L. Hall, J. B. K. Tio, C. A. McPherson, and F. A. Sadjadi, "Measuring Curved Surfaces for Robot Vision," *Computer*, Dec., 1982, pp. 42-54.

10. R. M. Haralick, "Using Perspective Transformations in Scene Analysis," *Computer Graphics and Image Processing*, Vol. 13, 1980, pp. 191-221.

11. R. M. Haralick, "Determining Camera Parameters from the Perspective Projection of a Rectangle," Virginia Polytechnic Institute and State University, June 1982, pp. 1-14.

12. P. W. Goode, "A Multifunction Recognition Operator for Telerobotic Vision," *Presentation at the AIAA Guidance , Navigation ,and Control Conference*, Williamsburg, Virginia, August 18-20, 1986, pp. 1-11.

13. N. E. Orlando, "Interfacing Intelligent Software to Robotic Peripherals," *Presentation at The First International Conference on Applications of Artificial Intelligence to Engineering Problems*, Southampton University, England, April 15-18, 1986.

14. R. W. Will and N. E. Orlando, "Design for a Goal-Oriented Telerobotic System," *Presentation at the AIAA Guidance , Navigation , and Control Conference*, Williamsburg, Virginia, August 18-28, 1986, pp.1-7.

15. P. Besl and R. Jain, "An Overview of Three-Dimensional Object Recognition," Dept. of EECS, The University of Michigan, Ann Arbor, Michigan, December 1984, pp. 1-85.

16. J. Champaneri and D. Livingston, "A Structured Light Method for Sensing Alignment during Automated Truss Assembly," *Proceedings of IEEE Southeastcon '88*, 1988.

17. L. P. Eisenhart, *Coordinate Geometry*, Ginn and Company, N.Y., 1939.

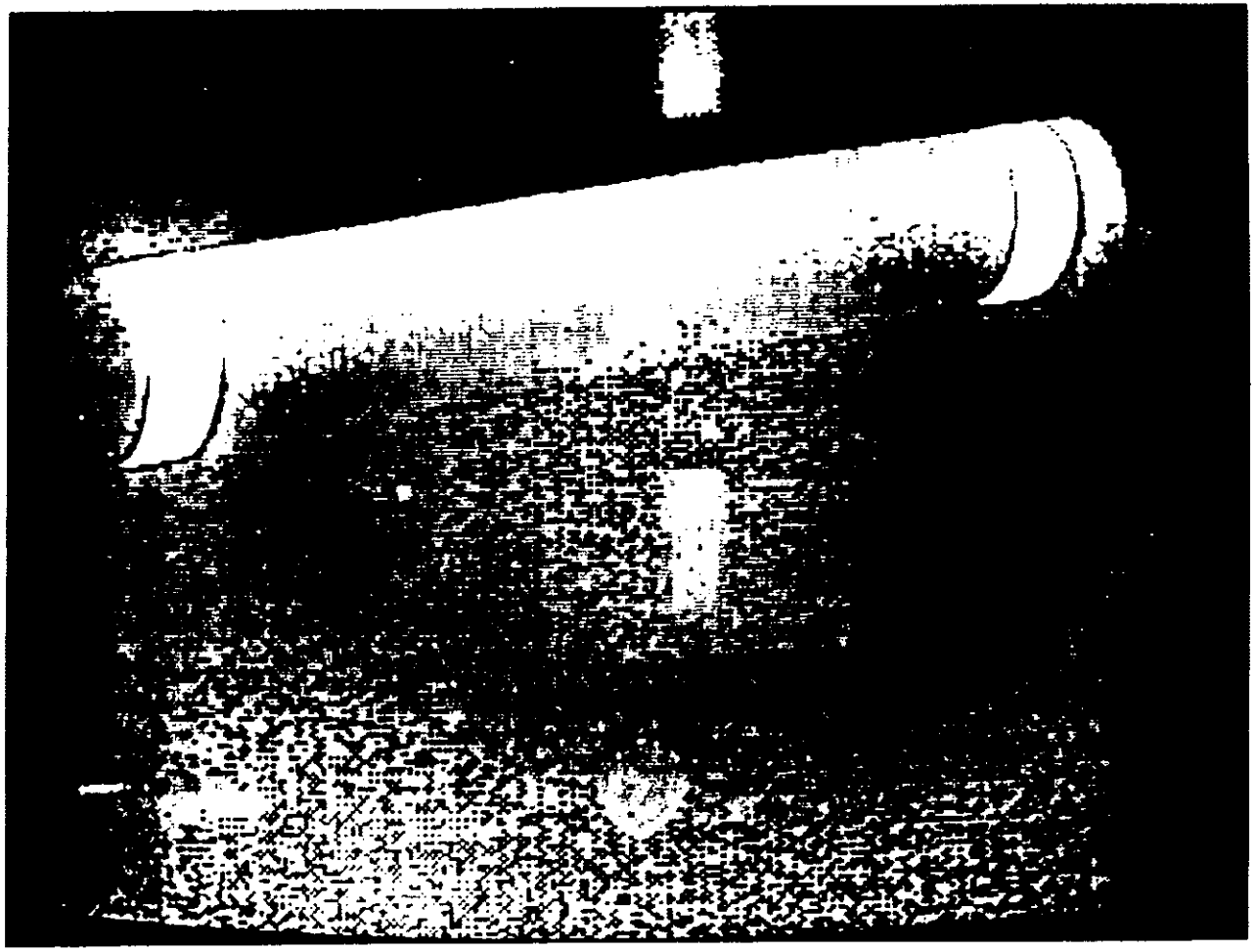


Plate 1. Captured image at sixteen levels of gray

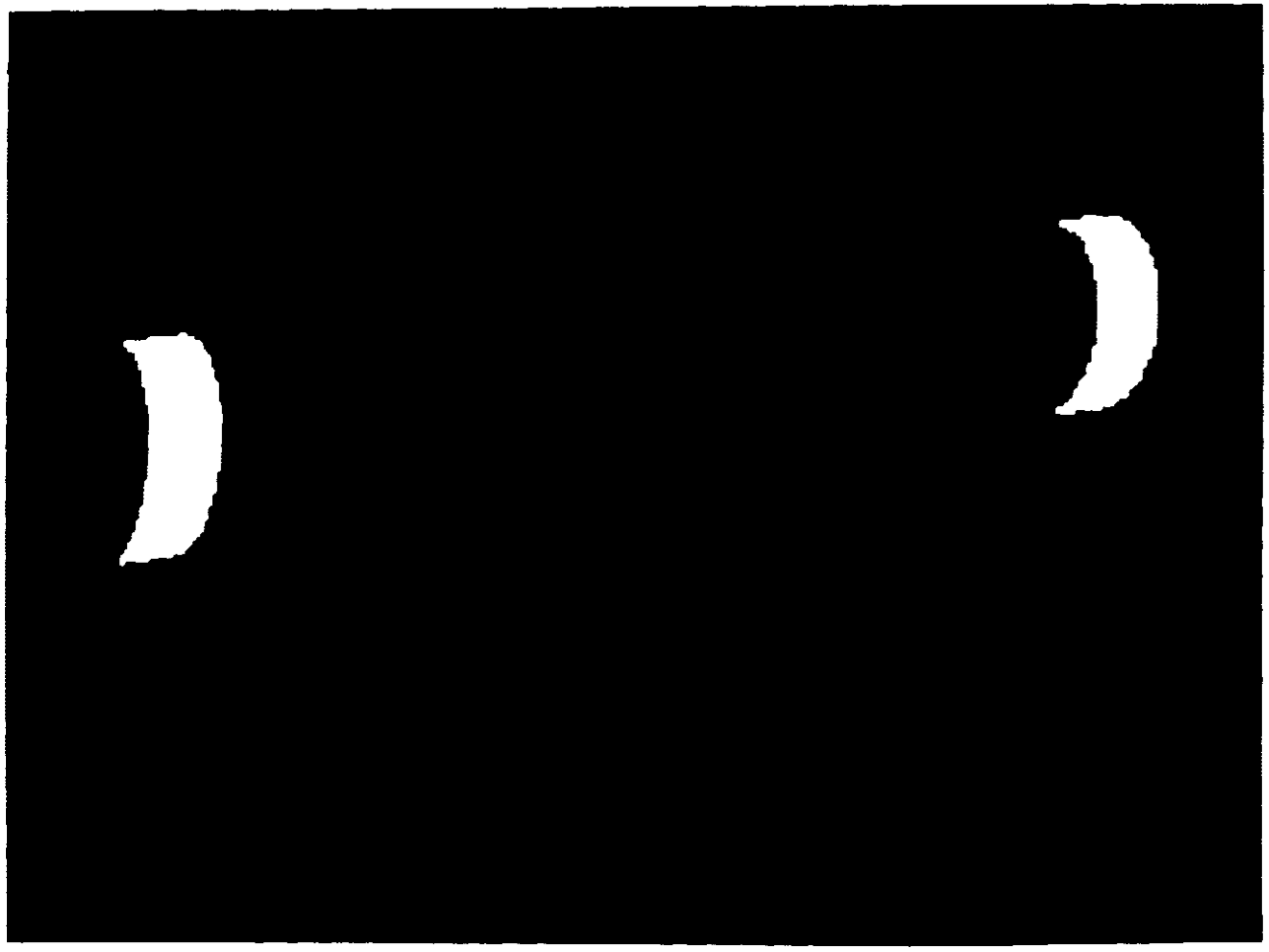


Plate 2. Binary thresholded image.

