

1994

3D information recovery of multiple mobile objects

Sy-Hung Kuo
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_theses

Recommended Citation

Kuo, Sy-Hung, "3D information recovery of multiple mobile objects" (1994). *Master's Theses*. 768.
DOI: <https://doi.org/10.31979/etd.afjt-hv52>
https://scholarworks.sjsu.edu/etd_theses/768

This Thesis is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Theses by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.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.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600



Order Number 1358192

3D information recovery of multiple mobile objects

Kuo, Sy-Hung, M.S.

San Jose State University, 1994

Copyright ©1994 by Kuo, Sy-Hung. All rights reserved.

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106



3D INFORMATION RECOVERY OF MULTIPLE MOBILE OBJECTS

A Thesis

Presented to

The Faculty of the Department of Computer Engineering
San Jose State University

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

by

Sy-Hung Kuo

May, 1994

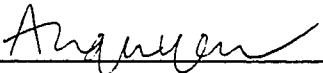
3D INFORMATION RECOVERY OF MULTIPLE MOBILE OBJECTS

© 1994

Sy-Hung Kuo

ALL RIGHTS RESERVED

APPROVED FOR THE DEPARTMENT
OF COMPUTER ENGINEERING

 April 18, 1994

Dr. An H. Nguyen, Computer Engineering

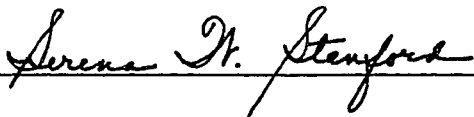


Dr. Ray R. Chen, Electrical Engineering



Dr. Haluk S. Ozemek, Computer Engineering

APPROVED FOR THE UNIVERSITY



ABSTRACT

3D INFORMATION RECOVERY OF MULTIPLE MOBILE OBJECTS

by Sy-Hung Kuo

This thesis is focused on the 3D information recovery of multiple mobile objects in 3D space. Since video images contain only 2D information, techniques must be developed in order to recover the third dimension which is important in identifying the exact location of the object in the 3D space. The research focuses primarily on the development of schemes to recover the 3D location and orientation of an object in the 3D space. Two schemes were developed. The first is based on single camera and the second is based on multiple cameras. Furthermore, an algorithm for tracking multiple mobile objects is also derived. The system is found not only to be effective in telerobotic control based on the top-down approach and machine vision, but also in virtual reality study.

ACKNOWLEDGMENTS

First of all, I would like to thank my parents for giving me the strength and the motivation to fulfill this thesis. Without their support, the thesis would never be accomplished.

I deeply thank my advisor, Professor An H. Nguyen from whom I learned not only science but also the philosophy of life. Without his valuable suggestions, I might not have had the opportunity to carry out the research. I would like to thank Professor Lawrence W. Stark from the University of California, Berkeley, for his precious advice and for supporting us with the video hardware equipment.

I would also like to thank Professor Ray R. Chen and Professor Haluk S. Ozemek who have been giving me their support since I first came to San Jose State University. They have provided me with many suggestions to improve the thesis.

I appreciate Dr. Michael McGreevy at NASA Ames for his precious suggestions and the frame grabber that he kindly lent us. I also want to thank Dr. Jen-Chang Chou who continually gave me encouragement and inspiration.

I am indebted to Victor Chiun and Ya-Mei Kao who read my entire manuscript and offered valuable advice and comments. I also want to thank Christine Yu who helped me collect the huge amount of experimental data.

A special thanks goes to many students in San Jose State University for giving me feedback and opinions to develop a robust and flexible computer vision system.

Table of Contents

Chapter 1 Introduction	1
Chapter 2 Camera Model	3
2.1 Spatial Description of 3D Objects	3
2.1.1 Discussion of Notations	3
2.1.2 Description of an Object Body in 3D Space	4
2.1.2.1 Description of a Frame	5
2.1.2.2 Orthogonal System	5
2.1.2.3 4x4 Homogeneous Transformation Matrix Consisting of 3x3 Rotation Matrix R and 3x1 Translation Matrix P	8
2.1.2.4 Inverse of the 4x4 Homogeneous Transformation Matrix	8
2.1.2.5 Camera Coordinates	8
2.2 Camera Model	9
2.2.1 Ideal Camera Model	9
2.2.2 Basic Geometry of the Camera Model	9
2.2.3 Camera Calibration	13
2.3 Camera Description	16
2.3.1 Universal General Solution for Deriving the Camera Transformation Matrix Based on Oblique Projections	16
2.3.2 Description of an Object in 3D Relative to Camera Coordinates	26

Chapter 3	Low Level Moment Image Processing for Computer Vision	29
3.1	A Survey	29
3.2	Moment image processing	30
3.3	Define Object Area and 2D Location	30
3.4	Central Moments	32
3.5	Define Object Orientation	33
3.6	Moment Binary Image Processing	37
3.7	Object Orientation with PAR Compensation	39
Chapter 4	3D Information Recovery Based on Orthogonal Projections	41
4.1	Single Camera Case	41
4.1.1	3D Recovery Technique with A Single Camera	42
4.1.2	The General Experimental Setup	52
4.1.3	The Experimental Setup for the Particular Single Camera Case	57
4.1.4	Experimental Results for Single Camera Vision System	59
4.2	Two-camera Case	63
4.2.1	3D Recovery Technique with Two Cameras	63
4.2.2	The Experimental Setup for the Particular Two-camera Case	67
4.2.3	Experimental Results for Two-camera Vision System	69

Chapter 5 Object Tracking Algorithm and Applications	72
5.1 A Self-adaptive Algorithm for Object Tracking	72
5.2 Virtual Reality Research	75
5.3 Telerobotic Control	82
Chapter 6 Conclusion and Future Research	85
6.1 Conclusion	85
6.2 Recommendations for Future Research	85
Appendix A Source Code Listing of the Developed Vision System	87
Appendix B Bibliography	162

List of Figures

Figure 2.1	$\{B\} = \{ {}^A_B R, {}^A P_{BORG} \}$.	4
Figure 2.2	Ideal camera model.	6
Figure 2.3	Physical setup of orthogonal system.	7
Figure 2.4	Basic geometry of the camera model.	11
Figure 2.5	The processes for acquiring the image coordinates.	12
Figure 2.6	Perspective project of an object.	15
Figure 2.7	Basic geometry for deriving Universal General Solution.	17
Figures 2.8	An initial condition (top) and the effect after the camera coordinate system aligned with the world coordinate system (bottom).	18
Figures 2.9	Move the camera coordinate system to the location of the object coordinate system (top), the result (bottom).	19
Figures 2.10	Rotate the camera coordinate system α degrees about its Y_C axis (top), the result (bottom).	22
Figures 2.11	Rotate the camera coordinate system $(-\beta)$ degrees about its X_C axis (top), the result (bottom).	23
Figures 2.12	Rotate the camera coordinate system 180 degrees about its Y_C axis (top), the result (bottom).	24
Figure 2.13	Project UP vector onto the camera's image plane to get UP' vector.	25

Figures 2.14	Rotate the camera coordinate system θ degrees about its Z_C axis (top), the result (bottom).	27
Figures 2.15	Translate the camera coordinate system to wP_o (top), the final result (bottom).	28
Figure 4.1	The basic geometry of a single camera.	41
Figure 4.2	The histograms of two selected windows in two cameras.	53
Figure 4.3	Two white ping-pong balls were chosen as the test object.	54
Figure 4.4	Two non-identical cameras were used in the developed vision system.	55
Figure 4.5	The computer setup for the developed vision system.	56
Figure 4.6	The experimental setup for calibrating single camera vision system.	58
Figures 4.7a	MEASURED AREA vs. MEASURED HEIGHT based on single camera system.	61
Figures 4.7b	MEASURED HEIGHT vs. ACTUAL HEIGHT based on single camera system.	62
Figure 4.8	The basic geometry setup for the two-camera vision system.	64

Figure 4.9	The experimental setup for calibrating two-camera vision system.	68
Figures 4.10	MEASURED HEIGHT vs. ACTUAL HEIGHT based on two-camera system.	71
Figure 5.1	A black rectangle target was attached to the helmet display.	76
Figure 5.2	Target tracking experiment 1.	77
Figure 5.3	Target tracking experiment 2.	77
Figure 5.4	Target tracking experiment 3.	78
Figure 5.5	Target tracking experiment 4.	78
Figure 5.6	Two ping-pong balls glued together can be a nice target.	79
Figure 5.7	A close view of two white ping-pong balls.	79
Figure 5.8	Experimental tracking patterns.	80
Figure 5.9	Experimental tracking patterns.	81
Figure 5.10	OSVEs for the 3D kinematic recovery of a robot.	83
Figure 5.11	In the kinematic recovery, full kinematic information about the robot, including link length and link direction, was utilized in the experiment.	84

List of Tables

Table 4.1	Experimental Data for Applying the Two Parameters Formula.	59
Table 4.2	Experimental Data for Applying the Three Parameters Formula.	60
Table 4.3	Experimental Data for Tracking the Object at the Center of the Grid Board.	69
Table 4.4	Experimental Data for Tracking the Object at the Corner of the Grid Board.	70

Chapter 1

Introduction

Many creatures on the earth have two eyes that help them detect a certain range of frequencies transmitted from their surrounding environment. In humans, vision plays a key role in our sense system. Among the five senses, it is one of our most important sources of getting information from the world around us. Without vision, the ability to perform our daily tasks is dramatically limited. Correspondingly, most of the robots in the factory today only perform simple tasks for lack of the visual feedback.

Today, computer vision system is being applied to many kinds of applications, including robotic control, machine vision, security system, remote sensing, and virtual reality study.

Some early work has been done in the field of vision systems. In the 1970s, Irwin Sobel at Stanford had done some early research in machine vision for robotic control. In the 1980s, Roger Tsai at IBM had done some important work in machine vision, especially in the camera calibration. In the 1990s, Professors Nguyen and Stark at UC Berkeley had developed a vision system for telerobotic and medical applications.

Basically, my research is focused on the 3D information recovery of multiple mobile objects in 3D space. Since video images contain only 2D information, techniques must be developed in order to recover the third dimension which is important in identifying the exact location of the object in the 3D space. This thesis focuses primarily on the development of schemes to recover the 3D location and orientation of an object in the 3D space. Two schemes were developed. The first is based on single camera and the second is based on multiple cameras. Furthermore, an algorithm for tracking multiple mobile objects is also derived. The system is found not only to be effective in telerobotic control based on the top-down approach and machine vision, but also in virtual reality study.

Organization of the thesis includes six chapters:

- Chapter 1: Introduction.
- Chapter 2: Camera Model.
- Chapter 3: Low Level Moment Image Processing for Computer Vision.
- Chapter 4: 3D Information Recovery Based on Orthogonal Projections.
- Chapter 5: Object Tracking Algorithm and Applications.
- Chapter 6: Conclusion and Future Research.

Chapter 2

Camera Model

2.1 Spatial Description of 3D Objects

2.1.1 Discussion of Notations

In my thesis, I use the following conventions. These conventions can be found in modern robotic text books Craig [1989].

- Variables written in uppercase represent vectors or matrices. Lowercase variables are scalars. For examples, $R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ is a rotation matrix; $P = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$ is a vector P which has three components p_x , p_y , and p_z .
- Leading subscripts and superscripts identify which coordinate system a quantity is described in. For instance, ${}^A P$ represents a position vector described in coordinate system $\{A\}$, and ${}^A_B R$ is a rotation matrix which specifies the relationship between coordinate systems $\{A\}$ and $\{B\}$ (see Figure 2.1).
- Trailing superscripts are used for indicating the inverse or transpose of a matrix, e.g., R^{-1} , R^T .

- Trailing subscripts are not subject to any strict convention but may indicate a vector component (e.g., x , y , or z) or may be used as a description as in P_{object} , the position of a object.

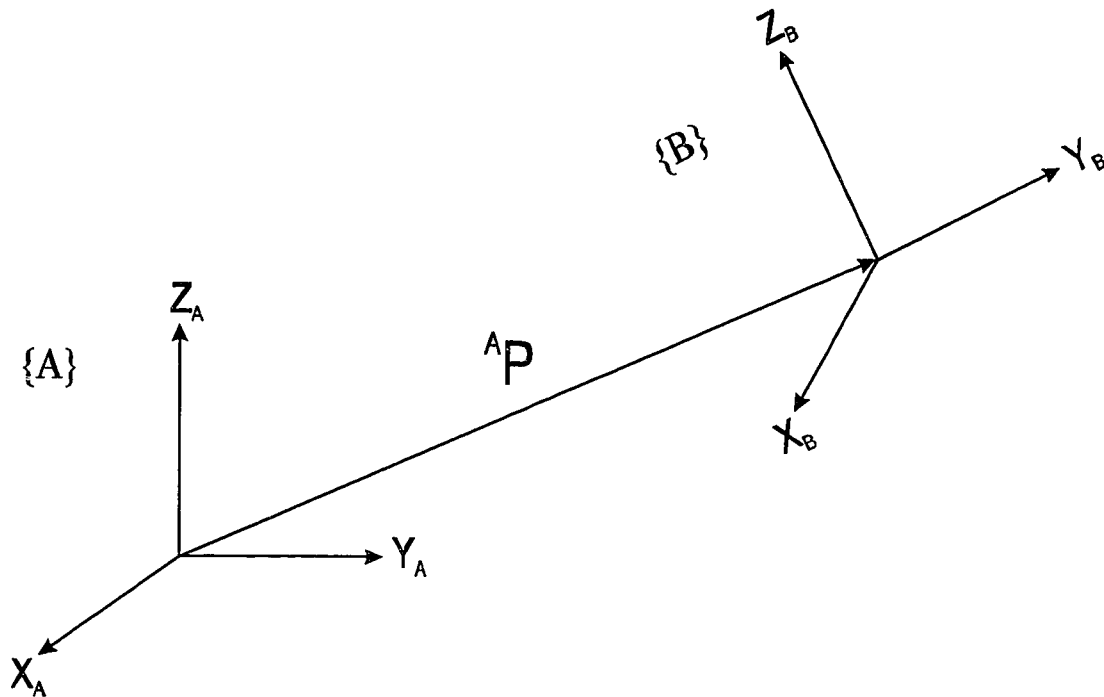


Figure 2.1 $\{B\} = \{{}_B^A R, {}^A P_{BORG}\}$.

2.1.2 Description of an Object Body in 3D Space

In computer vision and robotics we are constantly concerned with the location of objects in three-dimensional space. At a coarse but important level, these objects are described by just two attributes: their position and orientation.

2.1.2.1 Description of a Frame

In order to describe the position and the orientation of a body in space we will always attach a coordinate system, or a **frame**, rigidly to the object. We then proceed to describe the position and the orientation of this frame with respect to some reference coordinate systems.

A frame is a set of four vectors providing information about the position and the orientation. Equivalently, the description of a frame can be thought of as a position vector and a rotation matrix. Note that a frame is a coordinate system, where in addition to the orientation we give a position vector which locates its origin relative to some other embedding frame. For example, frame $\{B\}$ is described by a 3×3 ${}^A R_B$ and a 3×1 ${}^A P_{BORG}$, where ${}^A P_{BORG}$ is the vector which locates the origin of the frame $\{B\}$ relative to $\{A\}$ (see Figure 2.1):

$$\{B\} = \{{}^A R_B, {}^A P_{BORG}\}. \quad (\text{Eq. 2.1})$$

2.1.2.2 Orthogonal System

Consider the problem of projecting a 3D point $P = (x_p, y_p, z_p)$ onto some plane. There are many ways to do this. The simplest way is to discard one component, say the z -component, so that P projects to $P' = (x_p, y_p)$. This is equivalent to projecting the point onto the X_1, Y_1 -plane in Figure 2.2. An orthogonal system in terms of camera calibration for computer vision purpose is shown in Figure 2.3.

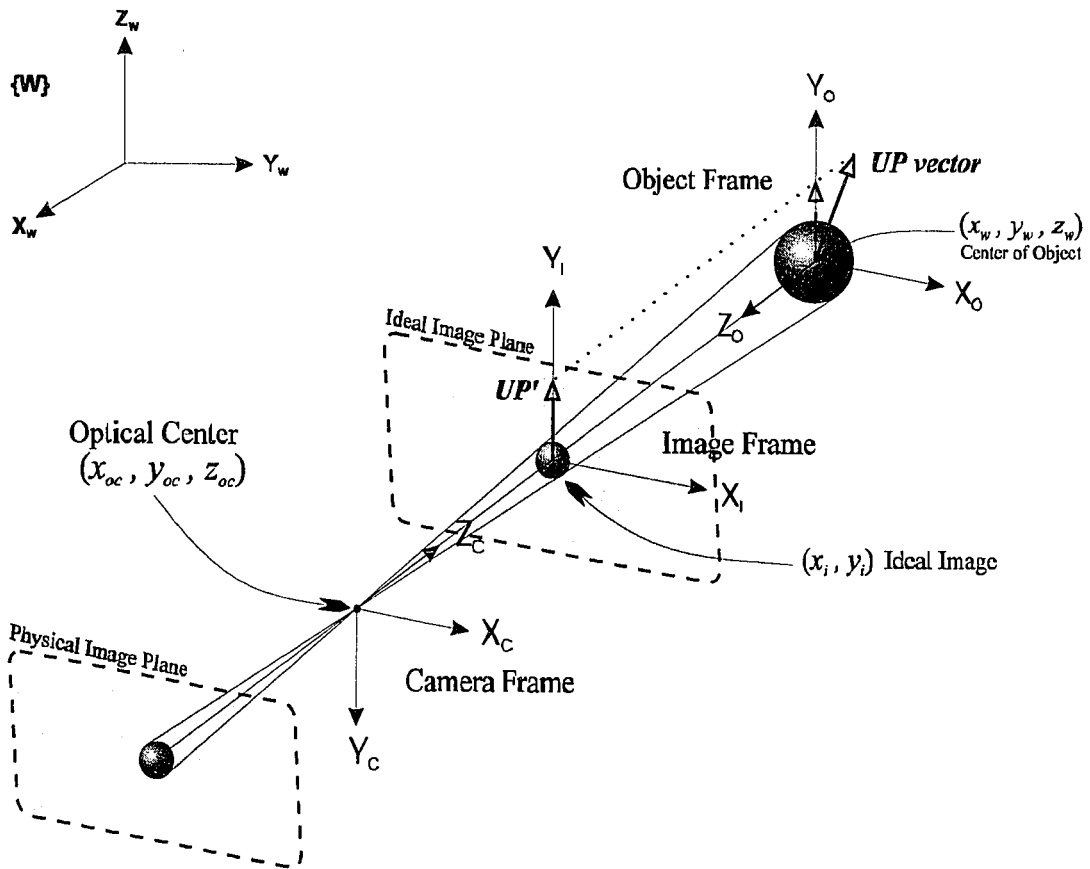


Figure 2.2 Ideal camera model.

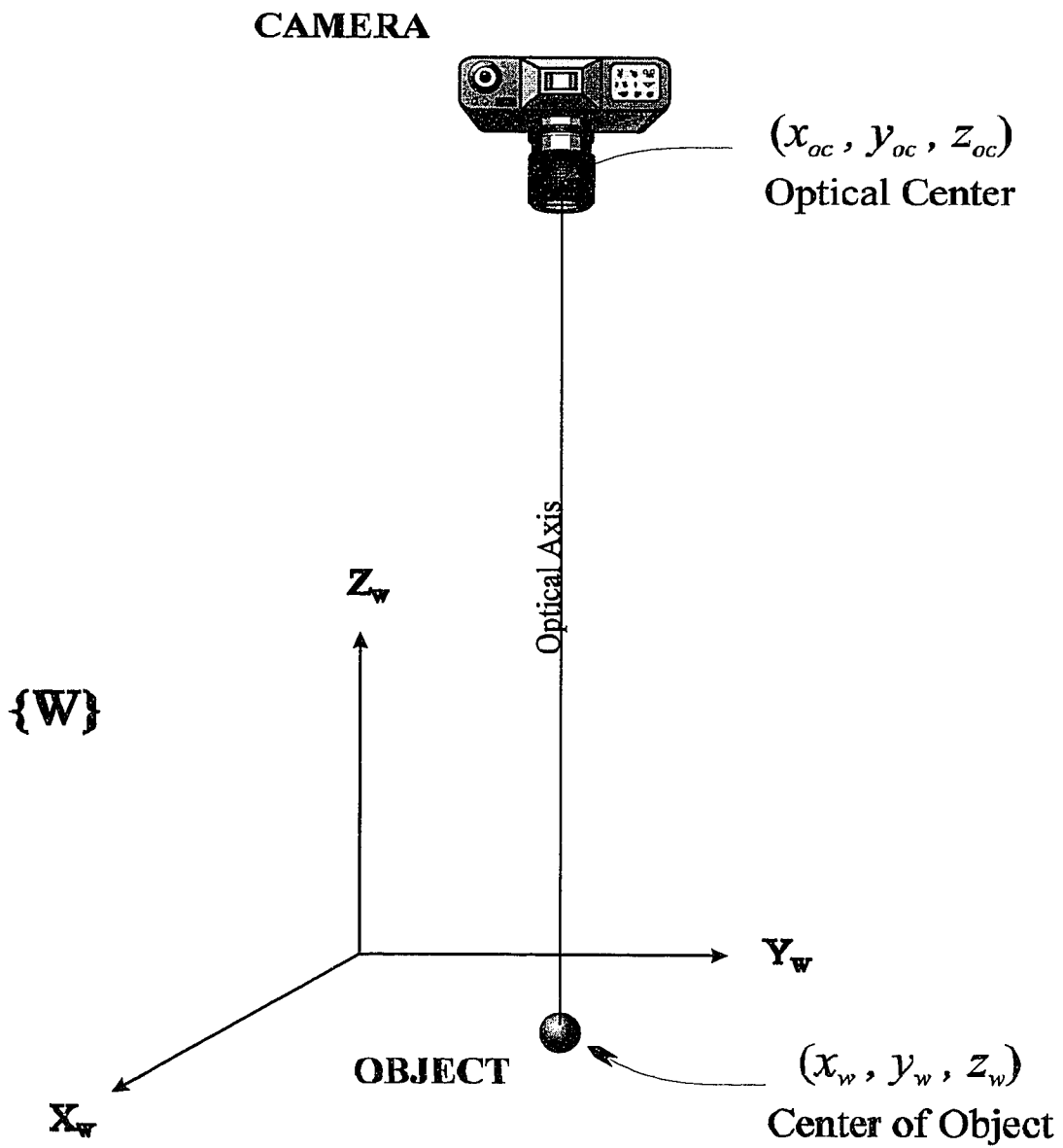


Figure 2.3 Physical setup of orthogonal system.

2.1.2.3 4x4 Homogeneous Transformation Matrix Consisting of 3x3 Rotation Matrix R and 3x1 Translation Matrix P

$$T = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ t_{41} & t_{42} & t_{43} & t_{44} \end{bmatrix} = \begin{bmatrix} R & P \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & P_x \\ r_{21} & r_{22} & r_{23} & P_y \\ r_{31} & r_{32} & r_{33} & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (\text{Eq. 2.2})$$

2.1.2.4 Inverse of the 4x4 Homogeneous Transformation Matrix

Due to the orthonormality of the coordinate system, $R^{-1} = R^T$,

$$T^{-1} = \begin{bmatrix} R & P \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} R^T & -R^T P \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (\text{Eq. 2.3})$$

2.1.2.5 Camera Coordinates

In my thesis, the camera coordinate system is defined as (X_C, Y_C, Z_C) . The Z_C axis lies along the camera optical axis, with the origin at the optical center of the lens and the positive direction outwards into the external world. The Y_C axis is chosen to be aligned with the projection of the UP vector on the camera's image plane (the UP vector assignment is the key role in my Oblique Projection scheme; see detailed description in section 2.31), and the choice of the X_C axis makes up the right-handed coordinate system. We choose this convention because it is consistent with the video display standard and our robotic applications.

2.2 Camera Model

2.2.1 Ideal Camera Model

The ideal camera model assumes that the lens can be replaced by a pin hole, so that the image of an object point could be found by drawing a straight line from the object through the pin hole, to the physical image plane. The location of this pin hole is called the optical center of the lens.

An ideal camera can be thought of as forming an image that is a point projection of its field of view (see detail description in section 2.4.1). This is schematically represented in Figure 2.2 by an optical center and an ideal image plane. The ideal image plane is a reflected version of the physical image plane formed by the lens.

Furthermore, the digitized image coordinate system must be aligned with the camera coordinate system. If the digitized image coordinate system is not aligned with the camera coordinate system, the image will appear to be rotated. In addition to the possibility of image rotation, there is the possibility of image translation, due to the center of the digitized image coordinate system not being coincident with the optical axis.

2.2.2 Basic Geometry of the Camera Model

Figure 2.4 illustrates the basic geometry of the camera model, in which the Z_c axis is the same as the optical axis. ${}^wP_O = (x_w, y_w, z_w)$ is the coordinates of the object point in the 3D world coordinate system. ${}^cP_O = (x_c, y_c, z_c)$ is the coordinates of the

object point in the 3D camera coordinate system, which is located at the optical center P_{oc} . (X_I, Y_I) is the image coordinate system centered at P_{ic} (intersection of the optical axis Z_C and the ideal image plane) and parallel to X_C and Y_C axes. f is the distance between the ideal image plane and the optical center. (x_i, y_i) are the image coordinates of (x_c, y_c, z_c) projected on the ideal image plane. ${}^D P_O = (x_d, y_d)$ is the digitized image coordinates displayed on the screen; it is measured in the number of pixels (see Figure 2.5).

Distortions due to lens aberrations or imperfect video camera and computer electronics are not taken into account in our camera model, because these have not been shown to be the major sources of errors. If they do become a problem, they can be measured and compensated for (see Sobel [1970] for detail). However, the distortion problem was studied by Tsai [1986].

