Sensor-based automated path guidance of a robot tool

Anietie Udo Ukpong
University of Nevada, Las Vegas

Follow this and additional works at: https://digitalscholarship.unlv.edu/rtds

Repository Citation
https://digitalscholarship.unlv.edu/rtds/223

This Thesis is brought to you for free and open access by Digital Scholarship@UNLV. It has been accepted for inclusion in UNLV Retrospective Theses & Dissertations by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Sensor-based automated path guidance of a robot tool

Ukpong, Anietie Udo, M.S.
University of Nevada, Las Vegas, 1992
SENSOR-BASED AUTOMATED PATH GUIDANCE OF A
ROBOT TOOL

by

Anietie Udo Ukpong

A thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Mechanical Engineering
(CAD/CAM, Robotics)

Department of Mechanical Engineering
University of Nevada, Las Vegas
July, 1992
The thesis of Mr. Anietie U. Ukpong for the degree of Master of Science in Mechanical Engineering is approved.

Chairperson, Professor Woosoon Yim, Ph.D

Examining Committee Member, Professor Mohammed Trabia, Ph.D

Examining Committee Member, Professor Robert Skaggs, Ph.D

Graduate Faculty Representative, Professor E. A Yfantis, Ph.D

Graduate Dean, Professor Ronald W. Smith, Ph.D

University of Nevada, Las Vegas

July, 1992
ABSTRACT

A structured lighting technique based on cross-shaped laser beam is used to guide a robot tool along an unknown geometry of workpiece. The system consists of two CCD cameras and two laser line projectors, and the camera’s image is processed by an IBM 386 microcomputer using Matrox Imager-AT image processing board. The objective of the research is to develop a robot capability for a simultaneous measurement of the orientation (surface normal) and position of a 3-dimensional unknown object for a precise tool path guidance and control. The proposed system can guide the robot manipulator while maintaining specific orientation between the robot end-effector and the workpiece and also generate a measured geometric CAD database.

The project is focused in two phases. The first phase involves the computer graphics simulation of an automated guidance and control of a robot tool using the proposed scheme. In the simulation, where both constant and variable tool orientation are considered, an object of known geometry is used for camera image data generation and subsequently determining the position and orientation of surface points based only on the simulated camera image information. Based on this surface geometry measurement technique, robot tool guidance and path planning algorithm is developed.

The second phase involves the laboratory experiment. The complete laboratory implementation of the proposed scheme is reserved for future work. However, to
demonstrate the validity of the proposed measurement method, the result of CCD image processing (grey to binary image conversion, thinning of binary image, detection of cross point, etc) and the calibration of the cameras/lighting source are performed.
ACKNOWLEDGMENTS

I express my profound gratitude to Dr. Woosoon Yim for his brilliant supervision of this work and for stimulating my interest in the area of CAD/CAM, robotics and factory automation. I remain specially indebted to him for all his suggestions, guidance, encouragement and constructive criticism of this project.

Much thanks also goes to Dr. Boehm, Chairman of Mechanical Engineering Department for all the financial support by way of teaching assistantship, also to Dr. E. Yfantis, Dr. Junichi Kanai of computer science department and to Dr. M. Trabia, Dr. R. Skaggs, Dr. Moujaes, Dr. Reynolds and Dr. Mauer all of Mechanical Engineering Department for their constructive criticisms of this project.

Finally, much thanks goes to my uncle Ikpe Ekpoudia, M.D, for all his moral and financial support and above all, for his ageless altruism towards the pursuit of knowledge, my mother Mrs. Hannah Ukpong, brothers S. Ukpong, M.D, Engr. Idongesit Akpan and the rest for all their prayers. I dedicate this work and others to come to my late father Elder U. A. Ukpong, who preferred to go without food so that his children could be educated.

I cannot end without thanking my creator for all his mercies and making my academic pursuit at the University of Nevada, Las Vegas a very interesting one.
# Contents

Abstract ................................................................. iii  
Acknowledgments ...................................................... v  

1 INTRODUCTION ......................................................... 1  
1.1 Robot and Flexible Environment .............................. 1  
1.2 Measurement of 3-D Information .............................. 2  
1.3 Overview of the Thesis ............................................. 3  

2 LITERATURE REVIEW .................................................. 5  

3 SYSTEM DESIGN ......................................................... 7  
3.1 Introduction .......................................................... 7  
3.2 System Configuration .............................................. 7  
3.3 Operational Principle ............................................. 10  
3.4 Interpolation of Detected Surface Points (X,Y,Z) ............ 14  
  3.4.1 Computation of Surface Normals ........................ 19  
3.5 Design Constraints ............................................... 21  

4 SIMULATION .............................................................. 22  
4.1 Introduction .......................................................... 22  
4.2 The Sensing Environment ........................................ 22  
4.3 Laser-Line Beams Position and Orientation .................. 24  
4.4 Ray Tracing to Get Intersection Points ....................... 26  
4.5 Camera Tracking .................................................. 32
4.5.1 Constant Orientation Tracking ....................................................... 33
4.5.2 Variable Orientation Tracking ....................................................... 34
4.5.3 Simulation Results ................................................................. 37

5 IMPLEMENTATION ......................................................................................... 48
  5.1 Overview ................................................................. 48
  5.2 Camera/Lighting Source Calibration ....................................................... 49
    5.2.1 Coordinate Systems ................................................................. 50
    5.2.2 Range Measurement: $z'$ ................................................................. 51
    5.2.3 $x'$ Coordinate ................................................................. 54
    5.2.4 Calibration ................................................................. 56
  5.3 Image Processing ................................................................. 57

6 CONCLUSION ....................................................................................................... 59
  6.1 Conclusion ................................................................. 59
  6.2 Recommendation ................................................................. 60
List of Figures

3.1 Coordinate Frame of the Proposed System .............................................. 8
3.2 Coordinate of Camera 1 with respect to Laser 2 .................................. 9
3.3 Coordinate of Camera 2 with respect to Laser 1 .................................. 10
3.4 Top View of Geometry Showing Camera/Plane-of-Light (Laser 2) Arrangement for Range Measurement using Camera 1 .................. 11
3.5 Interpolation of the Detected Surface Point(X, Y, Z) .......................... 14
3.6 Plane Approximation for Detection of Surface Normal ...................... 20

4.1 Initial Alignment of Tool Coordinate System ................................. 23
4.2 Euler angles (O, A, T) used for PUMA robot .................................. 24
4.3 Schematic diagram of the Simulated System using Ray Tracing Technique .................. 28
4.4 Intersection of Ray with Table Top, Top-Half and Lower-Half of the Sphere using Ray Tracing Technique .......................... 30
4.5 Geometry of Perspective Transformation ...................................... 32
4.6 Top View of the Workspace with Tracking Grid Path ...................... 33
4.7 Coordinate System for Variable Orientation Tracking ....................... 35
4.8 Rotation Procedures for Aligning Tool Axis to the Desired Orientation .......... 36
4.9 Tool Movement using Variable Orientation Algorithm ...................... 38
4.10 Status of Simulation Graphics Windows for Constant Orientation Tracking Showing Tool Movements and Locations .................. 39
4.11 Status of Simulation Graphics Windows for Variable Orientation Tracking Showing Tool Movements and Locations .................. 40
4.12 Overlain Surface Normals for the Constant Orientation Case ............... 41
4.13 Overlain Surface Normals for the Variable Orientation Case  
4.14 One-Pass Trajectory Over Sphere of Radius 100cm, Step Size = 2, for 
    Constant Tool Orientation, Grid Size is 110 X 110(cm)  
4.15 Two Dimensional Plot of An Entire Trajectory Over Sphere of Radius 
    40cm, Step Size = 1, for Constant Tool Orientation. Grid Size is 60 X 
    60(cm)  
4.16 Surface Contour Generated for a Sphere of Radius 20cm Using Con­ 
    stant Tool Orientation, Step Size = 1, Grid Size is 30 X 30(cm)  
4.17 Surface Contour Generated for a Sphere of Radius 40cm Using Con­ 
    stant Tool Orientation, Step Size = 1. Grid Size is 60 X 60(cm)  
4.18 Tool Trajectory for the Variable Tool Orientation Case, Radius = 
    100cm, Step Size = 2 and Initial Offset = 200cm. Notice that the 
    tool quickly attains the initial offset of 200cm as soon as it leaves the 
    sphere.  
5.1 Proposed Hardware/Interfacing  
5.2 Image Formation Using an Ideal Thin Lens  
5.3 Camera/Plane-of-light Geometry  
5.4 Side View of Camera/Plane-of-Light Geometry for An Illuminated 
    Field Point W in the Plane X' = 0  
5.5 Enlarged View of Camera Geometry  
5.6 Top View of Camera/Plane-of-Light geometry for An arbitrary Field 
    Point W, GW is the projection of W onto Z' axis  
5.7 Cross-Laser Before Thinning  
5.8 Cross-Laser After Thinning
Chapter 1

INTRODUCTION

1.1 Robot and Flexible Environment

The Robot Institute of America (RIA) has defined a robot [1] as a programmable, multifunction manipulator synthesized for the movement of tools, materials, parts, or specialized devices through variable programmed motions for the performance of a variety of tasks. Efforts in the area of developing an autonomous intelligent machine capable of performing various manufacturing tasks without supervision has increased lately. The same is true for sensing technology which is receiving increasing attention as a means of endowing the machine or mechanism with the capability of exhibiting a greater degree of intelligence and flexibility in dealing with its environment.

Hence, a robot that can see and feel is easier to train and program to perform a variety of complex tasks while at the same time requires less stringent controls than those of preprogrammed machines [2]. These types of systems can achieve high degree of universality that ultimately translates into lower production and maintenance costs.

The applications of 3-dimensional vision in robotics and manufacturing have been classified [3] into three major categories: visually guided manipulation, visual inspection and computer aided design and manufacturing. The central idea in the area of robot vision is that of the recovery of the missing depth information from the vision
system input (camera data) which is inherently two dimensional. Computer vision and robot vision are terms that have evolved over the years to include two separate but related functions that provides visual sensing for computer-driven robots. These two functions are *electro-optical imaging* and *image processing*. Electro-optical imaging converts optical radiations to appropriate electronic signals for input to the robot's computer, whereas image processing extracts useful information from the electronic image provided by the sensor. The image processing function is perhaps more complicated in view of the difficulty in deciding on what constitutes useful information and in choosing an algorithm to extract such information from the image.

Hence, robot vision may be defined \(2\) as the process of extracting, characterizing, and interpreting information from images of a three-dimensional world. It is subdivided into sensing, preprocessing, segmentation, description, recognition and interpretation. Sensing is the process of actually generating the visual image (it could be direct e.g by touch or indirect as in ultrasonic or sonic sensing), while preprocessing is concerned with enhancement of details with noise reduction. Segmentation then partitions the object image possibly into regions of interest for further processing, for instance, description, thinning, etc, and then recognition.

### 1.2 Measurement of 3-D Information

Various procedures have been developed for measuring 3-dimensional information. Direct measurements usually results in range data i.e value of the distances between a range sensor and a set of surface points. Indirect measurement is inferred from monocular images on the basis of target properties resulting in surface orientation.

Triangulation and time-of-flight procedures are the basic methods for determining range directly \(3\). Triangulation techniques are subdivided into two schemes: stereo, using ambient lighting and two cameras (a passive scheme), and structured light, using a projector of controlled illumination and a camera (an active scheme). The main drawback of the stereo vision technique, in addition to missing data points, is the correspondence problem, i.e, how to match corresponding points in stereo images
reliably and quickly [3]. Stereo vision technique would have provided the best means of estimating object’s range, but the effect of the correspondence problem is an increase in the measurement time.

The problem of correspondence is eliminated by using active lighting, a scheme in which one of the stereo cameras is replaced by a controlled illumination source called structured light. The structured light may consist of single or multiple light patterns each of which may be a straight line (a beam), planar or non-planar. The virtue of structured lighting technique is that it replaces natural features (edges, etc), if any, on object surface in the scene by artificial features in our case, cross-laser beam, that are very easily detected. False impressions caused by noisy ambient illuminations can be eliminated with this technique because it enhances gray to binary scale image conversions.

Our goal here is to provide a robot with the capability for a simultaneous measurement of the orientation (surface normal) and position of a three dimensional unknown object for a precise tool path guidance, control and inspection tasks. It can be used to generate a measured geometric database which can be used for rapid automatic contour programming, part analysis, production control and finally enhances the capability of guiding the robot manipulator in real time where maintaining specific orientation between the robot end-effector and the workpiece is critical.

1.3 Overview of the Thesis

A brief literature survey of the previous and current state of research in this area is presented in Chapter 2. Chapter 3 deals with the system design. The geometric configuration of the system along with the operational principle and the design constraints is also presented. Chapter 4 elucidates the computer graphics simulation of the proposed system. Ray tracing technique with an object of known geometry is used for camera image data generation. The trackability of the system and the subsequent results are presented. Chapter 5 explains the implementation procedure of the proposed system. Emphasis is on the calibration of the camera/lighting source and
the image processing of CCD camera image for the extraction of the center point of cross image. Finally, chapter 6 concludes the work and offers possible future research direction for the laboratory implementation of the scheme.
Chapter 2

LITERATURE REVIEW

Recent trends in machine vision and computer-aided manufacturing research have emphasized generating an appropriate surface geometry for machining, welding, and inspection. For years, efforts had been geared towards obtaining a three dimensional object geometry via depth measurement [4]. Various techniques have been developed in the past decades for inferring orientation using various sources, devices, information and heuristics [5]. Proposed methods have exploited regularity and surface structure constraints [6, 7, 8], occluding contours [9], photometric constraints [10, 11, 12], stereoscopy [13, 14], regular pattern [15], and spatial encoding [5, 16-18]. Laser ranging, an active sensing scheme, have also been adopted [16, 17].

Shape-from shading techniques for instance infers surface structure and orientation [5] by considering the photometric constraints imposed by the image formation process. However, these techniques suffer some set backs in characterizing the lighting geometry including reflectance and degree of specularity of the object surface. Some of these techniques also relies on more than one illumination source to infer orientation [11, 19].

Laser range sensors can yield a dense set of depth values from which surface structure can be obtained through surface fitting or approximation [17]. However, this active sensing method is only suitable for limited applications. There is a strong
physiological evidence in support of the stereo vision approach [5]. Stereo vision utilizes the disparity between projected positions of a point in two images to infer the depth at that point. However, correspondence between points in the two images is difficult to establish in view of the errors due to digitization and camera calibration.

Bounding volume representations can also be constructed if several occluding contours from different viewpoints are available [9]. However, for reasonable surface structure approximation, several views may be necessary and selection of view points may be critical. Surface structure can also be recovered if enough a priori knowledge about the properties of the object surfaces is known or if certain constraints can be imposed on the scene. Scenes containing objects with polyhedra faces have been extensively studied [6, 20].