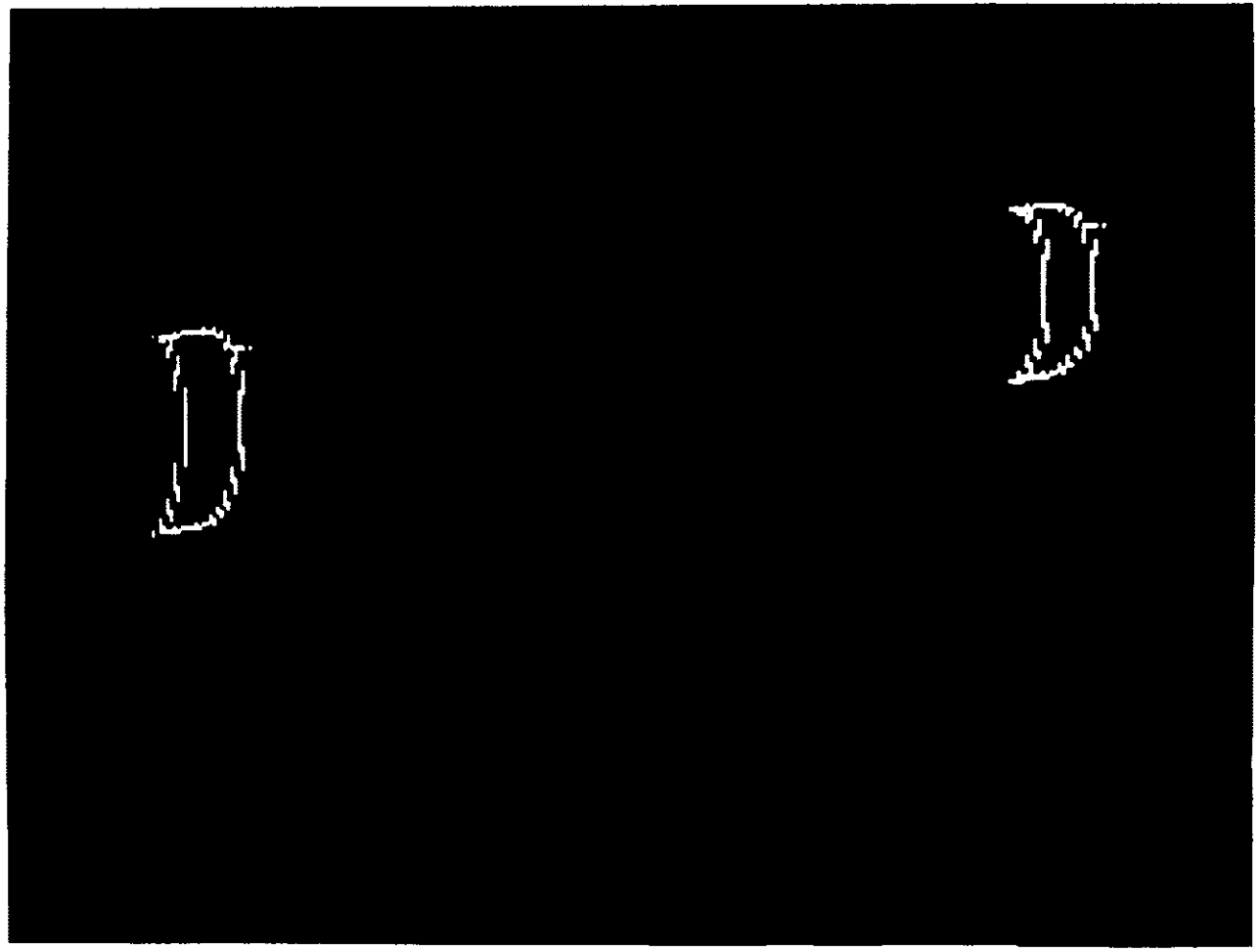


Plate 3. Image after edge detection.

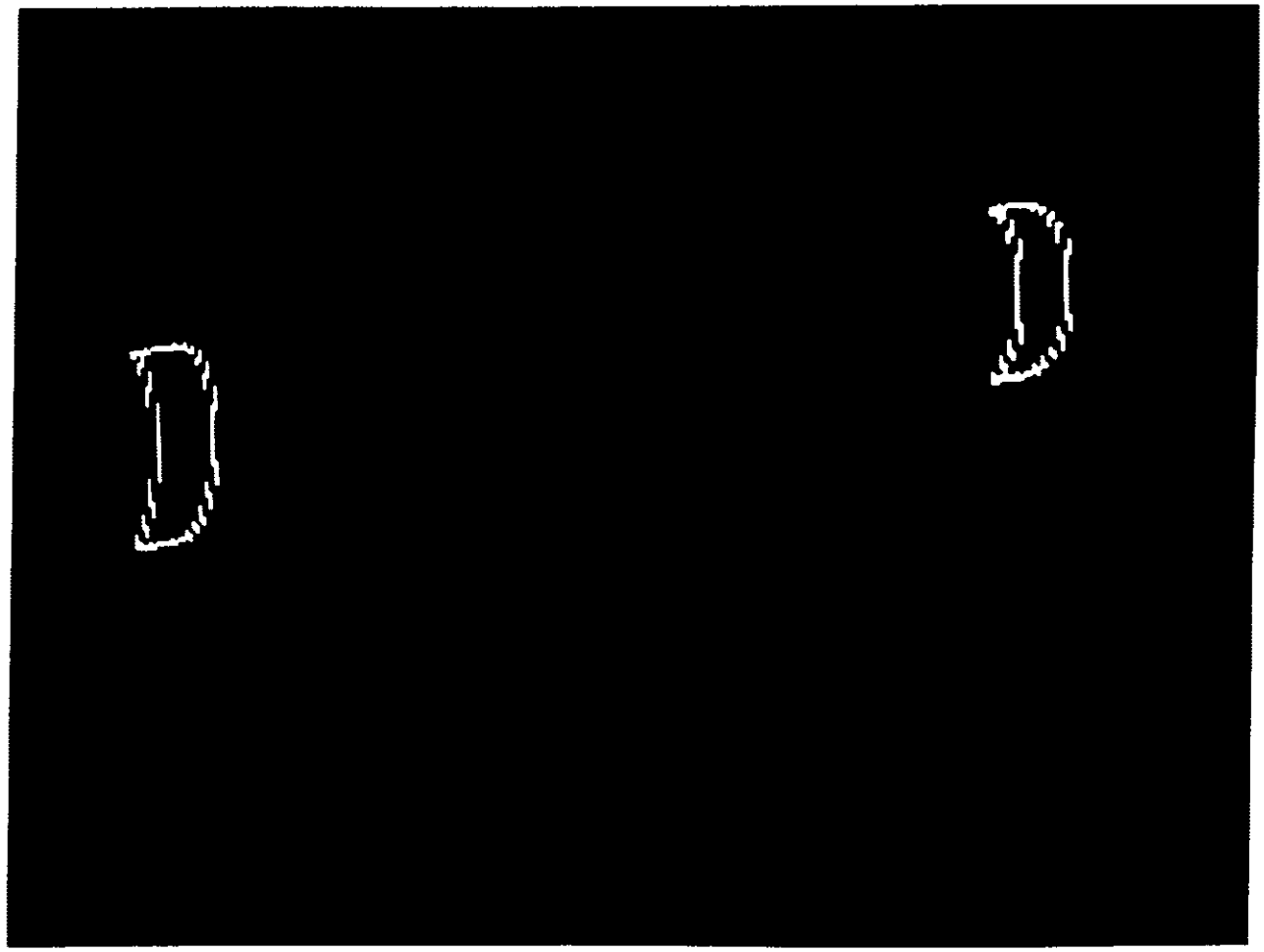


Plate 4. Filtered image.