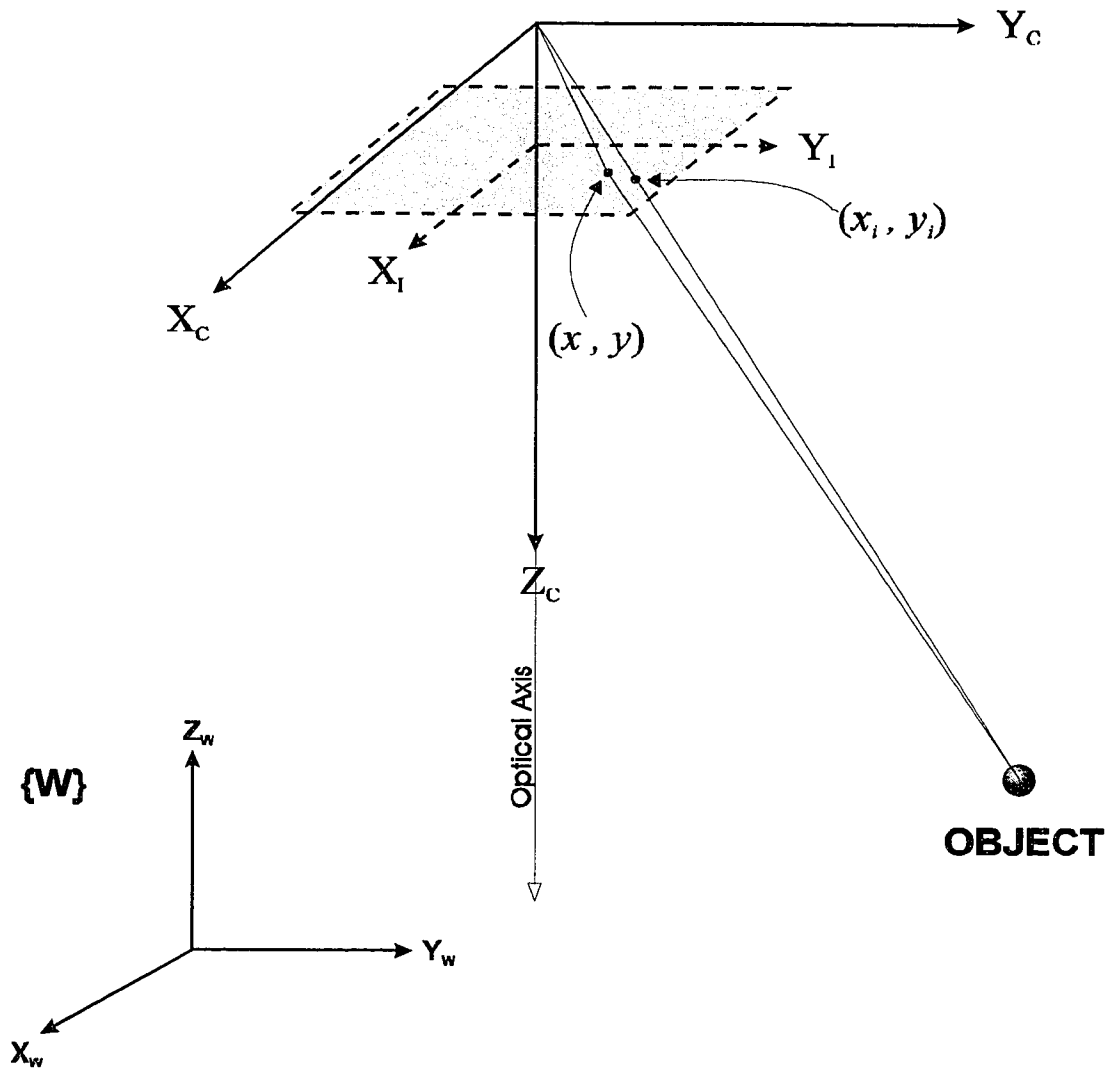


Figure 2.4 Basic geometry of the camera model.

Note: The projection of the object's center on the ideal image plane is (x_i, y_i) for the ideal case. However, with distortion, the actual projection of the object's center will be (x, y) .

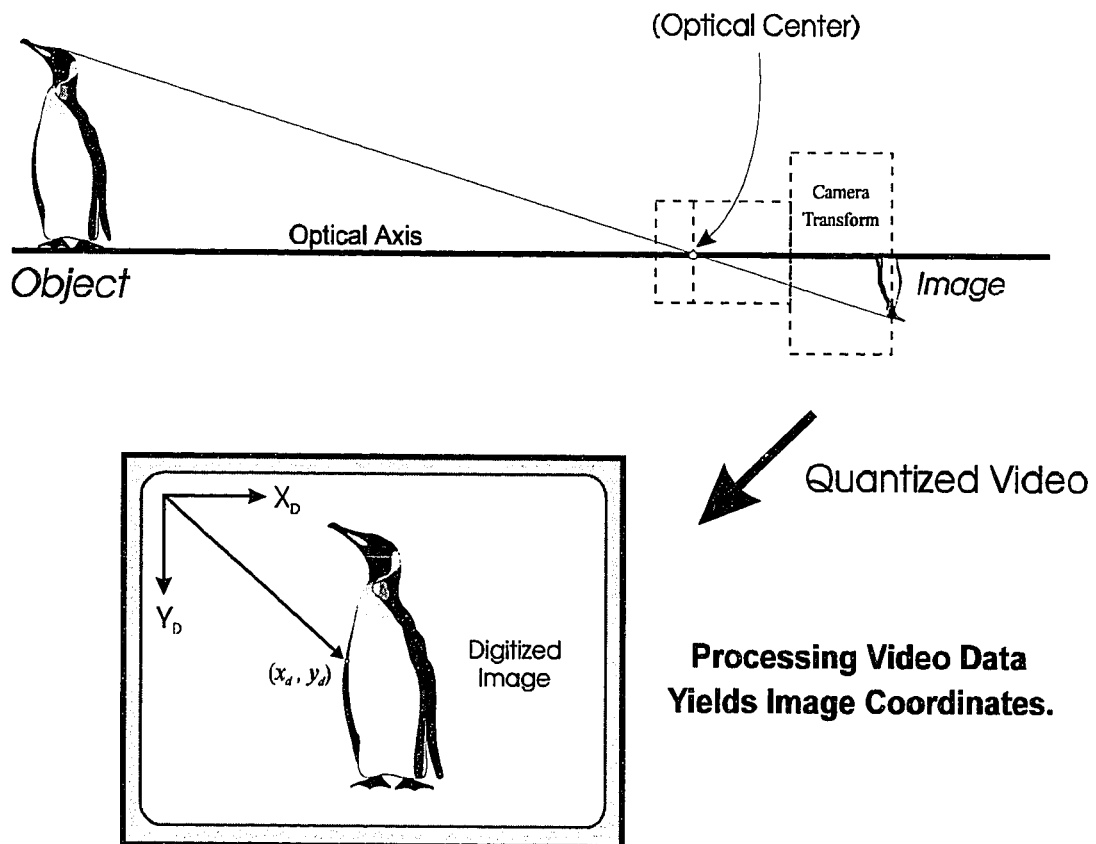


Figure 2.5 The processes for acquiring the image coordinates.

2.2.3 Camera Calibration

Camera calibration is the process of determining the internal camera geometric and optical characteristics (intrinsic parameters) and the 3D position and orientation of the camera frame relative to a world coordinate system (extrinsic parameters).

The overall transformation from (x_w, y_w, z_w) to (x_d, y_d) is shown as follows:

Step I: Transform from the object point ${}^wP_O = (x_w, y_w, z_w)$ in the world coordinate system to the object point ${}^cP_O = (x_c, y_c, z_c)$ in the camera coordinate system,

$${}^cP_O = {}^cT {}^wP_O = {}^wT^{-1} {}^wP_O = \begin{bmatrix} {}^wR^T & -{}^wR^T {}^wP_C \\ 0 & 0 & 0 & 1 \end{bmatrix} {}^wP_O, \quad (\text{Eq. 2.4})$$

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = {}^wT^{-1} {}^wP_O = \begin{bmatrix} {}^wR^T & -{}^wR^T {}^wP_C \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}, \quad (\text{Eq. 2.5})$$

where wR is the 3x3 camera rotation matrix relative to the world coordinate system, wP_C is the 3x1 camera translation matrix relative to the world coordinate system.

$${}^wR = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \quad {}^wP_C = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix}. \quad (\text{Eq. 2.6})$$

Parameters to be calibrated: wR and wP_C .

Step II: Transform from 3D camera coordinate (x_c, y_c, z_c) to ideal (undistorted) image coordinate (x_i, y_i) using perspective projection (see Figure 2.6) with pin hole camera geometry,

$$x_i = f \frac{x_c}{z_c}, \quad (\text{Eq. 2.7a})$$

$$y_i = f \frac{y_c}{z_c}. \quad (\text{Eq. 2.7b})$$

Parameters to be calibrated: effective focal length f .

Step III: Transform from ideal image coordinate (x_i, y_i) to digitized image coordinate (x_d, y_d) ,

$$x_d = \frac{x_i}{d_x} + c_x, \quad (\text{Eq. 2.8a})$$

$$y_d = \frac{y_i}{d_y} + c_y, \quad (\text{Eq. 2.8b})$$

where

(x_d, y_d) : row and column number of image pixel in the memory of frame grabber,

(c_x, c_y) : the center of digitized image coordinate system, their values are dependent on the image size,

d_x : distance between adjacent sensor elements in X direction,

d_y : distance between adjacent sensor elements in Y direction.

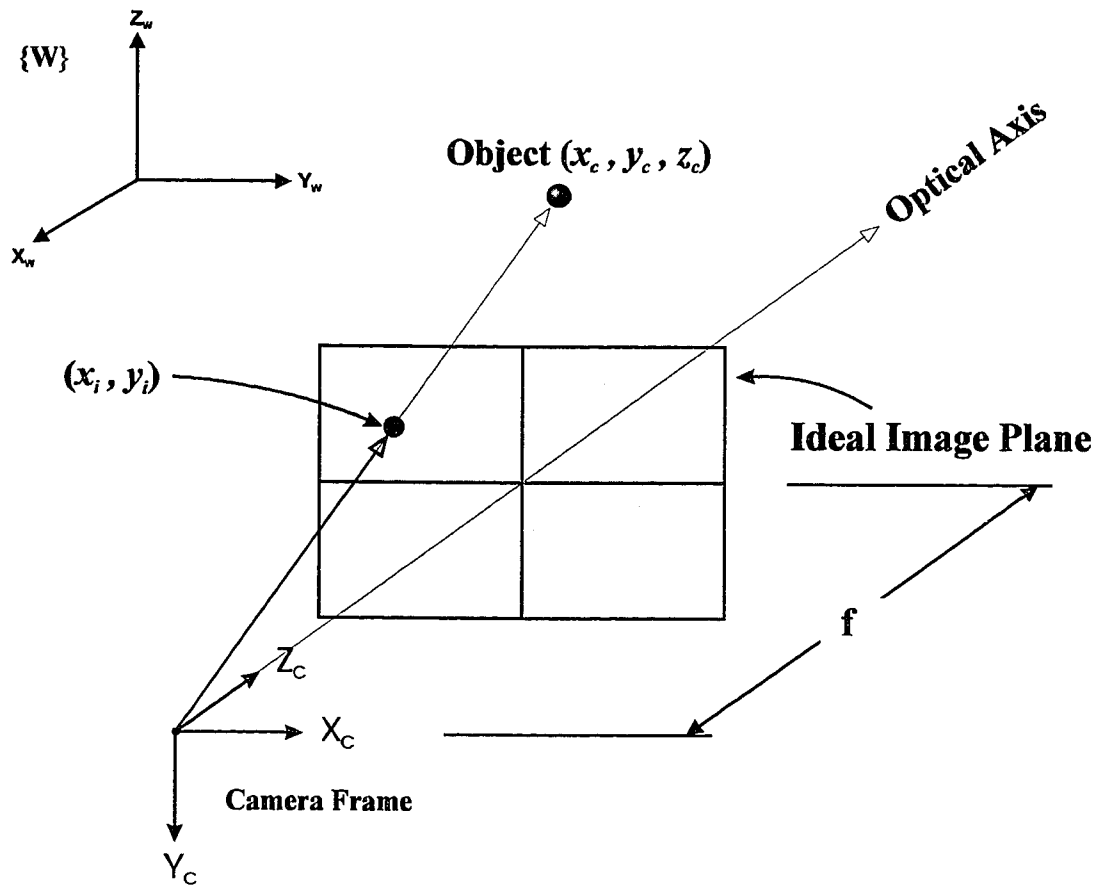


Figure 2.6 Perspective project of an object.

$$x_i = f \frac{x_c}{z_c}, \quad y_i = f \frac{y_c}{z_c}.$$

2.3 Camera Description

2.3.1 Universal General Solution for Deriving the Camera

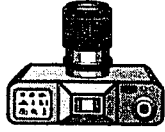
Transformation Matrix Based on Oblique Projections

Scenario: Given any two points in the space, one for the location wP_o of the object, one for the **future** location wP_c of the camera (see Figure 2.7). The goal is to find the description of the camera so that the object can be described in terms of the camera coordinates. I derive an algorithm to acquire a general solution to the problem in which the object's UP vector is taken into consideration. Therefore, by applying the algorithm, the Y_c axis of the camera coordinates can be aligned with the projection of the object's UP vector on the camera's image plane. The algorithm consists of nine steps as follows:

The nine-step Universal General Solution is derived under the following sequence :

Step 1: Initialize the camera orientation by replacing the camera rotation matrix with an identity matrix. This will align the camera coordinate system with the world coordinate system (see Figure 2.8, bottom).

Step 2: Move the camera coordinate system to the location of the object coordinate system (see Figures 2.9, top and bottom).



Initial Location of Camera

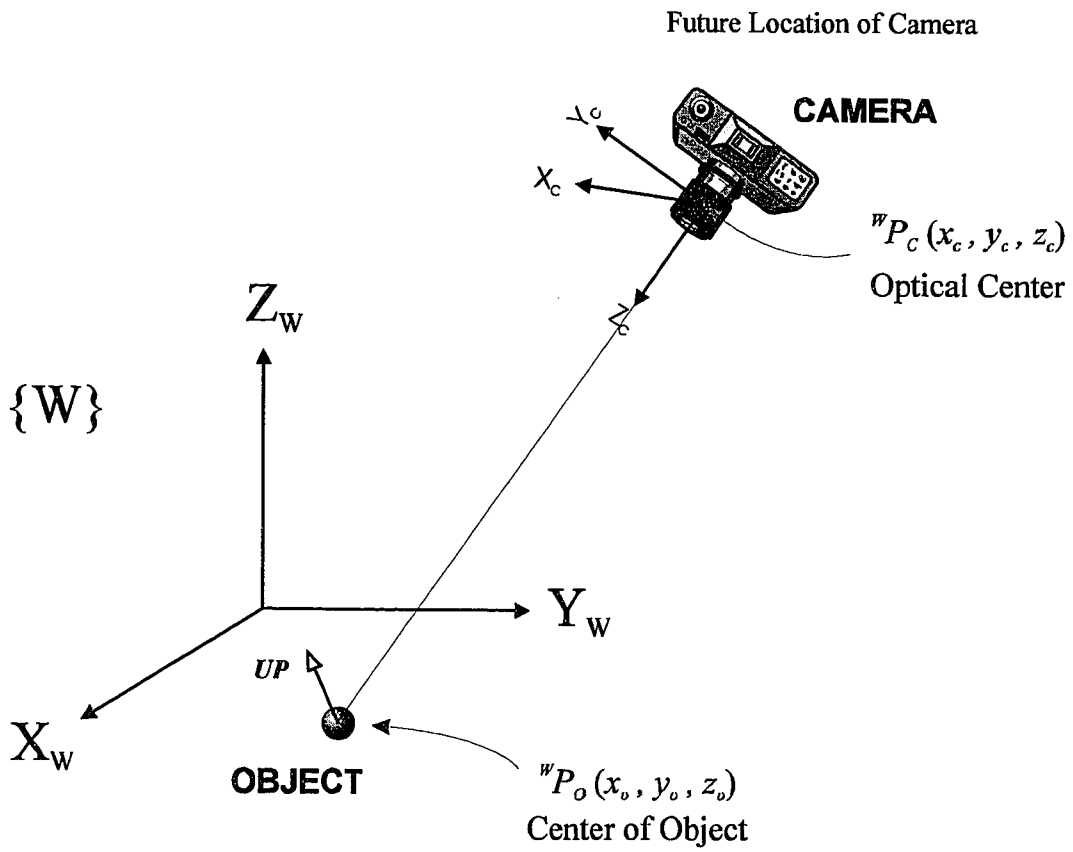
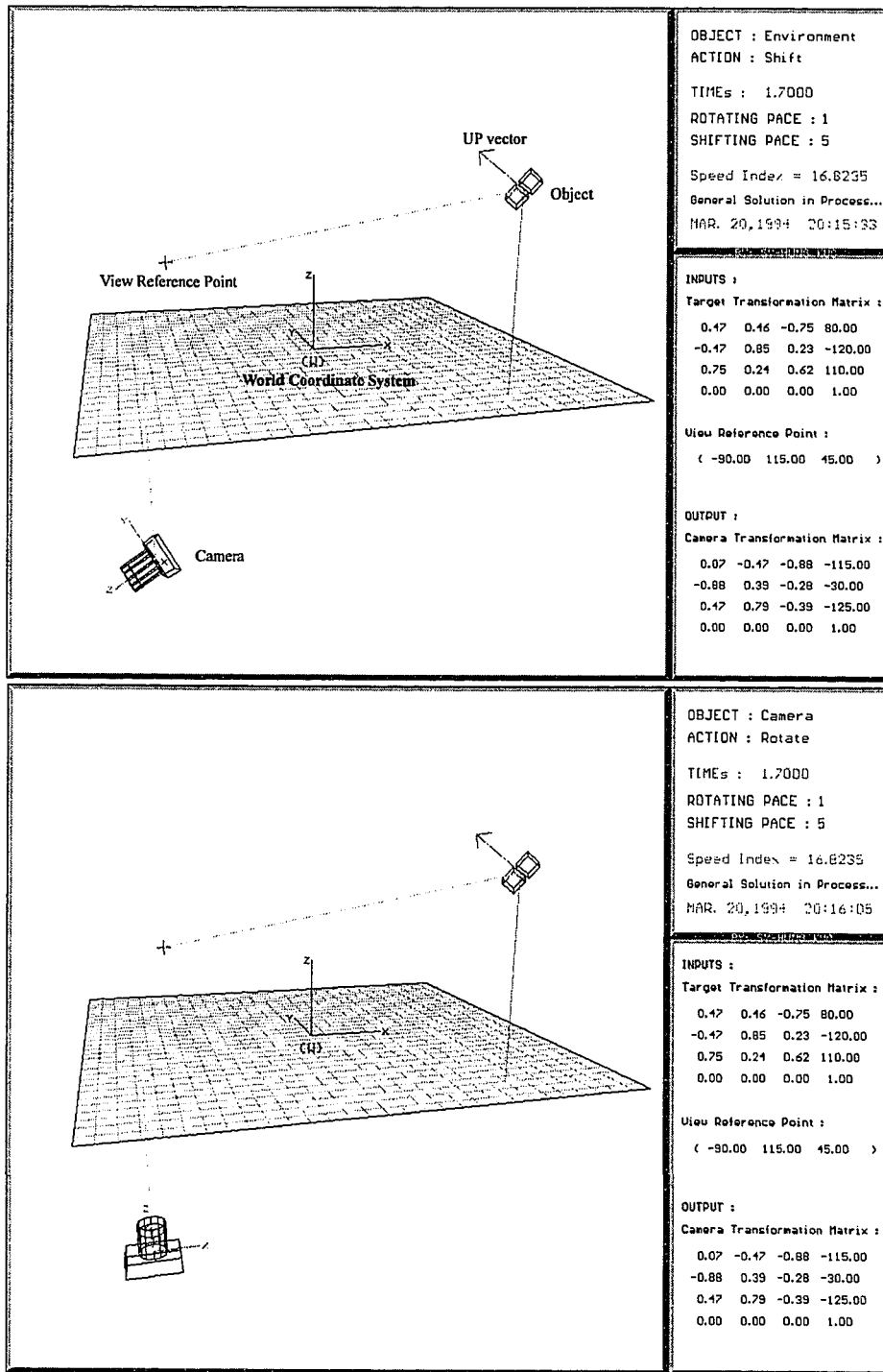
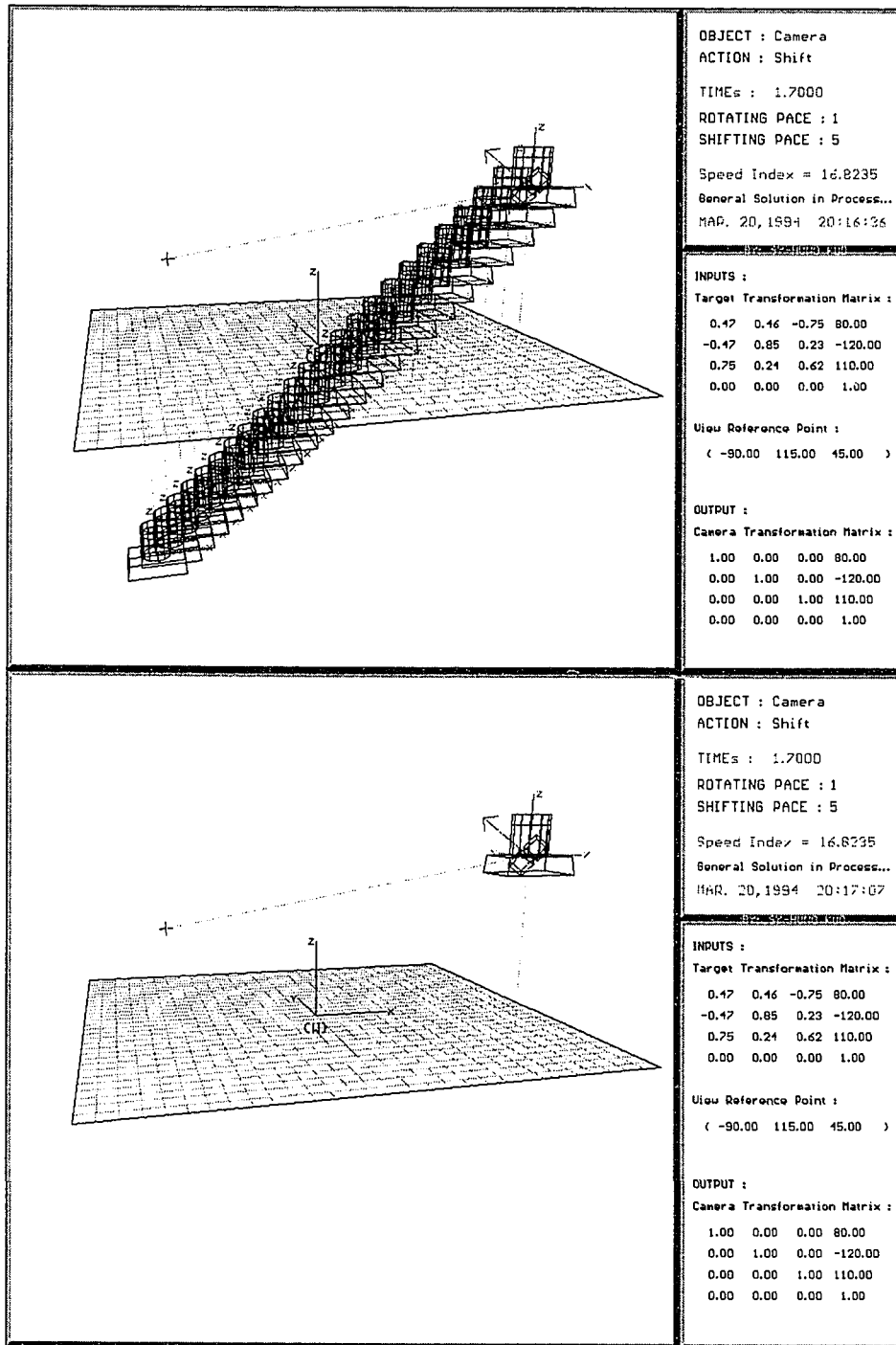


Figure 2.7 Basic geometry for deriving Universal General Solution.

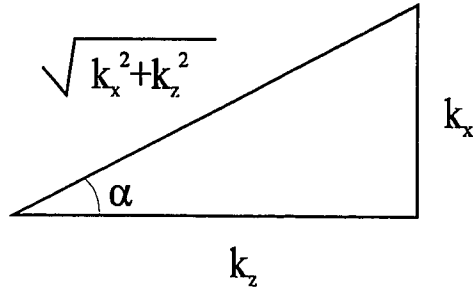


Figures 2.8 An initial condition (top) and the effect after the camera coordinate system aligned with the world coordinate system (bottom).



Figures 2.9 Move the camera coordinate system to the location of the object coordinate system (top), the result (bottom).