Using spatial encoding to recover surface structure was proposed recently [21-23], whereas, the passive sensing techniques such as shape-from-shading and stereoscopy gather information from ambient light reflections, spatial encoding methods cast modulated light onto the scene to obtain a spatial encoding for analysis. This technique was first proposed in [21-27] to extract surfaces with different orientations and later used to recover depth information using stereoscopic pairs of grid coded images [23-25]. However, grid coding for depth reconstruction under stereopsis principle poses some non trivial difficulties.

Our proposed measurement scheme takes on the structured illumination approach exploiting the regular pattern of two cross-shaped laser line beams to infer local surface orientation at their point of intersection. In this case, the camera system acquires the object’s image with the imposed cross laser line beams and the necessary calculations are done to interpret the image for analysis.

The system is expected to ensure high data acquisition rate at any point of interest, and the cross-shaped laser line beams will enable the easy integration of parametric curve to infer the surface orientation at the cross points.
Chapter 3

SYSTEM DESIGN

3.1 Introduction

In this chapter, the basic geometrical relationships along with the coordinate assignments are discussed in section 3.2. Operational principle is discussed in section 3.3 based on the triangulation principle. Cubic spline interpolation of the digitized camera information and the system design constraints are discussed in sections 3.4 and 3.5 respectively.

3.2 System Configuration

The proposed system configuration along with the coordinate frame assignment is shown in figure 3.1. The system consists of two CCD array cameras mounted on the robot arm and two laser line generators which project two laser line beams such that they form a cross-shaped pattern on the workpiece. The system is to be calibrated such that camera 1 and camera 2 are aligned along the x and y coordinate axis of the sensor respectively so as to subsequently simplify the detection of the laser 2 and laser 1 by camera 1 and camera 2 respectively as shown in figures 3.2 and 3.3.
Figure 3.1: Coordinate Frame of the Proposed System
Figure 3.2: Coordinate of Camera 1 with respect to Laser 2
3.3 Operational Principle

For geometrical simplicity, two cameras are mounted on the same stage as the sensor head. The camera axis has an offset with the sensor axis by 45° as shown in figure 3.1. As mentioned earlier, the proposed scheme uses a projection of two controlled light (laser beam) and two CCD cameras. These two laser beams form a
cross-shaped light pattern on the workpiece. The method is based on a triangulation, where distortions of the projected light pattern on the workpiece are used to calculate the range. As shown in figure 3.4, the distance $Z_a$ between a projection point and the reference plane is calculated by determining angle $\beta$ from digital images [2], where $\beta$ is the angle between the optical axis and the image axis. We define $X_{ca}$ and $Y_{ca}$ as the image position that is converted from Xpixel and Ypixel respectively, i.e the origin of $X_{ca}$ and $Y_{ca}$ is the center of the CCD array. Thus from figure 3.4,
\[
\beta = \tan^{-1}\left(\frac{X_{ca}}{\lambda}\right)
\]  \hspace{1cm} (3.1)

where

\[\lambda = \text{the focal length of the camera}\]

\[X_{ca} = \left[\frac{X_{pixel} - CCD_{maxpixel}}{2}\right] \text{CCDresolution}\]

\[Y_{ca} = \left[\frac{Y_{pixel} - CCD_{maxpixel}}{2}\right] \text{CCDresolution}\]

CCD_{maxpixel} = \text{number of pixels in CCD array in each axis}

CCDsize = \text{size of the square CCD}

CCDresolution = \frac{CCDsize}{CCD_{maxpixel}}

Thus, by arranging two cameras as shown in figures 3.2 and 3.3, camera 1 detects the \(Y_s\) aligned laser and camera 2 detects the \(X_s\) aligned laser. Using a geometry shown in figure 3.4, the depth of the projection point from camera 1 and camera 2 can be determined as

\[
Z_s = (d - \lambda \cos \alpha)\tan(\alpha - \beta) + \lambda \sin(\alpha)
\]  \hspace{1cm} (3.2)

where

\[\alpha = \text{the offset angle of the camera with respect to the sensor coordinate}\]

\[d = \text{the camera distance measured from the center of the CCD to the origin of the sensor frame}\]

The geometry shown in figure 3.4 can also be used to obtain \(X_s\) and \(Y_s\) position of the projection point for camera 1, i.e,

\[
\frac{\lambda}{q} = \cos \beta
\]
\[ q = \frac{\lambda}{\cos \beta} \]

\[ \frac{p}{\lambda \cos \beta} = \cos(\alpha - \beta) \]

\[ p = \frac{\lambda \cos(\alpha - \beta)}{\cos \beta} \]

hence,

\[ \frac{-Y_s}{Y_{ca}} = \frac{d - \lambda \cos \alpha}{\frac{\lambda \cos(\alpha - \beta)}{\cos \beta}} \]

thus,

\[
\begin{align*}
X_s &= 0 \\
Y_s &= \frac{Y_{ca}(d - \lambda \cos(\alpha)(\cos \beta))}{\lambda \cos(\alpha - \beta)}
\end{align*}
\]

for laser 2 \hspace{1cm} (3.3)

Similarly, the \(X_s\) and \(Y_s\) position of projection point for camera 2 becomes

\[
\begin{align*}
Y_s &= 0 \\
X_s &= \frac{Y_{ca}(d - \lambda \cos(\alpha))(\cos \beta)}{\lambda \cos(\alpha - \beta)}
\end{align*}
\]

for laser 1 \hspace{1cm} (3.4)

The \((X_s, Y_s, Z_s)\) location of projected point in the sensor frame are then transformed back to world by multiplying by the necessary transformation matrix.
3.4 Interpolation of Detected Surface Points \((X, Y, Z)\)

The discrete position information obtained in section 3.3 is transformed back to world coordinate and can be interpolated using several techniques, but most of these techniques are often plagued with disadvantages such as oscillatory tendencies, inability to produce smooth curve and/or requirements for gradient values at the data points. We then seek to interpolate the camera data points using 3rd order polynomial using cubic spline technique.

As shown in figure 3.5, for the \(i\)-th segment,

\[
X_i(t) = a_0 + a_1t + a_2t^2 + a_3t^3 \quad (0 \leq t \leq 1)
\]  

(3.5)

taking the first and second derivatives gives,
\[ X_i'(t) = a_1 + 2a_2 t + 3a_3 t^2 \]  \hspace{1cm} (3.6)

\[ X_i''(t) = 2a_2 + 6a_3 t \]  \hspace{1cm} (3.7)

\[ X_i(0) = X_i = a_0 \]  \hspace{1cm} (3.8)

\[ X_{i+1} = X_i(1) = a_0 + a_1 + a_2 + a_3 \]  \hspace{1cm} (3.9)

Also

\[ X_i' = X_i'(0) = a_1 \]  \hspace{1cm} (3.10)

\[ X_{i+1}' = X_i'(1) = a_1 + 2a_2 + 3a_3 \]  \hspace{1cm} (3.11)

\[ a_0 = X_i \]  \hspace{1cm} (3.12)

\[ a_1 = X_i' \]  \hspace{1cm} (3.13)

\[ a_2 + a_3 = X_{i+1} - X_i - X_i' \]  \hspace{1cm} (3.14)

\[ 2a_2 + 3a_3 = X_{i+1}' - X_i' \]  \hspace{1cm} (3.15)

Solving for \( a_3 \) from equations 3.13 and 3.14 gives,

\[ a_3 = X_{i+1}' - X_i' - 2X_{i+1} + 2X_i' + 2X_i \]  \hspace{1cm} (3.16)
\[ a_3 = X_{i+1}' + X_i' - 2(X_{i+1} - X_i) \] (3.17)

substituting for \( a_3 \) in equation 3.13, \( a_2 \) becomes,

\[ a_2 = X_{i+1} - X_i - X_i' - X_{i+1}' - X_i + 2X_{i+1} - 2X_i \] (3.18)

\[ a_2 = 3(X_{i+1} - X_i) - 2X_i' - X_{i+1}' \] (3.19)

substituting for \( a_2 \) and \( a_3 \) in equation 3.7 gives,

\[ X_i''(t) = 2[3(X_{i+1} - X_i) - 2X_i' - X_{i+1}'] + 6[X_{i+1} + X_i' - 2(X_{i+1} - X_i)]t \] (3.20)

For 2nd order continuity at the internal joints, we have

\[ X_i''(1) = X_{i+1}''(0) \] (3.21)

Expanding 3.20 gives,

\[ -2X_i - X_{i+1}' + 3X_{i+1}' + 3X_i' + 2X_{i+1} + X_i' = 3X_i' - 3X_{i+1} + 6X_{i+1} - 6X_i - 3X_{i+1} + 3X_i \] (3.22)

\[ X_i' + 4X_{i+1}' + X_{i+2}' = 3(X_{i+2} - X_i) \] (3.23)

for the first segment,

\[ X_{1}''(0) = 0 \Rightarrow 2a_2 = 0 \quad \text{i.e.} \ a_2 = 0 \] (3.24)
for the last segment,

\[ X_{n-1}''(1) = 2a_2 + 6a_3 = 0 \]  

(3.26)

hence

\[ 2x_n' + x_{n-1}' = 3(x_n - x_{n-1}) \]  

(3.27)

In summary,

\[
\begin{align*}
2X_1' + X_2' &= 3(X_2 - X_1) \\
X_1' + 4X_2' + X_3' &= 3(X_3 - X_1) \\
X_2' + 4X_3' + X_4' &= 3(X_4 - X_2) \\
&\vdots \\
X_{n-2}' + 4X_{n-1}' + X_n' &= 3(X_n - X_{n-2}) \\
2X_n' + X_{n-1}' &= 3(X_n - X_{n-1}) \\
\end{align*}
\]  

(3.28)

It should be noted that the above equations are only for the X component of the detected surface point. The steps can also be performed to derive equations for the Y and Z components.
Thus in matrix form

\[
\begin{bmatrix}
2 & 1 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\
1 & 4 & 1 & 0 & 0 & \ldots & 0 & 0 & 0 \\
0 & 1 & 4 & 1 & 0 & \ldots & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & \ldots & 1 & 4 & 1 \\
0 & 0 & 0 & 0 & 0 & \ldots & 0 & 1 & 2
\end{bmatrix}
\begin{bmatrix}
X'_1 \\
X'_2 \\
X'_3 \\
\vdots \\
X'_{n-1} \\
X'_n
\end{bmatrix}
= 
\begin{bmatrix}
3(X_2 - X_1) \\
3(X_3 - X_1) \\
3(X_4 - X_2) \\
\vdots \\
3(X_n - X_{n-2}) \\
3(X_n - X_{n-1})
\end{bmatrix}
\]  

(3.29)

Thus, the derivatives obtained from equation 3.28 are substituted appropriately to equations 3.10, 3.16, and 3.18 to compute the \( a_s \). The patch equation 3.5 is then subsequently evaluated. The above matrix could be reduced to any small number of data points.

Alternatively, we could interpolate the three points around the cross-laser beam point by a parabola. This could be achieved with a second order polynomial. After projecting the \( X,Y,Z \) position information to the \( X-Y \) plane, a particular convenient form will be

\[
Y = b_0 + b_1(X - X_0) + b_2(X - X_0)(X - X_1)
\]  

(3.30)

or

\[
Y = b_0 + b_1X - b_1X_0 + b_2X^2 + b_2X_0X_1 - b_2XX_0 - b_2XX_1
\]  

(3.31)

where


\[ b_i \] are the constant coefficients to be determined.

Collecting terms gives

\[ Y = c_0 + c_1 X + c_2 X^2 \] (3.32)

This is the familiar unique 2nd order polynomial joining three points.

Where

\[ c_0 = b_0 - b_1 X_0 + b_2 X_0 X_1 \]
\[ c_1 = b_1 - b_2 X_0 - b_2 X_1 \]
\[ c_2 = b_2 \]

Hence, a simple procedure utilizing divided difference can then be used to compute the coefficients \( b_i \). By setting \( X = X_0, X_1, X_2 \), we arrived at the followings

\[ b_0 = Y_0 \] (3.33)

\[ b_1 = \frac{Y_1 - Y_0}{X - X_0} \] (3.34)

and

\[ b_2 = \frac{\frac{Y_2 - Y_1}{X_2 - X_1} - \frac{Y_1 - Y_0}{X_1 - X_0}}{X_2 - X_0} \] (3.35)

### 3.4.1 Computation of Surface Normals

From figure 3.6, assume that

\[ \vec{r} = r(u, v) \]
defines the surface, the tangent vector to a parametric curves

\[ \vec{r} = r(u, v_0) \]

and

\[ \vec{r} = r(u_0, v) \]

where \( u_0 \) and \( v_0 \) are constants.

is a multiple of the vector \( \frac{\partial \vec{r}}{\partial u} \) and \( \frac{\partial \vec{r}}{\partial v} \) respectively. Hence the tangent plane at the intersection of these curves at \( r(u_0, v_0) \) contains these two tangent vectors [26], thus,
the normal to the surface at this point is a multiple of their vector product, i.e.

\[ \hat{n} = \pm \frac{\frac{\partial f}{\partial u} \times \frac{\partial f}{\partial v}}{\left| \frac{\partial f}{\partial u} \times \frac{\partial f}{\partial v} \right|} \quad (3.36) \]

where the derivatives are evaluated at \( u = u_0 \) and \( v = v_0 \), hence let \( \hat{n} = (1, m, n) \)

### 3.5 Design Constraints

The structural lighting technique could lead to specular reflection. This is a reflection from a mirror-like surface resulting in the no range measurement if the reflected light does not reach the camera. It can also lead to false (larger or smaller) measured range values if the reflected light subsequently is reflected by other surfaces before part of it reaches the camera [3]. Also, there could be occlusion problems resulting from sharp changes in depth. In this situation, not all the stripes projected will be visible to the camera.

Moreover, there is a tendency for multiple reflections of the incident beam if the target surfaces are either concave or semispecular [3], e.g. fillet joint of ground aluminum workpieces. This problem has been solved [3] by using a laser scanner, acting as a structured light, and a linear diode array, acting as a camera with a narrow rectangular field of view, within which true target points are visible and the most spurious ones are not. However, since most engineering components are usually of metallic luster, with a carefully controlled ambient lighting, the proposed system can yield very promising results.
Chapter 4

SIMULATION

4.1 Introduction

In this chapter, the computer graphics simulation is done for the proposed system to verify the tracking ability of the system. Section 4.2 explains the sensing environment using an object of known geometry for camera image data generation. The laser line beams position and orientation in space is presented in section 4.3, while the ray tracing technique to compute intersection points and the camera tracking are explained in sections 4.3 and 4.4 respectively. Simulation results are also presented.