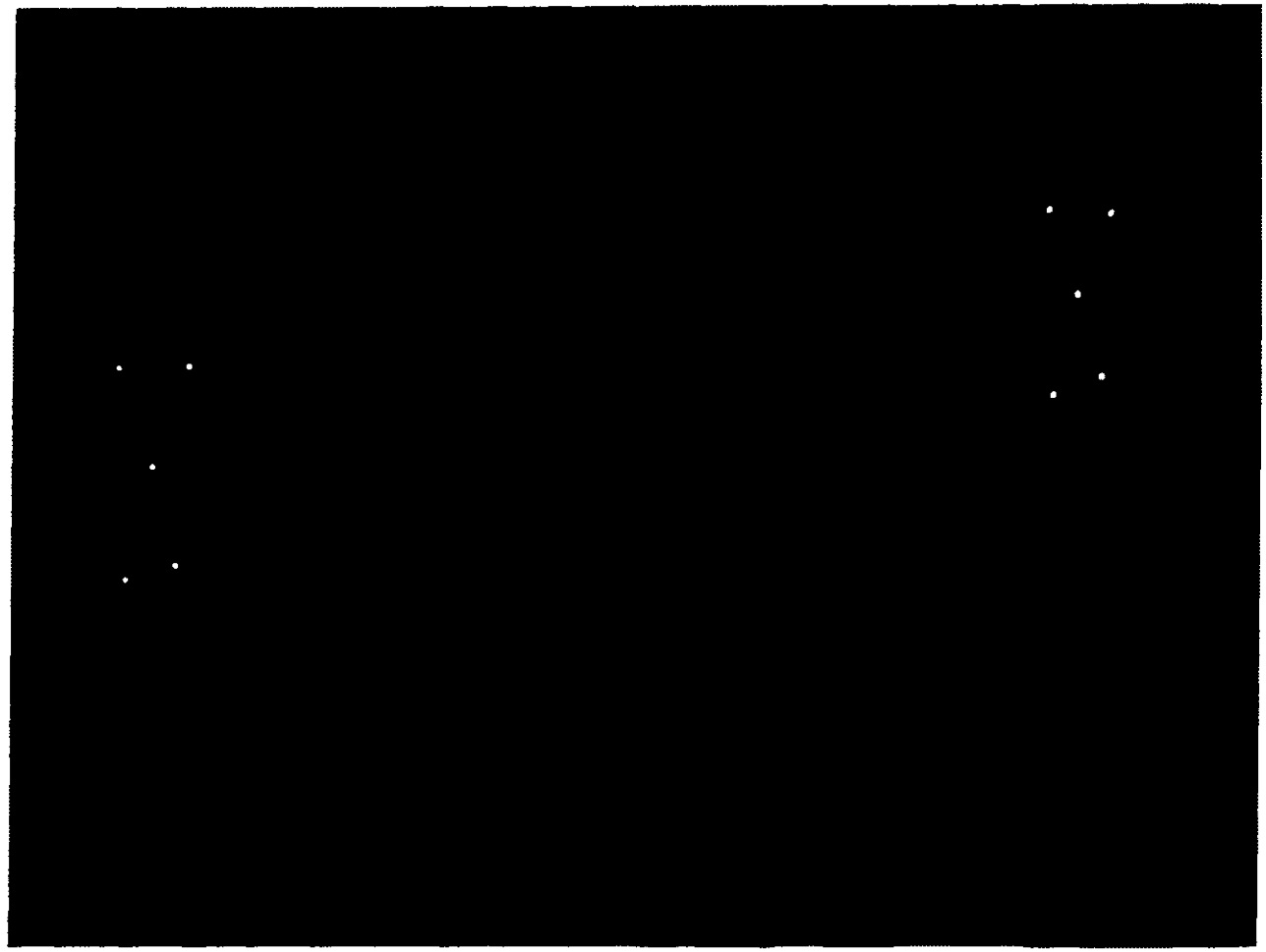


Plate 5. Image showing corner and center points

APPENDIX A

The programs that follow are executed in the same order as they appear, to determine the position of the cylinder in space. The first program thresholds the digitized image; the thresholded data thus obtained is used by the program "IMG_SEG" to segment the stripes and find edges around them. The remaining programs perform the task of filtering the edge data, finding the corners, and determining the position of the cylinder.

The program "IMG_SEG" and its associated subroutines were developed by the Automation Technology Branch at NASA, Langley Research Center.

Note: In the last program "ORION," the angles "ALPHA" and "BETA" mentioned in the comments correspond to the angles "beta" and "gamma," respectively, in the program code.