Step 3: Rotate the camera coordinate system α degrees about its Y_C axis
 (see Figures 2.10, top and bottom). α is computed as follows:



$$\alpha = \tan^{-1} \frac{k_x}{k_z} = \text{Atan2}(k_x, k_z), \quad (\text{Eq. 2.9})$$

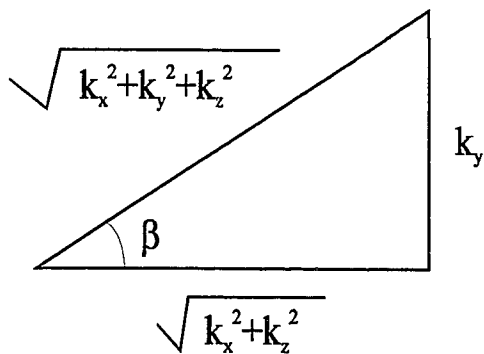
where

$$k_x = x_c - x_o,$$

$$k_y = y_c - y_o,$$

$$k_z = z_c - z_o.$$

Step 4: Rotate the camera coordinate system $(-\beta)$ degrees about its X_C axis
 (see Figures 2.11, Top and Bottom). β is computed as follows:



$$\beta = \tan^{-1} \frac{k_y}{\sqrt{k_x^2 + k_z^2}} = A \tan 2(k_y, \sqrt{k_x^2 + k_z^2}), \quad (\text{Eq. 2.10})$$

where

$$k_x = x_c - x_o,$$

$$k_y = y_c - y_o,$$

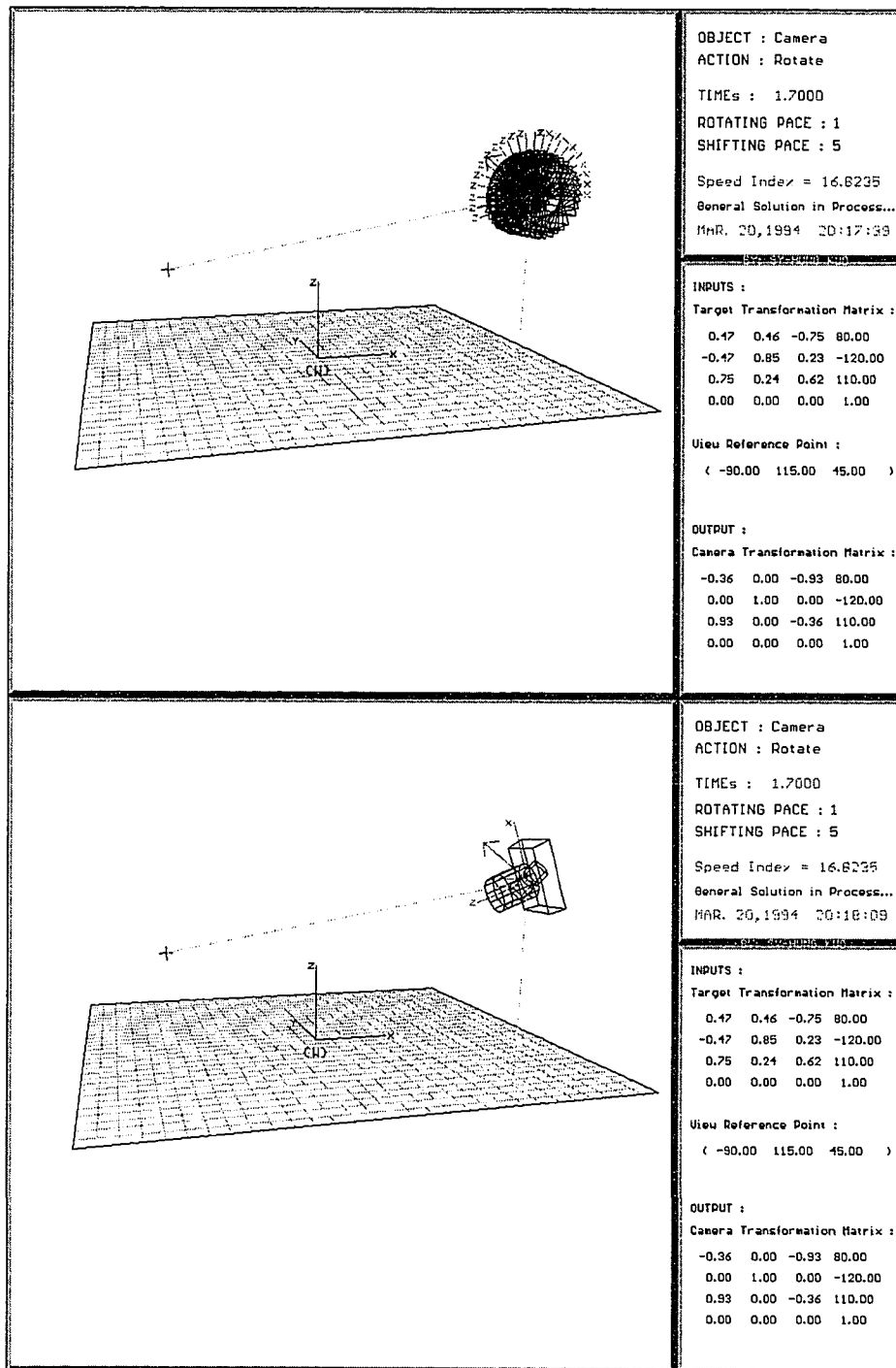
$$k_z = z_c - z_o.$$

Note: Step 3 and Step 4 are arranged to adjust the camera's Z_c axis to be in parallel with future camera axis that aligns with the straight line connecting the View Reference Point and the center of the object.

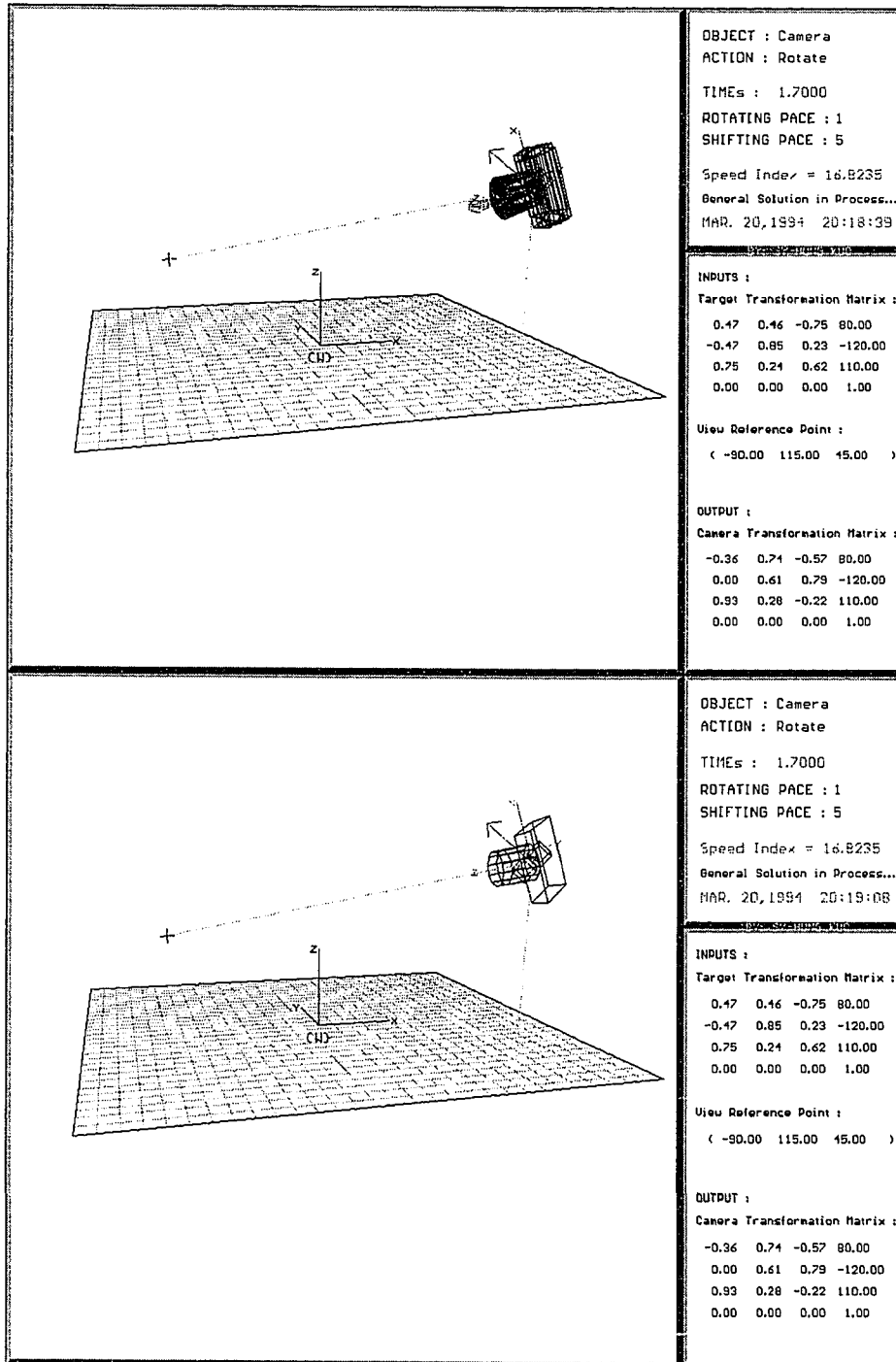
Step 5: Rotate the camera coordinate system 180 degrees about its Y_c axis, so that the camera's Z_c axis can point to the object directly (see Figures 2.12, top and bottom).

Step 6: Project UP vector onto the camera's image plane to get UP' vector (see Figure 2.13).

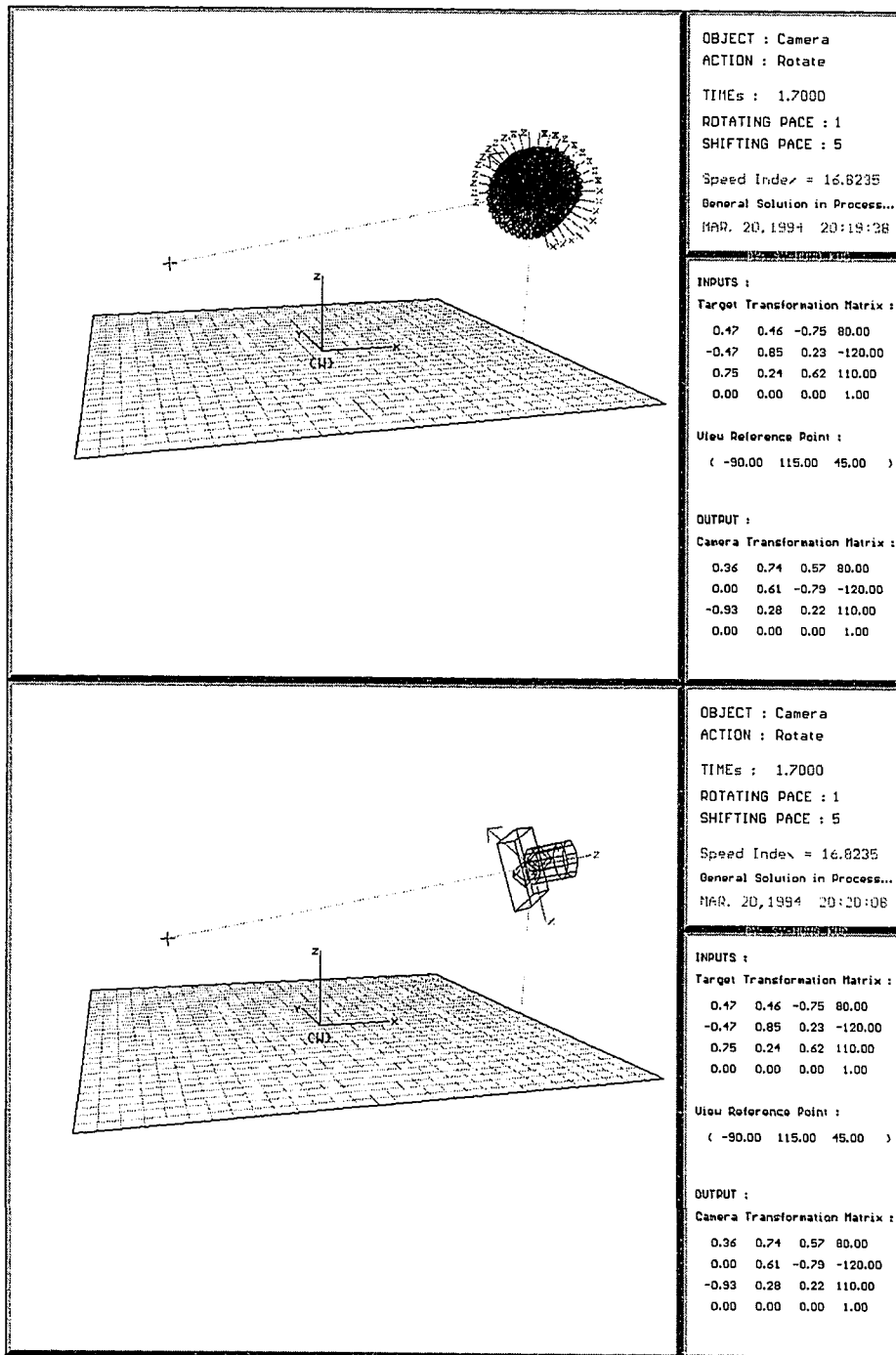
$$UP' = UP - (UP \cdot Z_c) Z_c. \quad (\text{Eq. 2.11})$$



Figures 2.10 Rotate the camera coordinate system α degrees about its Y_C axis (top), the result (bottom).



Figures 2.11 Rotate the camera coordinate system ($-\beta$) degrees about its X_C axis (top), the result (bottom).



Figures 2.12 Rotate the camera coordinate system 180 degrees about its Y_C axis (top), the result (bottom).

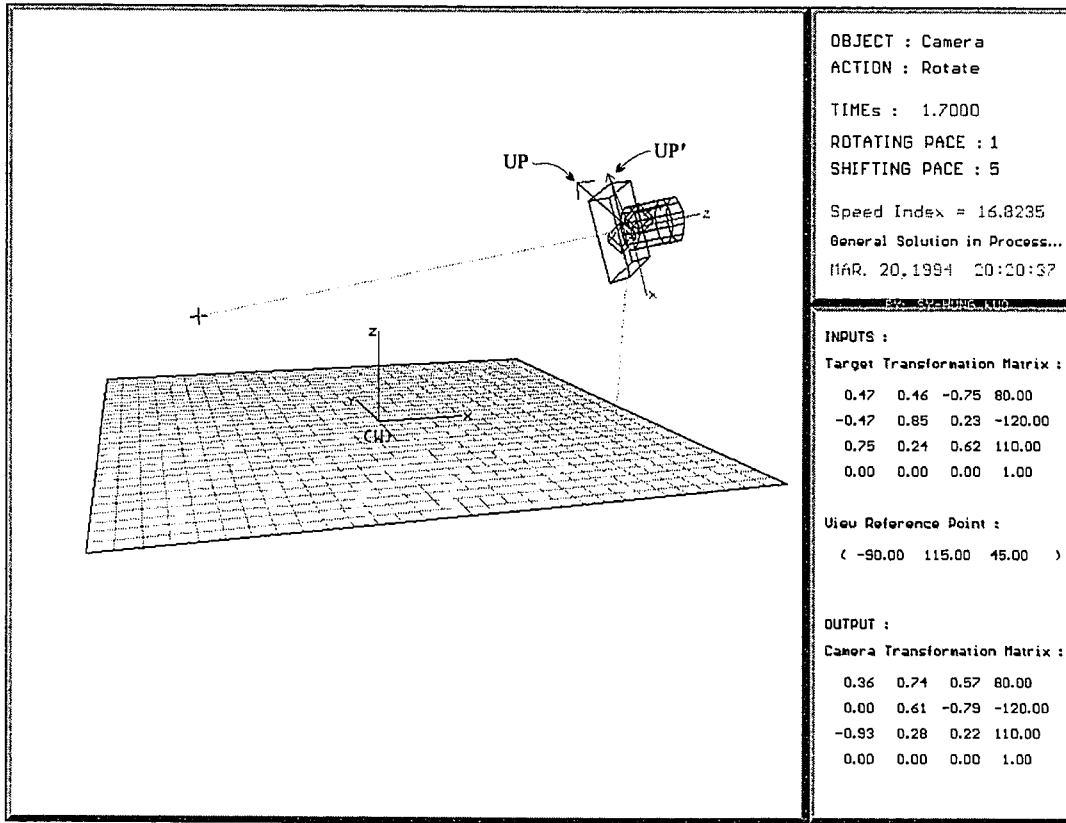


Figure 2.13 Project UP vector onto the camera's image plane to get UP' vector.

Step 7: Calculate θ between UP' vector and camera's Y_C axis.

$$\theta = \cos^{-1}\left(\frac{UP' \cdot Y_C}{|UP'| |Y_C|}\right). \quad (\text{Eq. 2.12})$$

Step 8: Rotate the camera coordinate system θ degrees about its Z_C axis.

It requires predicting the direction, either clockwise or counterclockwise (see Figure 2.14, top), for rotating the camera's Z_C axis to obtain the desired camera orientation (see Figure 2.14, bottom).

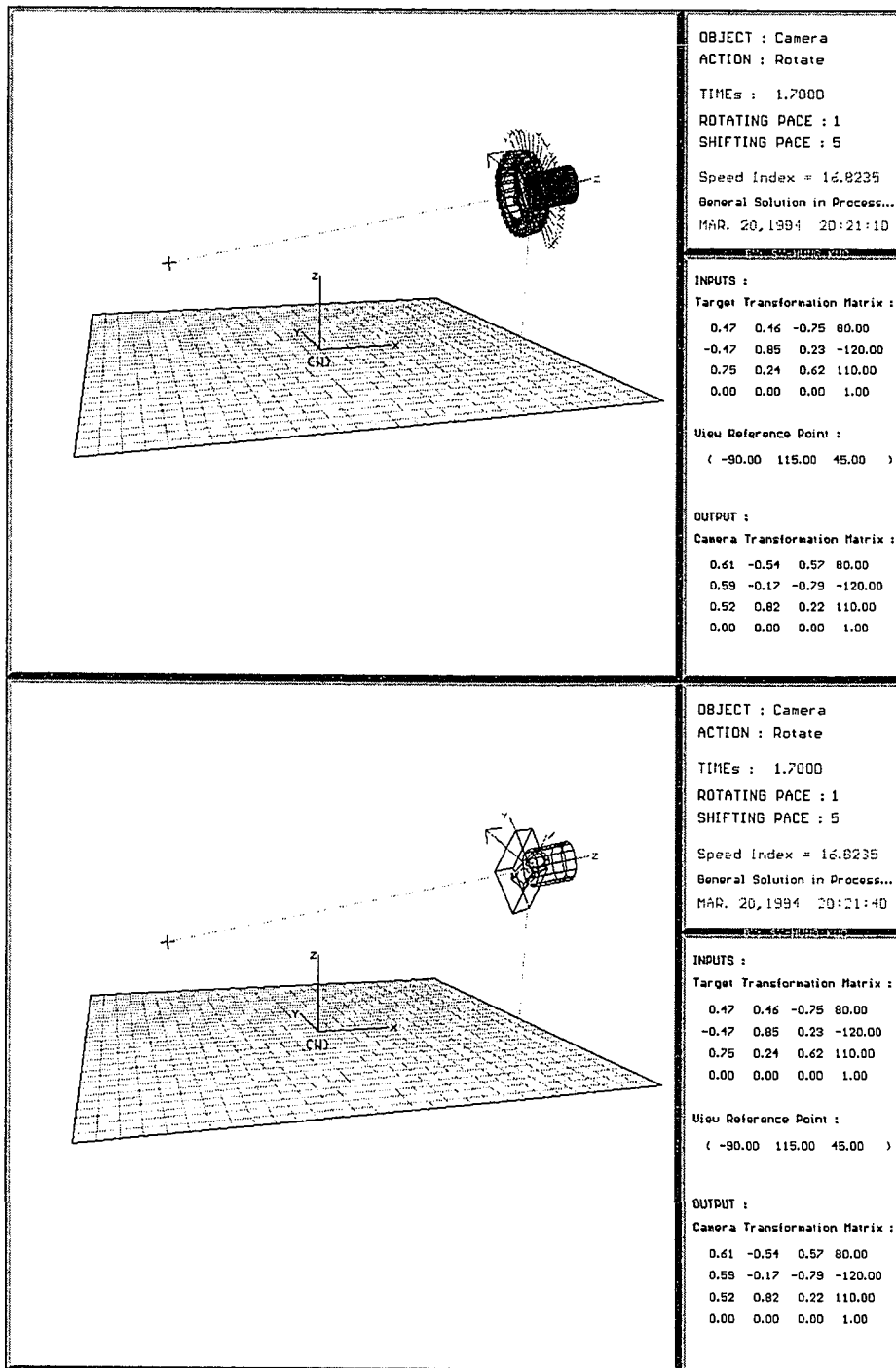
Step 9: Translate the camera coordinate system to wP_O .

Now we have a right view for camera to look at the object and capture a series of appropriate images for computer vision and robotic applications (see Figures 2.15, top and bottom).

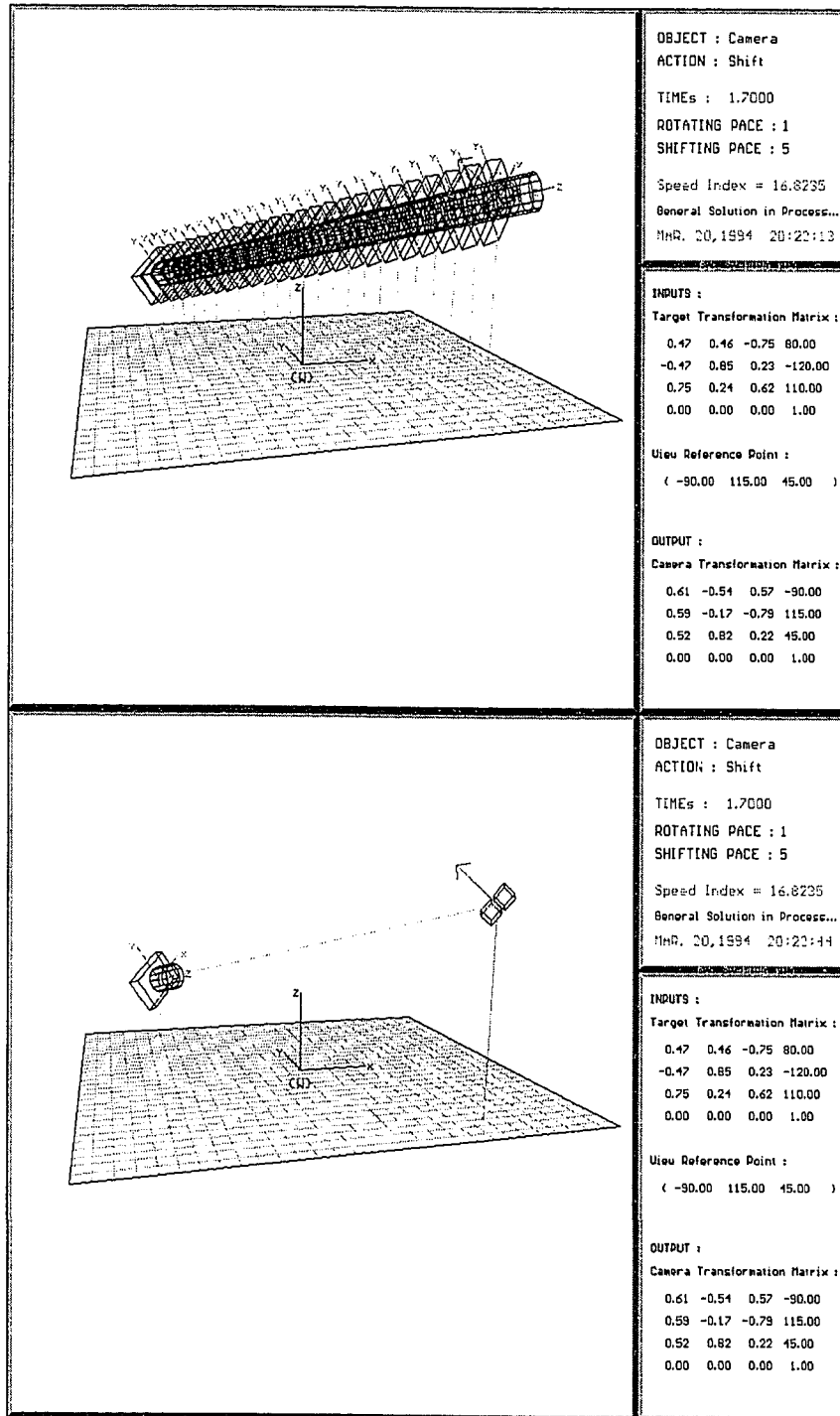
2.3.2 Description of an Object in 3D Relative to Camera Coordinates

Generally, we describe the position and the orientation of an object in space relative to a camera coordinate system with a 4x4 homogeneous transformation matrix. Assuming cT represents the transformation matrix of the object in the 3D space relative to the camera coordinate system; wT and oT represent the transformation matrices of the camera and the object in the 3D space relative to the world coordinate system respectively. We have

$${}^cT = {}^cT {}^wT {}^oT = {}^cT^{-1} {}^wT = \begin{bmatrix} {}^wR^T & -{}^wR^T {}^wP_C \\ 0 & 1 \end{bmatrix} \begin{bmatrix} {}^oR & {}^oP_O \\ 0 & 1 \end{bmatrix}. \quad (\text{Eq. 2.13})$$



Figures 2.14 Rotate the camera coordinate system θ degrees about its Z_C axis (top), the result (bottom).



Figures 2.15 Translate the camera coordinate system to ${}^W P_0$ (top), the final result (bottom).

Chapter 3

Low Level Moment Image Processing for Computer Vision

3.1 A Survey

Gray level image processing based on the moment approach appears to be more accurate than binary because the computation utilizes raw data. Furthermore, gray level processing does not require thresholding, which in certain situations is rather delicate and difficult to carry out automatically.

However, there are several drawbacks to gray level image processing using the moment approach that made the scheme less attractive to our application. First of all, gray level image processing requires a large number of calculations, since all computations are applied indiscriminately to all pixels. Secondly, the zeroth order moment does not yield the area information as binary case does. In the binary image processing, m_0 is the integration of all pixels inside the Regions of Interest (ROI). Thirdly, the centroid cannot be accurately computed because of the influence of the image background and the uneven light distribution of the object foreground. The orientation also suffers a similar effect inherent in the gray level scheme. As a result, the moment binary image processing scheme was chosen to obtain visual information for my object tracking subsystem. The price I had to pay was to develop a reliable procedure for automatic threshold level detection.

Binary image processing involves a thresholding step that separates the object's foreground from its background. All pixels above the threshold level are treated as one, otherwise they are treated as zero (or vice versa). As a matter of fact, binary images greatly reduce the number of multiplications and eliminate the expensive computations involved with the background pixels.

3.2 Moment image processing

Moments have been used in character recognition by Hu [1962], and later in object recognition by Smith [1971], Dudani [1977], and Casasent [1982]. Zahalak [1981] applied moments to reduce the computational costs involved in solving the Huxley microscopic muscle model. In my project, the first three order moments are used to extract the size, location and orientation information of an object from on-line video pictures. The moments generating function m_{pq} of order $(p + q)$ is defined as

$$m_{pq} = \iint x^p y^q f(x, y) dx dy, \quad (\text{Eq. 3.1})$$

where $f(x, y)$ is a bounded function, p and q are non-negative integers. For black-and-white video pictures, the value of $f(x, y)$ represents the intensity level of the pixel located at (x, y) ; for color images, it is the color attributes and the intensity of the pixel.

3.3 Define Object Area and 2D Location

Let $p = q = 0$ in (Eq. 3.1), the zeroth order moment of an object,

$$m_0 = \iint f(x,y) dx dy . \quad (\text{Eq. 3.2})$$

Similarly, $p = 1$ and $q = 0$, the first order moments in x ,

$$m_x = \iint x f(x,y) dx dy , \quad (\text{Eq. 3.3})$$

and $p = 0$ and $q = 1$, the first order moments in y ,

$$m_y = \iint y f(x,y) dx dy . \quad (\text{Eq. 3.4})$$

As analogous find the center of mass of a metal sheet, the zeroth and first order moments determine the location or the centroid (x_c, y_c) of the object,

$$x_c = \frac{m_x}{m_0} , \quad (\text{Eq. 3.5})$$

$$y_c = \frac{m_y}{m_0} . \quad (\text{Eq. 3.6})$$

In binary image processing, the area of the object measured in pixels is equal to m_0 .