4.2 The Sensing Environment

The simulation environment with the coordinate system used is shown in figure 4.3. Our simulated workspace is roughly 110 x 110(cm) on a table top, and a half sphere object is used for camera image data generation. Our object geometry is roughly 100cm in radius although the system could easily be scaled down to sensing objects of arbitrary sizes. The simulation uses two cameras with square CCD size of 5mm containing 500 CCD maximum pixels for a CCD resolution of 0.01mm. Two
laser line beams oriented in space with (O, A, T) angles are aligned with both cameras as shown in figures 3.2 and 3.3. Ray tracing technique is used to obtain the intersection of the ray with the chosen primitive (half sphere). The focal length of the lenses is taken as 10mm and the optical axis of the cameras are aligned 45° with the laser projection axis. Both the cameras and the laser projectors have an adjustable distance of about 200cm between them, which enables perspective transformation to be used for mapping 3-D intersected points on the primitive onto the image plane.

The inputs to the system include, the length of the laser line beam (laserlength), the (O, A, T) angles for the laser line beams, the initial starting offset of the end-effector, the camera offset relative to the projector axis $a$, the focal length $\lambda$, the size of the primitive, $R_s$, and the number of points on the laser line beam for discretisation, laserpts. As shown previously in figure 3.2 and figure 3.3, camera 1 is aligned along the x-axis of the sensor coordinate while camera 2 is aligned along the y-axis of the sensor coordinate. This enhances the easy detection and referencing of the two laser line beams by both cameras respectively. We choose to orient the laser line beams to follow a typical initial alignment of tool coordinate system (O, A, T euler angles) relative to the base coordinate [2] as shown in figure 4.1

![Figure 4.1: Initial Alignment of Tool Coordinate System](image)
4.3 Laser-Line Beams Position and Orientation

As explained earlier, the \((O, A, T)\) angles used by PUMA 560 and explained in figure 4.2 are used to orient the laser beam in space such that they coincide with a typical initial alignment of the tool coordinate system relative to the base frame, i.e.
\[
\begin{bmatrix}
      n_x & s_x & a_x \\
      n_y & s_y & a_y \\
      n_z & s_z & a_z \\
\end{bmatrix}
= \begin{bmatrix}
      0 & 1 & 0 \\
      0 & 0 & -1 \\
      -1 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
      R_{z_o} & R_{s_A} & R_{A,T} \\
\end{bmatrix}
\] (4.1)

\[
\begin{bmatrix}
      Co & -So & 0 \\
      So & Co & 0 \\
      0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
      0 & 1 & 0 \\
      0 & 0 & -1 \\
      -1 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
      Ca & 0 & Sa \\
      0 & 1 & 0 \\
      -Sa & 0 & Ca \\
\end{bmatrix}
\begin{bmatrix}
      Ct & -St & 0 \\
      St & Ct & 0 \\
      0 & 0 & 1 \\
\end{bmatrix}
\] (4.2)

\[
\begin{bmatrix}
      CoSt - SoSaCt & CoCt + SoSaSt & SoCa \\
      SoSt + CoSaCt & SoCt - CoSaSt & -CaCo \\
      -CaCt & CaSt & -Sa \\
\end{bmatrix}
\] (4.3)

where

\[
Co = \cos(O) \\
So = \sin(O) \\
Sa = \sin(A) \\
Ca = \cos(A) \\
Ct = \cos(T), \text{ etc.}
\]

If the initial cross-beam location relative to world is \((PX_s, PY_s, PZ_s, 1)^T\), then the transformation matrix is given by
where

\[ T_L^W = \begin{bmatrix} CoSt - SoSaCt & CoCt + SoSaSt & SoCa & PX_s \\ SoSt + CoSaCt & SoCt - CoSaSt & -CaCo & PY_s \\ -CaCt & CaSt & -Sa & PZ_s \\ 0 & 0 & 0 & 1 \end{bmatrix} \] (4.4)

4.4 Ray Tracing to Get Intersection Points

Let \( np = n_{\text{min}}, \ldots, 0, \ldots, n_{\text{max}} \)

where

\[ n_{\text{min}} = -\frac{\text{laserpts} - 1}{2} \]

and

\[ n_{\text{max}} = -n_{\text{min}} \]

The step increment in discretizing the beam is given as

\[ \Delta t = \frac{\text{laser length}}{\text{laserpts} - 1} \] (4.5)

The laser position is given by
\[
\begin{align*}
X_p &= PX_s + l_y(\Delta t \ast np) \\
Y_p &= PY_s + m_y(\Delta t \ast np) \\
Z_p &= PZ_s + n_y(\Delta t \ast np)
\end{align*}
\]

where \(l_y, m_y\) and \(n_y\) are the laser direction cosines obtained from

\[
\begin{align*}
l_y &= T_L^W[1,3] \\
m_y &= T_L^W[2,3] \\
n_y &= T_L^W[3,3]
\end{align*}
\]

We use a known geometry of half sphere to generate camera image data on CCD array. A ray of light is imagined to be passing through the discretized points outlined above towards and intersecting the defined primitive [27] as shown in figure 4.3. Define the ray as a straight line such that

\[
R_{\text{origin}} = R_0 = (X_p, Y_p, Z_p)
\]

\[
R_{\text{direction}} = R_d = (X_{\text{dir}}, Y_{\text{dir}}, Z_{\text{dir}})
\]

where

\[
X_{\text{dir}}^2 + Y_{\text{dir}}^2 + Z_{\text{dir}}^2 = 1 \quad \text{(i.e normalized)}
\]

alternatively, let \(R_d = (l, m, n)\). Hence, the set of points for consideration lies on the
Figure 4.3: Schematic diagram of the Simulated System using Ray Tracing Technique

The half sphere is located at \((X_c, Y_c, Z_c)\) with a radius of \(R_s\), the surface of the sphere is the sets of points

\[
(X_0 + lt - X_c)^2 + (Y_0 + mt - Y_c)^2 + (Z_0 + nt - Z_c)^2 = R_s^2
\]  

(4.12)
\[(X_0 - Xc) + lt]^2 + [(Y_0 - Yc) + mt]^2 + [(Z_0 - Zc) + nt]^2 = R_s^2 \quad (4.13)\]

Let

\[
\begin{align*}
X_0 - Xc &= X' \\
Y_0 - Yc &= Y' \\
Z_0 - Zc &= Z'
\end{align*}
\]

hence

\[
X'^2 + 2X'lt + l^2t^2 + Y'^2 + 2Y'mt + m^2t^2 + Z'^2 + 2Z'nt + n^2t^2 = R_s^2 \quad (4.14)
\]

\[
t^2 + 2t(X'l + Y'm + Z'n) + (X'^2 + Y'^2 + Z'^2 - R_s^2) = 0 \quad (4.15)
\]

Let

\[
\begin{align*}
\alpha &= (X'l + Y'm + Z'n) \\
\beta &= (X'^2 + Y'^2 + Z'^2 - R_s^2)
\end{align*}
\]

and solving for \(t\) yields

\[
t_{0,1} = -\alpha \pm \sqrt{\alpha^2 - \beta} \quad (4.16)
\]

Let

\[
\text{argument} = \alpha^2 - \beta
\]
argument > 0 corresponds to the intersection of the ray with the sphere or even the lower half of the sphere and argument < 0 corresponds to the intersection with the table top which is considered as an intersection as shown in figure 4.4

![Diagram showing laser beam and intersections](image)

Figure 4.4: Intersection of Ray with Table Top, Top-Half and Lower-Half of the Sphere using Ray Tracing Technique

On the X-Y plane,

\[ Z = 0 \quad (4.17) \]

\[ Z_0 + nt = 0 \]

hence,

\[ t = \frac{Z_0}{n} \quad (4.18) \]
and

\[ X = X_0 + l\left[-\frac{Z_0}{n}\right] \tag{4.19} \]

\[ Y = Y_0 + m\left[-\frac{Z_0}{n}\right] \tag{4.20} \]

Once \( t \) is calculated, we substitute it back to equation (4.17) to determine the intersecting point on the primitive

\[ \text{projectpoint} = (X_i, Y_i, Z_i) = (X_p + l \ast t, Y_p + m \ast t, Z_p + n \ast t) \tag{4.21} \]

A unit vector normal to the sphere on the intersected point is then calculated using

\[ \text{Actualnormal} = \left(\frac{X_i - X_c}{R_s}, \frac{Y_i - Y_c}{R_s}, \frac{Z_i - Z_c}{R_s}\right) \tag{4.22} \]

The virtual approximation to the manner in which image is formed by viewing a three-dimensional world is given using perspective transformation. A typical geometry for the transformation [2] is as shown in figure 4.5

The objective is to obtain the coordinate \((x, y)\) of the projection of the world points \((X, Y, Z)\) onto the image plane. Using similar triangles on figure 4.5 [19] gives

\[ \frac{x}{\lambda} = \frac{X}{\lambda - Z} \tag{4.23} \]

and

\[ \frac{y}{\lambda} = \frac{Y}{\lambda - Z} \tag{4.24} \]

hence
At this point, we are only dealing with the x,y camera data totally unconcerned about the primitive type and geometry. These image points are processed as discussed previously to compute the range and then converted back to world coordinates where cubic spline is used to interpolate the points for surface orientation measurement.

4.5 Camera Tracking

Camera tracking of a surface of unknown geometry is accomplished by considering the constant and variable tool orientation cases. In both cases, the translation of the
tool is based on predicting the surface orientation of the next surface point based on current surface point orientation. The possibility of having degeneracy, i.e. singularity problem involving the geometry of the manipulator and also the angles of the various joints that could endanger the trackability of the system was not considered. A typical tracking workspace is as shown in figure 4.6, where, the half sphere object is placed on a grided table top.

![Diagram](image)

Figure 4.6: Top View of the Workspace with Tracking Grid Path

### 4.5.1 Constant Orientation Tracking

In this case, the tool orientation is held constant in the Z-direction during the entire tracking operation. Movement is along the X and Y directions of the grid. Input to the tracking process includes, the initial tool position and an offset. At the initial tool position, the surface normal (l, m, n) and position (x, y, z) are evaluated. The spatial location of the surface at this point is approximated by a plane of
where \( d \) is computed using,

\[
d = -(lx + my + nz)
\]  

(4.28)

The estimated value of \( z \) at the next grid point can be obtained using the plane equation obtained in 4.27, i.e.

\[
z = \frac{(lx + my + d)}{n}
\]  

(4.29)

This value of \( z \) is added to the initial tool offset and the next spatial location and orientation computed by moving along this plane on a given grid.

### 4.5.2 Variable Orientation Tracking

Camera tracking in this case follows the same technique used for constant tool orientation case. However as figures 4.7 and 4.8 show, the ultimate objective is to align the present tool axis \( Z_a \) with the predicted tool axis \( Z_b \) before translating to the predicted location on the surface. Define 'old' tool frame as \( A \) and 'new' tool frame as \( B \).
as B. The orientation of frame B relative to frame A is given as

\[
\begin{bmatrix}
  l^a_b \\
  m^a_b \\
  n^a_b \\
  0
\end{bmatrix} = \left[ T^W_A \right]^{-1} \begin{bmatrix}
  l_b \\
  m_b \\
  n_b \\
  0
\end{bmatrix}
\] (4.30)

where

\[ T^W_A = \text{the transformation matrix of current tool frame relative to the world} \]

\[
\begin{aligned}
\{A\} & \quad \{B\} \\
X_a & \quad X_b \\
Y_a & \quad Y_b \\
Z_a & \quad Z_b \\
\{W\} & \quad \{\text{world frame}\}
\end{aligned}
\]

\[
[ T^W_A ]^{-1} \begin{bmatrix}
  l_b \\
  m_b \\
  n_b \\
  0
\end{bmatrix} = \begin{bmatrix}
  l^a_b \\
  m^a_b \\
  n^a_b \\
  0
\end{bmatrix}
\]

Figure 4.7: Coordinate System for Variable Orientation Tracking

This orientation is then used to calculate \( \beta_x \) and \( \beta_y \) angles for coordinate rotation, i.e.

\[
\beta_y = \tan^{-1} \left[ \frac{l^a_b}{\sqrt{l^a_b^2 + n^a_b^2}} \right]
\] (4.31)
Figure 4.8: Rotation Procedures for Aligning Tool Axis to the Desired Orientation

and

\[ \beta_z = tan^{-1}\left(\frac{m_b^a}{\sqrt{l_b^a + m_b^a}}\right) \] (4.32)

hence, rotation is first about the y-axis followed by rotation about the x-axis as shown in the figure 4.5.

\[ New T_L^W = T_L^W * Rot(y, \beta_y) * Rot(x, -\beta_x) \] (4.33)

It was observed correctly that the third column of

\[ New T_L^W = -(the \ predicted \ orientation) \]
The new sensorhead location being given as

\[ PX_s = PX_s + \text{offset} * l \]  \hspace{1cm} (4.34)

\[ PY_s = PY_s + \text{offset} * m \]  \hspace{1cm} (4.35)

\[ PZ_s = PZ_s + \text{offset} * n \]  \hspace{1cm} (4.36)

Figure 4.9 shows the tool movement and the corresponding orientation over a surface of arbitrary shape placed on a grid. As the result indicates, the variable tool orientation has enabled the easy computation of the orientation of the surface points especially at points where there is sharp changes in depth.

### 4.5.3 Simulation Results

The simulation program coded in Turbo Pascal is presented in the appendix. As stated earlier, our system could be easily scaled down to objects of any arbitrary size. Much of the results presented here has been for a half sphere of radius 100cm, 40cm and 20cm placed on 110 X 110(cm), 60 X 60(cm) and 30 X 30(cm) grid table top respectively. Typical simulation windows status for both cases are shown in figures 4.10 and 4.11.

The surface normals computed during tracking are Overlain as shown in figures 4.12 and 4.13 respectively for both cases. A one-pass trajectory using constant tool orientation over a sphere of radius 100cm with an initial offset of 120cm is shown in figure 4.14, while figure 4.15 shows a 2-D plot of an entire tool trajectory over a sphere of radius 40cm placed on a 60 X 60(cm) grid table top.

Also, figures 4.16 and 4.17 shows the three dimensional plots of the surface contour
Figure 4.9: Tool Movement using Variable Orientation Algorithm

generated, the step size along the grid in either the x-axis or y-axis could be varied depending on applications.