PROGRAM LISTINGS

```
C     PROGRAM THRESHOLD

C  THIS PROGRAM CONVERTS AN IMAGE FILE INTO A BINARY FILE.
C  THE THRESHOLD VALUE IS ENTERED BY THE USER (0 - 15).  THE INPUT
C  IS 'WORK.51' AND THE BINARY OUTPUT FILE CREATED IS 'WORK.52'
      integer k(160),a(160,16),ab1(160,8),ab2(160,8),ch(160),h(160,2)
      integer th,j,m,b,p,t1,t2,1,fs
      open(unit=2,file='work.51',status='old')
      open(unit=3,file='work.52',status='new')
      TYPE *,'ENTER THRESHOLD (0 -15)'
      READ (5,*) (th)
      TYPE *,'Enter no. of lines stored'
      READ (5,*) fs
      do 80 1=1,fs
      read(2,20) (k(j),j=1,160)
        DO 2 m=1,16
          DO 3 j=1,160
            a(j,m)=0
  3       CONTINUE
  2     CONTINUE
        DO 59 m=1,160
          p=k(m)
          b=1
  5       a(m,b)=mod(p,2)
          b=b+1
          p=p/2
            if (p.eq.0) then
              go to 30
            end if
            if (p.ne.1) then
              go to 5
            end if
          a(m,b)=1
 30       p=a(m,1)+2*a(m,2)+4*a(m,3)+8*a(m,4)
          t1=p
          p=0
          p=a(m,9)+2*a(m,10)+4*a(m,11)+8*a(m,12)
          t2=p

c                 HERE MAKE COMPARISON WITH THRESHOLD

            IF (t1.lt.th .AND. t2.lt.th) THEN
                ch(m)=0
              ELSE IF (t1.lt.th .AND. t2.ge.th) THEN
                ch(m)=3840
              ELSE IF (t1.ge.th .AND. t2.lt.th) THEN
                ch(m)=15
              ELSE IF (t1.ge.th .AND. t2.ge.th) THEN
                ch(m)=3855
            END IF
 59     CONTINUE
      WRITE(3,20) (ch(m),m=1,160)
      write(6,*) 1
 80   CONTINUE
 20   FORMAT(11i7)
 22   FORMAT(20(i3,tr1))
      close(unit=3,dispose='save')
      close(unit=2,dispose='save')
      end
```

```
CD0     program img_seg
CD0
C*****************************************************************
CD0
CD0   Decomposition of image into object contours
CD0
C*****************************************************************
CD1
CD1   PURPOSE
CD1
CD1       This program breaks image into separate objects and describes
CD1     the objects in terms of their centroid, first axis and contour in
CD1     x-y raster form.
CD1     PSEUDOCODE:
CD1     GRAB_IMAGE: Read input image, decompressing the data into pixel format
CD1     Group pixels into desired  nxn "cell" size, determine
CD1      and apply binary threshold to image cells.
CD1     Do until object blobs become smaller than noise cutoff
CD1       CENTROID: Sum neighboring cells to determine object centers of "mass"
CD1       CONTOUR: Plot a rough contour of blob of highest "mass"
CD1       AXIS: Calculate the centroid and first moment of object
CD1       TRACK: Track the pixel resolution contour of object
CD1       GRAB_IMAGE: Erase the current object from stored image
CD1     Enddo
CD1     OUT: Output object attributes to file
CD1
C*****************************************************************
CD2
CD2   DEFINITION OF INPUT
CD2
CD2      CALLING ARGUMENTS
CD2
CD2     SYMBOL          TYPE    DIM     DEFINITION
CD2
CD2     None
CD2
CD2      TERMINAL INPUTS
CD2
CD2     SYMBOL          TYPE    DIM     DEFINITION
CD2
CD2     None
C*****************************************************************
CD3
CD3   DEFINITION OF OUTPUT
CD3
CD3      CALLING ARGUMENTS
CD3
CD2     SYMBOL          TYPE    DIM     DEFINITION
CD3
CD3         N/A
CD3
CD3      TERMINAL OUTPUTS
CD3
CD2     SYMBOL          TYPE    DIM     DEFINITION
CD3
CD3         None
CD3
C*****************************************************************
CD4
CD4   COMMON VARIABLES
CD4
CD4      INPUT
CD4
```

```
CD4      None
CD4
CD4       OUTPUT
CD4      SYMBOL           TYPE     DIM      DEFINITION
CD4
CD4      object_index     integer           index describing object currently proces
sed
CD4      object_total     integer           total number of objects in image
CD4
C*****************************************************************
CD5
CD5    INTERNAL VARIABLES
CD5
CD2      SYMBOL           TYPE     DIM      DEFINITION
CD5
CD5      uinput       character*132         handles user input to getlib ions
CD5      segment.img_size.xpix    integer   horizontal image dimension in pixels
CD5      segment.img_size.ypix    integer   vertical image dimension in pixels
CD5      cellsize                 integer   dimension of square cell group in pixels
CD5      cutoff                   integer   greylevel value of binary threshold
CD5      segment.img_size.xcell             horizontal image dimension in cells
CD5      segment.img_size.ycell             vertical image dimension in cells
CD5
C*****************************************************************
CD6
CD6    EXTERNAL REFERENCES
CD6
CD6       I/O FILES
CD6
CD6          Compressed integer image files (ex. [karin.img_seg_temp]scene.t60
CD6
CD6       SCRATCH FILES
CD6
CD6          N/A
CD6
CD6       EXTERNAL ROUTINES
CD6
CD6         integer - getlib function which gets integer from user input
CD6
C*****************************************************************
CD7
CD7    FUNCTIONAL DESCRIPTION
CD7
C*****************************************************************
CD8
CD8    ASSUMPTIONS AND LIMITATIONS
CD8
C*****************************************************************
CD9
CD9    SPECIAL COMMENTS
CD9
CD9
C*****************************************************************
CD10
CD10   REFERENCES
CD10
CD10       None
CD10
C*****************************************************************
C     THE FOLLOWING CREATES A HELP LIBRARY MODULE
C*****************************************************************
CDX1 AREAL
CDX
```

```
CDX
C*****************************************************************
C
CP BEGIN program
C
        program img_seg
        implicit          none
        include           'img_seg.def'
        integer           threshold
        character*132     uinput  /' '/
        integer           integer
        real              theta(p_maxobj)

C**     Initialize object condition and image partitioning **C
        object_index        = 1
        object_total        = 0

        segment.img_size.xpix = integer('image x-dimension',uinput,'320')
        segment.img_size.ypix = integer('image y-dimension',uinput,'240')
        cellsize = integer('cellsize',uinput,'1')
        cutoff = integer('cutoff threshold',uinput,'4')
        segment.img_size.xcell          = segment.img_size.xpix/cellsize
        segment.img_size.ycell          = segment.img_size.ypix/cellsize

        do while(object_total .eq. 0)
          call grab_image
          call centroid(threshold)
          call contour
C         call axis(theta)
          call track
          object_index = object_index + 1
        enddo
        call out1(threshold,theta)
        close(unit=15,dispose='save')
        end
```

```
      subroutine grab_image

      implicit        none
      include         'img_seg.def/list'
      integer         x,y,px,py,data(160)
      integer*4       str$find_first_in_set
      character*132   uinput  /' '/
      character*132   filename
      integer         loc

      if(object_index .eq. 1)then
        segment.file_name.in=filename('Input file name',uinput,'.dat')
        segment.file_name.out=filename('Output file name',uinput,'.dat')
        loc = str$find_first_in_set (segment.file_name.out, ';')
        segment.file_name.pic=segment.file_name.out(1:loc-4)//'pic'
        open(unit=11,name=segment.file_name.in,status='old')
        open(unit=12,name=segment.file_name.pic(1:loc-1),status='new')
        open(unit=13,name=segment.file_name.out(1:loc-1),status='new')
        open(unit=15,file='b_image.dat',status='new')
        do y = 1,segment.img_size.ypix
          read(11,*) (data(x),x=1,segment.img_size.xpix/2)
          do x = 1,segment.img_size.xpix/2
            segment.image.pixel(2*x-1,y)=ibits(data(x),8,8)
            segment.image.pixel(2*x,y)=ibits(data(x),0,8)
          enddo
c         write(12,100) (segment.image.pixel(x,y),x=1,segment.img_size.xpix)
        enddo
      else
        do px = segment.bound.left(object_index-1),segment.bound.right(object_
index-1)
          do py = segment.bound.top(object_index-1,px),segment.bound.bottom(ob
ject_index-1,px)
            segment.image.cellmass(px,py)=0