From (Eq. 3.1), the second order moments are

$$m_{xx} = \iint x^2 f(x,y) dx dy , \quad (\text{Eq. 3.7})$$

$$m_{yy} = \iint y^2 f(x,y) dx dy, \quad (\text{Eq. 3.8})$$

and

$$m_{xy} = \iint xy f(x,y) dx dy. \quad (\text{Eq. 3.9})$$

3.4 Central Moments

Moments calculated in terms of the original coordinate $\{O\}$ are called ordinary moments. All moments based on the reference coordinate $\{C\}$ whose origin is located at the object's centroid (x_c, y_c) , are referred to as central moments and defined as

$${}^c m_{pq} = \iint {}^c x^p {}^c y^q f({}^c x, {}^c y) dx dy \quad (\text{Eq. 3.10})$$

$$= \iint (x - x_c)^p (y - y_c)^q f(x - x_c, y - y_c) dx dy. \quad (\text{Eq. 3.11})$$

The first three order central moments expressed in terms of ordinary moments are

$${}^c m_0 = m_0, \quad (\text{Eq. 3.12})$$

$${}^c m_x = 0, \quad (\text{Eq. 3.13})$$

$${}^c m_y = 0, \quad (\text{Eq. 3.14})$$

$${}^c m_{xx} = m_{xx} - x_c^2 m_0, \quad (\text{Eq. 3.15})$$

$${}^c m_{yy} = m_{yy} - y_c^2 m_0, \quad (\text{Eq. 3.16})$$

$${}^c m_{xy} = m_{xy} - x_c y_c m_0. \quad (\text{Eq. 3.17})$$

Note that ${}^c m_x$ and ${}^c m_y$ in (Eq. 3.13) and (Eq. 3.14) are zero, because the calculation is now based on the central coordinate located at the center of the object. From (Eq. 3.15) — (Eq. 3.17) the second order central moments are simply those ordinary moments without the influence of object size and location. Because of the independence of size and position, central second order moments play direct roles in the object orientation calculation.

3.5 Define Object Orientation

Consider a pixel P that belongs to the object image within a ROI. P is denoted as ${}^o P$ if referenced in terms of the ordinary frame coordinate $\{O\}$, or ${}^c P$ if expressed in the central frame coordinate $\{C\}$. The leading superscripts indicate the frames of reference. This convention of notation can be found in Craig [1989] or by others in the current robotic literature. ${}^o P$ can be described in terms of ${}^c P$ as follows,

$${}^o P = {}^o T {}^c P, \quad (\text{Eq. 3.18})$$

where ${}^o T$ is a 3x3 matrix which consists of a 2x2 rotation sub-matrix ${}^o R$ and translation vector ${}^o P_c$ in the form of,

$${}^o T = \begin{bmatrix} {}^o R & {}^o P_c \\ 0 & 1 \end{bmatrix}. \quad (\text{Eq. 3.19})$$

Since the matrix T deals with rotation as well as translation, it is referred to as the homogeneous transformation matrix. This matrix is the basic tool used to describe spatial relationships between objects in space.

The 2x2 rotation sub-matrix oR describes the rotational relationship between $\{C\}$ and $\{O\}$. As $\{C\}$ rotates about its origin located at the centroid of the object,

$${}^oR(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}, \quad (\text{Eq. 3.20})$$

the vector oP_c describes the translational relationship of the central coordinate $\{C\}$ with respect to $\{O\}$,

$${}^oP_c = \begin{bmatrix} {}^ox_c \\ {}^oy_c \end{bmatrix}. \quad (\text{Eq. 3.21})$$

From (Eq. 3.18), the pixel P expressed in terms of $\{C\}$,

$${}^cP = {}^cT^{-1} {}^oP. \quad (\text{Eq. 3.22})$$

Since $\{O\}$ and $\{C\}$ are orthonormal coordinate systems, the inverse matrix can be easily found,

$${}^cT^{-1} = {}^cT = \begin{bmatrix} {}^oR^T & {}^oP_c \\ 0 & 0 & 1 \end{bmatrix}. \quad (\text{Eq. 3.23})$$

Substituting (Eq. 3.20) into (Eq. 3.23), the coordinates of pixel P expressed in terms of $\{C\}$ are shown as follows,

$$\begin{bmatrix} {}^c x(\theta) \\ {}^c y(\theta) \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & -x_c \cos\theta - y_c \sin\theta \\ -\sin\theta & \cos\theta & x_c \sin\theta - y_c \cos\theta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \quad (\text{Eq. 3.24})$$

From the definition (Eq. 3.11), the second order central moment as a function of rotating angle θ is

$${}^c m_{xx}(\theta) = \iint {}^c x^2(\theta) f(x, y) dx dy. \quad (\text{Eq. 3.25})$$

Substituting (Eq. 3.24) into ${}^c m_{xx}$, the above ${}^c m_{xx}(\theta)$ is found to be

$${}^c m_{xx}(\theta) = \cos^2\theta (m_{xx} - x_c^2 m_0) + \sin^2\theta (m_{yy} - y_c^2 m_0) + 2\sin\theta \cos\theta (m_{xx} m_{yy} - x_c y_c m_0). \quad (\text{Eq. 3.26})$$

Or in terms of the central coordinate $\{C\}$, from (Eq. 3.15) — (Eq. 3.17)

$${}^c m_{xx}(\theta) = \cos^2\theta {}^c m_{xx} + \sin^2\theta {}^c m_{yy} + 2\sin\theta \cos\theta {}^c m_{xy}. \quad (\text{Eq. 3.27})$$

Recall that

$$\sin^2\theta = \frac{1}{2}(1 - \cos 2\theta),$$

$$\cos^2 \theta = \frac{1}{2}(1 + \cos 2\theta),$$

$$\sin 2\theta = 2 \sin \theta \cos \theta,$$

$$\cos 2\theta = \cos^2 \theta - \sin^2 \theta.$$

Thus ${}^c m_{xx}(\theta)$ expressed as a function of double angle θ ,

$${}^c m_{xx}(\theta) = \frac{{}^c m_{xx} + {}^c m_{yy}}{2} + \frac{{}^c m_{xx} - {}^c m_{yy}}{2} \cos 2\theta + {}^c m_{xy} \sin 2\theta. \quad (\text{Eq. 3.28})$$

Similarly,

$${}^c m_{yy}(\theta) = \frac{{}^c m_{xx} + {}^c m_{yy}}{2} - \frac{{}^c m_{xx} - {}^c m_{yy}}{2} \cos 2\theta - {}^c m_{xy} \sin 2\theta, \quad (\text{Eq. 3.29})$$

$${}^c m_{xy}(\theta) = -\frac{{}^c m_{xx} - {}^c m_{yy}}{2} \sin 2\theta + {}^c m_{xy} \cos 2\theta. \quad (\text{Eq. 3.30})$$

The object orientation α , at which ${}^c m_{xx}(\theta)$ is at its maximum, can be found by differentiating (Eq. 3.28) with respect to θ and then equating it to zero,

$$\frac{d}{d\theta} {}^c m_{xx}(\theta) = -({}^c m_{xx} - {}^c m_{yy}) \sin 2\theta + 2 {}^c m_{xy} \cos 2\theta = 0, \quad (\text{Eq. 3.31})$$

$$\alpha = \frac{1}{2} \tan^{-1} \left(\frac{2 {}^c m_{xy}}{{}^c m_{xx} - {}^c m_{yy}} \right). \quad (\text{Eq. 3.32})$$

At this angle ${}^c m_{yy}(\theta)$ is at its minimum and the cross moment ${}^c m_{xy}(\theta)$ is crossing its zero point. As a result, α can also be solved in the same manner as above for (Eq. 3.29) or by equating (Eq. 3.30) to zero.

3.6 Moment Binary Image Processing

Consider a digitized image $f(x, y)$, where x and y are horizontal and vertical coordinates of the pixel, respectively. For binary images, if $f(x, y)$ is greater than the threshold level, it takes the value of one; otherwise zero. Assuming the image $f(x, y)$ is bounded within a ROI of width $(X_{right} - X_{left})$ and height $(Y_{bottom} - Y_{top})$. From (Eq. 3.2) the zeroth order moment is

$$M_0 = \sum_{y=Y_{top}}^{Y_{bottom}} \sum_{x=X_{left}}^{X_{right}} 1. \quad (\text{Eq. 3.33})$$

Note that x increases from left to right of the screen, while y increases from top to bottom. (Eq. 3.33) is clearly the area of the object measured in pixels. Similarly, from (Eq. 3.3) and (Eq. 3.4), the first order moments are

$$M_x = \sum_{y=Y_{top}}^{Y_{bottom}} \sum_{x=X_{left}}^{X_{right}} x, \quad (\text{Eq. 3.34})$$

$$M_y = \sum_{y=Y_{top}}^{Y_{bottom}} \sum_{x=X_{left}}^{X_{right}} y. \quad (\text{Eq. 3.35})$$

And from (Eq. 3.5) and (Eq. 3.6) the centroid coordinates of the object measured in pixels are

$$X_c = \frac{M_X}{M_0}, \quad (\text{Eq. 3.36})$$

$$Y_c = \frac{M_Y}{M_0}. \quad (\text{Eq. 3.37})$$

As defined in (Eq. 3.7), (Eq. 3.8) and (Eq. 3.9), the second order moments of an object bounded by the upper and lower limits defined by a rectangular ROI are

$$M_{XX} = \sum_{y=Y_{top}}^{Y_{bottom}} \sum_{x=X_{left}}^{X_{right}} x^2, \quad (\text{Eq. 3.38})$$

$$M_{YY} = \sum_{y=Y_{top}}^{Y_{bottom}} \sum_{x=X_{left}}^{X_{right}} y^2, \quad (\text{Eq. 3.39})$$

$$M_{XY} = \sum_{y=Y_{top}}^{Y_{bottom}} \sum_{x=X_{left}}^{X_{right}} x \cdot y. \quad (\text{Eq. 3.40})$$

The second order central moments for digitized binary images are computed in the same manner as found in (Eq. 3.15), (Eq. 3.16), and (Eq. 3.17)

$${}^cM_{XX} = M_{XX} - M_X X_C, \quad (\text{Eq. 3.41})$$

$${}^cM_{YY} = M_{YY} - M_Y Y_C, \quad (\text{Eq. 3.42})$$

$${}^cM_{XY} = M_{XY} - M_X Y_C. \quad (\text{Eq. 3.43})$$

Horizontal and vertical distances between pixels for a particular video system are not the same. This can be thought of as the pixel not being square or non-unity Pixel Aspect Ratio (PAR). The non-unity PAR is the major source of error in the angle measurement.

The error in angle measurements due to the non-unity PAR can be very frustrating to many first time investigators of image processing (see Nguyen [1992]).

The problem can be effectively illustrated only when the image stored in the computer memory is printed out. Of course, the printer must be adjusted for a unity PAR before printing out the image. A laser printer is highly recommended.

3.7 Object Orientation with PAR Compensation

The Object orientation error can only be further improved by increasing the Length Width Ratio (LWR) of the object that was experimentally and analytically confirmed.

In order to obtain better results in angle computation, all pixels must be compensated for this non-unity PAR factor. This is dependent on the particular image processing system, especially the camera. The compensation must be carried out either pixel-by-pixel or by an overall adjustment in the second order moments before the angle

computation has taken place. To have an effective square pixel, either the height or the width needs to be adjusted. From (Eq. 3.32), the vertical adjustment is

$$\alpha = \frac{1}{2} \tan^{-1} \left(\frac{-2 \cdot PAR \cdot {}^cM_{XY}}{{}^cM_{XX} - PAR^2 \cdot {}^cM_{YY}} \right). \quad (\text{Eq. 3.44})$$

Note that the negative sign in the numerator is due to the image coordinate origin which is located at the upper left of the video image.

Chapter 4

3D Information Recovery Based on Orthogonal Projections

4.1 Single Camera Case

We have developed an effective technique using one camera to track and recover 3D information for a mobile object in the space. This method is applicable for tracking multiple mobile objects. However, this would degrade the system's performance. Figure 4.1 illustrates the basic geometry of a single camera.

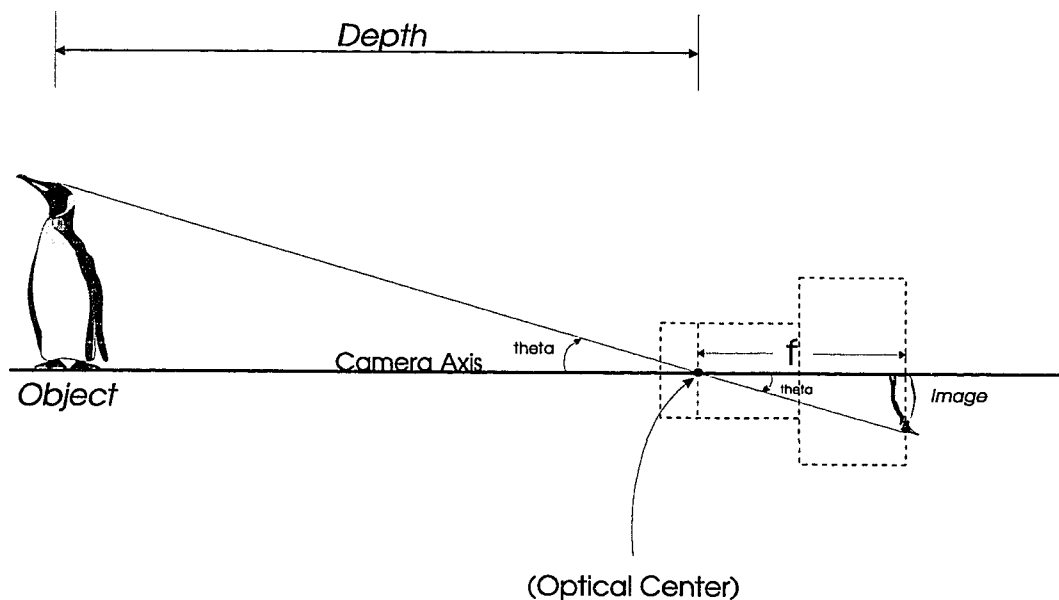


Figure 4.1 The basic geometry of a single camera.

4.1.1 3D Recovery Technique with A Single Camera

More precisely, the recovery technique for a single camera can be divided into three steps. The first step recovers the (x, y) location of the object; the second step recovers the orientation of the object; the third step recovers the depth of the object. When these three steps are well defined, the 3D information of the object can be easily obtained.

The Technique for Recovering the Centroid and the Size of the Object:

The first step of the recovery technique mainly relies on the low level moment image processing technique. In low level image processing, the moments have been chosen as the fastest and most robust algorithm for 2D image processing (see Nguyen [1992]). The zeroth-order moment provides information about sizes of objects; the first-order moments provide information about (x, y) locations of the objects. As mentioned in chapter 3, the digital moments generating function M_{pq} of order $(p + q)$ can be written as

$$M_{pq} = \sum_x \sum_y x^p y^q f(x, y), \quad (\text{Eq. 4.1})$$

where $f(x, y)$ is a bounded function, p and q are non-negative integers. For black-and-white video pictures, the value of $f(x, y)$ represents the intensity level of the pixel located at (x, y) . For binary images, the value of $f(x, y)$ is one if it is above a certain threshold level, otherwise it is assigned zero. Thus, let $p = q = 0$, the zeroth order moment of an object,

$$M_{00} = \sum_x \sum_y 1. \quad (\text{Eq. 4.2})$$

Similarly, $p = 1$ and $q = 0$, the first order moments in x ,

$$M_{10} = \sum_x \sum_y x, \quad (\text{Eq. 4.3})$$

and $p = 0$ and $q = 1$, the first order moments in y ,

$$M_{01} = \sum_x \sum_y y. \quad (\text{Eq. 4.4})$$

From (Eq. 4.2, 4.3, and 4.4), the centroid (x_c, y_c) of the object is found to be

$$x_c = \frac{M_{10}}{M_{00}}, \quad (\text{Eq. 4.5})$$

$$y_c = \frac{M_{01}}{M_{00}}. \quad (\text{Eq. 4.6})$$

In binary image processing, the size of the object measured in pixels is equal to M_{00} .

The Technique for Recovering the Orientation of the Object:

The second step of the recovery technique still relies on the low level moment image processing techniques. From (Eq. 4.1), the second order moments are

$$M_{20} = \sum_x \sum_y x^2, \quad (\text{Eq. 4.7})$$

$$M_{02} = \sum_x \sum_y y^2, \quad (\text{Eq. 4.8})$$

and

$$M_{11} = \sum_x \sum_y xy. \quad (\text{Eq. 4.9})$$

Moments computed after the original coordinate system is translated to the centroid (x_c, y_c) of the object are called central moments. It is defined as

$${}^c M_{pq} = \sum_x \sum_y (x - x_c)^p (y - y_c)^q f(x, y). \quad (\text{Eq. 4.10})$$

The second order central moments are

$${}^c M_{11} = M_{11} - y_c M_{10}, \quad (\text{Eq. 4.11})$$

$${}^c M_{20} = M_{20} - x_c M_{10}, \quad (\text{Eq. 4.12})$$

$${}^c M_{02} = M_{02} - y_c M_{01}. \quad (\text{Eq. 4.13})$$

The object orientation or the principal axes of the object expressed in terms of central moments is

$$\theta = \frac{1}{2} \tan^{-1} \left(\frac{2 \cdot {}^c M_{11}}{{}^c M_{20} - {}^c M_{02}} \right). \quad (\text{Eq. 4.14a})$$

At this angle, the second order central moments ${}^cM_{20}$ and ${}^cM_{02}$ are at their maximum and minimum values respectively, whereas the cross second order central moment ${}^cM_{11}$ is vanished.

For better results in angle computation, all pixels must be compensated for the non-unity PAR factor. After PAR has been taken into account, we have the object orientation

$$\theta = \frac{1}{2} \tan^{-1} \left(\frac{-2 \cdot PAR \cdot {}^cM_{11}}{{}^cM_{20} - PAR^2 \cdot {}^cM_{02}} \right). \quad (\text{Eq. 4.14b})$$

The negative sign in the numerator is due to the digitized image coordinate origin located at the upper left of the video image.

For consecutive rotations, let θ_i and θ_{i+1} be the consecutive angle values of θ and $\Delta\theta = \theta_i - \theta_{i+1}$, then the orientation of the object is computed based on the following algorithm:

When θ_i and θ_{i+1} have the same sign:

if ($\Delta\theta < 0$) direction = counterclockwise;

else direction = clockwise;

When θ_i is positive and θ_{i+1} is negative:

if ($45^\circ \leq \theta_i \leq 90^\circ$) direction = counterclockwise;

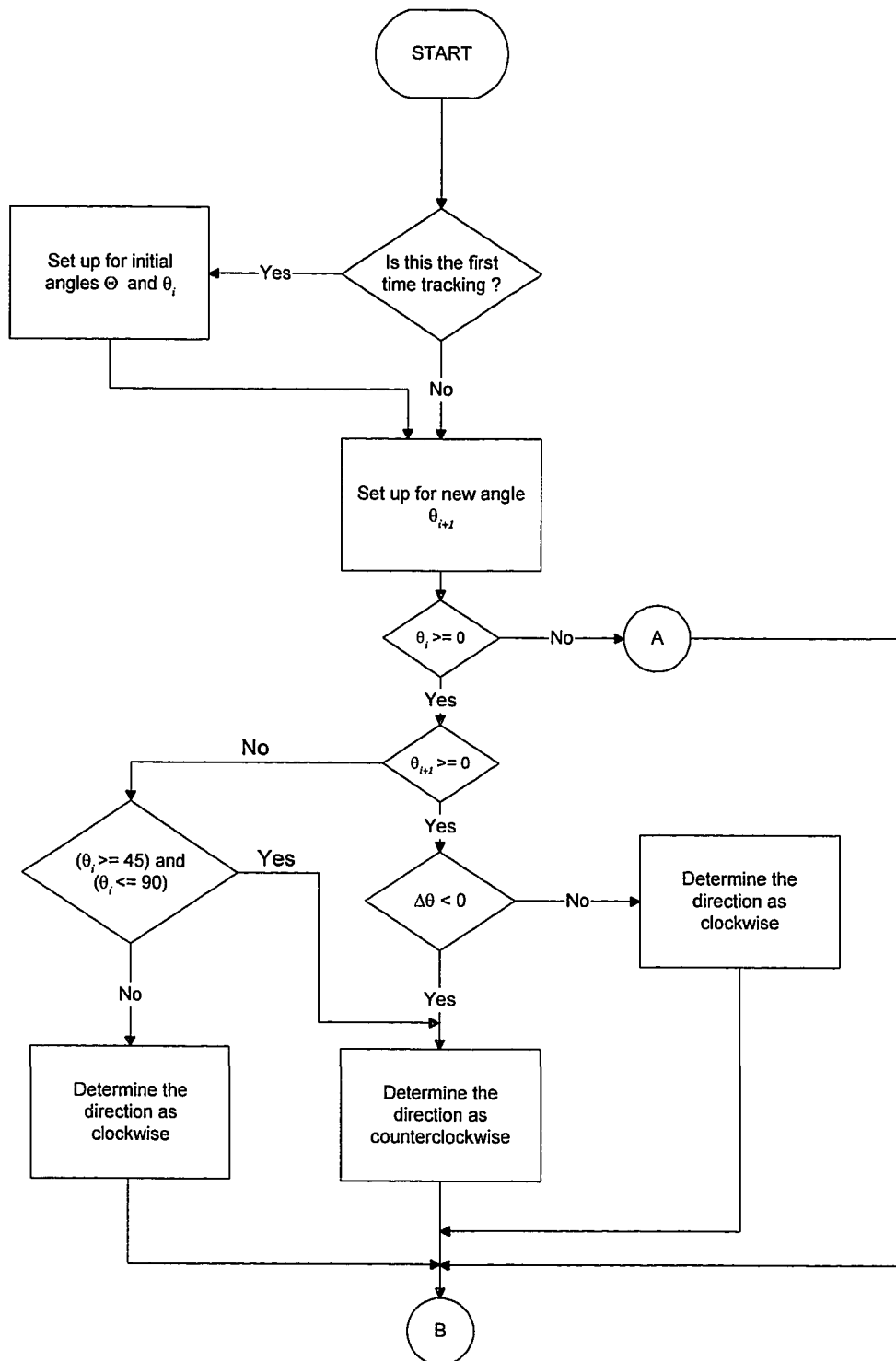
else direction = clockwise;

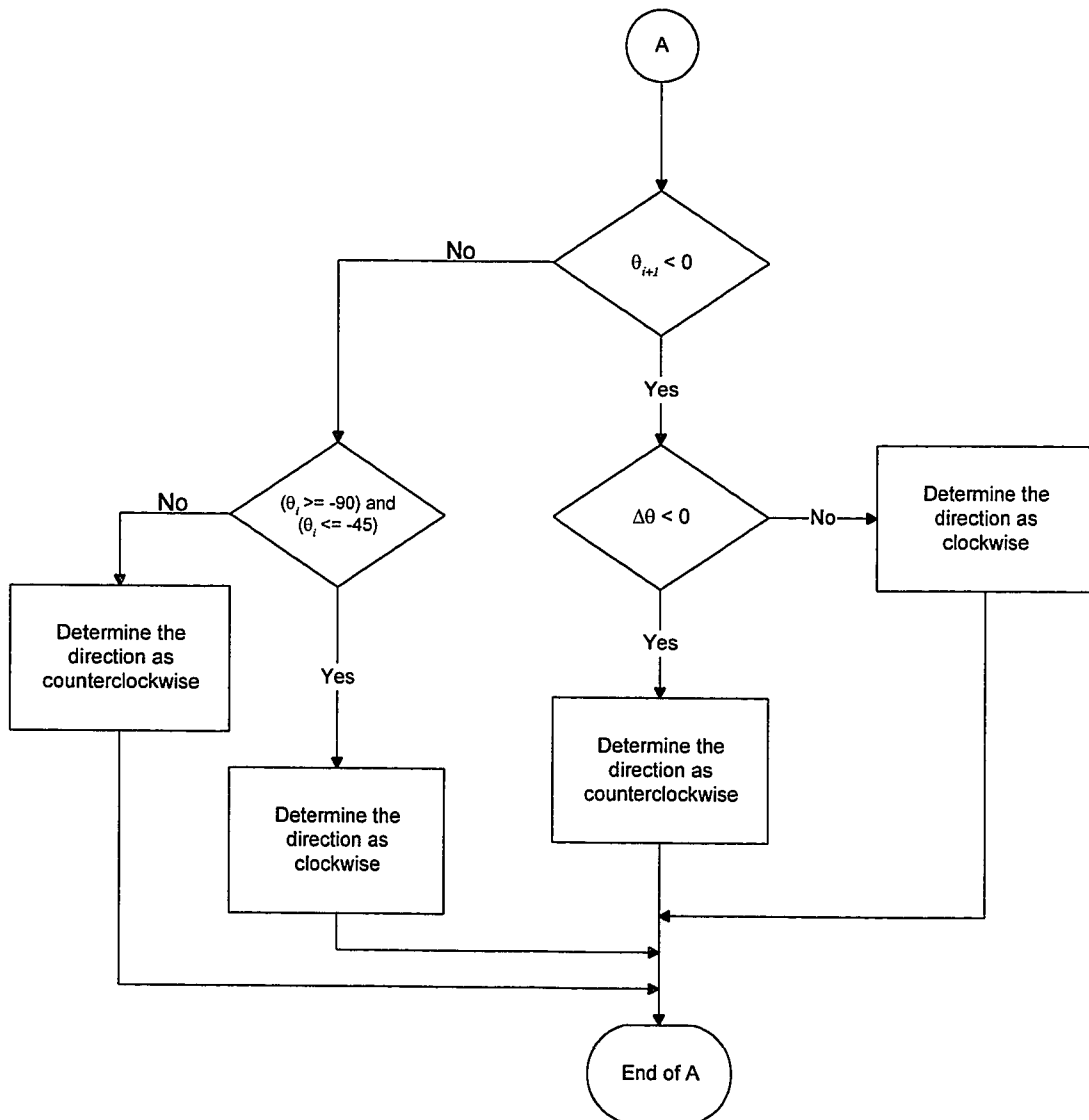
When θ_i is negative and θ_{i+1} is positive:

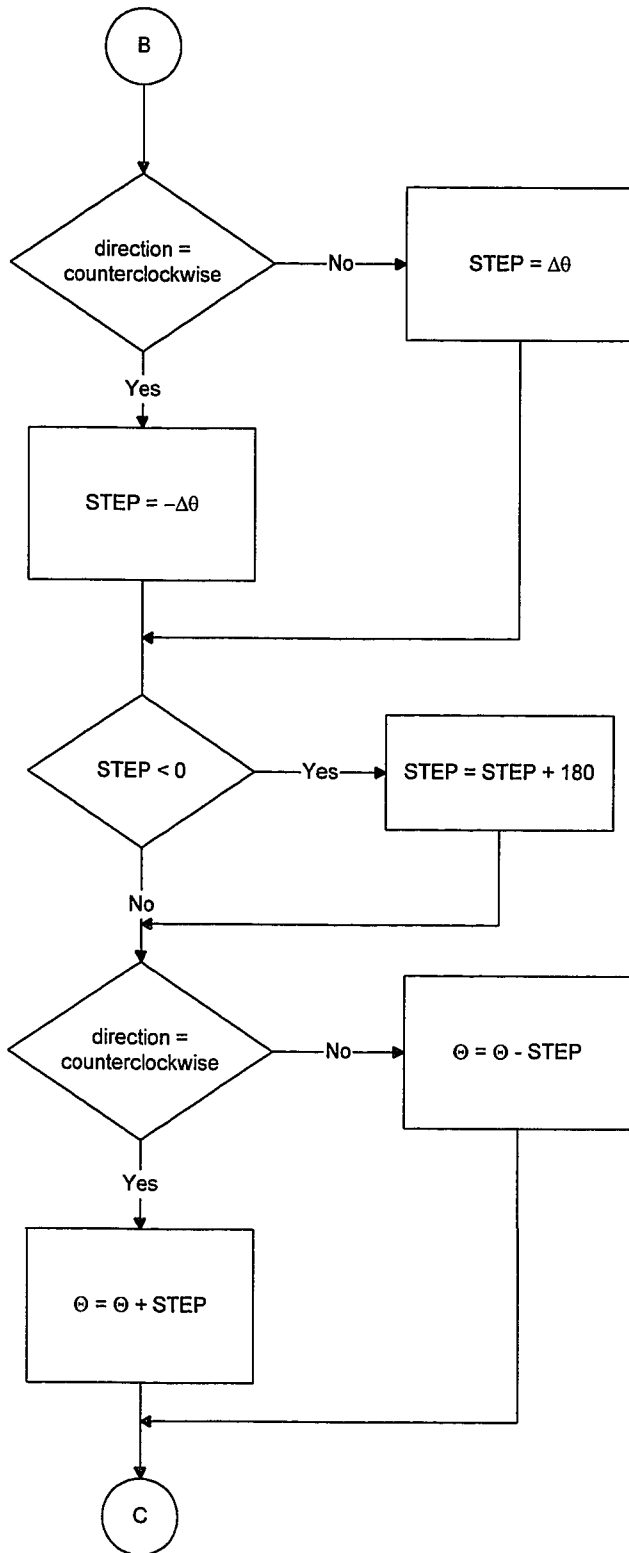
if ($-90^\circ \leq \theta_i \leq -45^\circ$) direction = clockwise;

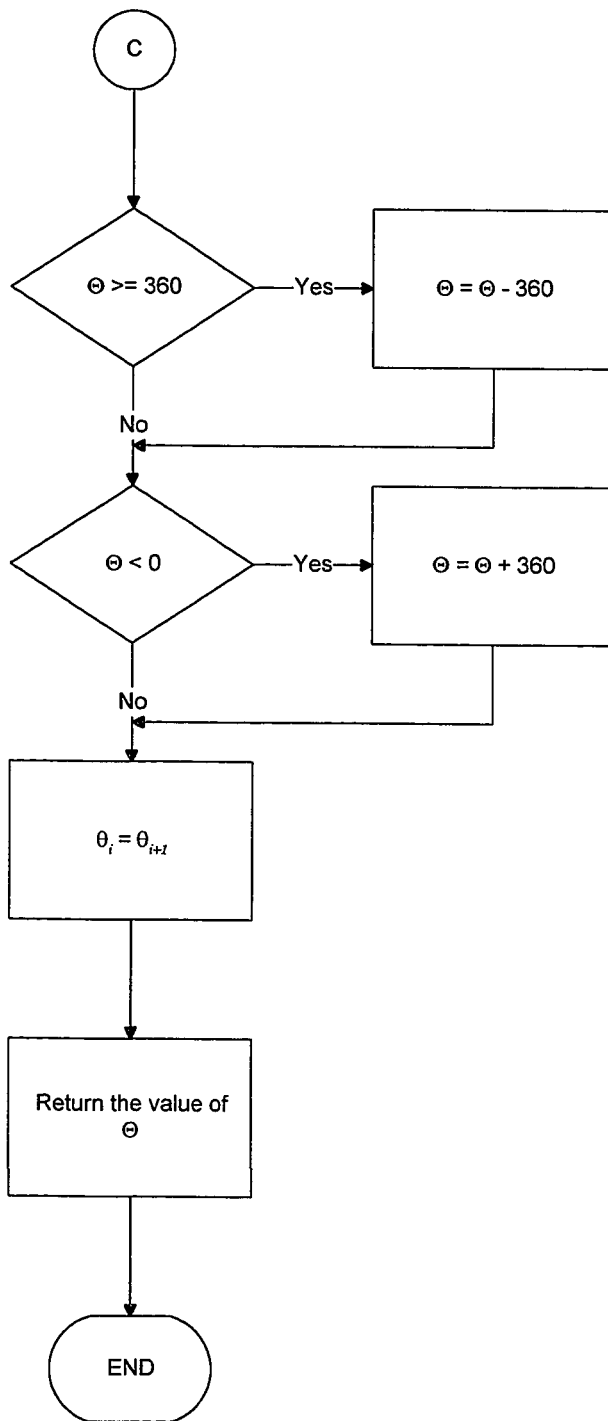
else direction = counterclockwise;

The flowcharts of the algorithm are shown as follows (the Θ is the updated angle):









The Technique for Recovering the Depth of the Object:

The third step of the recovery technique is based on an inverse ratio relationship between the area and the square of the depth of the object. In order to recover the depth of the object, a formula was derived to fulfill this requirement. The derivation process is shown as follows,

$$A_1 = \frac{a}{D_1^2 + bD_1 + c}, \quad (\text{Eq. 4.15})$$

$$A_2 = \frac{a}{D_2^2 + bD_2 + c}, \quad (\text{Eq. 4.16})$$

$$A_3 = \frac{a}{D_3^2 + bD_3 + c}, \quad (\text{Eq. 4.17})$$

where A_i represents the area of the object at a specific point in the space, D_i represents the depth of the object at a specific point in the space; a, b, c are parameters of the formulas.

$$A_1D_1^2 + A_1D_1b + A_1C = a, \quad (\text{Eq. 4.18})$$

$$A_2D_2^2 + A_2D_2b + A_2C = a, \quad (\text{Eq. 4.19})$$

$$A_3D_3^2 + A_3D_3b + A_3C = a. \quad (\text{Eq. 4.20})$$

(Eq. 4.18) – (Eq. 4.19) :

$$(A_1D_1 - A_2D_2)b = (A_2 - A_1)c + (A_2D_2^2 - A_1D_1^2). \quad (\text{Eq. 4.21})$$

(Eq. 4.18) – (Eq. 4.20) :

$$(A_1D_1 - A_3D_3)b = (A_3 - A_1)c + (A_3D_3^2 - A_1D_1^2). \quad (\text{Eq. 4.22})$$

(Eq. 4.21) \times $(A_3 - A_1)$ – (Eq. 4.22) \times $(A_2 - A_1)$:

$$b = \frac{(A_3 - A_1)(A_2D_2^2 - A_1D_1^2) - (A_2 - A_1)(A_3D_3^2 - A_1D_1^2)}{(A_3 - A_1)(A_1D_1 - A_2D_2) - (A_2 - A_1)(A_1D_1 - A_3D_3)}. \quad (\text{Eq. 4.23})$$

From (Eq. 4.22) :

$$c = \frac{(A_1D_1 - A_3D_3)b - (A_3D_3^2 - A_1D_1^2)}{(A_3 - A_1)}. \quad (\text{Eq. 4.24})$$

From (Eq. 4.18) :

$$a = A_1D_1^2 + A_1D_1b + A_1c,$$

$$A_1D_1^2 + A_1bD_1 + (A_1c - a) = 0,$$

$$D_1 = \frac{-(A_1b) \pm \sqrt{(A_1b)^2 - 4(A_1)(A_1c - a)}}{2A_1}. \quad (\text{Eq. 4.25})$$

From (Eq. 4.19) :

$$a = A_2D_2^2 + A_2D_2b + A_2c,$$

$$A_2D_2^2 + A_2bD_2 + (A_2c - a) = 0,$$

$$D_2 = \frac{-(A_2b) \pm \sqrt{(A_2b)^2 - 4(A_2)(A_2c - a)}}{2A_2}. \quad (\text{Eq. 4.26})$$

from (Eq. 4.20) :

$$a = A_3 D_3^2 + A_3 D_3 b + A_3 c,$$

$$A_3 D_3^2 + A_3 b D_3 + (A_3 c - a) = 0,$$

$$D_3 = \frac{-(A_3 b) \pm \sqrt{(A_3 b)^2 - 4(A_3)(A_3 c - a)}}{2A_3}. \quad (\text{Eq. 4.27})$$

The a, b, c parameters can either be solved at run-time, or be solved at the initialization stage.

4.1.2 The General Experimental Setup

- Threshold is very important for binary image processing. Due to the fact that the light in our laboratory is inconsistent, a threshold determined by histogram will be necessary (see Figure 4.2).
- We found out light is a very critical problem. The brightness in the laboratory changes with the sunshine. To minimize the effect of variations in the ambient light, we covered the windows in the laboratory with aluminum foils. By the way, the aluminum foils also block the heat from the laboratory, so they help to keep our laboratory cool in the summer time!

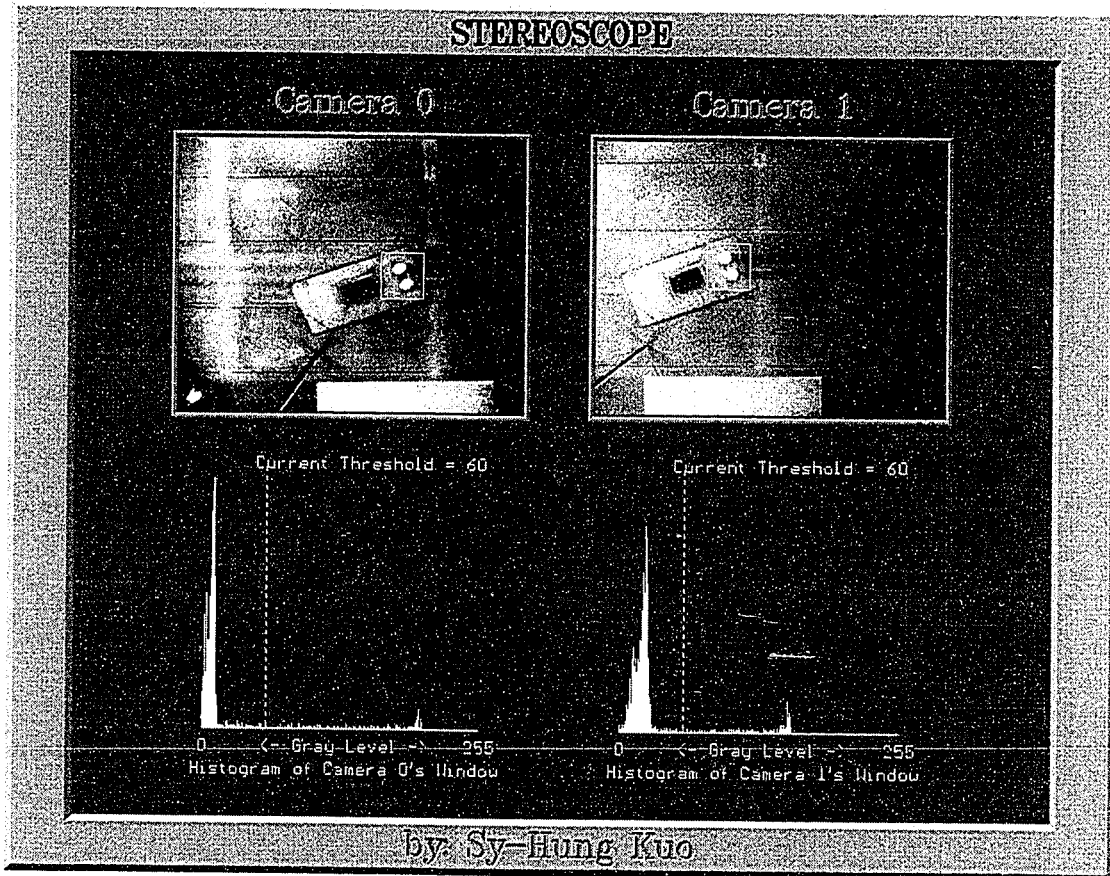


Figure 4.2 The histograms of two selected windows in two cameras.

- We have a 2.17 meters by 1.83 meters black board set up on the floor which makes an ideal background for the chosen white object. Another black board with white grid lines provides the test points for calibrating the cameras (see Figure 4.3). Two cameras are mounted on the ceiling. They are placed 20 centimeters apart and parallel to each other (see Figure 4.4). A tape ruler hangs on the ceiling so that the height of the object can be measured either in centimeters or in inches.

- Two white ping-pong balls are glued together to serve as the test object. The ping-pong balls were selected because their 2D images are shape invariant from different view angles. In addition to providing an excellent signal/noise ratio with the black background, the two ping-pong balls also yield an object orientation which is another important parameter besides the x , y , z coordinates (see Figure 4.3).

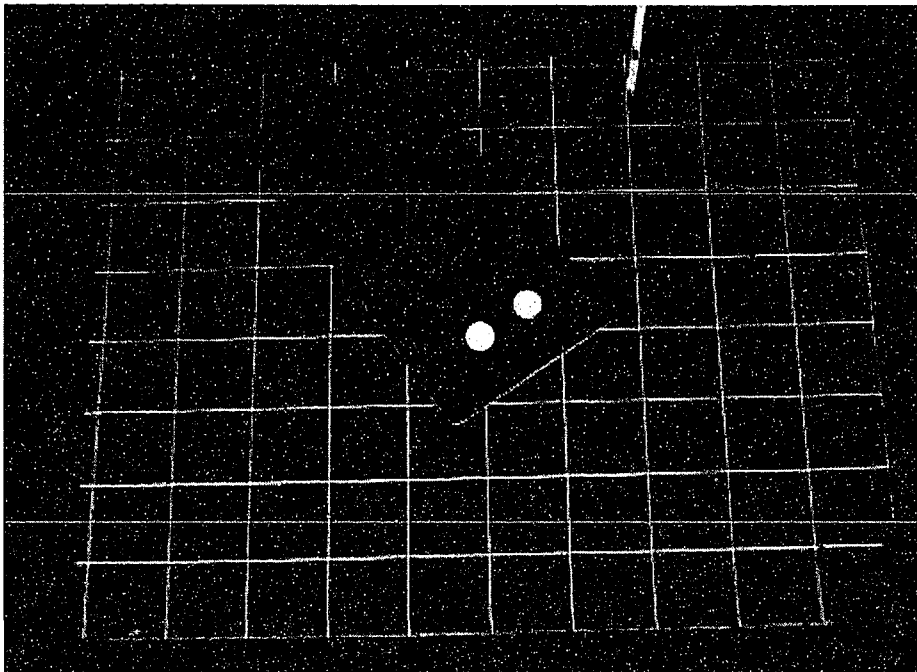


Figure 4.3 Two white ping-pong balls were chosen as the test object.

- Because of the limited ceiling height in our laboratory, the TOYO CCTV zoom lens have been adjusted to their minimum focal length of 12.5 mm. This helps to increase the field of view so that the object can be captured by the cameras as it is moving around (see Figure 4.4).



Figure 4.4 Two non-identical cameras were used in the developed vision system.

- Two computers are used to assist the project. One computer manipulates all the computational needs for image processing (the left PC). The other (the right PC) has the graphical 3D model of the object. The image processing computer passes calculated 3D information to the 3D modeling computer where the object is instantly displayed (see Figure 4.5).
- Three monitors are used in the project; two of them display the real-time views from the two cameras; the other one displays the digitized image from the frame grabber (see Figure 4.5).



Figure 4.5 The computer setup for the developed vision system.

4.1.3 The Experimental Setup for the Particular Single Camera Case

Figure 4.6 illustrates the setup for calibrating a camera using a black board with white grid lines. The distance between the grid lines is 10 cm apart. Sixteen cross points with heavy dots surrounding the center of the board were chosen to be the test points.

The 3D coordinates of the test points were measured and used as inputs to the calibration process. An RCA TV camera (model RCA 2000) with a TOYO CCTV zoom lens (focal length 12.5 mm - 75 mm, F1.8) were used to acquire image data. The camera was mounted on the ceiling at a distance of 280 cm above the grid board and looking straight down at it. For obtaining the desired field of view, the lens were set at the minimum focal length of 12.5 mm. Furthermore, in order to obtain a reliable result, histograms were regularly computed to determine the optimum threshold levels.

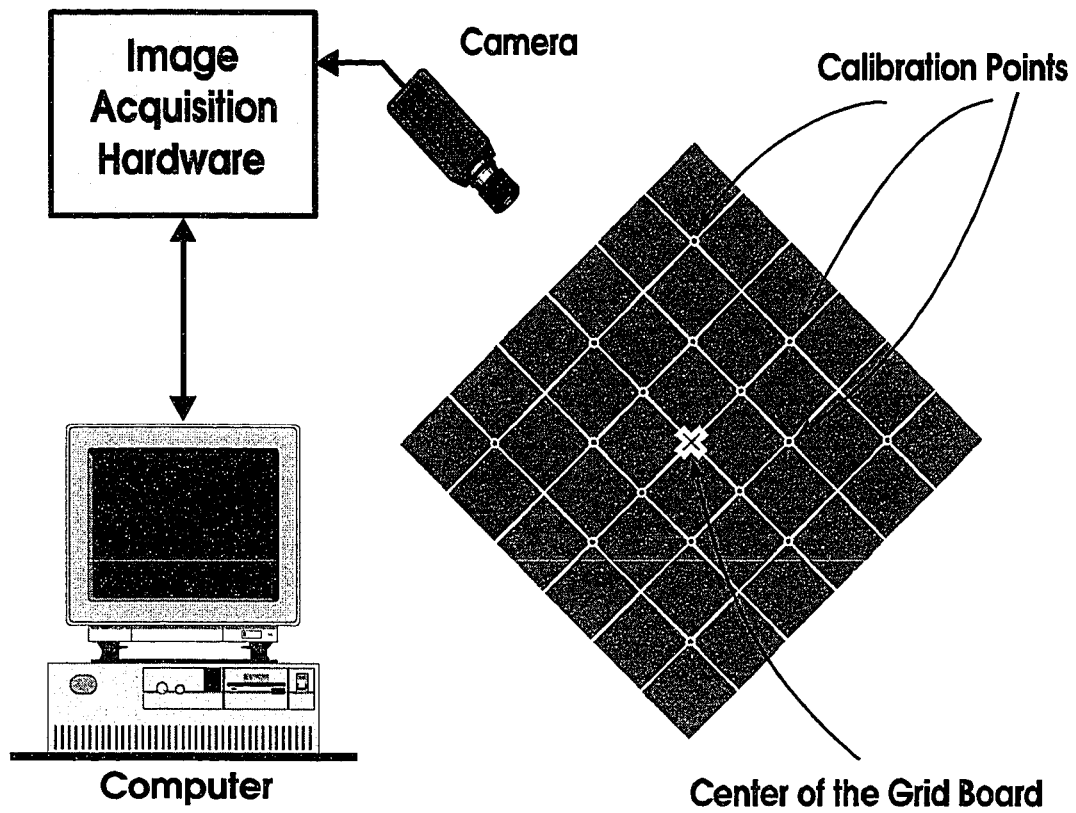


Figure 4.6 The experimental setup for calibrating single camera vision system.

4.1.4 Experimental Results for Single Camera Vision System

Table 4.1 Experimental Data for Applying the Two Parameters Formula:

Actual Height	Measured Area $= \sum_{i=1}^{25} Area_i / 25$	Measured Height $= \sum_{i=1}^{25} Height_i / 25$	Standard Deviation $= \sqrt{\frac{\sum_{i=1}^{25} (Height_i - \overline{Height})^2}{25}}$	Error of Measured Height
2.0 (cm)	145.0 (pixel)	1.9 (cm)	1.4 (cm)	0.1 (cm)
12.0	157.0	12.7	1.5	-0.7
22.0	169.0	22.4	1.4	-0.4
32.0	185.0	33.9	1.5	-1.9
42.0	201.0	45.1	1.4	-3.1
52.0	217.0	55.4	1.3	-3.4
62.0	238.0	65.1	1.2	-3.1
72.0	262.0	75.6	1.3	-3.6
82.0	286.0	84.9	0.9	-2.9
92.0	315.0	94.5	0.9	-2.5
102.0	347.0	103.9	0.9	-1.9
112.0	378.0	111.7	0.7	0.3
122.0	421.0	121.2	0.5	0.8
132.0	483.0	132.5	0.4	-0.5
142.0	548.0	142.4	0.4	-0.4
152.0	623.0	151.5	0.3	0.5
162.0	716.0	160.8	0.4	1.2
172.0	833.0	170.3	0.3	1.7
182.0	983.0	180.1	0.3	1.9
192.0	1222.0	191.5	0.2	0.5
202.0	1525.0	202.0	0.2	0

NOTE ①: Each measured cell takes the average of 25 times measurement.

NOTE ②: Please see Figure 4.7 (a, b), top.

$$\text{Root-mean-square Error} = \sqrt{\frac{\sum_{z=1}^{21} (Error_z)^2}{21}} = 1.9006 \text{ (cm)}$$

Table 4.2 Experimental Data for Applying the Three Parameters Formula:

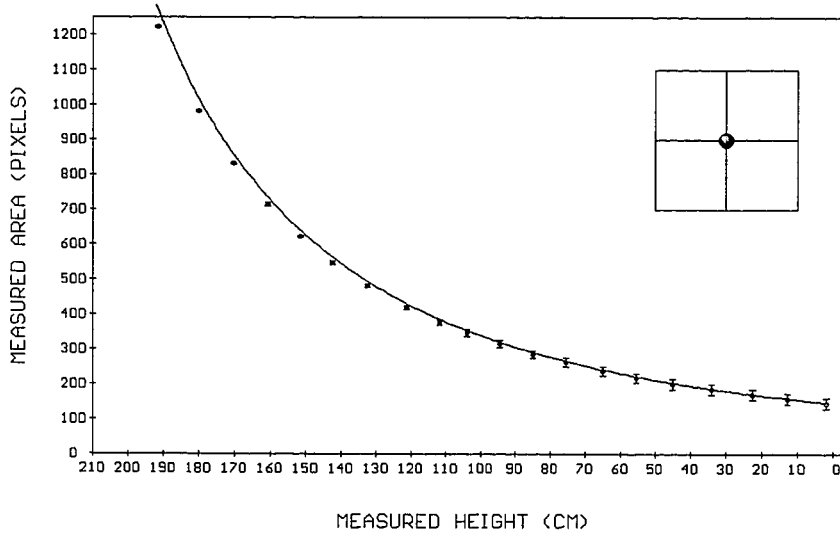
Actual Height	Measured Area $= \sum_{i=1}^{25} Area_i / 25$	Measured Height $= \sum_{i=1}^{25} Height_i / 25$	Standard Deviation $= \sqrt{\frac{\sum_{i=1}^{25} (Height_i - \overline{Height})^2}{25}}$	Error of Measured Height
2.0 (cm)	142.0 (pixel)	2.0 (cm)	1.4 (cm)	0 (cm)
12.0	154.0	12.1	1.4	-0.1
22.0	166.0	22.2	1.3	-0.2
32.0	180.0	32.8	1.4	-0.8
42.0	195.0	43.2	1.2	-1.2
52.0	214.0	53.6	1.3	-1.6
62.0	233.0	63.3	1.2	-1.3
72.0	256.0	73.1	1.1	-1.1
82.0	282.0	82.6	0.9	-0.6
92.0	311.0	92.4	0.8	-0.4
102.0	347.0	102.1	0.7	-0.1
112.0	390.0	111.9	0.5	0.1
122.0	434.0	121.5	0.5	0.5
132.0	497.0	131.7	0.4	0.3
142.0	559.0	141.0	0.4	1.0
152.0	646.0	150.7	0.3	1.3
162.0	751.0	160.9	0.4	1.1
172.0	883.0	171.2	0.3	0.8
182.0	1049.0	181.7	0.3	0.3
192.0	1261.0	191.8	0.2	0.2
202.0	1525.0	202.0	0.2	0

NOTE ①: Each measured cell takes the average of 25 times measurement.

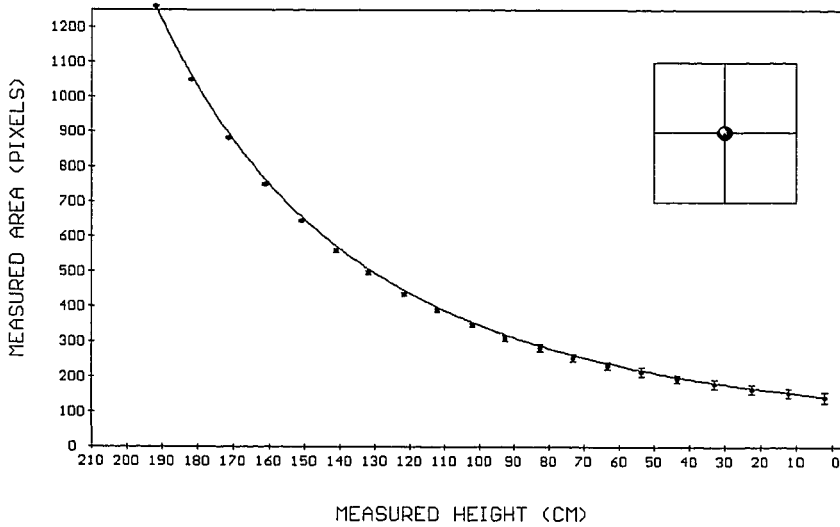
NOTE ②: Please see Figure 4.7 (a, b), bottom.

$$\text{Root-mean-square Error} = \sqrt{\frac{\sum_{z=1}^{21} (Error_z)^2}{21}} = 0.7910 \text{ (cm)}$$

Formula: $AREA = A / (DISTANCE + B)^2$



Formula: $AREA = A / (DISTANCE^2 + B * DISTANCE + C)$

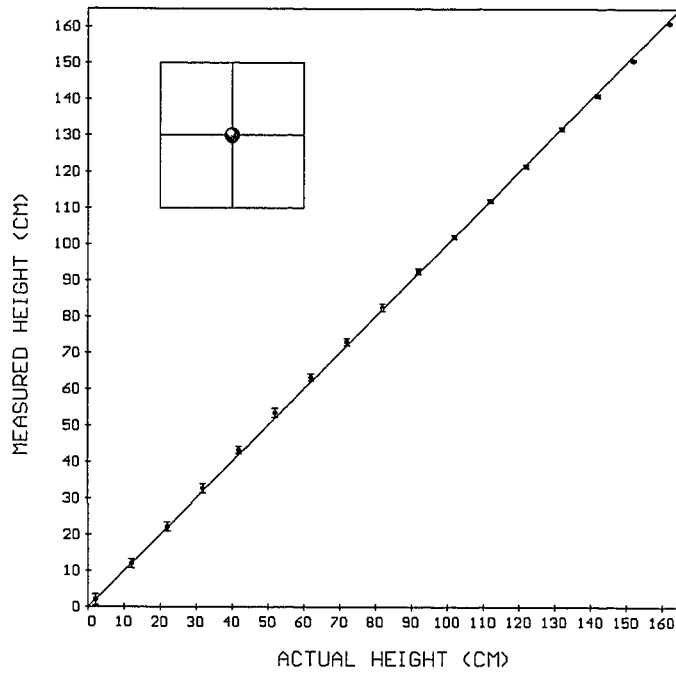
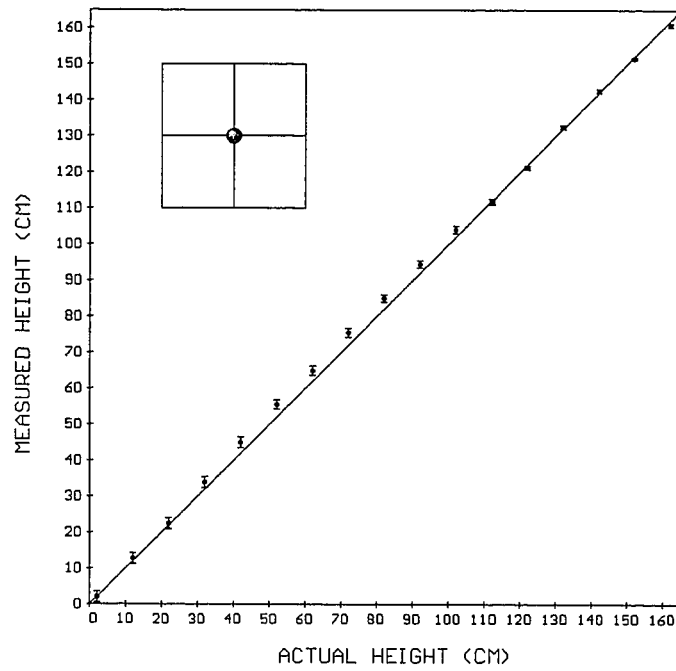


Figures 4.7a MEASURED AREA vs. MEASURED HEIGHT based on single camera system.