Figure 4.18 shows the tool trajectory obtained for the variable tool orientation case, observe that the sensor head now takes on different positions and orientations, the initial offset of 200cm is quickly attained as soon as the tool leaves the sphere.
Figure 4.10: Status of Simulation Graphics Windows for Constant Orientation Tracking Showing Tool Movements and Locations
Figure 4.11: Status of Simulation Graphics Windows for Variable Orientation Tracking Showing Tool Movements and Locations
Figure 4.12: Overlaid Surface Normals for the Constant Orientation Case
Figure 4.13: Overlain Surface Normals for the Variable Orientation Case
Figure 4.14: One-Pass Trajectory Over Sphere of Radius 100cm, Step Size = 2, for Constant Tool Orientation, Grid Size is 110 X 110(cm)
Figure 4.15: Two Dimensional Plot of An Entire Trajectory Over Sphere of Radius 40cm, Step Size = 1, for Constant Tool Orientation. Grid Size is 60 X 60(cm)
Figure 4.16: Surface Contour Generated for a Sphere of Radius 20cm Using Constant Tool Orientation, Step Size = 1, Grid Size is 30 X 30(cm)
Figure 4.17: Surface Contour Generated for a Sphere of Radius 40 cm Using Constant Tool Orientation, Step Size = 1. Grid Size is 60 X 60 cm
Figure 4.18: Tool Trajectory for the Variable Tool Orientation Case, Radius = 100cm, Step Size = 2 and Initial Offset = 200cm. Notice that the tool quickly attains the initial offset of 200cm as soon as it leaves the sphere.
Chapter 5

IMPLEMENTATION

5.1 Overview

The proposed hardware/interfacing is as shown in figure 5.1. Two laser diode drivers generate two laser beams which pass through two parallel projectors to form a cross-shaped pattern on the workpiece. The camera/lighting source is to be calibrated so as to establish the relationship between 3-dimensional world point and the corresponding 2-dimensional image points. The system is designed to be mounted on a Unimate Puma 560 robot arm. The cross-shaped laser beam is projected on an object surface and the cameras pick up their respective images for processing using the Imager-AT image processing board on an IBM 386 microcomputer. Grey to binary conversion of the image and the image thinning are performed to obtain the center point of the cross pattern image formed on CCD array.

The complete implementation of the proposed system is however reserved for future research. In this chapter, initial camera/lighting source calibration procedures and some necessary image processing procedure are discussed.
5.2 Camera/Lighting Source Calibration

Camera calibration involves establishing correspondence between points in the physical world and those in the camera's image. It is only when such correspondence has been established that the camera's 2-dimensional raster coordinates could be translated into real world locations for objects within the camera's threshold. The technique described in [28] has been fully relied upon in doing this calibration. The calibration is to be done every time the system is either dismantled or adjusted. In attempting to do the calibration, rather than physically measuring the camera/lighting source parameters to a lesser degree of accuracy, the measured world data points will be fitted to the parameterized expressions to be developed. An optimization procedure will then be used to obtain optimal set of parameters giving the closest fit.
5.2.1 Coordinate Systems

A simplistic image forming geometry is shown in figure 5.2, where an ideal lens converges all the rays from the object through it optical center onto the image plane.

![Diagram of Image Formation Using an Ideal Thin Lens](image)

Figure 5.2: Image Formation Using an Ideal Thin Lens

Image Coordinates

We define \((u, v)\) as the distance of the image point from the image center, where \(u\) and \(v\) are parallel to the \(x\) and \(y\) external axes of the camera respectively.

Pixel Coordinates

Let integer pairs \((i, j)\) identify the discrete pixel taken from the center of raster as their column and row numbers respectively. Define \(m\) as the pixel spacing (distance from the center of one pixel to the center of the adjacent pixel) in the \(u\)-direction and \(n\) as the spacing in the \(v\)-direction. Thus, the image coordinates \((u, v)\) is related to the pixel coordinates by \(u = m \times i\) and \(v = n \times j\).
Camera/Plane-of-Light Coordinates

Camera/Plane-of-light geometry is shown in figure 5.3. Two real world (external) coordinate systems for each camera/light source configuration is used. The z-axis lies along the camera axis, whereas x-axis is parallel to the pixel row and y-axis is perpendicular to both. The z'-axis of the plane-of-light is located within the plane of light and the origin of the lighting source is selected arbitrarily to lie at the face of one of the light projectors. The z'-axis is also made to be within the plane-of-light while y'-axis is perpendicular to it.

![Diagram of Camera/Plane-of-light Geometry](image)

Figure 5.3: Camera/Plane-of-light Geometry

5.2.2 Range Measurement: z'

As described in [28], the objective here is to compute $r_1$ representing the unknown $z'$ of the object as shown on figure 5.4, which is the side view of camera/light source arrangement.
Figure 5.4: Side View of Camera/Plane-of-Light Geometry for An Illuminated Field Point W in the Plane $X' = 0$

As shown in figures 5.4 and 5.5,

$$\tan \beta = \frac{v}{f} \quad (5.1)$$

and

$$\frac{r_1}{\sin \beta} = \frac{r_4}{\sin \alpha} \quad (5.2)$$

The aim here is to compute the range given by $r_1 + A$, i.e. the object's $z'$ coordinate.

Since

$$\alpha = \theta - \beta$$
then noting that $u = m \ast i$ and $v = n \ast j$ then

$$r_1 = \frac{r_4 \sin \beta}{\sin \theta \cos \beta - \cos \theta \sin \beta} \quad (5.3)$$

noting that $u = m \ast i$ and $v = n \ast j$ then

$$r_1 = \frac{C_1 j}{1 - C_2 j} \quad (5.4)$$

where

$$C_1 = r_3 \left(\frac{n}{f}\right) (\csc \theta)^2 \quad (5.5)$$
and

\[ C_2 = \left( \frac{n}{f} \right) \cot \theta \quad (5.6) \]

From equation 5.4, \( z' \) becomes

\[ z' = \frac{C_1 j}{1 - C_2 j} + A \quad (5.7) \]

### 5.2.3 \( x' \) Coordinate

Figure 5.6 shows the world point \( w(x', y', z') \) for which \( x' \) need not be zero unlike the case above. In this case, \( y' = 0 \), since point is within the plane of light, hence \( z' \) derived in equation 5.7 is still applicable. From figure 5.6 and using similar triangles,

\[ x' = \left( \frac{r_2}{f_1} \right) u \quad (5.8) \]

where

\[ f_1 = f^2 + v^2 \quad (5.9) \]

hence

\[ x' = \frac{C_3 i \sqrt{C_4 + [z' - F]^2}}{\sqrt{1 + [\frac{v}{f}]^2}} \quad (5.10) \]
Figure 5.6: Top View of Camera/Plane-of-Light geometry for An arbitrary Field Point $W$, $GW$ is the projection of $W$ onto $Z'$ axis

where

$$C_3 = \frac{m}{f} \quad (5.11)$$

$$C_4 = r_3^2 \quad (5.12)$$

and $F$ is the $z'$ coordinate of the optical center of the lens.

In making the above calibration, we have made use of several assumptions including

- No distortion of the lens, i.e an ideal lens model is assumed. This implies that there is no nodal separation, thus rays incident and exit only on one optical center.
• The pixel raster is perfectly aligned with the external coordinate system.

• No raster translation.

• No Camera/light source misalignment.

5.2.4 Calibration

The plane-of-light is selected as origin from which other positions are referenced. It is also the origin from which several field points together with their corresponding image points are carefully measured. Each of the field points was captured by the camera into the buffer, thresholded and the pixel coordinate located by moving a cross-hair across the image plane. The parameters interested in their optimal values include

• \( r_3 \), i.e. the distance between the lens and the plane-of-light (origin)

• \( \theta \), the angle between the optical axis and the plane of light

• \( m \), the pixel spacing in the u-direction (column spacings in \( \text{mm/pixel} \))

• \( n \), the pixel spacing in the v-direction (row spacings in \( \text{mm/pixel} \))

Where \( f \) is the focal length of the lens written on the lens mount. We seek for five independent parameters \( (\frac{m}{f}, \frac{n}{f}, r_3, F, \theta) \). As discussed in [28], inverse transformation from \((x', y', z')\) to the \((u, v)\) coordinates yield the best result, hence

\[
\begin{align*}
  u &= \frac{x' \sqrt{1 + \left(\frac{mn}{f^2}\right)^2}}{C_3 \sqrt{C_4 + [z' - F]^2}} \\
  v &= \frac{z' - A}{C_1 + C_2(z' - A)}
\end{align*}
\]
Thus, for each field point \((x', y', z')\) is computed image location \((u_{\text{comp}}, v_{\text{comp}})\) given by equations 5.13 and 5.14 respectively and then compared with actual \((u, v)\). An optimization procedure is then used to obtain the optimal parameters that minimizes:

\[
obj = \sum_{i=1}^{i=\text{data points}} [(u_{\text{comp}} - u)^2 + (v_{\text{comp}} - v)^2] \quad (5.15)
\]

where the sum is taken over all the 70 data points measured. It should be noted that for the best result especially for robotics applications, considerations should be given to all the assumptions as attempted by [28].

### 5.3 Image Processing

A computer program using microsoft C that calls several image processing routines from the imager-AT library is presented in the appendix. Figures 5.7 and 5.8 shows the cross-laser before and after thinning. An imager-AT processing board [29] was used to process the image captured by the camera into the frame buffer. The cameras were connected to A/D channels 1 and 2 respectively. An optimal threshold was selected using the histogramming approach, and a grey to binary scale image conversion carried out before calling the thinning routine.

However, due to the limited statistical computation capability of the our image processing board, pattern matching with the thinned image in order to locate points surrounding and including the cross-points could not be implemented.
Figure 5.7: Cross-Laser Before Thinning

Figure 5.8: Cross-Laser After Thinning
Chapter 6

CONCLUSION

6.1 Conclusion

This thesis presents a simulation algorithm for surface-tracking an object of unknown geometry using the principle of active lighting via triangulation. The advantages of the proposed cross-shaped technique over other structured beam techniques such as dots, lines, etc can be summarized as follows:

- It enhances the easy detection of the surface normal to a three-dimensional surface at any given point by computing two slope vectors in the x and y directions.

- It leads to an increase in the data acquisition rate on any particular point of interest.

- The intersection point of the cross-shaped beam can provide an easy reference point in range measurement and other video tracking purposes.

- The four wings of the cross-shaped light beam provide an easy integration of parametric curve and surface representation with measured data.
Moreover, the technique proposed does not require any computationally intensive geometric modeling to infer orientation (surface normal). Thus, this presents a real opportunity for real time guidance of a robot tool especially in cases where specific orientation between the robot end-effector and the workpiece is critical. Results obtained from both constant and variable tool orientation tracking are very encouraging and should give an impetus that will lead to the much needed experimental verification. The experimental construction is inexpensive and we expect our approach to be very time efficient especially in the field of robotics, factory automation and geometric modeling.

6.2 Recommendation

For future laboratory implementation of the scheme, it is desirable to

- Acquire an imager-AT/NP image processing board with high statistical processing capability to be used for feature extraction and pattern matching.

- The cameras/lighting source should be recalibrated to account for all the assumptions outlined here as well as in [11].

- Implementation of the proposed system for PUMA 560 to verify the proposed constant and variable orientation tracking algorithms.
Bibliography


APPENDIX:

Computer graphics simulation program in Turbo Pascal.

Image processing programs in MicroSoft C.
This program simulates the video camera image of cross-shaped laser beam illuminated on the half sphere object on the flat table and overlays the normals.

PROGRAM CrossLaserSimulation;
uses Crt,Graph,Dos;
type Matrix4x4 = array [1..4,1..4] of real;
Matrix4x1 = array [1..4] of real;
Matrix4xN = array [1..4,-10..10] of real;
\{number of col depending on LaserPts\}
\{N=(LaserPts-1) div 2\}
MatrixMx3 = array [-1..1,1..3] of real;
MatrixNx4 = array [-10..10,1..4] of real;
MatrixMx4 = array [-1..1,1..4] of real;
\{arrays for curve fitting\}
Matrix4x3 = array [1..4,1..3] of real;
Matrix3x4 = array [1..3,1..4] of real;
Matrix4x3x3 = array [1..4,1..3,1..3] of real;
Matrix3x3 = array [1..3,1..3] of real;
Matrix3x1 = array [1..3] of real;
MatrixNNx110 = array [1..1,1..110] of integer;
\{\}
MatrixSTEPx4 := array[1..10,1..4] of real;\}
const
dum1 : Matrix3x3 = ((2,1,0),
(1,4,1),
(0,1,2));
dum2 : Matrix3x3 = ((1,0,0),
(0,1,0),
(0,0,1));
at : Matrix4x4 = ((1,0,0,0),
(0,0,1,0),
(0,1,0,0),
(0,0,0,0));
att : Matrix4x4 = ((0,0,0,0),
(1,0,0,0),
(0,1,0,0),
(0,0,0,0));
de : Matrix3x3 = ((0,0,0),
(0,0,0),
(0,0,0));

label A;
var \{homogeneous coord transformation\}
worldTsphere, \{transform of laser head coord w.r.t world coord\}
worldTlaser,newworldTlaser,
laserTcamera1,
laserTcamera2,
worldTcamera1,
worldTcamera2,
camera1Tworld,
camera2Tworld,
cm1, cm2 : Matrix4x4;
{ position vectors }
XYZworld, Spn2,
XYZsensor : Matrix4x1; { detected XYZ proj point of
laser beam }

XYZlaser1,
XYZlaser2 : Matrix4xN; { storage of XYZsensor }
XYZlaser11,
XYZlaser22 : MatrixNx4;
XYZlaser1W,
XYZlaser2W : Matrix4xN; { storage of XYZworld }
MeasuredXYZcrsPts, Me: Matrix4x1; { XYZworld of
cross-intersection point }
ActualXYZcrsPts : Matrix4x1; { Actual proj position of cross-
intersection pt }
xsaold, ysaold, xsa, ysa : MatrixNNx110;
{ sensor parameters }
LaserLen, hh, rr, bb, cc : real; { length of laser
line projector }
CameraF : real; { focal length of camera 1 & 2 }
CameraAngle : real; { angle between opt axis and
Xs/Ys axis of laser head coordinate }
CameraDist : real; { dist from laser head to CCD }
CCDSize : real; { size of square CCD }
CCDmaxPixel : integer; { number of pixel in CCD array }
CCDresolution : real; { CCDsize/CCDmaxPixel }
{ simulation parameters }
OangS, AangS, TangS: real; { O, A, T angle of sensor system }
PxS, Pz, PyS, PzS : real; { position of sensor system }
xc, yc, zc, radius : real; { location of sphere and radius }
Laserpts : integer; { number of laser dots for simulation }
deltat : real; { step for discretization of
laser line }
xp, yp, zp : real; { position coord used in Ray tracing }
xdir, ydir, zdir : real; { dir cosine used in Ray tracing }
imageXpixel,
imageYpixel : integer; { actual image position in
cameras 1 & 2 after
perspective transformation

projpt,
Comp1,Comp2  : Matrix4x1;  \{ projected laser point
w.r.t world coord \}
XorY          : integer;  \{ 1 or 2, designate two line laser \}
nmax, nmin   : integer;  \{ index for discrete laser
point iteration \}
funcKey       : boolean;
delTrans, delOAT : real;  \{ step for translation and
rotation in main simulation window \}
\{ for turbo graphics \}
XWmin,YWmin,XWmax,YWmax :real;  \{ size of main view window \}
scaleX, scaleY,sX,sY     :real;  \{ scale factor FROM World
TO View Port \}
viewPhi, viewTheta      :real;  \{ for 3-D orthographic projection \}
SCF                      :real;  \{ Screen adjustment Correction Factor \}
xz, ye, ze              :real;  \{ transformed position to eye
coord system \}
Xscr,Yscr,xscrols,yscrols :integer;  \{ screen coordinate \}
XVmin,YVmin,XVmax,YVmax,
YVmx,YVmn               :integer;  \{ viewport size for each window \}
GraphDriver : integer;  \{ The Graphics device driver \}
GraphMode   : integer;  \{ The Graphics mode value \}
MaxX, MaxY  : word;  \{ The maximum resolution of the screen \}
ErrorCode   : integer;  \{ Reports any graphics errors \}
MaxColor    : word;  \{ The maximum color value available \}
OldExitProc : Pointer;  \{ Saves exit procedure address \}
\{ for spline fitting of digitized images \}
aa, ad, dd   : Matrix3x3;
ap, ap1, ap2 : Matrix3x4;
pp, pp1, pp2 : MatrixMx4;
ac         : Matrix4x3x3;
pt, CompNorm, Spn1  : Matrix3x1;
am, ai       : Matrix3x3;
theta, gamma, fac1, fac2, delt, px, py, pxx, pyy : real;
size, segment, tt,
index1, index2, kk, mm : integer;
\{ Surface Normal point parameters \}
c1, c2, c3, c4, rad, s  : real;
ActualNormal              : Matrix3x1;
Slope1, Slope2, Vec, AProd, CompNormVec, grad : Matrix3x1;
\{ TrackSensor Head parameters \}
delX, delY, offset, xm, ym, nextp1, d, nextp2, nextp3 : real;
CNorm, PredictNorm, NewLoc : Matrix3x1;
FileVar, FF : Text;