            do x = (px-1)*cellsize+1,px*cellsize
              do y = (py-1)*cellsize+1,py*cellsize
                segment.image.pixel(x,y) = 0
              enddo
            enddo
          enddo
        enddo
      endif

      close(unit=11)
100   format(<segment.img_size.xpix>(I4))
      goto 99
666   type*, 'read err:'
99    return
      end
```

```
      subroutine centroid(threshold)

C     Locates centroid of major mass in binary thresholded image.
C     Process involves summing greyvalues for each n X n picture
C     cell in image, thresholding out sparse picture cells, and
C     ranking remaining cells in terms of greatest connectivity to the
C     remaining "higher mass" cells, the highest rank becoming centroid.

      implicit        none
      include         'img_seg.def'
      integer         threshold
      integer         sumtotal,x,y,cellcolumn,cellrow,k,c,count,last
      integer         px(4800),py(4800),connectivity(320,240),superconnectivit
y(320,240)
      logical         flag

      flag            = .true.
      sumtotal        = 0

C     Calculate cell masses; determine threshold level.

      do cellrow = 1,segment.img_size.ycell
        do cellcolumn = 1,segment.img_size.xcell
          if(object_index .eq. 1)then
            segment.image.cellmass(cellcolumn,cellrow) = 0
            do y = cellsize*(cellrow-1)+1,cellsize*(cellrow-1)+cellsize
              do x = cellsize*(cellcolumn-1)+1,cellsize*(cellcolumn-1)+cellsiz
e
c               segment.image.cellmass(cellcolumn,cellrow) = segment.image.cel
lmass(cellcolumn,cellrow) + segment.image.pixel(x,y)
                if(segment.image.pixel(x,y).gt. cutoff) segment.image.cellmass
(cellcolumn,cellrow) =
     1          segment.image.cellmass(cellcolumn,cellrow) + 1
              enddo
            enddo
          endif
          sumtotal = sumtotal + segment.image.cellmass(cellcolumn,cellrow)
        enddo
      enddo
      c = nint(4.0*sumtotal/(segment.img_size.xcell*segment.img_size.ycell*cel
lsize*cellsize))+1
      if(object_index .eq. 1) threshold = nint(c*cellsize**2/8.0)
      if(cellsize .eq. 1) threshold = 1

C     threshold cellmasses and store locations of 'winning' cells
      do cellrow = 1,segment.img_size.ycell
        do cellcolumn = 1,segment.img_size.xcell
          segment.image.bigcell(cellcolumn,cellrow)=0
          connectivity(cellcolumn,cellrow) = 0
        enddo
      enddo
      k = 0
      do cellrow = 2,segment.img_size.ycell-1
        do cellcolumn = 2,segment.img_size.xcell-1
c         type*,cellrow,cellcolumn,segment.image.cellmass(cellcolumn,cellrow)
          if(segment.image.cellmass(cellcolumn,cellrow) .ge. threshold)then
            segment.image.bigcell(cellcolumn,cellrow) = 1
              k = k + 1
              px(k) = cellcolumn
              py(k) = cellrow
          endif
        enddo
      enddo
```

```
        last = k
        if((last*1.0/(segment.img_size.xcell*segment.img_size.ycell)).lt. .0025)
object_total = object_index - 1
C       write(13,*) 'cellfrac:',(last*1.0/(segment.img_size.xcell*segment.img_si
ze.ycell))

C       determine connectivity of winning cells



        do k=1,last
          connectivity(px(k),py(k)) = segment.image.bigcell(px(k)-1,py(k))+segme
nt.image.bigcell(px(k)+1,py(k))+
     1                              segment.image.bigcell(px(k),py(k)-1)+segme
nt.image.bigcell(px(k),py(k)+1)
        enddo

        count = 0
        do while(flag.and.count.lt. 4)
          count = count + 1
          do k=1,last
            superconnectivity(px(k),py(k)) = connectivity(px(k)-1,py(k))+
     1                                       connectivity(px(k)+1,py(k))+
     2                                       connectivity(px(k),py(k)-1)+
     3                                       connectivity(px(k),py(k)+1)+
     4                                       connectivity(px(k),py(k))
          enddo


C         determine centroid
          c = 1
          flag = .false.
          do k=2,last
            if(superconnectivity(px(k),py(k)).eq.
     1         superconnectivity(px(c),py(c))    )then
              flag = .true.
            endif
            if(superconnectivity(px(k),py(k)).gt.
     1         superconnectivity(px(c),py(c))    )then
              flag = .false.
              c = k
            endif
          enddo

          do k=1,last
            connectivity(px(k),py(k))=superconnectivity(px(k),py(k))
          enddo

          segment.centroid.xcell(object_index) = px(c)
          segment.centroid.ycell(object_index) = py(c)
        enddo
c       if(count .ge. 4) write(5,*) 'centroid tie, object',object_index
200     format(' ',<segment.img_size.xcell>(I1))
300     format(' ',<segment.img_size.xcell>(I8))
        return
        end
```

```
      subroutine contour

      implicit        none
      include         'img_seg.def'
      logical         flag,yflag
      integer         ymax(320),ymin(320),direction,x,y,xmax
      integer         px,py,sum,ixx,iyy

      flag = .true.
      x = segment.centroid.xcell(object_index)
      ymin(x) = segment.centroid.ycell(object_index) - 1
      ymax(x) = segment.centroid.ycell(object_index) + 1
      do while(flag .and. (x .le. segment.img_size.xcell))
        direction = - 1
        yflag = .true.
        do while(ymin(x) .ge. 1 .and. ymin(x) .lt. segment.centroid.ycell(obje
ct_index) .and. yflag)
          ymin(x) = ymin(x) + direction
          if(segment.image.bigcell(x,ymin(x))) then
            if(direction .eq. 1) yflag = .false.
          else
            direction = 1
          endif
        enddo
        direction = 1
        yflag = .true.
        do while(ymax(x) .le. segment.img_size.ycell .and. ymax(x) .gt. segmen
t.centroid.ycell(object_index) .and. yflag)
          ymax(x) = ymax(x) + direction
          if(segment.image.bigcell(x,ymax(x))) then
            if(direction .eq. -1) yflag = .false.
          else
            direction = -1
          endif
        enddo
        if((ymax(x) .eq. segment.centroid.ycell(object_index)).and.(ymin(x) .e
q. segment.centroid.ycell(object_index)))flag =.fals
        if(ymin(x) .le. 0) ymin(x) = 1
        if(ymax(x) .ge. segment.img_size.ycell) ymax(x) = segment.img_size.yce
ll
        segment.bound.top(object_index,x) = ymin(x)
        segment.bound.bottom(object_index,x) = ymax(x)
        x = x + 1
        ymin(x)= ymin(x-1)
        ymax(x)= ymax(x-1)
        if(ymin(x) .lt. 2) ymin(x) = 2
        if(ymax(x) .gt. segment.img_size.ycell-1) ymax(x) = segment.img_size.y
cell -1
        if(ymin(x).ge. segment.centroid.ycell(object_index))ymin(x) = segment.
centroid.ycell(object_index) - 1
        if(ymax(x).le. segment.centroid.ycell(object_index))ymax(x) = segment.
centroid.ycell(object_index) + 1
      enddo
      segment.bound.right(object_index) = x-1

      flag = .true.
      x = segment.centroid.xcell(object_index)-1
      ymax(x) = ymax(segment.centroid.xcell(object_index))
      ymin(x) = ymin(segment.centroid.xcell(object_index))
      do while(flag .and. (x .ge. 0))
        direction = -1
        yflag = .true.
        do while(ymin(x) .ge. 1 .and. ymin(x) .lt. segment.centroid.ycell(obje
```

```
ct_index) .and. yflag)
            ymin(x) = ymin(x) + direction
            if(segment.image.bigcell(x,ymin(x))) then
              if(direction .eq. 1) yflag = .false.
            else
              direction = 1
            endif
          enddo


          direction = 1
          yflag = .true.
          do while(ymax(x) .le. segment.img_size.ycell .and. ymax(x) .gt. segmen
t.centroid.ycell(object_index) .and. yflag)
            ymax(x) = ymax(x) + direction
            if(segment.image.bigcell(x,ymax(x))) then
              if(direction .eq. -1) yflag = .false.
            else
              direction = -1
            endif
          enddo
          if((ymax(x) .eq. segment.centroid.ycell(object_index)).and.(ymin(x) .e
q. segment.centroid.ycell(object_index)))flag =.fals
          if(ymin(x) .le. 0) ymin(x) = 1
          if(ymax(x) .ge. segment.img_size.ycell) ymax(x) = segment.img_size.yce
ll
          segment.bound.top(object_index,x) = ymin(x)
          segment.bound.bottom(object_index,x) = ymax(x)