Note ①: Two parameters case (top), three parameters case (bottom).

Note ②: The curves in two figures represent the theoretical ideal lines.

Note ③: MEASURED HEIGHT = 280 (cm) - DISTANCE (from camera to object).



Figures 4.7b MEASURED HEIGHT vs. ACTUAL HEIGHT based on single camera system.

Note ①: Two parameters case (top), three parameters case (bottom).

Note ②: The 45 degrees straight lines represent the theoretical ideal lines.

4.2 Two-camera Case

4.2.1 3D Recovery Technique with Two Cameras

Comparing the two-camera case with the single camera case, the difference lies with the method of how the depth of the object is recovered. In the two-camera case, the binocular disparity is used to resolve the depth computation problem. Figure 4.8 illustrates the basic geometry setup for the vision system based on the two-camera case. The derivation processes for recovering the depth are shown as follows.

From Figure 4.8, if we apply the similar triangular method to the camera 0, the ratio of x_0 (the object's x coordinate measured in the image plane of camera 0) and λ_0 (the focal length of camera 0),

$$\frac{x_0}{\lambda_0} = -\frac{X_0}{Z_0 - \lambda_0} = \frac{X_0}{\lambda_0 - Z_0}. \quad (\text{Eq. 4.28})$$

X_0 and Z_0 indicate that the camera 0 was at the origin of the world coordinate system.

Similarly, for camera 1, we moved the camera 1 to the origin of the world coordinate system, with the camera 0 and the object shifted with it without changing their original relative positions. We have

$$\frac{x_1}{\lambda_1} = -\frac{X_1}{Z_1 - \lambda_1} = \frac{X_1}{\lambda_1 - Z_1}. \quad (\text{Eq. 4.29})$$

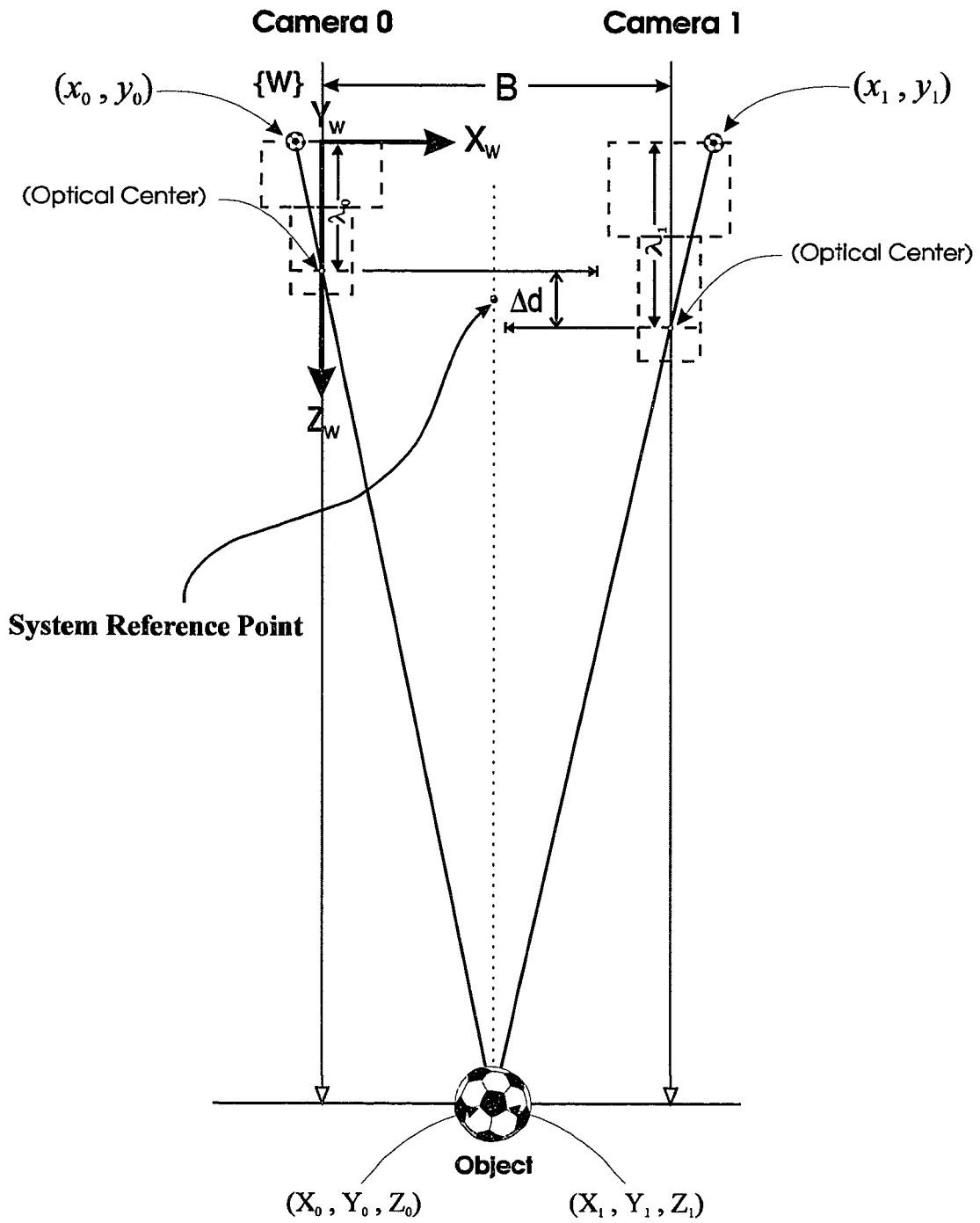


Figure 4.8 The basic geometry setup for the two-camera vision system.

We utilize two non-identical TV cameras and they are not placed at the same distance away from the object under study. Therefore, an extra Δd parameter must be added to the (Eq. 4.30). See Figure 4.8 for illustration.

$$Z_0 = Z_1 + \Delta d. \quad (\text{Eq. 4.30})$$

Since the separation between the two cameras is B , it follows that

$$X_1 = X_0 + B. \quad (\text{Eq. 4.31})$$

Substituting (Eq. 4.30), (Eq. 4.31) into (Eq. 4.28) gives

$$\frac{x_0}{\lambda_0} = \frac{X_1 - B}{\lambda_0 - (Z_1 + \Delta d)}. \quad (\text{Eq. 4.32})$$

From (Eq. 4.29) we have

$$X_1 = \frac{x_1(\lambda_1 - Z_1)}{\lambda_1}. \quad (\text{Eq. 4.33})$$

Substituting (Eq. 4.33) into (Eq. 4.32) gives

$$\frac{x_0}{\lambda_0} = \frac{x_1(\lambda_1 - Z_1) - \lambda_1 B}{\lambda_1 \lambda_0 - \lambda_1 (Z_1 + \Delta d)}. \quad (\text{Eq. 4.34})$$

Expanding (Eq. 4.34) gives

$$x_0\lambda_1\lambda_0 - x_0\lambda_1Z_1 - x_0\lambda_1\Delta d = \lambda_0x_1\lambda_1 - \lambda_0x_1Z_1 - \lambda_0\lambda_1B, \quad (\text{Eq. 4.35})$$

$$Z_1 = \frac{\lambda_0\lambda_1x_1 - \lambda_0\lambda_1B - \lambda_0\lambda_1x_0 + x_0\lambda_1\Delta d}{(\lambda_0x_1 - \lambda_1x_0)}, \quad (\text{Eq. 4.36})$$

$$Z_1 = \frac{\lambda_0\lambda_1(x_1 - B - x_0) + x_0\lambda_1\Delta d}{(\lambda_0x_1 - \lambda_1x_0)}. \quad (\text{Eq. 4.37})$$

If the System Reference Point (SRP) is located in the middle of the two optical centers (see Figure 4.8), then the depth measured from the SRP to the object is

$$\text{Depth} = Z_1 - \lambda_1 + \frac{\Delta d}{2}, \quad (\text{Eq. 4.38})$$

where

B : the length of the baseline,

λ_0 : the focal length of camera 0,

λ_1 : the focal length of camera 1,

Δd : the offset between two cameras' optical centers,

Depth : the depth of the object, measured from a System Reference Point of the two-camera system to the location of the object.

4.2.2 The Experimental Setup for the Particular Two-camera Case

Please refer to section 4.1.2 for the details of experimental setup. Figure 4.9 illustrates the setup for calibrating two cameras using a black board with white grid lines. The distance between the grid lines is 10 cm apart. An RCA TV camera (model RCA 2000) and a PANASONIC TV camera (model WV-1410) were used to acquire image data; both attached with TOYO CCTV zoom lens (focal length 12.5 mm - 75 mm, F1.8). However, in the two-camera case, both cameras must be carefully set up so that the two optical axes are parallel to each other. Since the height of the ceiling is fixed, to obtain a practicable field of view, the two cameras were separated with a distance of 20 cm. Similar to the single camera case, in order to deal with the light inconsistency problem, the histogram was used intensively to provide the optimum threshold levels. The Δd , the offset between the two cameras' optical centers, was measured to be 2.5 cm.

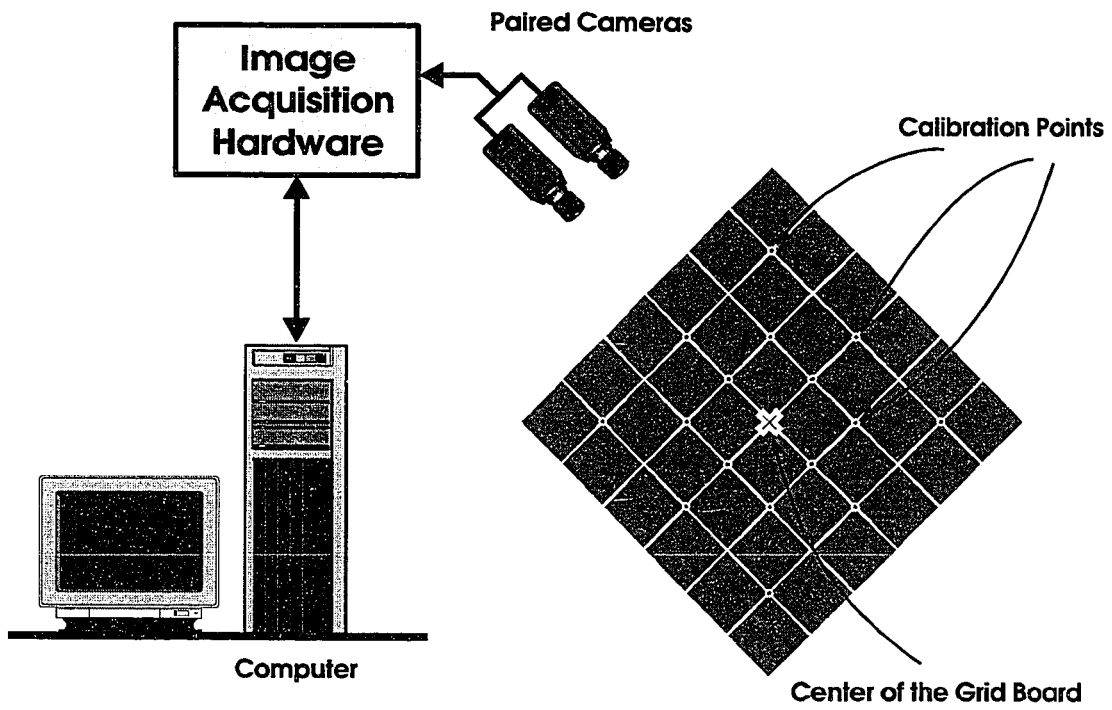


Figure 4.9 The experimental setup for calibrating two-camera vision system.

4.2.3 Experimental Results for Two-camera Vision System

Table 4.3 Experimental Data for Tracking the Object at the Center of the Grid Board:

Actual Height (cm)	Measured Height (cm) $= \sum_{i=1}^{25} Height_i / 25$	Standard Deviation $= \sqrt{\frac{\sum_{i=1}^{25} (Height_i - \overline{Height})^2}{25}}$	Error of Measured Height
2.0	2.8	0.7	-0.8
12.0	12.5	0.9	-0.5
22.0	22.7	0.8	-0.7
32.0	32.3	0.6	-0.3
42.0	42.5	0.9	-0.5
52.0	53.1	0.6	-1.1
62.0	62.7	0.9	-0.7
72.0	72.6	0.8	-0.6
82.0	83.0	0.5	-1.0
92.0	92.8	0.5	-0.8
102.0	101.8	0.6	0.2
112.0	112.6	0.6	-0.6
122.0	122.4	0.8	-0.4
132.0	131.5	0.6	0.5
142.0	142.3	0.8	-0.3
152.0	151.9	0.6	0.1

NOTE ①: Each measured cell takes the average of 25 times measurement.

NOTE ②: Please see Figure 4.10, top.

$$\text{Root-mean-square Error} = \sqrt{\frac{\sum_{z=1}^{16} (Error_z)^2}{16}} = 0.6290 \text{ (cm)}$$

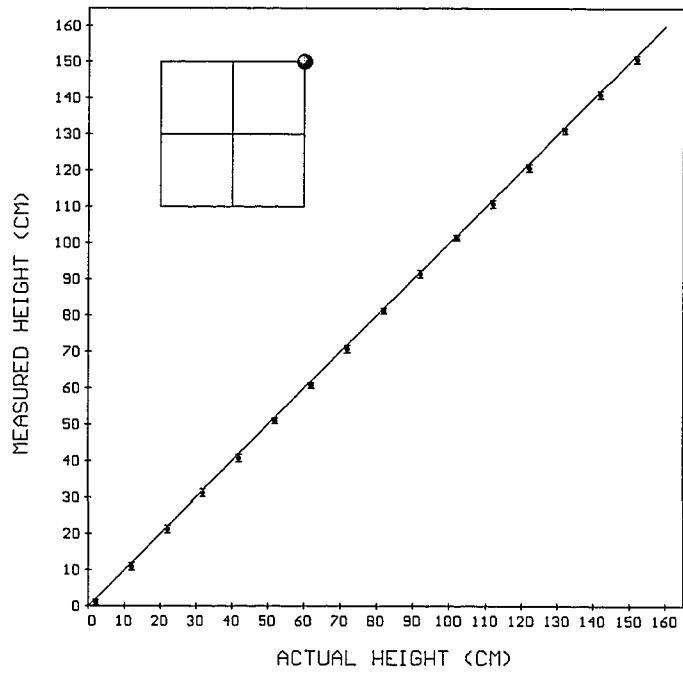
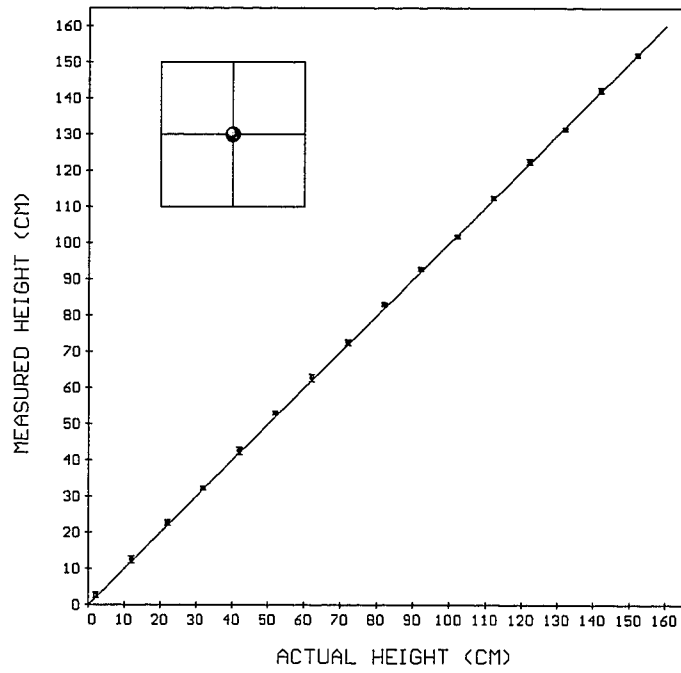
Table 4.4 Experimental Data for Tracking the Object at the Corner of the Grid Board:

Actual Height (cm)	Measured Height (cm) = $\sum_{i=1}^{25} Height_i / 25$	Standard Deviation = $\sqrt{\frac{\sum_{j=1}^{25} (Height_j - \overline{Height})^2}{25}}$	Error of Measured Height
2.0	0.9	0.7	1.1
12.0	11.1	1.0	0.9
22.0	21.2	1.0	0.8
32.0	31.3	0.9	0.7
42.0	40.8	1.0	1.2
52.0	50.9	0.8	1.1
62.0	60.7	0.8	1.3
72.0	70.7	1.0	1.3
82.0	81.2	0.8	0.8
92.0	91.4	0.9	0.6
102.0	101.5	0.7	0.5
112.0	110.8	0.9	1.2
122.0	120.8	1.0	1.2
132.0	131.0	0.8	1.0
142.0	141.1	0.9	0.9
152.0	150.8	0.9	1.2

NOTE ①: Each measured cell takes the average of 25 times measurement.

NOTE ②: Please see Figure 4.10, bottom.

$$\text{Root-mean-square Error} = \sqrt{\frac{\sum_{z=1}^{16} (Error_z)^2}{16}} = 1.0173 \text{ (cm)}$$



Figures 4.10 MEASURED HEIGHT vs. ACTUAL HEIGHT based on two-camera system.

Note ①: Measured at the center (top), measured at the corner (bottom).

Note ②: The 45 degrees straight lines represent the theoretical ideal lines.

Chapter 5

Object Tracking Algorithm and Applications

5.1 A Self-adaptive Algorithm for Object Tracking

For the object tracking purpose, a window that follows the object's moving track will be necessary. The window should be big enough to include the whole object as well as to cover the next possible position of the moving object. Since the size of the object becomes larger when it is closer to the camera, the window should quickly adjust its size to accommodate the practical needs. Furthermore, the window should be able to perform the boundary checks in order to prevent the window from moving beyond the frame grabber's memory. A five-step self-adaptive algorithm that satisfies all the requirements is derived and shown as follows:

STEP 1. Self-adaptive Window Tuning:

$$\textit{Width_of_Window} = \textit{MagicFactor} \times \sqrt{\textit{Area_of_Object}}, \quad (\text{Eq. 5.1})$$

where

MagicFactor := 2.3,

Width_of_Window : the width of the window,

Area_of_Object : the area of the object.

STEP 2. Always move the window's center to the center of the object, so that the window can cover the largest range for tracking the object's next move:

$$X_{wc} = X_{oc}, \quad (\text{Eq. 5.2a})$$

$$Y_{wc} = Y_{oc}, \quad (\text{Eq. 5.2b})$$

where

(X_{wc}, Y_{wc}) : the coordinates that represent the center of the window relative to the digitized image coordinate system,

(X_{oc}, Y_{oc}) : the coordinates that represent the center of the object relative to the digitized image coordinate system.

STEP 3. Object moving prediction:

$$Dir_vector_X = New_X_{oc} - Old_X_{oc}, \quad (\text{Eq. 5.3a})$$

$$Dir_vector_Y = New_Y_{oc} - Old_Y_{oc}, \quad (\text{Eq. 5.3a})$$

where

$(Dir_vector_X, Dir_vector_Y)$: the 2D vector that represents the object's moving direction relative to the digitized image coordinate system.

(New_X_{oc}, New_Y_{oc}) : the coordinates that represent the new location of the object's center relative to the digitized image coordinate system.

(Old_X_{oc}, Old_Y_{oc}) : the coordinates that represent the old location of the object's center relative to the digitized image coordinate system.

STEP 4. If lose tracking of the object, immediately double the window's width and height based on the object's moving direction calculated in Step 3.

STEP 5. Dynamic boundary checking:

if $((X_{wc} - Width_of_Window) < fg_X_{min}) X_{wc} = Width_of_Window;$

else

if $((X_{wc} + Width_of_Window) > fg_X_{max}) X_{wc} = fg_X_{max} - Width_of_Window;$

if $((Y_{wc} - Width_of_Window) < fg_Y_{min}) Y_{wc} = Width_of_Window;$

else

if $((Y_{wc} + Width_of_Window) > fg_Y_{max}) Y_{wc} = fg_Y_{max} - Width_of_Window;$

where

(fg_X_{min}, fg_Y_{min}) : the coordinates which represents the top left pixel in the frame grabber's memory relative to the digitized image coordinate system,

(fg_X_{max}, fg_Y_{max}) : the coordinates which represents the bottom right pixel in the frame grabber's memory relative to the digitized image coordinate system.

5.2 Virtual Reality Research

Virtual Reality is a display and a control technology that can envelop a person in an interactive computer-generated or computer-mediated virtual environment. Virtual Reality creates artificial worlds of sensory experience, or immerses the user in representations of real spatial environments that might otherwise be inaccessible by virtual distance, scale, time, or physical incompatibilities of the user and the environment. The essential defining characteristics of Virtual Reality systems are that they perceptually surround and include the user in the display space, and provide familiar, intuitive interactions with depicted environments and objects.

Virtual Reality deals with a computer synthesized world in which a human can perceive the objects and events existing in the world. The necessary components of a Virtual Reality system are captured or designed digital models of environments and objects. Computer graphics system is capable of generating usefully detailed and sufficiently interactive environments and objects. User interface devices (e.g., helmet displays, gesture tracker, etc.) support a sense of presence in a virtual environment.

Generally, a magnetic tracker which detects head position and orientation is mounted on the helmet. Our developed vision system can easily replace such a tracker with several characteristics (see Figures 5.1 — 5.7). It is easier to set up in a laboratory. It covers a larger environment area. It not only provides head position and orientation information, but also is cost effective. Figure 5.8 and Figure 5.9 illustrate the tracking patterns when our vision system was applied to a Virtual Research helmet display.

The developed system provides an effective tool for our future research and exploring virtual environments for telerobotic and space exploration applications.

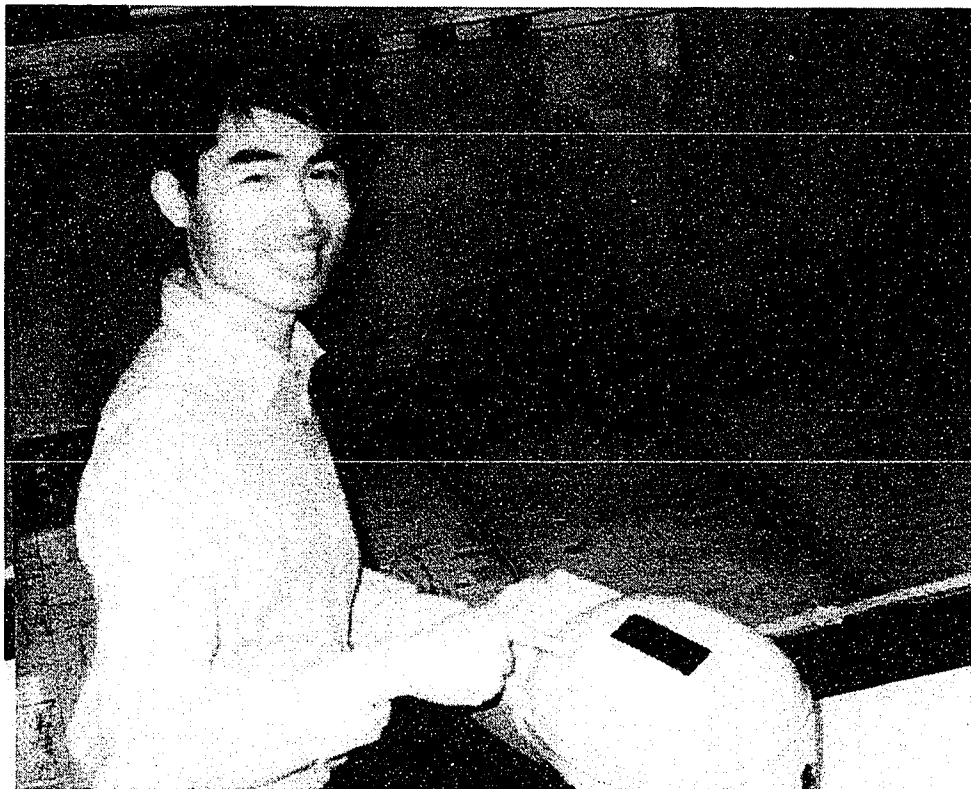


Figure 5.1 A black rectangle target was attached to the helmet display.