*********** get simulation parameters ***************
procedure SimulParameter;
begin
write ('Enter view angle Phi, theta in degree for main window');
writeln (' - Phi: angle to +Z axis');
writeln (' - Th: angle to +X axis');
readln(viewPhi, viewTheta);
viewPhi := viewPhi * pi / 180.;
viewTheta := viewTheta * pi / 180.;
delOAT := delOAT * pi / 180;
{
} cameral and 2 parameters
CameraF := 15.;
CameraAngle := 0.785; { 45 degrees }
CameraDist := 200.; { 200 }
CCDSize := 5.;
CCDmaxPixel := 500; { 500x500 }
CCDresolution := CCDSize / CCDmaxPixel; { size of individual pixel }
{ define sphere object }
xc := 0.; yc := 0.; zc := 0.; { zc should be always zero in this program }
radius := 100;
{ Initial location of sensor system w.r.t. sphere }
OangS := 0.; { Y / |Z sensor (or laser head) }
AangS := 90.; { Z }
TangS := 0.; { X /

} Y world
PxS := -110.; { initially laser head is at the north of sphere }
PyS := 0.; { - with optical axis of camera intersect with }
{ laser beam at the object sphere surface }
PzS := 200.;
{ size and discretization of cross laser beam }
LaserLen := 20.;
Laserpts := 21; { if you change this value, dim of Matrix4xN should be changed accordingly }
{ window size for main port }
XWmin := -300.; { make (XWmax - XWmin) = (YWmax - YWmin) }
YWmin := -200.; { for non-distorted image in the }
XWmax := 900.; { main simulation window }
YWmax := 1000.;
{spline parameters}
size := 3;
theta := viewTheta;
gamma := viewPhi;
{Define Step size for discretisation}
delX := 2.;
delY := 2.;
offset := 200.;
s := 10.;
xm := 2*radius/delX;
ym := 2*radius/delY;
end;{ SimulParameter }

{**** forward procedure Simulation ****************************************} procedure Simulation(color:word);forward;
{*********** control simulation program ************************************}
procedure WaitToGo;
{ Wait for the user to abort the program or continue with different parameters }
const
Esc = #27;
UpArrow = #72; {+Z}
var
ch:char;
i,j : integer;
begin
funcKey:=false;
ch := ReadKey;
if ch ≠ #0 then funcKey:=false
else begin
   funcKey:=true;
   ch:=ReadKey;
end;
Simulation(0);{erase previous plot}
case ch of
   UpArrow: PzS := offset; {+delTrans}
   Esc: Exit;
else
   funcKey:=false;
end;
end; { WaitToGo }
procedure Wait;
{ Wait for the user to abort the program or continue with different parameters }
const
Esc = #27;
UpArrow = #72; \{+Z\}

var
  ch:char;
  i,j : integer;
begin
  funcKey:=false;
  ch := ReadKey;
  if ch ≠ #0 then funcKey:=false
  else begin
    funcKey:=true;
    ch:=ReadKey;
  end;
  case ch of
    UpArrow: PzS := offset; \{+delTrans;\}
    Esc:   Halt(1);
    else
      funcKey:=false;
  end;
end; \{ Wait\}

{************************** Miscellaneous Procedures **************************}

procedure PrintMatrix (p:matrix4x4; RowMax, ColMax:integer);
var
  I, J : integer;
begin
  for I:= 1 to RowMax do
    begin
      for J := 1 to ColMax do
        begin
          write (p[I,J]:10:5);
        end;
      writeln;
    end;
end; \{ PrintMatrix \}

procedure RotX (Ang: real; var T: Matrix4x4);
begin
  T[1,1]:=1.; T[1,2]:=0.; T[1,3]:=0.;T[1,4]:=0.;
  T[2,1]:=0.; T[2,2]:=cos(Ang); T[2,3]:=-sin(Ang);T[2,4]:=0.;
  T[3,1]:=0.; T[3,2]:=sin(Ang); T[3,3]:=cos(Ang),T[3,4]:=0.;
  T[4,1]:=0.; T[4,2]:=0.; T[4,3]:=0.;T[4,4]:=1.;
end; \{ RotX \}

procedure RotY (Ang: Real; var T: Matrix4x4);
begin
T[1,1]:=cos(Ang); T[1,2]:=0.; T[1,3]:=sin(Ang); T[1,4]:=0.;
T[2,1]:=-sin(Ang); T[2,2]:=0.; T[2,3]:=cos(Ang); T[2,4]:=0.;
T[3,1]:=0.; T[3,2]:=0.; T[3,3]:=1.; T[3,4]:=0.;
T[4,1]:=0.; T[4,2]:=0.; T[4,3]:=0.; T[4,4]:=1.;
end; { RotY }

procedure RotZ (Ang: Real; var T: Matrix4x4);
begin
T[1,1]:=cos(Ang); T[1,2]:=-sin(Ang); T[1,3]:=0.; T[1,4]:=0.;
T[2,1]:=sin(Ang); T[2,2]:=cos(Ang); T[2,3]:=0.; T[2,4]:=0.;
T[3,1]:=0.; T[3,2]:=0.; T[3,3]:=1.; T[3,4]:=0.;
T[4,1]:=0.; T[4,2]:=0.; T[4,3]:=0.; T[4,4]:=1.;
end; { RotZ }

procedure Trans (x,y,z: Real; var T: Matrix4x4);
begin
T[1,1]:=1.; T[1,2]:=0.; T[1,3]:=0.; T[1,4]:=x;
T[2,1]:=0.; T[2,2]:=1.; T[2,3]:=0.; T[2,4]:=y;
T[3,1]:=0.; T[3,2]:=0.; T[3,3]:=1.; T[3,4]:=z;
T[4,1]:=0.; T[4,2]:=0.; T[4,3]:=0.; T[4,4]:=1.;
end; { Trans }

procedure OATzero (var T: Matrix4x4);
{ transform matrix for O=A=T=0 deg }
begin
T[1,1]:=0.; T[1,2]:=1.; T[1,3]:=0.; T[1,4]:=0.;
T[2,1]:=0.; T[2,2]:=0.; T[2,3]:=-1.; T[2,4]:=0.;
T[3,1]:=-1.; T[3,2]:=0.; T[3,3]:=0.; T[3,4]:=0.;
T[4,1]:=0.; T[4,2]:=0.; T[4,3]:=0.; T[4,4]:=1.;
end; { OATzero }

procedure MatMpy (var A,B: Matrix4x4; var C: Matrix3x4 ; M,N,L: Integer);
var
I,J,K : integer;
begin
for I:=1 to M do
  for J:=1 to L do
  begin
    C[I,J] := 0.;
    for K:=1 to N do
  end;
end; { MatMpy }

procedure MatMpy1 (var A:Matrix3x4; B:Matrix4x4; var C:Matrix3x4 ;
  M,N,L: Integer);
var
I,J,K : integer;
begin
  for I:=1 to M do
    for J:=1 to L do begin
      C [I,J] := 0.;
      for K:=1 to N do
    end;
end; { MatMpy1 }

procedure MatMpy4x1 (A: Matrix4x4; B:Matrix4x1; var C:Matrix4x1);
var I,K :integer;
begin
  for I:=1 to 4 do
    begin
      C [I] := 0.;
      for K:=1 to 4 do
    end;
end; { MatMpy4x1 }

procedure Orthoinverse (var A, AI: Matrix4x4);
{ Inverse of transformation matrix A }
var I,J: integer;
begin
  for I:=1 to 3 do
    for J:=1 to 3 do
      AI[I,J] := A[J,I];
  for I:=1 to 3 do
  for J:=1 to 4 do
    AI[4,J] := A[4,J];
end; { Orthoinverse }

procedure atan2 < » ( arg1, arg2 : real; var angle: real);
var arg4 : real;
begin
  if ((arg1 > 0) and (arg2 = 0)) then
    angle := pi/2
  else if ((arg1 = 0) and (arg2 < 0)) then
    angle := pi
  else if ((arg1 < 0) and (arg2 = 0)) then
    angle := -pi/2
  else if ((arg1 = 0) and (arg2 > 0)) then
angle := 0.
else if ((arg1 = 0) and (arg2 = 0)) then
    angle := 0.
else
    begin
        arg4 := abs(arg1/arg2);
        angle := ArcTan(arg4);
        if((arg1 > 0) and (arg2 > 0)) then
            angle := angle
        else if((arg1 > 0) and (arg2 < 0)) then
            angle := pi - angle
        else if((arg1 < 0) and (arg2 < 0)) then
            angle := -(pi - angle)
        else if((arg1 < 0) and (arg2 > 0)) then
            angle := -angle
    end;
end;

{*********** homogeneous transformation for sensor system ***********}

procedure SensorTrans1;
{ determine transformation matrix for laser head, camera1, and camera2}
var
    Matrix1, Matrix2, Matrix3, Matrix4 : Matrix4x4;
    i,j : integer;
    arg1, arg2, arg3, angle : real;
begin
{ laser head w.r.t. world }
    Comp1[1] := -CompNorm[1];
    Comp1[3] := -CompNorm[3];
    Comp1[4] := 0.;
    OrthoInverse (worldTlaser,newworldTlaser);
    MatMpy4xl(newworldTlaser,Comp1,Comp2);
    arg1 := Comp2[1];
    arg3 := Comp2[1]*Comp2[1] + Comp2[3]*Comp2[3];
    arg2 := sqrt(arg3);
    atan2(arg1, arg2, angle);
    RotY(angle,Matrix1);
    MatMpy4x4(WorldTlaser,Matrix1,Matrix3,4,4,4);
    arg1 := Comp2[2];
    arg3 := Comp2[1]*Comp2[1] + Comp2[3]*Comp2[3];
    arg2 := sqrt(arg3);
    atan2(arg1, arg2, angle);
    RotX(-angle,Matrix2);
MatMpy(Matrix3,Matrix2,Matrix4,4,4,4); {3,2}
Matrix4[1,4] := PxS;
Matrix4[2,4] := PyS;
Matrix4[3,4] := PzS;
Matrix4[4,4] := 1.;
for i:= 1 to 4 do
begin
    for j := 1 to 4 do
    begin
        worldTlaser[i,j] := Matrix4[i,j];
    end;
end;
{ cameral w.r.t laser head }
Trans (CameraDist,0.,0.,Matrixl);
RotY(—CameraAngle, Matrix2);
MatMpy (Matrix1, Matrix2, laserTcamera1, 4,4,4);
{ camera2 w.r.t. laser head }
Trans (0.,CameraDist,0.,Matrixl);
RotZ (pi/2., Matrix2);
MatMpy (Matrix1, Matrix2, Matrix3, 4,4,4);
RotY (—CameraAngle, Matrix1);
MatMpy (Matrix3, Matrix1, laserTcamera2, 4,4,4);
{ cameral and 2 w.r.t. world coord }
MatMpy (worldTlaser, laserTcamera1, worldTcamera1, 4,4,4);
MatMpy (worldTlaser, laserTcamera2, worldTcamera2, 4,4,4);
end; {SensorTransform}
{ *********************************************** }
procedure SensorTransform;
{ determine transformation matrix for laser head, cameral, and camera2 }
var
    Matrix1, Matrix2, Matrix3 : Matrix4x4;
begin
    { laser head w.r.t. world }
    RotZ (OangS, Matrix1);
    OATzero (Matrix2);
    MatMpy (Matrix1, Matrix2, Matrix3,4,4,4);
    RotY (AangS, Matrix1);
    MatMpy (Matrix3, Matrix1, Matrix2,4,4,4);
    RotZ (TangS, Matrix1);
    MatMpy (Matrix2, Matrix1, Matrix3,4,4,4);
    Trans (PxS,PyS,PzS,Matrixl);
    MatMpy (Matrix1, Matrix3,WorldTlaser,4,4,4);
    { cameral w.r.t laser head }

Trans (CameraDist,0.,0.,Matrix1);
RotY (-CameraAngle, Matrix2);
MatMpy (Matrix1, Matrix2, laserTcamera1, 4,4,4);
{ camera2 w.r.t. laser head }
Trans (0.,CameraDist,0.,Matrix1);
RotZ (pi/2., Matrix2);
MatMpy (Matrix1, Matrix2, Matrix3, 4,4,4);
RotY (-CameraAngle, Matrix1);
MatMpy (Matrix3, Matrix1, laserTcamera2, 4,4,4);
{ camera1 and 2 w.r.t. world coord }
MatMpy (worldTlaser, laserTcamera1, worldTcamera1, 4,4,4);
MatMpy (worldTlaser, laserTcamera2, worldTcamera2, 4,4,4);
end; {SensorTransform}
*************** procedures for Turbo graphics ***************
begin
{  when using Crt and graphics, turn off Crt' s memory-mapped writes  } DirectVideo
{$F+}
procedure MyExitProc;
begin
  ExitProc := OldExitProc; { Restore exit procedure address }
  CloseGraph; { Shut down the graphics system }
end; { MyExitProc }
{$F-}
procedure Initializegraphics;
{ Initialize graphics and report any errors that may occur }
var
  InGraphicsMode : boolean; { Flags initialization of graphics mode }
  PathToDriver : string; { Stores the DOS path to *.BGI & *.CHR }
begin
  { when using Crt and graphics, turn off Crt' s memory-mapped writes } DirectVideo
  := False;
  OldExitProc := ExitProc; { save previous exit proc }
  ExitProc := @MyExitProc; { insert our exit proc in chain }
  PathToDriver := ''; { default path }
  repeat
    {$IFDEF Use8514} { check for Use8514 $DEFINE }
      GraphDriver := IBM8514;
      GraphMode := IBM8514Hi;
    {$ELSE}
      GraphDriver := Detect; { use autodetection }
    {$ENDIF}
  InitGraph(GraphDriver, GraphMode, PathToDriver);
  ErrorCode := GraphResult; { preserve error return }
  if ErrorCode ≠ grOK then { error? }
  begin