          x = x - 1
          ymin(x)= ymin(x+1)
          ymax(x)= ymax(x+1)
          if(ymin(x) .lt. 2) ymin(x) = 2
          if(ymax(x) .gt. segment.img_size.ycell-1) ymax(x) = segment.img_size.y
cell -1
          if(ymin(x).ge. segment.centroid.ycell(object_index))ymin(x) = segment.
centroid.ycell(object_index) - 1
          if(ymax(x).le. segment.centroid.ycell(object_index))ymax(x) = segment.
centroid.ycell(object_index) + 1
        enddo
        segment.bound.left(object_index) = x+1
        if(segment.bound.left(object_index).eq.0)segment.bound.left(object_index
)=1

        do x = segment.bound.left(object_index),segment.bound.right(object_index
)
          segment.image.bigcell(x,segment.bound.top(object_index,x))=8
          segment.image.bigcell(x,segment.bound.bottom(object_index,x))=8
        enddo
        segment.image.bigcell(segment.centroid.xcell(object_index),
     +                        segment.centroid.ycell(object_index))=5

c       Find centroid of object_index segment.image.pixel mass
        ixx = 0
        iyy = 0
        sum = 0
        do px = segment.bound.left(object_index),segment.bound.right(object_inde
x)
          do py = segment.bound.top(object_index,px),segment.bound.bottom(object
_index,px)
            do x = (px-1)*cellsize+1,px*cellsize
              do y = (py-1)*cellsize+1,py*cellsize
c               ix  = segment.image.pixel(x,y)*x
c               iy  = segment.image.pixel(x,y)*y
c               ixx = ix + ixx
c               iyy = iy + iyy
```

```
                if(segment.image.pixel(x,y) .gt. cutoff) then
                  ixx = x + ixx
                  iyy = y + iyy
                  sum = sum + 1
                endif
              enddo
            enddo
          enddo
        enddo
        if(sum .ne. 0)then
          segment.centroid.xpix(object_index) = ixx/sum
          segment.centroid.ypix(object_index) = iyy/sum
        else
          write(5,*) 'centroid singularity, object',object_index
          write(13,*) 'centroid singularity, object',object_index
        endif

C       xmax=jmin0(80,segment.img_size.xcell)


C       if(segment.img_size.xcell .gt. 80) write(12,*) 'image truncated to fit b
uffer'
C       do y=1,segment.img_size.ycell
C         write(12,200) (segment.image.bigcell(x,y),x=1,xmax)
C       enddo

100     format(<segment.bound.right(object_index)-segment.bound.left(object_inde
x)+1>(' ',I3))
200     format(' ',<segment.img_size.xcell>(I1))
        return
        end
```

```
      subroutine track

      implicit        none
      include         'img_seg.def'
      integer         x,y,xtest,ytest,count,edgecount,case,xmax
      integer         xlast,ylast,xstart,ystart,threshold,theta
      logical         flag


      flag = .true.
      edgecount = 0
      x = segment.centroid.xcell(object_index)
      y = segment.centroid.ycell(object_index)
      do while(((segment.image.bigcell(x,y-1).gt. 0) .or.
     1        (segment.image.bigcell(x,y-2).gt. 0)).and.(y .gt. 2))
        y = y - 1
      end do
      xlast = x - 1
      ylast = y
      xstart = x
      ystart = y
      do while((x .ne. xstart) .or. (y .ne. ystart) .or.(edgecount .eq. 0))
     1                         .and. (flag .eq. .true.))
        count = 1
        flag = .false.
        if(x .eq. xlast) then
          if(y .lt. ylast)then
            case = 1
          else
            case = 3
          endif
        else
          if(x .gt. xlast)then
            case = 2
          else
            case = 4
          endif
        endif

        xlast = x
        ylast = y
        do while((count .le. 4) .and. (flag .eq. .false.))
          count = count + 1
            x = xlast
            y = ylast
          go to (10,20,30,40), case
10          x = xlast-1
            goto 50
20          y = ylast-1
            goto 50
30          x = xlast+1
            goto 50
40          y = ylast+1

50        case = jmod(case,4)+1
          if(segment.image.bigcell(x,y).gt. 0) then
            flag = .true.
            edgecount=edgecount + 1
            segment.contour.x(object_index,edgecount) = x
            segment.contour.y(object_index,edgecount) = y
c           segment.image.bigcell(x,y) = jmod(edgecount,9)+1
            segment.image.bigcell(x,y) = 8
          endif
```

```
c            type*,'xlast,ylast,flag,count,case:',xlast,ylast,flag,count,case-1
           end do
         end do
         segment.contour.count(object_index) = edgecount
         if(flag .eq. .false.) write(5,*) 'tracking error at x,y = ',x,y,', objec
t',object_index
         if(flag .eq. .false.) write(13,*) 'tracking error at x,y = ',x,y,', obje
ct',object_index
         xmax=jmin0(80,segment.img_size.xcell)
         write (12,*) 'object: ',object_index
         if(segment.img_size.xcell .gt. 80) write(12,*) 'image truncated to fit b
uffer'
         do y=1,segment.img_size.ycell
           write(12,100) (segment.image.bigcell(x,y),x=1,xmax)
         enddo
         write(12,*) ' '
100      format(' ',<xmax>(I1))
         return
         end
```

```
      subroutine      outl
      implicit        none
      include         'img_seg.def'
      integer         threshold,count
      real            theta(p_maxobj)

      write(15,22) object_total

      do object_index=1,object_total
        write(15,22) segment.contour.count(object_index)
        do count=1,segment.contour.count(object_index)
          write(15,24) (segment.contour.x(object_index,count),
     1                  segment.contour.y(object_index,count))
        enddo
      enddo
22    format(1i7)
24    format(2i7)
      close(unit=15,dispose='save')
      return
      end
```

```
C  PROGRAM CHECK_REPEAT

C  THE INPUT FILE TO THIS PROGRAM IS 'B_IMAGE.DAT'.  THE PURPOSE OF
c  THIS PROGRAM TO CHECK IF REPEAT VALUES EXIST IN CONTOUR DATA.
C  ANY POINT THAT REPEATS, IS REMOVED.  THIS DATA IS STORED IN A FILE
C  'B_IMAGE1.DAT'.  THE SECOND HALF OF THIS PROGRAM USES THIS FILE
C  'B_IMAGE1.DAT' AND DELETES ALL SINGLE POINTS AND POINTS WHICH FORM
C  AN UNCLOSED CURVE.  A NEW FILE B_IMAGE2.DAT IS CREATED WHICH STORES
C  THE REFINED FORM OF THE CONTOUR DATA.
C  THE FIRST INTEGER IN THE FILES B_IMAGE*.DAT INDICATES THE NUMBER OF
C  OBJECTS WHICH IS INVARIABLY IN THIS CASE.  THE SECOND INTEGER INDICATES
C  THE NUMBER OF POINTS IN THE FIRST OBJECT.  THE PAIR OF INTEGERS WHICH
C  FOLLOW, DENOTE THE X AND Y COORDINATES OF THE OBJECT CONTOUR.  THE PAIRS
C  OF INTEGERS IS FOLLOWED BY A SINGLE INTEGER WHICH INDICATES THE NUMBER OF
C  POINTS IN THE SECOND OBJECT AND SO ON.
      integer i,j,k,pt1(2,1000),tot_pts1,d,m,n,l,t_pt
      integer tpt1,tpt2,count(1000),pt_1(2,1000),pt_2(2,1000)
      integer lx,rx,ty,by,pt2(2,1000),tot_pts2
      integer flag1,flag2,any_n1,any_n2
      open(unit=1,file='b_image.dat',status='old')
      open(unit=2,file='b_image1.dat',status='new')
      read(1,20) (i)
      if (i.ne.2) then
         go to 50
      end if
      write(2,20) (i)
 20   format(1i7)
 22   format(2i7)
      read(1,20) (tot_pts1)
      do j=1,tot_pts1
        read(1,22) (pt1(i,j),i=1,2)
      enddo
      read(1,20) (tot_pts2)
      do j=1,tot_pts2
        read(1,22) (pt2(i,j),i=1,2)
      enddo
      do k=1,tot_pts1-1
        t_pt=tot_pts1
        d=0
        j=k+1
          do i=j,t_pt
           l=i-d
           if ((pt1(1,k).eq.pt1(1,l)).AND.(pt1(2,k).eq.pt1(2,l))) then
             do m=l+1,tot_pts1
               pt1(1,m-1)=pt1(1,m)
               pt1(2,m-1)=pt1(2,m)
             enddo
             tot_pts1=tot_pts1-1
c            write(6,*) (tot_pts1)
             d=d-1
           end if
          enddo
      enddo
      write(2,20) (tot_pts1)
      do n=1,tot_pts1
        write(2,22) (pt1(i,n),i=1,2)
      enddo
c                                                    ***********
      do k=1,tot_pts2-1
        t_pt=tot_pts2
        d=0
        j=k+1
          do i=j,t_pt
```

```
            l=i-d
             if ((pt2(1,k).eq.pt2(1,l)).AND.(pt2(2,k).eq.pt2(2,l))) then
               do m=l+1,tot_pts2
                 pt2(1,m-1)=pt2(1,m)
                 pt2(2,m-1)=pt2(2,m)
               enddo
               tot_pts2=tot_pts2-1
c              write(6,*) (tot_pts2)
               d=d-1