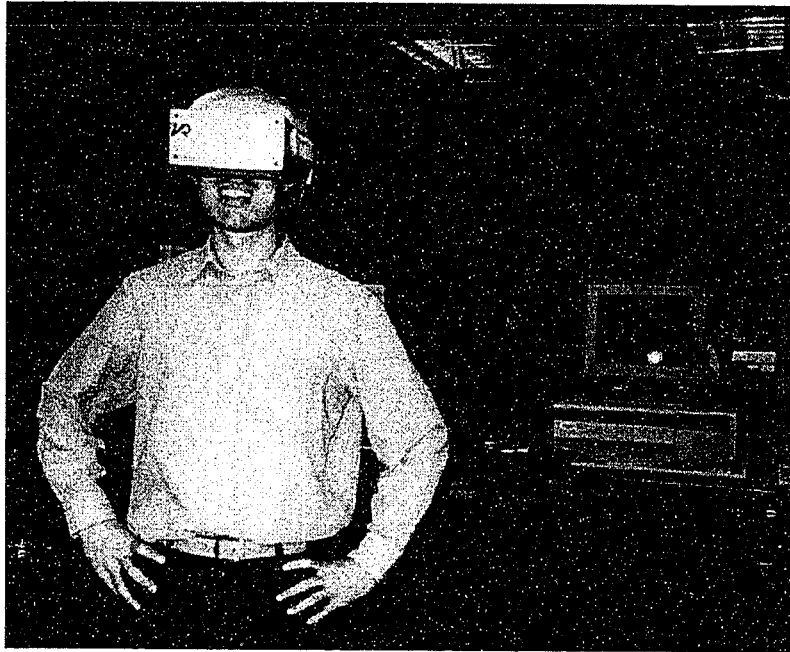


Figure 5.2 Target tracking experiment 1.

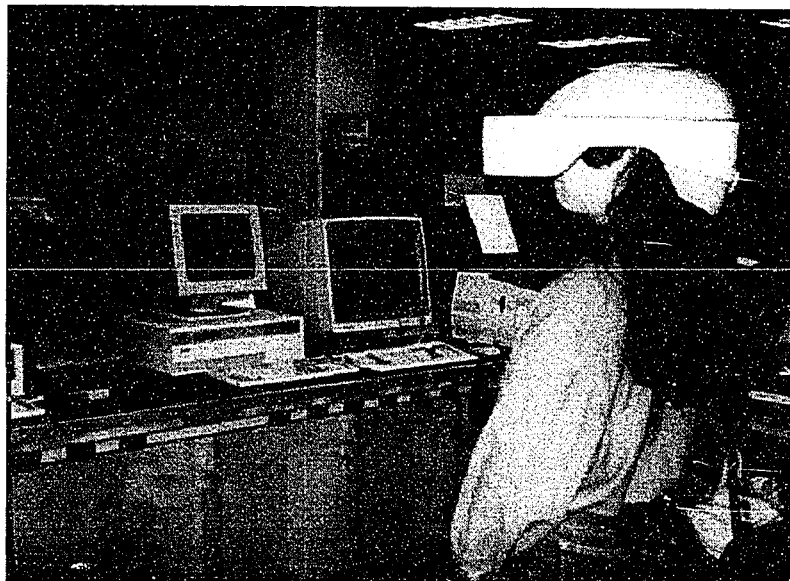


Figure 5.3 Target tracking experiment 2.



Figure 5.4 Target tracking experiment 3.



Figure 5.5 Target tracking experiment 4.

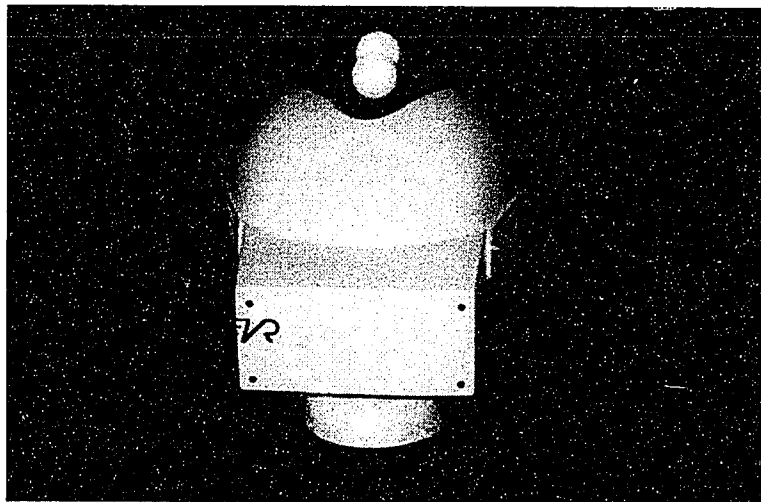


Figure 5.6 Two ping-pong balls glued together can be a nice target.

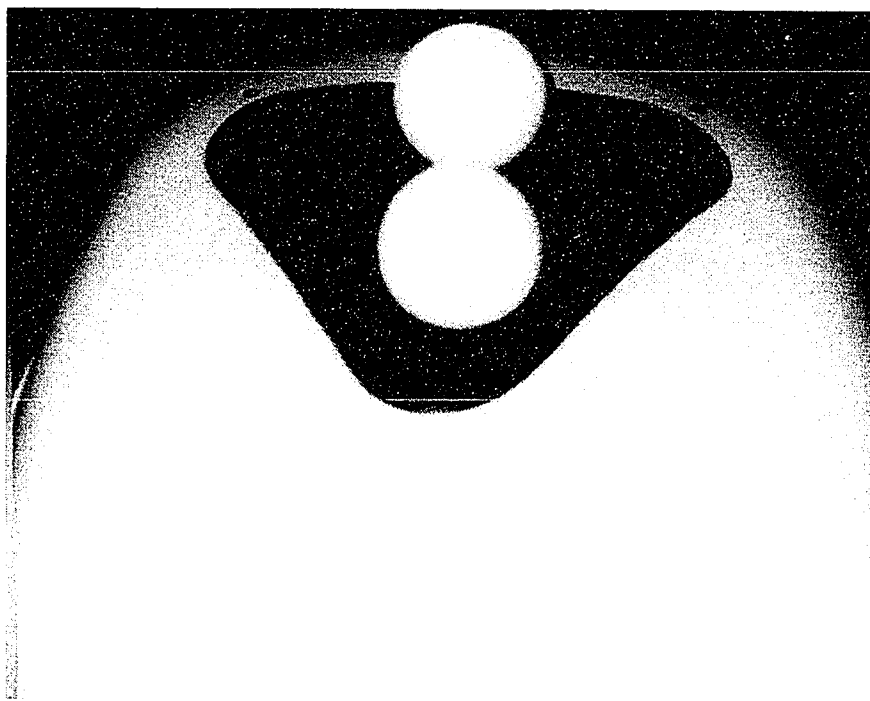


Figure 5.7 A close view of two white ping-pong balls.

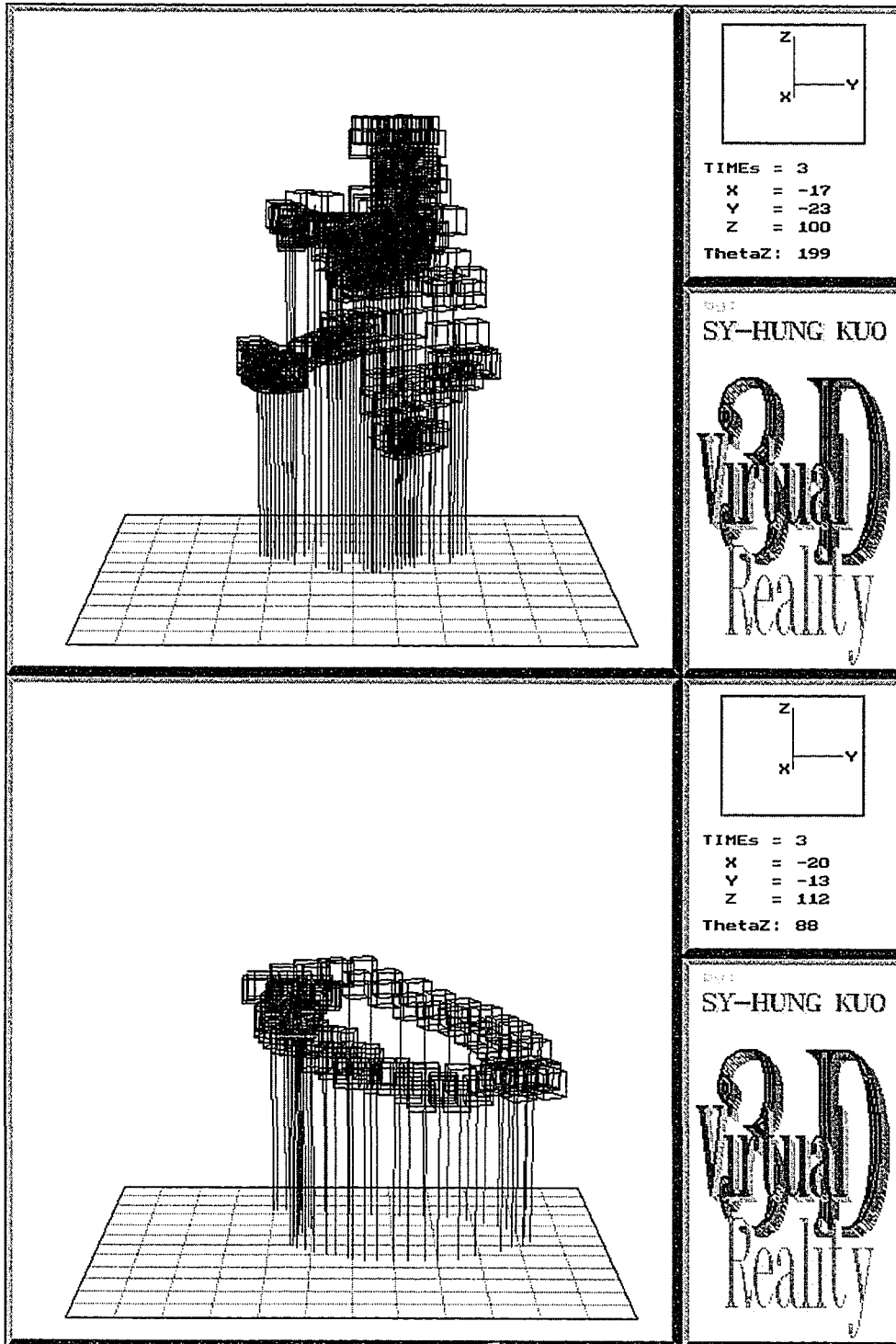


Figure 5.8 Experimental tracking patterns.

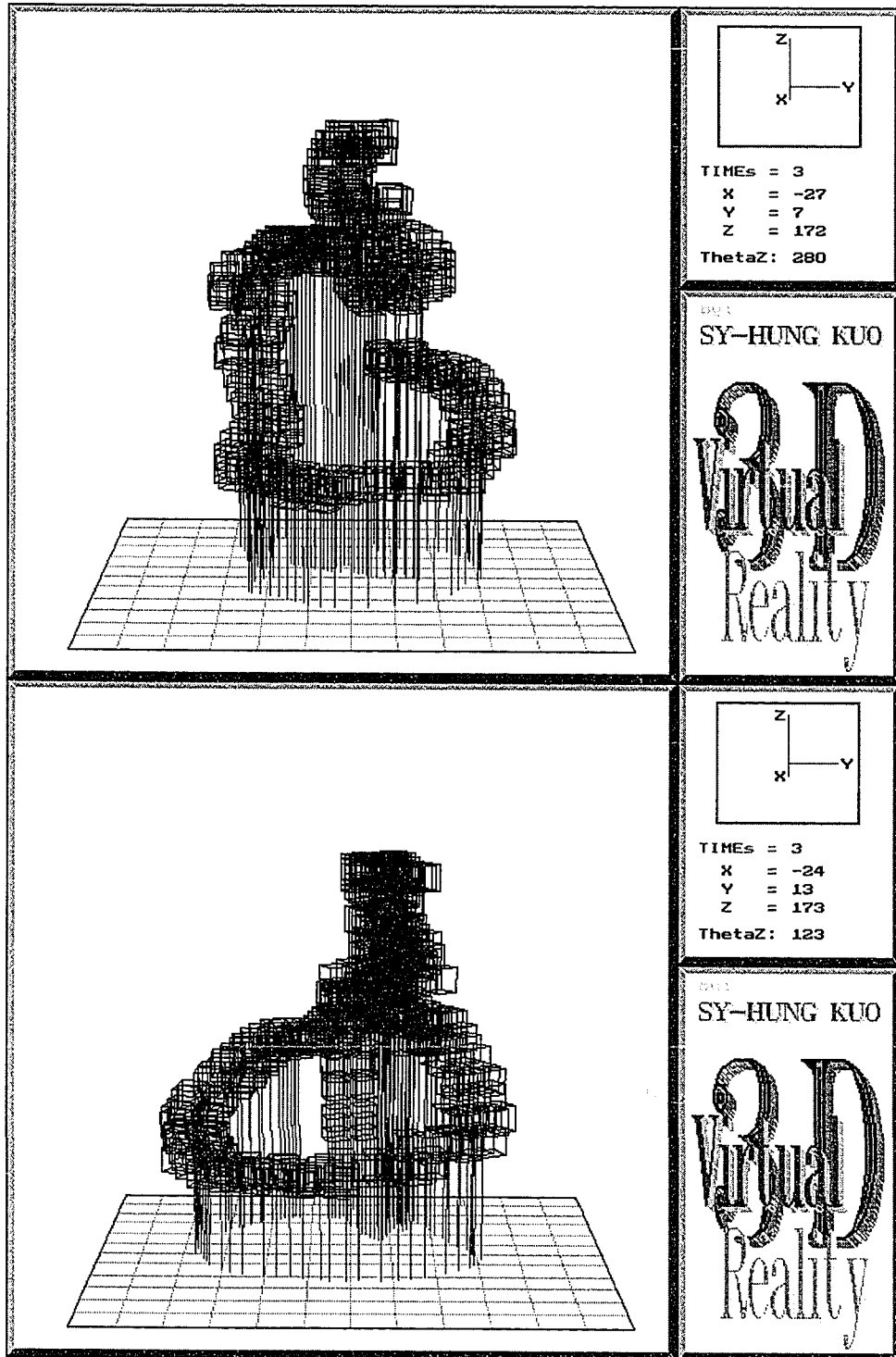


Figure 5.9 Experimental tracking patterns.

5.3 Telerobotic Control

Telerobotics studies the remote control of distant robots. Normally, a human operator is in the loop and acts as a supervisor. The top-down model control approach developed by Nguyen and Stark [1993] has been successfully applied in the control of a remote robot. Our vision system provides accurate visual information for the kinematic recovery of the robots, which is essential in telerobotic control. Supposing that the autonomous operation is inoperative, the human operator, with correct kinematic information of the robot, can directly intervene and bring the robot under control.

In robotic applications based on the top-down model approach, OSVEs (On the Scene Visual Enhancements) are placed at the strategic locations on the robot Nguyen [1992]. Under the control of the 3D model, our vision system tracks and simultaneously computes the 3D information of multiple OSVEs on the robot. This provides the necessary information for kinematic recovery which is crucial in controlling a remote robot based on the visual feedback (see Figures 5.10 and 5.11).

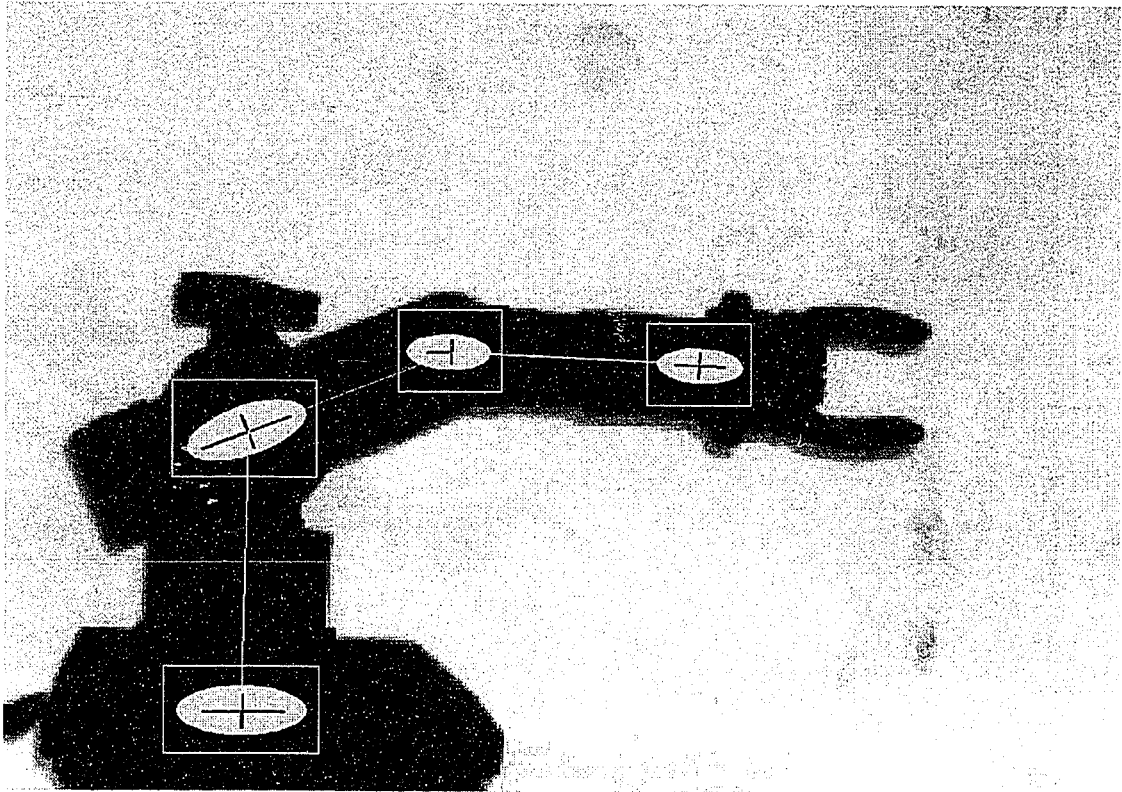


Figure 5.10 OSVEs for the 3D kinematic recovery of a robot.

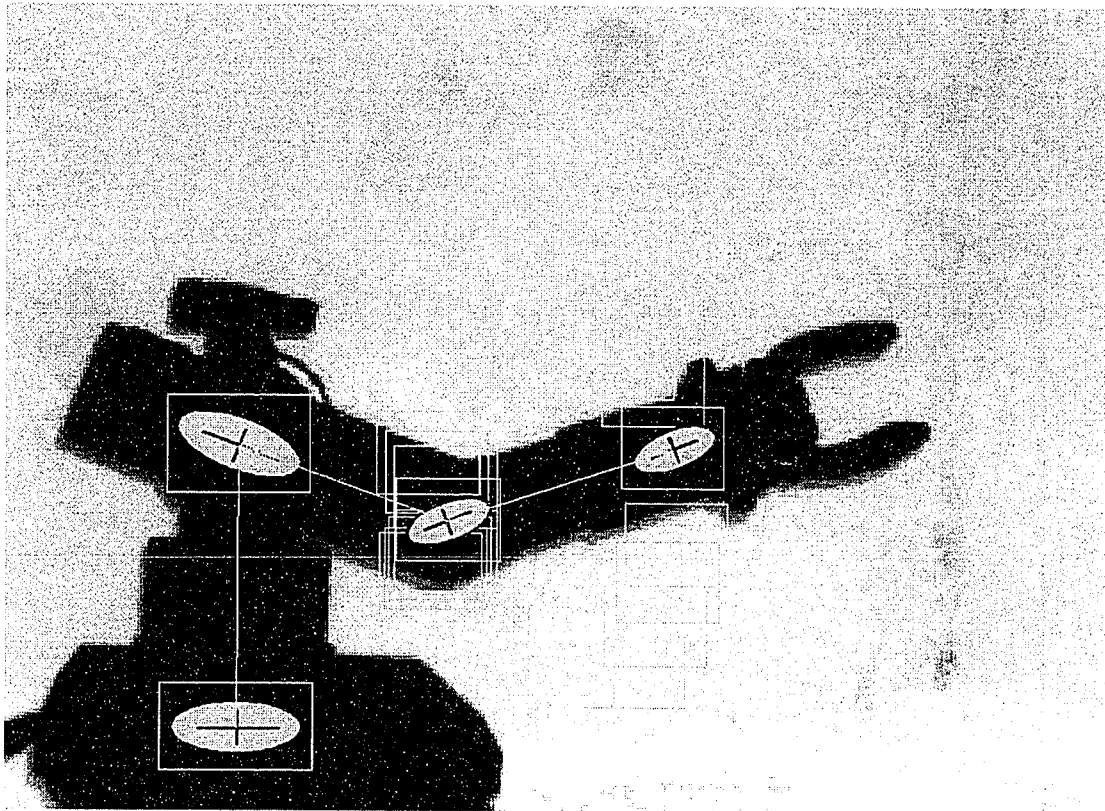


Figure 5.11 In the kinematic recovery, full kinematic information about the robot, including link length and link direction, was utilized in the experiment.

Chapter 6

Conclusion and Future Research

6.1 Conclusion

From the experimental processes and results I ascertained that a two-camera system is superior to single camera system. The two-camera system is more robust and more flexible. It is object independent and area invariant. On the other hand, it requires a very careful setup during the camera calibration process because the two cameras must be perfectly aligned to each other.

The single camera system has its advantage of faster performance, considering the manipulation of only one frame at a time compared with that of two frames at a time for a two-camera system. The single camera system can be useful when the size of the object is fixed and the object is symmetric. The single camera system is also easier to set up. Depending on the situation, the single camera system might be a better choice.

6.2 Recommendations for Future Research

A top-down model control method should be added to the system, so that object recognition can be achieved by accessing a 3D model knowledge base and comparing the outline of the object to a suitable 3D model in the knowledge base. When the objects can be recognized, more applications will be available.

A pair of small CCD cameras can be mounted on a stepping motor to provide a larger field of view. If two small CCD cameras are not available, two mirrors mounted on two stepping motors can provide the reflected version of images to two TV cameras and enlarge the field of view.

Networking multiple computers can increase the system's overall performance. For instance, two computers can deal with image processing, so that the computation needs for recovering 3D information based on the two-camera case can be divided into two sequences and processed by the two computers concurrently. At the same time, the third computer controls the stepping motor mentioned earlier, and the fourth computer displays the 3D graphical model of the object.

Most of the computation burden in the developed vision system is due to the low level moment image processing. If we can implement the algorithm in hardware with a parallel processing architecture and use the current FPGA technology, the processing time can be significantly decreased.

Appendix A

Source Code Listing of the Developed Vision System

```

/*****
/* KUO_DEF.H : Data Type Declarations and Constant Definitions */
/* for the Computer Vision Project */
/* */
/* by: Sy-Hung Kuo */
*****/

typedef unsigned char    byte;
typedef unsigned int     word;
typedef unsigned long    dword;

#define ABS(a)           ((a) < 0) ? -(a) : (a)
#define fABS(a)          ((a) < 0.0) ? -(a) : (a)
#define MIN(a,b)        ((a) < (b)) ? (a) : (b)
#define MAX(a,b)        ((a) > (b)) ? (a) : (b)
#define SQR(a)           ((a) * (a))

#define VGA_WIDTH        1024
#define VGA_HEIGHT       768

/* Stereoscope Background Color */
#define ERASE_Color      99

#define XSAM_BASE_ADDR   0xD000
#define XSAMSIZE         1024
#define XSAM_OFF         0

/* Switch to next memory segment in the frame grabber */
#define SegSwitch        0x308
#define SegLimit         0xFFFF

/* Monitor Specifications */
#define Xmax0             256
#define Xmax1             272
#define Xmax2             320
#define Ymax              240

/* Default Parameters Setting */
#define CamMax            6 /* Maximum Cameras Number */
#define CamNum            3 /* Current Number of Cameras */
#define HalfofWidth       30 /* Half of the Width of the Windows */
#define Window_Center_X   160 /* X Component of Window's Center */
#define Window_Center_Y   120 /* Y Component of Window's Center */
#define AREA_MIN          4 /* Minimum Number of Pixels of AREA */

```

```
/* often used constants or macros */
#define PI          3.14159265358979323846
#define RADIAN      (PI / 180.0)
#define DEGREE      (180.0 / PI)
#define PI180       RADIAN /* PI180 := PI/180 */
#define DEGREEdiv2  (DEGREE / 2.0)

/* Video System's Pixel Aspect Ratio (PAR) */
#define VIDEO_PAR   0.86

/* Define the length of a Cross. The Cross is used to represent the */
/* orientation of a mobile object in the 3D space. */
#define CrossXY     20

/* Power Switch */
#define ON           0x01
#define OFF          0x00
```

```

/*****
/*  Kuo_GLOB.H : Global Variables Declarations and Initializations      */
/*                                     for the Computer Vision Project    */
/*                                     by: Sy-Hung Kuo                    */
*****/

byte dig_buf[XSAMSIZE], dis_buf[XSAMSIZE];
byte far *LineBuffer, far *LineBuffer0;
byte far *XPixel, far *XPixel0;
byte far *ImageBuffer;

/*
  0: 256x240 (non linear).
  1: 272x240 ( < 64K).
  2: 320x240 ( > 64K).
*/
word XSAM_FLAG = 2;

word XMAX = Xmax2, YMAX = Ymax;

word THRESHOLD = 60;
byte CamID = 0x00;
unsigned long Image_Size;

struct region
{
  int xc, yc, ax, by, angle, thres;
  float fxc, fyc;
  word area;
};

struct region *WindowPTR[CamNum], Window[CamNum];
struct region *objectPTR[CamNum], object[CamNum];

```

```
/*
*****
/* BIOSKEY.H : Keyboard Interface Definitions */
/*
/* by: Sy-Hung Kuo
/*
*****

```

```
// BIOS Keyboard Interface (Part I)
```

```
#define KEY_ESC      0x11B
#define KEY_SPACE   0x3920
#define KEY_LEFT    0x4B00
#define KEY_RIGHT   0x4D00
#define KEY_UP      0x4800
#define KEY_DOWN    0x5000
#define KEY_NumLEFT 0x4B34
#define KEY_NumRIGHT 0x4D36
#define KEY_NumUP   0x4838
#define KEY_NumDOWN 0x5032
#define KEY_ENTER   0x1C0D
#define KEY_SPACE   0x3920
#define KEY_HOME    0x4700
#define KEY_END     0x4F00
#define KEY_PGUP    0x4900
#define KEY_PGDN    0x5100
#define KEY_INS     0x5200
#define KEY_DEL     0x5300
#define KEY_MINUS   0x4A2D
#define KEY_PLUS    0x4E2B
#define KEY_F1      0x3B00
#define KEY_F2      0x3C00
#define KEY_F3      0x3D00
#define KEY_F4      0x3E00
#define KEY_F5      0x3F00
#define KEY_F6      0x4000
#define KEY_F7      0x4100
#define KEY_F8      0x4200
#define KEY_F9      0x4300
#define KEY_F10     0x4400