Writeln('Graphics error: ', GraphErrorMsg(ErrorCode));
if ErrorCode = grFileNotFound then { Can't find driver file } begin
  Writeln('Enter full path to BGI driver or type <Ctrl-Break> to quit: ');
  Readln(PathToDriver);
  Writeln;
end
else
  Halt(1); { Some other error: terminate }
end;
until ErrorCode = grOK;
Randomize; { init random number generator }
MaxColor := GetMaxColor; { Get the maximum allowable drawing color }
MaxX := GetMaxX; { Get screen resolution values }
MaxY := GetMaxY;
end; { InitializeGraphics }
{************** Draw border of each window ******************}
procedure DrawBorder;
{ Draw a border around the current view port }
var
  ViewPort : ViewPortType;
begin
  SetColor(MaxColor);
  SetLineStyle(SolidLn, 0, NormWidth);
  GetViewSettings(ViewPort);
  with ViewPort do
  begin
    rectangle(0,0,x2-xl,y2-yl);
  end; { DrawBorder }
{************** Definition of full screen window ******************}
procedure FullPort;
{ Set the view port to the entire screen }
begin
  SetViewPort(0, 0, MaxX, MaxY, ClipOn);
end; { FullPort }
{************** Definition of main simulation window ******************}
procedure MainPort;
{ Set the view port to the main simulation screen }
begin
  XVmin:=0; YVmin:=TextHeight('M')+4;
  XVmax:=(MaxX*2) div 3; YVmax:=MaxY-(TextHeight('M')+4);
  ScaleX:=(XVmax-XVmin)/(XWmax-XWmin);
  sX := ScaleX;
  ScaleY:=(YVmax-YVmin)/(YWmax-YWmin);
  sY := ScaleY;
SCF:=(XVmax—XVmin)/(YVmax—YVmin);  \{Screen scaling adjustment factor\}
SetViewport(XVmin+1,YVmin+1,XVmax—1,YVmax—1, ClipOn);
\{ writeln(sX,sY:5); \}
end; \{MainPort\}

procedure MainWindow(Header : string);
\{ Make a main simulation window and view port \}
begin
  ClearDevice; \{ Clear the screen \}
  SetTextStyle(DefaultFont, HorizDir, 1); \{ Default text font \}
  SetTextJustify(CenterText, TopText); \{ Left justify text \}
  FullPort; \{ Full screen view port \}
  OutTextXY(MaxX div 2, 2, Header); \{ Draw the header \}
  \{ Draw main simulation window \}
  MainPort ;
  SetViewport(XVmin,YVmin,XVmax,YVmax,ClipOn);
  DrawBorder; \{ Put a border around it \}
  SetViewport(XVmin+1,YVmin+1,XVmax—1,YVmax—1, ClipOn);
end; \{ MainWindow \}

\{*************** Definition of simulation result display window ***********\}

procedure StatusPort;
\{ Set the status report port to the main simulation screen \}
begin
  XVmin:=0; YVmin:=TextHeight('M')+4;
  XVmax:=MaxX*2 div 3 —1; YVmax:=MaxY div 5+(TextHeight('M')+4);
  SetViewport(XVmin+1,YVmin+1,XVmax—1,YVmax—1, ClipOn);
end; \{ StatusPort \}

procedure StatusWindow(Header : string);
begin
  SetTextStyle(DefaultFont, HorizDir, 1); \{ Default text font \}
  SetTextJustify(CenterText, TopText); \{ Left justify text \}
  StatusPort;
  OutTextXY(XVmax div 2, 2, Header); \{ Draw the header \}
  \{ Draw main simulation window \}
  SetViewport(XVmin,YVmin,XVmax,YVmax,ClipOn);
  DrawBorder; \{ Put a border around it \}
  SetViewport(XVmin+1,YVmin+1,XVmax—1,YVmax—1, ClipOn);
end; \{ StatusWindow \}

\{************** Definition of cameral image window ***********************\}

procedure Image1Port (XImin,YImin, \{window size definition\}
  XImax,YImax :real);
\{ Set the view port to the cameral image screen \}
begin
  XVmin:=(MaxX*2) div 3; YVmin:=TextHeight('M')+4;

XVmax:=MaxX; YVmax:=TextHeight('M')+(MaxY-(TextHeight('M')+4)*2) div 2;
ScaleX:=(XVmax−XVmin)/(XImax−Xlmin);
ScaleY:=(YVmax−YVmin)/(YImax−Ylmin);
SCF:=(XVmax−XVmin)/(YVmax−YVmin); {Screen scaling adjustment factor}
SetViewPort(XVmin+1,YVmin+1,XVmax−1,YVmax−1, ClipOn);
end; {Image1Port}
procedure Image1Window(Header : string);
{ Make a camera1 image window and view port }
begin
SetColor(MaxColor); { Reset the colors }
SetTextStyle(DefaultFont, HorizDir, 1); { Default text font }
SetTextJustify(CenterText, TopText); { Left justify text }
Image1Port(0,0,CCDsize,CCDsize);
OutTextXY((XVmax—XVmin) div 2,2, Header); { Draw the header }
{ Draw main window }
SetViewPort(XVmin,YVmin,XVmax,YVmax,ClipOn);
DrawBorder; { Put a border around it }
SetViewPort(XVmin+1,YVmin+1,XVmax−1,YVmax−1,ClipOn);
end; { Image1Window }
{************** Definition of camera2 image window ***************}
procedure Image2Port(XImin,YImin,XImax,YImax:real);
{ Set the view port to the camera2 image screen }
begin
XVmin:=(MaxX*2) div 3;
YVmin:=TextHeight('M')+(MaxY−(TextHeight('M')+4)*2) div 2;
XVmax:=MaxX; YVmax:=MaxY−TextHeight('M')−4;
ScaleX:=(XVmax−XVmin)/(XImax−Xlmin);
ScaleY:=(YVmax−YVmin)/(YImax−Ylmin);
SCF:=(XVmax−XVmin)/(YVmax−YVmin); {Screen scaling adjustment factor}
SetViewPort(XVmin+1,YVmin+1,XVmax−1,YVmax−1, ClipOn);
end; {Image2Port}
procedure Image2Window(Header : string);
{ Make a camera2 window and view port }
begin
SetColor(MaxColor); { Reset the colors }
SetTextStyle(DefaultFont, HorizDir, 1); { Default text font }
SetTextJustify(CenterText, TopText); { Left justify text }
Image2Port(0,0,CCDsize,CCDsize);
OutTextXY((XVmax−XVmin) div 2,2, Header); { Draw the header }
{ Draw main window }
SetViewPort(XVmin,YVmin,XVmax,YVmax,ClipOn);
DrawBorder; { Put a border around it }
SetViewPort(XVmin+1,YVmin+1,XVmax−1,YVmax−1,ClipOn);
end; { Image2Window }

{********** Make status line at the bottom of screen **********}
procedure StatusLine(Msg : string);
{ Display a status line at the bottom of the screen }
begin
  FullPort;
  SetColor(MaxColor);
  SetTextStyle(DefaultFont, HorizDir, 1);
  SetTextStyle(CenterText, TopText);
  SetLineStyle(SolidLn, 0, NormWidth);
  SetFillStyle(EmptyFill, 0);
  Bar(0, MaxY−(TextHeight('M')+4), MaxX, MaxY);  { Erase old status line }
  Rectangle(0, MaxY−(TextHeight('M')+4), MaxX, MaxY);
  OutTextXY(MaxX div 2, MaxY−(TextHeight('M')+2), Msg);
  { Go back to the main window }
  {SetViewPort(1, TextHeight('M')+5, MaxX−1, MaxY−(TextHeight('M')+5), ClipOn);}
end; { StatusLine }

{********* Print out simulation results on the screen *********}
procedure ReportStatus;
{ Display the laser head location and pose of surface point }
const
  X = 10;
var
  Y : word;
  OangSD,AangSD,TangSD: real;

function Real2str(L : real) : string;
{ Converts an real to a string for use with OutText, OutTextXY }
var
  S : string;
begin
  Str(L:8:3, S);
  real2Str := S;
end; { real2Str }
procedure WriteOut(S : string);
{ Write out a string and increment to next line }
begin
  OutTextXY(X, Y, S);
  Inc(Y, TextHeight('M')+2);
end; { WriteOut }

begin { ReportStatus }
  Y := 4+TextHeight('M');
StatusWindow('Status report after Simulation');
SetTextJustify(LeftText, TopText);
WriteOut('Laser Head XYZ : '+real2Str(WorldTlaser[1,4])+', '+real2Str(WorldTlaser[2,4])+', '+real2Str(WorldTlaser[3,4]));
OangSD:=OangS*180/pi;
AangSD:=AangS*180/pi;
TangSD:=TangS*180/pi;
WriteOut('Laser Head OAT angle: '+real2Str(OangSD)+' , '+real2Str(AangSD)+' , '+real2Str(TangSD));
writeout('Measured Cross Pts : '+real2Str(MeasuredXYZcrsPts[1])+', '+real2Str(MeasuredXYZcrsPts[2])+', '+real2Str(MeasuredXYZcrsPts[3]));
writeout('Actual Cross Pts : '+real2Str(ActualXYZcrsPts[1])+', '+real2Str(ActualXYZcrsPts[2])+', '+real2Str(ActualXYZcrsPts[3]));
writeout('Actual Normal Pts : '+real2Str(ActualNormal[1])+', '+real2Str(ActualNormal[2])+', '+real2Str(ActualNormal[3]));
writeout('Computed Normal Pts : '+real2Str(CompNorm[1])+', '+real2Str(CompNorm[2])+', '+real2Str(CompNorm[3]));
end; { ReportStatus }

*********** Calculate eye coordinate based on given view angle ***********
procedure eyeCoordinate (xw,yw,zw :real; var xe,ye,ze : real); {world coord}
var xe,ye,ze : real; {eye coord}
begin
xe:=— xw*sin(viewTheta)—yw*cos(viewTheta);
ye:=— xw*cos(viewTheta)*cos(viewPhi)— yw*sin(viewTheta)*cos(viewPhi)
+zw*sin(viewPhi);
ze:=— xw*cos(viewTheta)*sin(viewPhi)—yw*sin(viewTheta)*sin(viewPhi)
—zw*cos(viewPhi);
end; {eyeCoordinate}

******** perspective transformation for camera image of projection points ***
procedure PerspectiveTransform (XYZ: Matrix4x1; var ximage,yimage:real); {XY image position}
begin
ximage:=cameraF*XYZ[1]/(cameraF—XYZ[3]);
yimage:=cameraF*XYZ[2]/(cameraF—XYZ[3]);
end; {PerspectiveTransform}

************** Draw half sphere on main simulation window **************
procedure DrawSphere;
var
Th :real; {zero to pi}
Phi :real; {zero to 2*pi}
ThMax, PhiMax, PhiMin : real;
Xsw, Ysw, Zsw : real;
Xsw0, Ysw0 : real;
ThStep, PhiStep : real;
xscrOld, yscrOld : integer;
I, J : integer;

const
  MaxThStep : integer = 20; {number of wire frame}  
  MaxPhiStep : integer = 20;
begin { DrawSphere }
  ThMax := pi; { half sphere }
  PhiMin := 0.;
  PhiMax := pi;
  ThStep := ThMax / MaxThStep;
  PhiStep := (PhiMax — PhiMin) / MaxPhiStep;
  Phi := 0.;
  Xsw0 := radius; Ysw0 := 0;
  SetColor (MaxColor);
  SetLineStyle (SolidLn, 0, NormWidth);
  for i := 0 to MaxPhiStep do
    begin
      phi := i * PhiStep + PhiMin;
      eyeCoordinate (Xsw0, Ysw0, 0., xe, ye, ze);
      xscrOld := round((scaleX * (xe — XWmin)) * SCF);
      yscrOld := round(scaleY * (YWmin — ye)) + (YVmax — YVmin);
      Th := 0.;
      for j := 0 to MaxThStep do
        begin
          Th := j * ThStep;
          Xsw := radius * cos(Th) * cos(Phi);
          Ysw := radius * sin(Th) * cos(Phi);
          Zsw := radius * sin(Th) * sin(Phi);
          eyeCoordinate (Xsw, Ysw, Zsw, xe, ye, ze);
          xscr := round((scaleX * (xe — XWmin)) * SCF);
          yscr := round(scaleY * (YWmin — ye)) + (YVmax — YVmin);
          line (xscrOld, yscrOld, xscr, yscr);
          xscrOld := xscr;
          yscrOld := yscr;
        end;
      Phi := Phi + PhiStep;
    end;
  for i := 0 to MaxPhiStep do
begin
Phi:=PhiMin;
Th:=i*ThStep;
Xsw:=radius*cos(Th);
Ysw:=radius*sin(Th)*cos(Phi);
Zsw:=radius*sin(Th)*sin(Phi);
eyeCoordinate (Xsw,Ysw,Zsw,xe,ye,ze);
xscrOld:=round((scaleX*(xe—XWmin))*SCF);
yscrOld:=round(scaleY*(YWmin—ye))+(YVmax−YVmin);
for j:=0 to MaxPhiStep do
begin
Phi:=j*PhiStep+PhiMin;
Xsw:=radius*cos(Th);
Ysw:=radius*sin(Th)*cos(Phi);
Zsw:=radius*sin(Th)*sin(Phi);
eyeCoordinate (Xsw,Ysw,Zsw,xe,ye,ze);
xscr:=round((scaleX*(xe—XWmin))*SCF);
yscr:=round(scaleY*(YWmin—ye))+(YVmax—YVmin);
line (xscrOld,yscrOld,xscr,yscr);
xscrOld:=xscr;
yscrOld:=yscr;
end;
end; {for}
end; {DrawSphere}

end; {DrawSphere}

*************** Draw coordinate frame for laser head, camera1, ***********
camera2, and sphere. World coord is at the center ***********