             end if
           enddo
       enddo
       write(2,20) (tot_pts2)
       do n=1,tot_pts2
         write(2,22) (pt2(i,n),i=1,2)
       enddo
       close(unit=1,dispose='save')
       close(unit=2,dispose='save')

c THIS PART OF THE PROGRAM REMOVES SINGLE POINTS AND UNCLOSED CURVES

       open(unit=1,file='b_image1.dat',status='old')
       open(unit=2,file='b_image2.dat',status='new')
       read(1,20) (i)
          if (i.ne.2) then
             go to 50
          end if
       write(2,20) (i)
       read(1,20) (tpt1)
c        write(6,*) (tpt1)
         do j=1,tpt1
            read(1,22) (pt_1(i,j),i=1,2)
         enddo
       read(1,20) (tpt2)
c        write(6,*) (tpt2)
         do j=1,tpt2
            read(1,22) (pt_2(i,j),i=1,2)
         enddo

       any_n1=0
 52    flag1=0
         do i=1,tpt1
            count(i)=0
         enddo
         do i=1,tpt1
            do j=1,tpt1
               lx=pt_1(1,i)-1
               rx=pt_1(1,i)+1
               ty=pt_1(2,i)-1
               by=pt_1(2,i)+1
               if (((lx).eq.(pt_1(1,j))).AND.
     +      ((pt_1(2,i)).eq.(pt_1(2,j)))) then
                  count(i)=count(i)+1
               end if
               if (((rx).eq.(pt_1(1,j))).AND.
     +      ((pt_1(2,i)).eq.(pt_1(2,j)))) then
                  count(i)=count(i)+1
               end if
               if (((ty).eq.(pt_1(2,j))).AND.
     +      ((pt_1(1,i)).eq.(pt_1(1,j)))) then
                  count(i)=count(i)+1
               end if
               if (((by).eq.(pt_1(2,j))).AND.
```

```
     +      ((pt_1(1,i)).eq.(pt_1(1,j)))) then
                 count(i)=count(i)+1
               end if
             enddo
           enddo
          d=0
             do i=1,tpt1
                if (count(i).le.1) then
                  do j=i,tpt1-1
                    pt_1(1,j+d)=pt_1(1,j+d+1)
                    pt_1(2,j+d)=pt_1(2,j+d+1)
                  enddo
                  flag1=1
                  d=d-1
                end if
     c          write(6,*) (d)
             enddo
             if (flag1.eq.1) then
                any_n1=1
             end if
     c       write(6,*) (d)
            tpt1=tpt1+d
             if (flag1.eq.1) then
                go to 52
             end if
             if (any_n1.eq.0) then
                type*, 'NO NOISE IN STRIPE 1 !'
             else
                type*, 'NOISE REDUCTION IN STRIPE 1 COMPLETE'
             end if
            write(2,20) (tpt1)
             do i=1,tpt1
                write(2,22) (pt_1(1,i),pt_1(2,i))
             enddo

            any_n2=0
      53    flag2=0
             do i=1,tpt2
                count(i)=0
             enddo
             do i=1,tpt2
                do j=1,tpt2
                   lx=pt_2(1,i)-1
                   rx=pt_2(1,i)+1
                   ty=pt_2(2,i)-1
                   by=pt_2(2,i)+1
                   if (((lx).eq.(pt_2(1,j))).AND.
     +             ((pt_2(2,i)).eq.(pt_2(2,j)))) then
                      count(i)=count(i)+1
                   end if
                   if (((rx).eq.(pt_2(1,j))).AND.
     +             ((pt_2(2,i)).eq.(pt_2(2,j)))) then
                      count(i)=count(i)+1
                   end if
                   if (((ty).eq.(pt_2(2,j))).AND.
     +             ((pt_2(1,i)).eq.(pt_2(1,j)))) then
                      count(i)=count(i)+1
                   end if
                   if (((by).eq.(pt_2(2,j))).AND.
     +             ((pt_2(1,i)).eq.(pt_2(1,j)))) then
                      count(i)=count(i)+1
                   end if
                enddo
```

```
        enddo
      d=0
         do i=1,tpt2
            if (count(i).le.1) then
              do j=i,tpt2-1
                pt_2(1,j+d)=pt_2(1,j+d+1)
                pt_2(2,j+d)=pt_2(2,j+d+1)
              enddo
              flag2=1
              d=d-1
            end if
c           write(6,*) (d)
         enddo
         if (flag2.eq.1) then
            any_n2=1
         end if
c       write(6,*) (d)
       tpt2=tpt2+d
         if (flag2.eq.1) then
            go to 53
         end if
         if (any_n2.eq.0) then
            type*, 'NO NOISE IN STRIPE 2 !'
         else
            type*, 'NOISE REDUCTION IN STRIPE 2 COMPLETE'
         end if
        write(2,20) (tpt2)
         do i=1,tpt2
            write(2,22) (pt_2(1,i),pt_2(2,i))
         enddo
       close(unit=1,dispose='save')
       close(unit=2,dispose='save')
       go to 51
  50   type*, 'NUMBER OF OBJECTS NOT TWO, CHECK THRESHOLD'
  51   i=1
       end
```

```
C      PROGRAM FIND_CORNER

C  THIS PROGRAM FINDS THE PERPENDICULAR DISTANCE FROM EACH POINT IN THE
C  OBJECT CONTOUR DATA TO THE LINE PASSING THROUGH TWO POINTS ON EITHER
C  SIDE OF THE CONCERNED POINT.  THE DISTANCE AT WHICH TWO POINTS ARE
C  CHOSEN ON EITHER SIDE OF THE CONCERNED POINT IS ENTERED BY THE USER
C  WHEN THE PROGRAM PROMTS TO ENTER THIS SPAN.  THE INPUT FILE IS
C  'B_IMAGE2.DAT' AND THE OUTPUT FILE CREATED IS 'CORN.DAT'.  THE FIRST
C  INTEGER IN THIS FILE INDICATES THE NUMBER OF OBJECTS WHICH IS TWO HERE.
C  THE SECOND INTEGER INDICATES THE THE TOTAL NUMBER OF POINTS IN THE
C  FIRST OBJECT CONTOUR DATA.  THE INTEGERS THAT FOLLOW REPRESENT THE
C  ABOVE MENTIONED 'PERPENDICULAR DISTANCE'.  THE NEXT INTEGER THAT
C  FOLLOWS, INDICATES THE NUMBER OF POINTS IN THE SECOND OBJECT CONTOUR
C  DATA, AND SO ON.
      integer i,j,k,tot_obj,tot_pts1,tot_pts2,pt1(2000,2),pt2(2000,2),sp
      integer v1,v2,xo1,xo2,yo1,yo2,xt1,xt2,yt1,yt2,n11,n12
      real dis1(1000),dis2(1000),d11,d12
      open(unit=1,file='b_image2.dat',status='old')
      open(unit=2,file='corn.dat',status='new')
      read(1,22) (tot_obj)
 22   format(1i7)
      if (tot_obj.ne.2) then
         go to 30
      end if
      write(2,22) (tot_obj)
      read(1,22) (tot_pts1)
        do 40 i=1,tot_pts1
          read(1,24) (pt1(i,j),j=1,2)
 40     continue
      write(2,22) (tot_pts1)
 24   format(2i7)
      read(1,22) (tot_pts2)
        do 41 i=1,tot_pts2
          read(1,24) (pt2(i,j),j=1,2)
 41     continue
      type*, 'ENTER SPAN IN PIXEL UNITS - '
      read(5,*) (sp)
        do 42 i=1,tot_pts1
          v1=i-sp
            if (v1.le.0) then
              v1=tot_pts1+v1
            end if
          v2=mod(i+sp,tot_pts1)
            if (v2.eq.0) then
              v2=tot_pts1
            end if
          yo1=pt1(i,2)
          xo1=pt1(i,1)
          yt1=pt1(v1,2)-pt1(v2,2)
          xt1=pt1(v1,1)-pt1(v2,1)
          n11=xo1*yt1 - yo1*xt1 - pt1(v1,1)*yt1 + pt1(v1,2)*xt1
          n11=abs(n11)
          d11=sqrt(real(yt1**2 + xt1**2))
          dis1(i) = n11/d11
        write(2,26) (dis1(i))

c                              ************************************

 42     continue
        write(2,22) (tot_pts2)
        do 43 i=1,tot_pts2
          v1=i-sp
            if (v1.le.0) then
```

```
            v1=tot_pts2+v1
          end if
        v2=mod(i+sp,tot_pts2)
          if (v2.eq.0) then
            v2=tot_pts2
          end if
        yo2=pt2(i,2)
        xo2=pt2(i,1)
        yt2=pt2(v1,2)-pt2(v2,2)
        xt2=pt2(v1,1)-pt2(v2,1)
        nl2=xo2*yt2 - yo2*xt2 - pt2(v1,1)*yt2 + pt2(v1,2)*xt2
        nl2=abs(nl2)
        dl2=sqrt(real(yt2**2 + xt2**2))
        dis2(i) = nl2/dl2
      write(2,26) (dis2(i))
 43     continue
 26   format(e18.6)
c     amin1=angl(1)
c     imin=1
c     do 45 i=2,tot_pts1
c       if (ang(i).lt.amin1) then
c         amin1=ang(i)
c         imin=i
c       end if
c45   continue
c     dif2=angl(1)-amin1
c     do 46 i=1,tot_pts1
c       if ((angl(i)-amin1).lt.dif2) then
c         dif2=angl(i)-amin1
c         imin2=i
c       end if
c     dif3=ang
        go to 35
 30   type*, 'NO. OF OBJECTS IS NOT TWO CHECK THRESHOLD'
 35   i=1
      close(unit=2,dispose='save')
      close(unit=1,dispose='save')
      end
```

```
C      PROGRAM CORNER

c   THIS PROGRAM FINDS THE FOUR CORNERS IN THE CONTOUR OF THE TWO OBJECTS.
C   THIS IS DONE BY FINDING THE FOUR PEAKS IN THE 'PERPENDICULAR DISTANCE'
C   DATA FORM THE FILE 'CORN.DAT'.  FROM THE FOUR CORNER POINTS THE CENTER
C   OF THE QUADRILATERAL IS FOUND, WHICH IS THE POINT OF INTEREST.  THE
C   FOUR POINTS AND ITS CENTER FOR BOTH OBJECTS IS STORED IN A FILE CALLED
C   'POINTS.DAT'.