*************** of sphere ***************

procedure DrawCoordinate (len :integer; {size of axis}
LabelSize,color :word; {axis label}
T :Matrix4x4; {coord frame to be drawn}
header :string); {name of frame}

var xtip, ytip, ztip : real;
axis : integer;

begin
SetTextStyle(DefaultFont, HorizDir, LabelSize);
SetTextJustify(CenterText, CenterText);
SetLineStyle(SolidLn, 0, NormWidth);
SetColor(color);
for axis:=1 to 3 do
begin
  eyeCoordinate (T[1,4],T[2,4],T[3,4],xe,ye,ze);
xscr:=round((scaleX*(xe—XWmin))*SCF);
yscr := round(scaleY *(YWmin— ye)) + (YVmax— YVmin);
moveTo (xscr, yscr);
xtip := T[1,4] + T[1, axis] * len;
eyeCoordinate (xtip, ytip, ztip, xe, ye, ze);
xscr := round((scaleX *(xe— XWmin)) * SCF);
yscr := round(scaleY *(YWmin— ye)) + (YVmax— YVmin);
lineTo (xscr, yscr);
xtip := T[1,4] + T[1, axis] *(len+15);
ytip := T[2,4] + T[2, axis] *(len+15);
ztip := T[3,4] + T[3, axis] *(len+15);
neyeCoordinate (xtip, ytip, ztip, xe, ye, ze);
xscr := round((scaleX *(xe— XWmin)) * SCF);
yscr := round(scaleY *(YWmin— ye)) + (YVmax— YVmin);
CASE axis OF
1 : begin
   OutTextXY (xscr, yscr, 'X');
   OutTextXY (xscr+10, yscr+10, header);
end;
2 : OutTextXY (xscr, yscr, 'Y');
3 : OutTextXY (xscr, yscr, 'Z');
end;
end;
end;

{ ************* calculate projection point of laser beam on ***************
 ** half sphere object surface using Ray Tracing Technique **** }
procedure GetProjPoint (xp, yp, zp, xdir, ydir, zdir: real; { laser position }
 var projpt: Matrix4x1; n: integer); { projection point position }

var
IntersectExist : boolean; { true=laser beam intersect to object }
alpha, { parameters used for calculating }
beta, { intersection point with sphere }
argument,
t, t1, t2 : real; { for Ray tracing }
i : integer;
begin
alpha := (xp— xc)*xdir+(yp— yc)*ydir+(zp— zc)*zdir;
beta := sqr(xp— xc)+sqr(yp— yc)+sqr(zp— zc)— sqa(radius);
argument := sqa(alpha)— beta;
IntersectExist := argument ≥ 0.
{ writeln(n:5); }

if IntersectExist
then begin
  t1 := -alpa-sqrt(argument);  { first intersection }
  t2 := -alpa+sqrt(argument);  { second intersection }
  if t1<t2
    then t := t1
    else t := t2;  { smaller t is an actual intersection }
  projpt[3] := zp+zdir\*t;
  if projpt[3] \geq 0.  { intersect with upper half sphere}
     then begin
    projpt[1] := xp+xdir\*t;
    projpt[2] := yp+ydir\*t;
    projpt[3] := zp+zdir\*t;
    projpt[4] := 1.;
    if n = 0 then
      begin
        writeln(ActualNormal[1],projpt[1]:5);
      end;
     end
  end
else begin  { intersect with lower half sphere}
  t := -zp/zdir;  { is treated as intersection to flat }
  projpt[1] := xp+xdir\*t;  { table top }
  projpt[2] := yp+ydir\*t;
  projpt[3] := 0.;
  projpt[4] := 1.;
  end;
else begin  { intersect with flat table top }
  t := -zp/zdir;
  projpt[1] := xp+xdir\*t;
  projpt[2] := yp+ydir\*t;
  projpt[3] := 0.;
  projpt[4] := 1.;
  if n = 0 then
    begin
      ActualNormal[1] := 0.;
      ActualNormal[2] := 0.;
      ActualNormal[3] := 1.;
    end;
end;
end; { GetProjPoint }

{**** for n-th laser beam ***************************************************}
procedure GetProjPointPlot(n:integer);
var m: integer;
begin
MainPort;
 xp:=PxS+worldTlaser[1,XorY]*deltat*n;
 yp:=PyS+worldTlaser[2,XorY]*deltat*n;
 zp:=PzS+worldTlaser[3,XorY]*deltat*n;
 GetProjPoint (xp,yp,zp,xdir,ydir,zdir,projpt,n);
 { get actual coord of cross intersection point }
if (n=0) then
begin
 for m := 1 to 4 do
 ActualXYZcrsPts[m]:=projpt[m];
end;
 eyeCoordinate (projpt[1],projpt[2],projpt[3],xe,ye,ze);
 xscr:=round((scaleX*(xe—XWmin))*SCF);
 yscr:=round(scaleY*(YWmin— ye)) + (YVmax— YVmin);
end;

{************* Get image pixellocation for the camera ***************
 ************* located at "cameraTworld" and plot it ***************
 ************* on the screen. Viewport def procedure ***************
 ************* should be called first.******************************}
procedure PutlmagePixel (cameraTworld:matrix4x4); {camera location}
var imagex, imagey :real; {image position on focal plane of camera}
 dum :real;
 imagexyz :matrix4x1; {laser proj point w.r.t. camera frame}
begin
 Matmpy4xl (cameraTworld,projpt,imagexyz);
 perspectiveTransform (imagexyz, imagex, imagey); {image on focal plane}

dum:=imagey+CCDsize/2.;
imageXpixel:=trunc(dum/CCDresolution)+1; { in terms of pixel}
 dum:=imagey+CCDsize/2.;
imageYpixel:=trunc(dum/CCDresolution)+1; { in terms of pixel}

{plot on screen}
xscr:=round((scaleX*(imagey+CCDsize/2))*SCF);
 yscr:=round(scaleY*—CCDsize/2—imagey))+(YVmax— YVmin);
 PutPixel (yscr,xscr, MaxColor);
end; { PutlmagePixel }
procedure GetXYZ (cameraNo :integer; { camera1 or 2} xpixel, ypixel :integer); { image pixel location }
var
  xca, yca :real; {image position converted from xpixel, ypixel
                   i.e. origin of xca,yca is center of CCD}
  beta :real; {angle between opt axis and image axis }
  dumF,dumD,dumA :real;
begin
  xca:=(xpixel— CCDmaxPixel/2)*CCDresolution;
yca:=(ypixel— CCDmaxPixel/2)*CCDresolution;
beta:= arctan(xca/CameraF);
dumA:=CameraAngle— beta;
dumF:=CameraF*sin(CameraAngle);
dumD:=(CameraDist— dumF);
XYZsensor[3]:= dumD*(sin(dumA)/cos(dumA))+dumF;
if cameraNo=1 then { Camera1 detect Ys coord and Camera2 detect
                      Xs coord }
begin
  XYZsensor[2]:=— (yca*dumD*cos(beta)) / (CameraF *cos(dumA));
  XYZsensor[1]:=0.; {laser1 is aligned with Xs axis}
end;
if cameraNo=2 then
begin
  XYZsensor[1]:=(yca*dumD*cos(beta))/(CameraF*cos(dumA));
  XYZsensor[2]:=0.; {laser2 is aligned with Ys axis}
end;
XYZsensor[4]:=l; {homogeneous coordinate}
{convert XYZsensor to world frame}
MatMpy4xl (worldTlaser, XYZsensor, XYZworld);
end;
{**************************** Get measured XYZ position of cross point ***************}
procedure GetCrossPosition;
var n:integer;
begin
  for n:= 1 to 4 do
    MeasuredXYZcrsPts[n]:=XYZlaser1W[n,0];
end; {GetCrossPosition}
procedure CubicSplineFit(CameraNo :integer; XYZminX,XYZminZ,ScaleX,ScaleY,SCF:real);
var
i, j, k : integer;
begin
  for k := 1 to 3 do
  begin
    for j := 1 to 3 do
    begin
      am[k,j] := dum1[k,j];
      ai[k,j] := dum2[k,j];
    end;
  end;
  for k := 1 to 3 do
  begin
    for j := 1 to 3 do
    begin
      dd[k,j] := de[k,j];
    end;
  end;
SetLineStyle(SolidLn, 0, NormWidth);
if CameraNo = 1 then
begin
  for i := -1 to 1 do
  begin
    for j := 1 to 4 do
    begin
      ap[2+i,j] := pp1[i,j];
    end;
  end;
end;
if CameraNo = 2 then
begin
  for i := -1 to 1 do
  begin
    for j := 1 to 4 do
    begin
      ap[2+i,j] := pp2[i,j];
    end;
  end;
end;
{begin spline fitting}
  for i := 1 to size-2 do
  begin
    for j := 1 to 3 do
    begin
      ...
ad[i+1,j] := 3.*(ap[i+2,j] — ap[i,j]);
end;
end;
am[size,size] := 2.;
for j := 1 to 3 do
begin
ad[size,j] := 3.*(ap[size,j] — ap[size-1,j]);
end;
for i := 1 to size-1 do
begin
facl := 1./(am[i,i]);
for j := 1 to size do
begin
am[i,j] := am[i,j]*facl;
ai[i,j] := ai[i,j]*facl;
end;
for j := 1 to size do
begin
am[i+1,j] := am[i+1,j] — am[i,j];
ai[i+1,j] := ai[i+1,j] — ai[i,j];
end;
end;
for i := 3 to 1 do
begin
fac1 := 1./(am[i,i]);
fac2 := 1./(am[i-1,i]);
for j := 1 to size do
begin
am[i,j] := am[i,j]*fac1;
am[i-1,j] := am[i-1,j]*fac2;
ai[i,j] := ai[i,j]*fac1;
ai[i-1,j] := ai[i-1,j]*fac2;
end;
for j := 1 to size do
begin
am[i-1,j] := am[i-1,j] — am[i,j];
ai[i-1,j] := ai[i-1,j] — ai[i,j];
end;
end;
fac1 := 1./(am[1,1]);
for j := 1 to size do
begin
\begin{verbatim}
   am[1,j] := am[1,j]*fac1;
   ai[1,j] := ai[1,j]*fac1;
end;
for k := 1 to 3 do
for j := 1 to size do
for i := 1 to size do
begin
   dd[i,k] := dd[i,k] + ad[j,k]*ai[i,j];
end;
for k := 1 to 3 do
for j := 1 to size—1 do
begin
   ac[1,j,k] := ap[j,k];
   ac[2,j,k] := dd[j,k];
   ac[3,j,k] := 3.*(ap[j+1,k] - ap[j,k]) - 2.*dd[j,k] - dd[j+1,k];
   ac[4,j,k] := -2.*(ap[j+1,k] - ap[j,k]) + dd[j+1,k] + dd[j,k];
end;
pxx := ap[1,1] + ((ap[1,3])*((sin(theta)/cos(theta)))*cos(gamma));
pyy := ap[1,2] - f((ap[1,3])*((sin(theta)/cos(theta)))*sin(gamma));
xscr:=round((scaleX*(pxx—XYZminX))*SCF);
yscr:=round(scaleY*(XYZminZ—pyy)+(YVmax—YVmin));
SetColor(5);
MoveTo (xscr ,yscr);
for segment := 1 to size—1 do
for tt := 1 to 100 do
begin
   delt := tt/100.;
   for k := 1 to 3 do
begin
   pt[k] := ac[1,segment,k] + (ac[2,segment,k])*delt
   + (ac[3,segment,k])*delt*delt +
   (ac[4,segment,k])*delt*delt*delt;
   grad[k] := ac[2,segment,k] +2.*(ac[3,segment,k]*delt)
   + 3.*(ac[4,segment,k]*delt*delt);
   if ((segment = 1) and (delt = 1.0)) then
   begin
   if CameraNo = 1 then
   begin
   pt[k] := ac[1,segment,k] + (ac[2,segment,k])*delt
   + (ac[3,segment,k])*delt*delt +
   (ac[4,segment,k])*delt*delt*delt;
   Slope1[k] := ac[2,segment,k] +
   2.*(ac[3,segment,k]*delt) +
   
end;
\end{verbatim}
3.*(ac[4, segment, k]*delt*delt);
Slope1[k] := grad[k];
end;
if CameraNo = 2 then
begin
Slope2[k] := ac[2, segment, k] +
2.*(ac[3, segment, k]*delt) +
3.*(ac[4, segment, k]*delt*delt);
Slope2[k] := grad[k];
end;
end;
end;
px := pt[l] +
((pt[3])*((sin(theta)/cos(theta)))*(cos(gamma)));
py := pt[2] +
((pt[3])*((sin(theta)/cos(theta)))*(sin(gamma)));
if CameraNo = 2 then
xscr:=round(ScaleX*((px— XYZminX))*SCF)
else
xscr:=round(ScaleX*((px— XYZminX))*SCF);
yscr:=round(ScaleY*(XYZminZ— py)+(YVmax— YVmin));
SetColor(segment*5);
LineTo(xscr,yscr);
end;
GraphDefaults;
end; { CubicSplineFit}
procedure CompNormal(var Slpl,Slp2,CompNorm : Matrix3xl);
var
i,j,k : integer;
Vect : Matrix3xl;
Prod : real;
begins
for k := 1 to 3 do
begin
CompNorm[k] := Vect[k]/sqrt(Prod);
end;
end; {CompNormal}
procedure PredictNormal(kk,mm:integer;Spn2:Matrix4xl;
CompNorm:Matrix3xl;var PzS,d :real;var
PredictNorm:Matrix3xl);
var
i,j : integer;
pd,sp1,sp2,sp3 : real;
{pz : real}
begin
for i := 1 to 3 do
begin
  PredictNorm[i] := CompNorm[i];
end;
eyeCoordinate(Spn2[1],Spn2[2],Spn2[3],xe,ye,ze);
xsaold[kk,mm] := round(sX*(xe - XWmin)*SCF);
ysaold[kk,mm] := round(sY*(YWmin - ye)) + (YVmx - YVmn);
nextp1 := Spn2[1] + 50*PredictNorm[1];
nextp3 := Spn2[3] + 50*PredictNorm[3];
eyeCoordinate(nextp1,nextp2,nextp3,xe,ye,ze);
xsa[kk,mm] := round(sX*(xe - XWmin)*SCF);
ysa[kk,mm] := round(sY*(YWmin - ye)) + (YVmx - YVmn);
pz := -((pd + PredictNorm[1]*hh + PredictNorm[2]*rr)/PredictNorm[3]);
PzS := pz + 200.*CompNorm[3];
end;
{************ Plot measured XYZ position of each discretized laser ************
******** beam on screen *******************}
procedure PlotXYZsensor;
var i,n:integer;
  XYZminX,XYZminY,XYZmaxX,XYZmaxY,XYZminZ,XYZmaxZ:real;
begin
  GetXYZ (1,CCDmaxPixel,CCDmaxPixel);
  XYZminY:=XYZsensor[2];
  XYZminZ:=XYZsensor[3];
  GetXYZ (1,1,1);
  XYZmaxY:=XYZsensor[2];
  XYZmaxZ:=XYZsensor[3];
  Image1Port (XYZminY,XYZminZ,XYZmaxY,XYZmaxZ);
for n := -10 to 10 do
begin
  xscr:=round((scaleX*(XYZlaser2[2,n]-XYZminY))*SCF);
  yscr:=round(scaleY*(XYZminZ-XYZlaser2[3,n]))+(YVmx-YVmn);
  if(n = 0)then
begin
    SetColor(maxcolor-2);
    PutPixel (xscr,yscr,maxcolor-2);
    circle (xscr,yscr,1);
SetColor(1);
end
else
   PutPixel (xscr, yscr, max
   Slope2[k] := grad[k];
   end;
end;
px := pt[1] + ((pt[3])*((sin(theta)/cos(theta)))*(cos(gamma)));
py := pt[2] + ((pt[3])*((sin(theta)/cos(theta)))*(sin(gamma)));
if CameraNo = 2 then
   xscr:=round(ScaleX*((px— XYZminX)))*SCF)
else
   xscr:=round(ScaleX*((px— XYZminX)))*SCF);
yscr:=round(ScaleY*(XYZminZ— py)+(YVmax— YVmin)*color— 1);
end;
XYZminX := XYZminY;
CubicSplineFit(l, XYZminX, XYZminZ, ScaleX, ScaleY, SCF);
GetXYZ (2, CCDmaxPixel, CCDmaxPixel);
XYZminX:=XYZsensor[1];
XYZminZ:=XYZsensor[3];
GetXYZ (2, 1, 1);
XYZmaxX:=XYZsensor[1];
XYZmaxZ:=XYZsensor[3];
Image2Port (XYZminX, XYZminZ, XYZmaxX, XYZmaxZ);
for n:= nmin to nmax do (+1)
   begin
      xscr:=round((scaleX*(XYZlaser1[1,n]— XYZminX))*SCF);
yscr:=round(scaleY *(XYZminZ— XYZlaser1[3,n]))+(YVmax— YVmin);
if (n=0) then
   begin
      SetColor(maxcolor— 2);
      SetLineStyle(SolidLn, 0, ThickWidth);
      PutPixel (xscr, yscr, maxcolor—2);
      circle (xscr, yscr, 1);
      SetColor(1);
   end
else
   PutPixel (xscr, yscr, maxcolor—1);
end;
CubicSplineFit(2, XYZminX, XYZminZ, ScaleX, ScaleY, SCF);
CompNormal(Slope1, Slope2, CompNorm); {Computes the normal}
end;
procedure TrackSensorHead;
var
i,j:integer;
begin
PredictNormal(kk,mm,Spn2,CompNorm,PzS,d,PredictNorm);
SensorTrans1;
Simulation(maxcolor—3);
PlotXYZsensor;
end;
{*************** Main simulation procedure ***********************}
procedure Simulation(Color:word);
var
n,i,j,k,m:integer;
InView:boolean;
begin
OrthoInverse(worldTcamera1,camera1Tworld);
OrthoInverse(worldTcamera2,camera2Tworld);
MainPort;
DrawCoordinate(100,1,color,WorldTlaser,'Laser Head'); {sensor coord}
DrawCoordinate(70,1,color,WorldTcamera1,'Camera1'); {camera 1, on
X axis of sensor frame}
DrawCoordinate(70,1,color,WorldTcamera2,'Camera2'); {camera 2, on
Y axis of sensor frame}
deltat:=laserLen/(Laserpts—1); {discretize cross laser line}
nmin:=(LaserPts—1) div 2;
nmax:=—nmin;
xdir:=worldTlaser[1,3]; {dir cosine of laser beam}
ydir:=worldTlaser[2,3];
zdir:=worldTlaser[3,3];
SetLineStyle(SolidLn, 0, ThickWidth); { line style for projected laser}
SetColor(color); {SetColor(MaxColor—3);} 
eyeCoordinate(worldTlaser[1,4],worldTlaser[2,4],worldTlaser[3,4],
xe,ye,ze);
xscrorld:=round((scaleX*(xe—XWmin))*SCF);
yscrorld:=round(scaleY*(YWmin—ye)+(YVmax—YVmin));
XorY:=1;
GetProjPointPlot(0);
line (xscrorld,yscrorld,xscr,yscr);
{START of SIMULATION}
for XorY:=1 to 2 do {1: laser aligned with X axis}
    {2: laser aligned with Y axis}
    begin
    GetProjPointPlot(nmin);
for n := nmin to nmax do
  begin
    GetProjPointPlot(n); {draw projected laser beam}
    Line(xscrold,yscrold,xscr,yscr);
    xscrold := xscr;
    yscrold := yscr;
    {plot camera 1 image and get actual image pixel location}
    ImagelPort(0,0,CCDsize,CCDsize);
    PutlmagePixel (cameralTworld);
    {Is image inside of camera view ?}
    InView := (imageXpixel >= 1) and (imageXpixel <= CCDmaxPixel) and
             (imageYpixel >= 1) and (imageYpixel <= CCDmaxPixel);
    if (XorY = 2) and (InView) then
      begin
        GetXYZ (1, imageXpixel, imageYpixel);
        for m := 1 to 4 do
          begin
            XYZlaser2[m,n] := XYZsensor[m];
            XYZlaser2W[m,n] := XYZworld[m];
            XYZlaser2[m,n] := XYZlaser2[m,n]; {XYZlaser2[m,n]}
          end;
        if ((n >= -1) and (n <= 1)) then
          begin
            for m := 1 to 4 do
              begin
                pp1[n,m] := XYZworld[m];
              end;
          end;
      end;
  end;
  {plot camera 2 image and get actual image pixel location}
  Image2Port(0,0,CCDsize,CCDsize);
  PutlmagePixel (camera2Tworld);
  InView := (imageXpixel >= 1) and (imageXpixel <= CCDmaxPixel) and
           (imageYpixel >= 1) and (imageYpixel <= CCDmaxPixel);
  if (XorY = 1) and (InView) then
    begin
      GetXYZ (2, imageXpixel, imageYpixel);
      for m := 1 to 4 do
        begin
          XYZlaser1W[m,n] := XYZsensor[m];
          XYZlaser1[m,n] := XYZworld[m];
        end;
    end;
XYZlaser1[n,m] := XYZlaser1[m,n];  \{XYZlaser1[m,n]\}
end;
if ((n ≥ -1) and (n ≤ 1)) then
begin
    for m := 1 to 4 do
    begin
        pp2[n,m] := XYZworld[m];
    end;
end;
if n = 0 then
begin
    for m := 1 to 4 do
    begin
        Spn2[m] := XYZworld[m];
    end;
end;
{Simulation}
procedure SmallMain;
begin
    Image1Port(0,0,CCDsize,CCDsize);
    ClearViewPort;
    SetColor(MaxColor);
    SetTextStyle(DefaultFont, HorizDir, 1);  \{ Default text font \}
    SetTextJustify(CenterText, TopText);  \{ Left justify text \}
    OutTextXY((XVmax−XVmin) div 2,2, 'Camera1 Image');  \{ Draw the header \}
    Image2Port(0,0,CCDsize,CCDsize);
    ClearViewPort;
    OutTextXY((XVmax−XVmin) div 2,2, 'Camera2 Image');  \{ Draw the header \}
    TrackSensorHead;
    StatusPort;
    ClearViewPort;
GetCrossPosition;
    ReportStatus;
end;
{ ****************************************** Main Program ******************************************}
var
  i,j,ii,ji :integer;
begin
  Assign(FileVar,'op.dat');
Assign(FF,'p.dat');
Rewrite(FileVar);
Rewrite(FF);
SimulParameter; \{obtain simulation parameters\}
InitializeGraphics; \{initialize BGI graphics\}
MainWindow ('Simulation of CrossLaser Beam Tack ing');
DrawSphere;
Trans (xc,yc,zc,worldTsphere);
DrawCoordinate (100,1,MaxColor—4,worldTsphere,''); \{object frame,
also world frame at the center of sphere\}
Image1Window('Camera1 Image');
Image2Window('Camera2 Image');
SensorTransform;
Simulation(MaxColor—3);
PlotXYZsensor;
StatusLine('Esc to quit, Arrow & PgDn&Up for XYZ motion, Alt-OAT for Euler Angle'); GetCrossPosition;
ReportStatus;
repeat
  WaitToGo;
  if funcKey then
  begin
    for kk := 1 to 1 do
      begin
        for mm := 1 to 110 do
          begin
            delX := delX;
            if ((kk mod 2) = 0) then
              begin
                hh := PxS — delX;
                PxS := hh + 200.*CompNorm[1];
              end;
            if ((kk mod 2) ≠ 0) then
              begin
                hh := PxS + delX;
                PxS := hh + 200.*CompNorm[1];
              end;
            rr := PyS;
            PyS := rr + 200.*CompNorm[2];
            SmallMain;
            writeln(FileVar,PxS,PzS:10);
            writeln(FF,PxS,PyS,PzS:10);
Simulation(0);
Pxs := hh;
PyS := rr;
end;

for mm := 1 to 1 do
begin
  bb := PyS + delY;
  PyS := bb + 200.*CompNorm[2];
  cc := Pxs;
  Pxs := cc + 200.*CompNorm[1];
  SmallMain;
  Simulation(0);
  writeln(FF,Pxs,PyS,PzS:10);
  PyS := bb;
  Pxs := cc;
end;
end;

until (kk = 1); {forever}
Close(FileVar);
Close(FF);
Clearviewport;
setcolor(5);
MainWindow ('Overlaying with computed normals');
StatusLine('Esc to quit, Up Arrow to begin overlaying');
repeat
  Wait;
  if funcKey then
  begin
    Drawsphere;
    for jj:= 1 to 1 do
    begin
      for ii := 1 to 110 do
      begin
        SetColor(ii+1);{MaxColor — 4};}
        Line(xsaold[jj,ii],ysaold[jj,ii],xsa[jj,ii],ysa[jj,ii]);
      end;
    end;
  end;
until (1 < 0);
end.
This program captures the cross-shaped light beam into the frame buffer, Grey scale image converted to binary scale using the histogram approach, the binary image is then thinned.

#include <stdio.h>
#include <debugat.h>
define maxX 511
define maxY 420
void mainty();
unsigned long int buf[258];
int ntimes,minlist[5],maxlist[5],size,t,small,x,y;
main()
{
    int chan,i,j,h,k,ff,maxk,u2,k1;
    int x1,y1,x2,y2;
    unsigned long m;
    extern int t,small,x,y;
    mainty();
    printf("enter channel <1 or 2> s l n");
    scanf(" %d", & chan);
    im_opmode(2,3);
    im_procwin(272,114,452,364); /* processing window can be */
    ntimes = 3; /* set up to be dynamic */
    size = 6;
    im_clear(4,0);
    im_clear(5,0);
    im_opmode(2,0);
    im_outpath(0,— 1,0,0);
    im_chann(chan); /* selects A/D channel */
    im_sync(1,0); /* selects camera sync */
    im_video(1,0); /* selects video buffer for output */
    im_video(1,1);
    im_snapshot(0); /* takes snapshot to frame buffer 0 */
    im_histo(0,buf); /* generates the histogram of image in buffer */
    /* smoothening the histogram to obtain optimal threshold */
    im_threshold(ntimes,minlist,maxlist,size,buf);
    /*
     for(i = 0; i < 5; i++)
     {
        printf(" %d %d s l n",minlist[i],maxlist[i]);
     }*/
* /
im_opmode(0,0);
for(i = 272; i < 452; i++)
{
    for(j = 114; j < 364; j++)
    {
        m = im_pixr(i,j);
        if(m < minlist[4])
        {
            m = 0; im_pixw(i,j,m);
        }
        else
        {
            m = 255;
            im_pixw(i,j,m);
        }
    }
im_outpath(0,-1,0,0);
im_opmode(2,0);
im_thin(0,1,2,4,1); /* thins binary image in buffer 0 */
im_outpath(1,-1,0,0);
}
void mainty()
{
    im_init(0xD000,0x300); /* initialize MVP-AT */
im_clear(5,0);
im_clear(4,0);
im_outpath(0,-1,0,0); /* selects buffer 0 for output */
}
This program is used to read off the pixel coordinates in the camera frame for all the field points taken. The image of the field point is grabbed into the frame buffer and its pixel coordinate determined by moving the cross-hair to that point.

```
#include <stdio.h>
#include <middle.h>
#include <struct.h>
#include <debugat.h>
#include "stepper.h"
#define CLS printf ("\x1B[2J\n")
#define Loc(q,r) printf ("\x1B[%d;%dH",q,r)
int xp = 255;
int yp = 239;
int inc = 25;
update () /* updates AT screen */
{
    int i;
    Loc (18,25) ; printf ("%08d",xp);
    Loc (19,26) ; printf ("%3d",inc);
    Loc (18,55) ; printf ("%08d",yp);
    Loc (19,56) ; printf ("%3d",inc);
    Loc (25,8); printf
        ("Arrow keys move cursor...PgUp,PgDn change step...");
}
cursor (nx,ny) /* draws and updates the cursor */
int nx,ny;
{  
im_opmode (1,0);    /* graphics mode */
im_drawmode (2);    /* XOR drawing mode */
im_setcolor (255);    /* WHITE */
im_move (xp,yp);      /* move to previous position */
im_circle (10);      /* draw a circle, erasing previous */
im_rmove (-20,-20);  /* erase the previous crosshair */
im_rline (40,40);
im_rmove (-40,0);
im_rline (40,-40);
im_move (nx,ny);      /* move to new position */
im_circle (10);      /* draw new circle...etc. etc. */
im_rmove (-20,-20);
im_rline (40,40);
```
im_rmove (-40,0);
im_rline (40,-40);
xp = nx;
yp = ny;
}/* update cursor position */
snap()/* takes a snapshot and draws the first cursor */
{
im_opmode (2,0); /* image-processing mode */
im_snapshot (0); /* snap to page 0 */
im_opmode (1,0); /* graphics mode */
im_drawmode (2); /* XOR */
im_setcolor (255); /* WHITE */
im_move (xp,yp); /* draw first cursor */
im_circle (10);
im_rmove (-20,-20);
im_rline (40,40);
im_rmove (-40,0);
im_rline (40,-40);
}
start()/* initialize MVP-AT */
{
FILE *fp;
int i;
im_init (0xD000,0x300); /* clear pages 2 & 3 */
im_clear (0x300);
im_clear (5,0);
im_clear (4,0);
im_chan (1);
im_sync (1,0);
im_video (1,1);
im_outpath (0,-1,0,0);
CLS;
snap();
update ();
}
main()/* select video buffer for output */
{
char c;
int done=0;
start();
while (!done) {
    while (!kbhit ());
    c = getch ();
    if (c == 0) {
c = getch();
switch (c) {
    case 72: cursor (xp, yp—inc); break;
    case 73: inc*=5; break;
    case 75: cursor (xp—inc, yp); break;
    case 77: cursor (xp+inc, yp); break;
    case 79: done = 1; break;
    case 80: cursor (xp, yp+inc); break;
    case 81: inc = (inc>1 ? inc/5 : 1); break;
    case 82: snap(); break;
    default: putchar (7);
}
update();
}
else {
    if (c< '7' && c>'0') {
        c -= 49;
    }
    else putchar(7);
}
}