       integer i,j,k,t_pt1,t_pt2,iil(4),ii2(4),m,i2,sml
       integer pt1(1000,2),pt2(1000,2),pr1_1(2),pr1_2(2)
       integer pr2_1(2),pr2_2(2),c1x,c1y,c2x,c2y
       real dis1(1000),dis2(1000),td1,td2,sm,ndis1(4),ndis2(4)
       real pp1_1(2),pp1_2(2),pp2_1(2),pp2_2(2)
       real cc1x,cc1y,cc2x,cc2y
       open(unit=1,file='corn.dat',status='old')
       open(unit=2,file='b_image2.dat',status='old')
       open(unit=3,file='points.dat',status='new')
 20    format(1i7)
 22    format(e18.6)
 24    format(2i7)
       read(2,20) (i)
       read(2,20) (i)
         do j=1,i
            read(2,24) (pt1(j,k),k=1,2)
         enddo
       read(2,20) (i)
         do j=1,i
            read(2,24) (pt2(j,k),k=1,2)
         enddo

       read(1,20) (i)
       read(1,20) (t_pt1)
         do i=1,t_pt1
            read(1,22) (dis1(i))
         enddo
       read(1,20) (t_pt2)
         do i=1,t_pt2
            read(1,22) (dis2(i))
         enddo
       do j=1,4
       td1=dis1(1)
         do i=2,t_pt1
            if (dis1(i).gt.td1) then
               td1=dis1(i)
            end if
         enddo
         do i=1,t_pt1
            if (dis1(i).eq.td1) then
               iil(j)=i
               go to 50
            end if
         enddo
 50      k=iil(j)
         dis1(k)=0
         do i=1,8
            m=mod(k+i,t_pt1)
            if (m.eq.0) then
               m=0
            end if
            dis1(m)=0
            m=k-i
            if (m.le.0) then
              m=t_pt1+m
```

```
            end if
            dis1(m)=0
          enddo
        enddo
        write(6,*) (ii1(i),i=1,4)
        do j=1,4
        td2=dis2(1)
          do i=2,t_pt2
            if (dis2(i).gt.td2) then
              td2=dis2(i)
            end if
          enddo
          do i=1,t_pt2
            if (dis2(i).eq.td2) then
              ii2(j)=i
              go to 51
            end if
          enddo
51        k=ii2(j)
          dis2(k)=0
          do i=1,8
            m=mod(k+i,t_pt2)
            if (m.eq.0) then
              m=0
            end if
            dis2(m)=0
            m=k-i
            if (m.le.0) then
              m=t_pt2+m
            end if
            dis2(m)=0
          enddo
        enddo
        write(6,*) (ii2(i),i=1,4)

          do i=1,3
            do j=i+1,4
              if (ii1(i).gt.ii1(j)) then
                k=ii1(i)
                ii1(i)=ii1(j)
                ii1(j)=k
              end if
            enddo
          enddo
          do i=1,3
            do j=i+1,4
              if (ii2(i).gt.ii2(j)) then
                k=ii2(i)
                ii2(i)=ii2(j)
                ii2(j)=k
              end if
            enddo
          enddo
        write(6,*) (ii1(i),i=1,4)
        write(6,*) (ii2(i),i=1,4)

          do i=1,4
            i2=mod(i+1,4)
              if (i2.eq.0) then
                i2=4
              end if
            ndis1(i)=(pt1(ii1(i),1)-pt1(ii1(i2),1))**2 +
     +               (pt1(ii1(i),2)-pt1(ii1(i2),2))**2
```

```
      ndis1(i)=sqrt(ndis1(i))
    enddo

    do i=1,4
      i2=mod(i+1,4)
        if (i2.eq.0) then
          i2=4
        end if
      ndis2(i)=(pt2(ii1(i),1)-pt2(ii1(i2),1))**2 +
     +          (pt2(ii1(i),2)-pt2(ii1(i2),2))**2
      ndis2(i)=sqrt(ndis2(i))
    enddo

    sm=ndis1(1)
      do i=2,4
        if (sm.gt.ndis1(i)) then
          sm=ndis1(i)
        end if
      enddo
      do i=1,4
        if (sm.eq.ndis1(i)) then
          sml=i
        end if
      enddo
     pr1_1(1)=sml
      i=mod(sml+1,4)
        if (i.eq.0) then
          i=4
        end if
     pr1_1(2)=i
     pr1_2(1)=mod(pr1_1(2)+1,4)
        if (pr1_2(1).eq.0) then
          pr1_2(1)=4
        end if
     pr1_2(2)=mod(pr1_2(1)+1,4)
        if (pr1_2(2).eq.0) then
          pr1_2(2)=4
        end if

    sm=ndis2(1)
      do i=2,4
        if (sm.gt.ndis2(i)) then
          sm=ndis2(i)
        end if
      enddo
      do i=1,4
        if (sm.eq.ndis2(i)) then
          sml=i
        end if
      enddo
     pr2_1(1)=sml
      i=mod(sml+1,4)
        if (i.eq.0) then
          i=4
        end if
     pr2_1(2)=i
     pr2_2(1)=mod(pr2_1(2)+1,4)
        if (pr2_2(1).eq.0) then
          pr2_2(1)=4
        end if
     pr2_2(2)=mod(pr2_2(1)+1,4)
        if (pr2_2(2).eq.0) then
          pr2_2(2)=4
```

```
        end if

      pp1_1(1)=(pt1(ii1(pr1_1(1)),1) + pt1(ii1(pr1_1(2)),1))/2
      pp1_1(2)=(pt1(ii1(pr1_1(1)),2) + pt1(ii1(pr1_1(2)),2))/2
      pp1_2(1)=(pt1(ii1(pr1_2(1)),1) + pt1(ii1(pr1_2(2)),1))/2
      pp1_2(2)=(pt1(ii1(pr1_2(1)),2) + pt1(ii1(pr1_2(2)),2))/2

      pp2_1(1)=(pt2(ii2(pr2_1(1)),1) + pt2(ii2(pr2_1(2)),1))/2
      pp2_1(2)=(pt2(ii2(pr2_1(1)),2) + pt2(ii2(pr2_1(2)),2))/2
      pp2_2(1)=(pt2(ii2(pr2_2(1)),1) + pt2(ii2(pr2_2(2)),1))/2
      pp2_2(2)=(pt2(ii2(pr2_2(1)),2) + pt2(ii2(pr2_2(2)),2))/2

      cc1x = (pp1_1(1) + pp1_2(1))/2
      cc1y = (pp1_1(2) + pp1_2(2))/2
      cc2x = (pp2_1(1) + pp2_2(1))/2
      cc2y = (pp2_1(2) + pp2_2(2))/2

      c1x=int(cc1x)
      c1y=int(cc1y)
      c2x=int(cc2x)
      c2y=int(cc2y)

    write(3,20) (2)
    write(3,20) (5)
    write(3,24) (pt1(ii1(pr1_1(1)),i),i=1,2)
    write(3,24) (pt1(ii1(pr1_1(2)),i),i=1,2)
    write(3,24) (pt1(ii1(pr1_2(1)),i),i=1,2)
    write(3,24) (pt1(ii1(pr1_2(2)),i),i=1,2)
    write(3,24) (c1x,c1y)

    write(3,20) (5)
    write(3,24) (pt2(ii2(pr2_1(1)),i),i=1,2)
    write(3,24) (pt2(ii2(pr2_1(2)),i),i=1,2)
    write(3,24) (pt2(ii2(pr2_2(1)),i),i=1,2)
    write(3,24) (pt2(ii2(pr2_2(2)),i),i=1,2)
    write(3,24) (c2x,c2y)

  close(unit=1,status='save')
  close(unit=2,status='save')
  close(unit=3,status='save')

  end
```

```
C     PROGRAM ORION

C     THIS PROGRAM COMPUTES THE ORIENTATION USING THE TWO POINTS
C     OBTAINED FROM THE PROGRAM 'CORNER.FOR', THAT IS THE CENTER OF
C     THE QUADRILATERAL.  THIS DATA IS IN THE FILE 'POINTS.DAT'.
C     THE OUTPUT ORIENTATION ANGLES IS SHOWN ON THE SCREEN.
C     'ALPHA' IS THE INCLINATION OF THE CYLINDER AXIS WITH RESPECT TO
C     THE HORIZONTAL ABOUT A HORIZONTAL AXIS.
C     'BETA' IS THE ORIENTATION ABOUT THE VERTICAL AXIS.
      integer i,j,k,x1,y1,x2,y2
      real r,xp1,yp1,xp2,yp2,x_ap1,y_ap1,x_ap2,y_ap2,z0,hy1,hy2
      real a_phi1,a_phi2,th,alpha1,alpha2,xn1,yn1,zn1,xn2,yn2,zn2
      real y_p1,y_p2,xs1,ys1,zs1,xs2,ys2,zs2,beta,gamma,bn1,bn2,br
      real gn1,gn2,gr,tp
      open(unit=1,file='points.dat',status='old')
 22   format(1i7)
 24   format(2i7)
      read(1,22) (i)
      read(1,22) (i)
      read(1,24) (i,j)
      read(1,24) (i,j)
      read(1,24) (i,j)
      read(1,24) (i,j)
      read(1,24) (x1,y1)
      read(1,22) (i)
      read(1,24) (i,j)
      read(1,24) (i,j)
      read(1,24) (i,j)
      read(1,24) (i,j)
      read(1,24) (x2,y2)
c     type*, 'input x1,y1 values'
c     read(5,*) (x1,y1)
c     type*, 'input x2,y2 values'
c     read(5,*) (x2,y2)
        if (x1.gt.x2) then
           tp = x1
           x1 = x2
           x2 = tp
           tp = y1
           y1 = y2
           y2 = tp
        end if
      r = 2.2/80
      xp1 = -119.5 + y1
      yp1 =  159.5 - x1
      xp2 = -119.5 + y2
      yp2 =  159.5 - x2
      x_ap1 = r*xp1
      y_ap1 = r*yp1
      x_ap2 = r*xp2
      y_ap2 = r*yp2
      z0 = 25.4
      hy1 = sqrt((z0)**2 + (y_ap1)**2)
      hy2 = sqrt((z0)**2 + (y_ap2)**2)
      a_phi1 = atand(y_ap1/z0)
      a_phi2 = atand(y_ap2/z0)
      th = 28
      alpha1 = a_phi1 - th
      alpha2 = a_phi2 - th
      xn1 = x_ap1
      yn1 = hy1*(sind(alpha1))
      zn1 = -hy1*(cosd(alpha1))
      xn2 = x_ap2
```

```
yn2 = hy2*(sind(alpha2))
zn2 = -hy2*(cosd(alpha2))
y_p1 = 359
y_p2 = 359 + 278
ys1 = y_p1
xs1 = (y_p1/yn1)*xn1
zs1 = (y_p1/yn1)*zn1
ys2 = y_p2
xs2 = (y_p2/yn2)*xn2
zs2 = (y_p2/yn2)*zn2
write(6,*) (xs1,ys1,zs1)
write(6,*) (xs2,ys2,zs2)
bn1 = (ys1 - ys2)**2 + (zs1 - zs2)**2
bn1 = sqrt(bn1)
bn2 = (xs1 - xs2)**2 + (ys1 - ys2)**2 + (zs1 - zs2)**2
bn2 = sqrt(bn2)
br = bn1/bn2
beta = acosd(br)
  if (xs1.lt.xs2) then
     beta = -beta
  end if
gn1 = (ys1 - ys2)**2
gn1 = sqrt(gn1)
gn2 = (ys1 - ys2)**2 + (zs1 - zs2)**2
gn2 = sqrt(gn2)
gr = gn1/gn2
gamma = acosd(gr)
  if (zs1.lt.zs2) then
     gamma = -gamma
  end if
write(6,*) ('beta = ',beta)
write(6,*) ('gamma = ',gamma)
close(unit=1,dispose='save')
end
```