```

```
// BIOS Keyboard Interface (Part II)
```

```
#define ALT_PRESSED 0x0008
#define CTRL_PRESSED 0x0004
#define LfSHIFT_PRESSED 0x0002
#define RtSHIFT_PRESSED 0x0001
#define SHIFT_PRESSED 0x0003

```

```
// BIOS Keyboard Interface (Part III)
```

```
#define KEY_a      0x1E61
#define KEY_A      0x1E41
#define KEY_b      0x3062
#define KEY_B      0x3042
#define KEY_c      0x2E63
#define KEY_C      0x2E43
#define KEY_d      0x2064

```

```
#define KEY_D          0x2044
#define KEY_e          0x1265
#define KEY_E          0x1245
#define KEY_f          0x2166
#define KEY_F          0x2146
#define KEY_g          0x2267
#define KEY_G          0x2247
#define KEY_h          0x2368
#define KEY_H          0x2348
#define KEY_i          0x1769
#define KEY_I          0x1749
#define KEY_j          0x246A
#define KEY_J          0x244A
#define KEY_k          0x256B
#define KEY_K          0x254B
#define KEY_l          0x266C
#define KEY_L          0x264C
#define KEY_m          0x326D
#define KEY_M          0x324D
#define KEY_n          0x316E
#define KEY_N          0x314E
#define KEY_o          0x186F
#define KEY_O          0x184F
#define KEY_p          0x1970
#define KEY_P          0x1950
#define KEY_q          0x1071
#define KEY_Q          0x1051
#define KEY_r          0x1372
#define KEY_R          0x1352
#define KEY_s          0x1F73
#define KEY_S          0x1F53
#define KEY_t          0x1474
#define KEY_T          0x1454
#define KEY_u          0x1675
#define KEY_U          0x1655
#define KEY_v          0x2F76
#define KEY_V          0x2F56
#define KEY_w          0x1177
#define KEY_W          0x1157
#define KEY_x          0x2D78
#define KEY_X          0x2D58
#define KEY_y          0x1579
#define KEY_Y          0x1559
#define KEY_z          0x2C7A
#define KEY_Z          0x2C5A
#define KEY_0          0xB30
#define KEY_1          0x231
#define KEY_2          0x332
#define KEY_3          0x433
#define KEY_4          0x534
#define KEY_5          0x635
#define KEY_6          0x736
#define KEY_7          0x837
#define KEY_8          0x938
#define KEY_9          0xA39
```

```
// BIOS Keyboard Interface (Part IV)
#define Alt_A      0x1E00
#define Alt_B      0x3000
#define Alt_C      0x2E00
#define Alt_D      0x2000
#define Alt_E      0x1200
#define Alt_F      0x2100
#define Alt_G      0x2200
#define Alt_H      0x2300
#define Alt_I      0x1700
#define Alt_J      0x2400
#define Alt_K      0x2500
#define Alt_L      0x2600
#define Alt_M      0x3200
#define Alt_N      0x3100
#define Alt_O      0x1800
#define Alt_P      0x1900
#define Alt_Q      0x1000
#define Alt_R      0x1300
#define Alt_S      0x1F00
#define Alt_T      0x1400
#define Alt_U      0x1600
#define Alt_V      0x2F00
#define Alt_W      0x1100
#define Alt_X      0x2D00
#define Alt_Y      0x1500
#define Alt_Z      0x2C00
```

```
// BIOS Keyboard Interface (Part V)
#define SYM_COMMA  0x332C
#define SYM_PERIOD 0x342E
#define SYM_LESS   0x333C
#define SYM_GREATER 0x343E
#define SYM_SLASH  0x352F
#define SYM_QUESTION 0x353F
```

```
/*
*****
/* MONITOR.H : Virtual Monitor Specifications */
/*
/* by: Sy-Hung Kuo */
*****
*/
```

```
#define Monitor_x0a 76
#define Monitor_x0b 396
#define Monitor_y0a 60
#define Monitor_y0b 300
#define Monitor_x1a 464
#define Monitor_x1b 784
#define Mon_OffsetX 89
#define Mon_OffsetY 55
```

```

/*****
/*  Computer Vision System:                                     */
/*                                                                 */
/*                               by: Sy-Hung Kuo                 */
/*-----*/
/*  Filename: KUO_MAIN.C                                       */
/*-----*/
/*  This file contains the main function.                       */
/*                                                                 */
/*.....*/
/*                                                                 */
/*  Last Modified : April 1, 1994                               */
/*****

#include <stdio.h>
#include <conio.h>
#include <io.h>
#include <alloc.h>
#include <fcntl.h>
#include <dos.h>
#include <math.h>
#include <string.h>
#include <process.h>
#include <graphics.h>

#include "bioskey.h"
#include "monitor.h"
#include "kuo_def.h"
#include "kuo_glob.h"

void main(void)
{ int i;

  clrscr();
  setup_3D_screen();

  XPixel = XPixel0 = MK_FP(XSAM_BASE_ADDR, 0x0000);

  Image_Size = (long)XMAX*(long)YMAX;
  if ((ImageBuffer = malloc(Image_Size)) == NULL)
  {
    closegraph();
    printf("\nMemory Not Enough: Cannot allocate ImageBuffer."); getch();
    exit(1);
  }

  if ((LineBuffer0 = malloc(4*XMAX)) == NULL)
  {
    closegraph();
    printf("\nMemory Not Enough: Cannot allocate LineBuffer0."); getch();
    exit(2);
  }
}

```



```

LineBuffer = LineBuffer0;

init_fgrabber();

for (i=0; i<CamNum; ++i)
{
    WindowPTR[i] = (struct region *)&Window[i];
    WindowPTR[i]->xc = Window_Center_X;
    WindowPTR[i]->yc = Window_Center_Y;
    WindowPTR[i]->ax = WindowPTR[i]->by = HalfofWidth;
    WindowPTR[i]->angle = 0;
}

load_xcode_to_buf(XSAM_FLAG);

draw_monitors_frame();
next_picture(0x01);
equalize_picture(Monitor_x1a, Monitor_y0a, 0x01, 1);
next_picture(0x00);
equalize_picture(Monitor_x0a, Monitor_y0a, 0x00, 1);
window_region((struct region *)&Window[0x00]);

for (;;) {
    central_Moment_Loop();
    if ((bioskey(0) == KEY_ESC) && (bioskey(2) & CTRL_PRESSED)) break;
}

closegraph();

free(ImageBuffer);
free(LineBuffer0);
}

```

```

/*****
/* Computer Vision System: */
/* */
/* by: Sy-Hung Kuo */
/*-----*/
/* Filename: STEREO.C */
/*-----*/
/* This file contains: */
/* */
/* + the subroutines to track multiple mobile objects */
/* in the 3D space. */
/* */
/* + the subroutines to transmit and receive data through RS-232C. */
/* */
/* + the subroutines to recover 3D position and orientation of */
/* multiple mobile objects. */
/* */
/* + the subroutine to automatically move and adjust */
/* multiple windows. */
/* */
/*.....*/
/* */
/* Last Modified : April 1, 1994 */
/*****

```

```

#include <stdio.h>
#include <io.h>
#include <alloc.h>
#include <fcntl.h>
#include <dos.h>
#include <math.h>
#include <string.h>
#include <graphics.h>
#include <time.h>

```

```

#include "bioskey.h"
#include "monitor.h"
#include "kuo_def.h"

```

```

#define MagicFactor 2.3

```

```

#define SetupTimes 25
#define TestTimes 20
#define Setup_X 150
#define Setup_Y 380
#define Text_X 60
#define Fix4_X 40
#define Text_Y 360
#define ABS_X 80
#define ABS_Y 420
#define AVG_X 80
#define AVG_X2 120
#define AVG_Y 540

```

```

#define AVG_DefY    35
#define AVG_CntY    3

#define COM1    0x3F8
#define COM2    0x2F8

#define fg_Xmax    (XMAX-1)
#define fg_Ymax    (YMAX-1)
#define fg_Xmin    0
#define fg_Ymin    0

#define Camera0_to_Ground    280.0
#define Camera1_to_Ground    277.5

#define First_YES    1
#define First_NO    0

#define YES    1
#define NO    0

#define CounterClockwise    1
#define Clockwise    0

extern struct region
{
    int xc, yc, ax, by, angle, thres;
    float fxc, fyc;
    word area;
};

extern struct region *objectPTR[CamNum], Window[CamNum];
extern byte far *XPixel0, *XPixel;
extern word XMAX, YMAX;
extern byte CamID;

float A1[CamNum], D1[CamNum], A2[CamNum], D2[CamNum], A3[CamNum], D3[CamNum];
float aa[CamNum], bb[CamNum], cc[CamNum];
float camera_org_x[CamNum], camera_org_y[CamNum];
double camera_delta_x[CamNum], camera_delta_y[CamNum], FocalLen[CamNum];
float Baseline, delta_D, delta_X[CamNum];
float XX[CamNum], YY[CamNum], ZZ;
float yt_world_coord[CamNum], yb_world_coord[CamNum];
float yt_image_coord[CamNum], yb_image_coord[CamNum];
float x_gain_base[CamNum], y_gain_base[CamNum];
float xl_world_coord[CamNum], xr_world_coord[CamNum];
float xl_image_coord[CamNum], xr_image_coord[CamNum];
word COMx;
long oldtheta = 0L, newtheta, theta;
int FirstCheck[CamNum];
int AA, CtoG[CamNum], MaxArea[CamNum];

/* Function Prototypes */
void one_camera_tracking_float(void);

```

```

void one_camera_tracking_int(void);
void intelligent_setup(byte camera);
void intelligent_setup2(byte camera);
void intelligent_setup3(byte camera);
void two_camera_tracking_float(void);
void two_camera_tracking_float2(void);
void two_camera_tracking_int(void);
void auto_tracking_and_resizing_window(byte camera);
void calculate_float_xy(byte camera);
void calculate_int_xy(byte camera);
void send_serial_3Dx(void);
void send_serial_2cam_3Dx(void);
void cal_coeffs(byte camera);
void default_setting(void);
void cal_base_gain(void);
word track_orientation(int angle, byte camera);
void init_UART_kuo(word COMp);
void out_char(byte c);
void show_picture(int x, int y);
void show_one_camera_text(void);
void show_two_camera_text(void);
void warning_sound(void);

// Objects tracking subroutine for single camera vision system
// (floating point version)
void one_camera_tracking_float(void)
{ static char outstr[80] = "";
  static char outstr2[AVG_CntY][80] = {"", "", ""};
  static float sum, avg, DefSum, Def, StdDev, Height[TestTimes];
  int j, x, ct, py, LocY, sumA, avgA;

  cleanup_display_board();
  setcolor(255);
  set_triplex_font(4);
  show_one_camera_text();

  if (CamID == 0x00) x = Monitor_x0a;
  else x = Monitor_x1a;
  next_picture(CamID);
  equalize_picture(x, Monitor_y0a, CamID, 0);

  default_setting();
  intelligent_setup3(CamID);
  cal_coeffs(CamID);
  cal_base_gain();
  SandTextxy2(Text_X+Fix4_X, Text_Y, "Real Time : (Floating Point Version)");
  set_default_font(1);

  /* START here */
  LocY = 0;
  init_UART_kuo(COM1);
  FirstCheck[CamID] = First_YES;
  for (;;) {
    for (ct=0; ct<TestTimes; ++ct)
    {

```

```

next_picture(CamID);
centroid_float((struct region *)&Window[CamID], 0, CamID);
auto_tracking_and_resizing_window(CamID);
fgrabber(ON);

if (kbhit()) if(bioskey(0) == KEY_ESC) return;

set_default_font(1);
if (objectPTR[CamID]->area > 0)
{
    send_serial_3Dx();
    setcolor(ERASE_Color);
    outtextxy(ABS_X+Fix4_X, ABS_Y, outstr);
    sprintf(outstr, "HEIGHT=%4.1f, AREA=%5d", ZZ, AA);
    setcolor(255);
    outtextxy(ABS_X+Fix4_X, ABS_Y, outstr);
}
Height[ct] = ZZ;
if (ct == 0) { sum = ZZ; sumA = AA; }
else { sum += ZZ; sumA += AA; }
}
avg = sum / TestTimes;
avgA = sumA / TestTimes;
DefSum = 0;
for (j=0; j<TestTimes; ++j)
{
    Def = Height[j] - avg;
    DefSum += (Def * Def);
}
StdDev = (float) sqrt((double)DefSum/(double)TestTimes);
py = AVG_Y + AVG_DefY * LocY;
setcolor(ERASE_Color);
outtextxy(AVG_X2, py, (char *)&outstr2[LocY][0]);
sprintf((char *)&outstr2[LocY][0], "MH=%4.1f, SD=%4.1f, MA=%d", avg, StdDev,
    avgA);
setcolor(255);
outtextxy(AVG_X2, py, (char *)&outstr2[LocY][0]);
if (++LocY == AVG_CntY) LocY = 0;
}
}

// Objects tracking subroutine for single camera vision system
// (integer version)
void one_camera_tracking_int(void)
{ static char outstr[80] = "";
  int x;

  cleanup_display_board();
  setcolor(255);
  set_triplex_font(4);
  show_one_camera_text();

  if (CamID == 0x00) x = Monitor_x0a;
  else                x = Monitor_x1a;
  next_picture(CamID);

```

```

equalize_picture(x, Monitor_y0a, CamID, 0);

default_setting();
intelligent_setup3(CamID);
cal_coeffs(CamID);
cal_base_gain();
SandTextxy2(Text_X+Fix4_X, Text_Y, "Real Time : (Integer Version)");
set_default_font(1);

/* START here */
init_UART_kuo(COM1);
FirstCheck[CamID] = First_YES;
for (;;) {
    next_picture(CamID);
    centroid_int((struct region *)&Window[CamID], 0, CamID);
    auto_tracking_and_resizing_window(CamID);

    fgrabber(ON);
    if (objectPTR[CamID]->area > 0)
    {
        send_serial_3Dx();
        setcolor(ERASE_Color);
        set_default_font(1);
        outtextxy(ABS_X+Fix4_X, ABS_Y, outstr);
        sprintf(outstr, "HEIGHT=%4.1f, AREA=%5d", ZZ, AA);
        setcolor(255);
        outtextxy(ABS_X+Fix4_X, ABS_Y, outstr);
    }
    if (kbhit()) if(bioskey(0) == KEY_ESC) break;
}
}

/* find the minimum area for the object on the ground */
void intelligent_setup(byte camera)
{ int ct, i;
  word MinArea;

  set_triplex_font(4);
  SandTextxy2(Setup_X, Setup_Y, "Please wait for Intelligent Setup ...");

  ct = 0;
  for (i=0; i<SetupTimes; i++)
  {
    next_picture(camera);
    centroid_int((struct region *)&Window[camera], 0, camera);
    auto_tracking_and_resizing_window(CamID);
    if (objectPTR[camera]->area > AREA_MIN)
    {
        if (ct == 0) MinArea = objectPTR[camera]->area;
        else
            if (objectPTR[camera]->area < MinArea) MinArea = objectPTR[camera]->area;
        ct++;
    }
  }
}
if (MinArea > AREA_MIN) A1[camera] = (float) MinArea;

```

```

    set_triplex_font(4);
    Erase_SandTextxy(Setup_X, Setup_Y, "Please wait for Intelligent Setup ...");
}

/* find the most frequently appeared area for the object on the ground */
void intelligent_setup2(byte camera)
{ int ct, i, j, SameAreaFlag, count, Most;
  static struct xobject
  {
    int area;
    int freq;
  } ob[SetupTimes];

  for (i=0; i<SetupTimes; i++)
    ob[i].area = ob[i].freq = 0;
  set_triplex_font(4);
  SandTextxy2(Setup_X, Setup_Y, "Please wait for Intelligent Setup ...");

  ct = 0;
  for (j=0; j<SetupTimes; j++)
  {
    SameAreaFlag = 0;
    next_picture(camera);
    centroid_int((struct region *)&Window[camera], 0, camera);
    auto_tracking_and_resizing_window(CamID);
    if (objectPTR[camera]->area > AREA_MIN)
    {
      if (ct == 0)
      {
        ob[0].area = objectPTR[camera]->area;
        ob[0].freq++;
        count = 1;
      }
      else
        for (i=0; i<count; i++)
        {
          if (objectPTR[camera]->area == ob[i].area)
          {
            SameAreaFlag = 1;
            ob[i].freq++;
            break;
          }
        }
      if (!SameAreaFlag)
      {
        ob[count].area = objectPTR[camera]->area;
        ob[count++].freq++;
      }
      ct++;
    }
  }

  for (i=0; i<count; i++)
  {

```

```

        if (i == 0) { ct = 0; Most = ob[0].freq; }
        else
            if (ob[i].freq > Most) { ct = i; Most = ob[i].freq; }
    }

    A1[camera] = (float) ob[ct].area;

    set_triplex_font(4);
    Erase_SandTextxy(Setup_X, Setup_Y, "Please wait for Intelligent Setup ...");
}

/* find the average area for the object on the ground */
void intelligent_setup3(byte camera)
{ int ct, i;
  word SumArea;

  set_triplex_font(4);
  SandTextxy2(Setup_X, Setup_Y, "Please wait for Intelligent Setup ...");

  ct = 0;
  for (i=0; i<SetupTimes; i++)
  {
    next_picture(camera);
    centroid_int((struct region *)&Window[camera], 0, camera);
    auto_tracking_and_resizing_window(CamID);
    if (ct == 0) SumArea = objectPTR[camera]->area;
    else SumArea += objectPTR[camera]->area;
    ct++;
  }
  A1[camera] = (float) ((float)SumArea/(float)SetupTimes);

  set_triplex_font(4);
  Erase_SandTextxy(Setup_X, Setup_Y, "Please wait for Intelligent Setup ...");
}

// Objects tracking subroutine for two-camera vision system
// (floating point version)
void two_camera_tracking_float(void)
{ static char outstr[80] = "";
  static char outstr2[AVG_CntY][80] = {"", "", ""};
  static float sum, avg, DefSum, Def, StdDev, Height[TestTimes];
  int i, j, ct, py, LocY;

  cleanup_display_board();
  setcolor(255);
  set_triplex_font(4);
  show_two_camera_text();
  SandTextxy2(Text_X, Text_Y, "Real Time : (Floating Point Version 1.0c)");
  set_default_font(1);
  next_picture(0x00); equalize_picture(Monitor_x0a, Monitor_y0a, 0x00, 0);
  next_picture(0x01); equalize_picture(Monitor_x1a, Monitor_y0a, 0x01, 0);

  /* START here */
  LocY = 0;

```



```

init_UART_kuo(COM1);
default_setting();
cal_base_gain();
FirstCheck[0] = FirstCheck[1] = First_YES;

for (;;) {
    for (ct=0; ct<TestTimes; ++ct)
    {
        for (i=0; i<CamNum; ++i)
        {
            if (CamID == 0x00) CamID = 0x01; else CamID = 0x00;
            next_picture(CamID);
            centroid_float((struct region *)&Window[CamID], 0, CamID);
            auto_tracking_and_resizing_window(CamID);

            calculate_float_xy(CamID);
            fgrabber(ON);
        }
        if (kbhit()) if(bioskey(0) == KEY_ESC) return;

        set_default_font(1);
        if (objectPTR[CamID]->area > 0)
        {
            send_serial_2cam_3Dx();
            setcolor(ERASE_Color);
            outtextxy(ABS_X, ABS_Y, outstr);
            sprintf(outstr, "HEIGHT=%4.1f, X0=%4.1f, X1=%4.1f", ZZ, XX[0x00],
                XX[0x01]);
            setcolor(255);
            outtextxy(ABS_X, ABS_Y, outstr);
        }
        Height[ct] = ZZ;
        if (ct == 0) sum = ZZ;
        else sum += ZZ;
    }

    avg = sum / TestTimes;
    DefSum = 0;
    for (j=0; j<TestTimes; ++j)
    {
        Def = Height[j] - avg;
        DefSum += (Def * Def);
    }
    StdDev = (float) sqrt((double)DefSum/(double)TestTimes);
    py = AVG_Y + AVG_DefY * LocY;
    setcolor(ERASE_Color);
    outtextxy(AVG_X, py, (char *)&outstr2[LocY][0]);
    sprintf((char *)&outstr2[LocY][0], "Avg.=%4.1f, Std. Deviation=%4.1f", avg,
        StdDev);
    setcolor(255);
    outtextxy(AVG_X, py, (char *)&outstr2[LocY][0]);
    if (++LocY == AVG_CntY) LocY = 0;
}
set_default_font(1);
}

```

```

// Objects tracking subroutine for two-camera vision system
// (floating point version 2)
void two_camera_tracking_float2(void)
{ static char outstr[80] = "";
  static char outstr2[AVG_CntY][80] = {"", "", ""};
  float min, max, sum, avg;
  int i, ct, py, LocY;

  cleanup_display_board();
  setcolor(255);
  set_triplex_font(4);
  show_two_camera_text();
  SandTextxy2(Text_X, Text_Y, "Real Time : (Floating Point Version 1.0a)");
  set_default_font(1);
  next_picture(0x00); equalize_picture(Monitor_x0a, Monitor_y0a, 0x00, 0);
  next_picture(0x01); equalize_picture(Monitor_x1a, Monitor_y0a, 0x01, 0);

  /* START here */
  LocY = 0;
  init_UART_kuo(COM1);
  default_setting();
  cal_base_gain();
  FirstCheck[0] = FirstCheck[1] = First_YES;
  for (;;) {
    for (ct=0; ct<TestTimes; ++ct)
    {
      for (i=0; i<CamNum; ++i)
      {
        if (CamID == 0x00) CamID = 0x01; else CamID = 0x00;
        next_picture(CamID);
        centroid_float((struct region *)&Window[CamID], 0, CamID);
        auto_tracking_and_resizing_window(CamID);

        calculate_float_xy(CamID);
        fgrabber(ON);
      }
      if (kbhit()) if(bioskey(0) == KEY_ESC) return;

      set_default_font(1);
      if (objectPTR[CamID]->area > 0)
      {
        send_serial_2cam_3Dx();
        setcolor(ERASE_Color);
        outtextxy(ABS_X, ABS_Y, outstr);
        sprintf(outstr, "HEIGHT=%4.1f, X0=%4.1f, X1=%4.1f", ZZ, XX[0x00],
          XX[0x01]);
        setcolor(255);
        outtextxy(ABS_X, ABS_Y, outstr);
      }
      if (ct == 0) min = max = sum = ZZ;
      else sum += ZZ;
      if (ZZ < min) min = ZZ;
      if (ZZ > max) max = ZZ;
    }
    avg = sum / TestTimes;
  }
}

```

```

    py = AVG_Y + AVG_DefY * LocY;
    setcolor(ERASE_Color);
    outtextxy(AVG_X, py, (char *)&outstr2[LocY][0]);
    sprintf((char *)&outstr2[LocY][0], "Avg=%4.1f, Max=%4.1f, Min=%4.1f", avg,
        max, min);
    setcolor(255);
    outtextxy(AVG_X, py, (char *)&outstr2[LocY][0]);
    if (++LocY == AVG_CntY) LocY = 0;
}
set_default_font(1);
}

// Objects tracking subroutine for two-camera vision system
// (integer version)
void two_camera_tracking_int(void)
{ static char outstr[80] = "";
  int i;

  cleanup_display_board();
  setcolor(255);
  set_triplex_font(4);
  show_two_camera_text();
  SandTextxy2(Text_X, Text_Y, "Real Time : (Integer Version)");
  set_default_font(1);
  next_picture(0x00); equalize_picture(Monitor_x0a, Monitor_y0a, 0x00, 0);
  next_picture(0x01); equalize_picture(Monitor_x1a, Monitor_y0a, 0x01, 0);

  /* START here */
  init_UART_kuo(COM1);
  default_setting();
  cal_base_gain();
  FirstCheck[0] = FirstCheck[1] = First_YES;
  for (;;) {
    for (i=0; i<CamNum; ++i)
    {
      if (CamID == 0x00) CamID = 0x01; else CamID = 0x00;
      next_picture(CamID);
      centroid_int((struct region *)&Window[CamID], 0, CamID);
      auto_tracking_and_resizing_window(CamID);

      calculate_int_xy(CamID);
      fgrabber(ON);
    }

    if (objectPTR[CamID]->area > 0)
    {
      send_serial_2cam_3Dx();
      setcolor(ERASE_Color);
      set_default_font(1);
      outtextxy(ABS_X, ABS_Y, outstr);
      sprintf(outstr, "HEIGHT=%4.1f, X0=%4.1f, X1=%4.1f", ZZ, XX[0x00],
          XX[0x01]);
      setcolor(255);
      outtextxy(ABS_X, ABS_Y, outstr);
    }
  }
}

```

```

    if (kbhit()) if(bioskey(0) == KEY_ESC) break;
}
set_default_font(1);
}

void auto_tracking_and_resizing_window(byte camera)
{ int MagicWing;

/* automatically objects tracking and windows adjusting */
/* Algorithm is created by: Sy-Hung Kuo      06/15/1993 */
if (objectPTR[camera]->area > AREA_MIN)
{
    Window[camera].xc = objectPTR[camera]->xc;
    Window[camera].yc = objectPTR[camera]->yc;
    MagicWing = (int) (MagicFactor * (float) sqrt(objectPTR[camera]->area));
    if (MagicWing < HalfofWidth) MagicWing = HalfofWidth;
    Window[camera].ax = Window[camera].by = MagicWing;
}
else
{
    Window[camera].xc = (fg_Xmax - fg_Xmin) / 2;
    Window[camera].yc = (fg_Ymax - fg_Ymin) / 2;
    Window[camera].ax = Window[camera].xc;
    Window[camera].by = Window[camera].yc;
}

/* check boundaries */
if ((Window[camera].xc-Window[camera].ax) < fg_Xmin)
    Window[camera].xc = Window[camera].ax;
else
    if ((Window[camera].xc+Window[camera].ax) > fg_Xmax)
        Window[camera].xc = fg_Xmax - Window[camera].ax;
if ((Window[camera].yc-Window[camera].by) < fg_Ymin)
    Window[camera].yc = Window[camera].by;
else
    if ((Window[camera].yc+Window[camera].by) > fg_Ymax)
        Window[camera].yc = fg_Ymax - Window[camera].by;
}

void calculate_float_xy(byte camera)
{
    static float reg;

    if (objectPTR[camera]->area > 0)
    {
        reg = (float) sqrt((double) objectPTR[camera]->area / (double) A1[camera]);
        XX[camera] = (float) ((objectPTR[camera]->fxc - camera_org_x[camera]) *
            x_gain_base[camera] / reg);
        YY[camera] = (float) ((objectPTR[camera]->fyc - camera_org_y[camera]) *
            y_gain_base[camera] / reg);
        delta_X[camera] = (float) ((double) -(objectPTR[camera]->fxc -
            camera_org_x[camera]) * camera_delta_x[camera]);
    }
}

```

```

void calculate_int_xy(byte camera)
{
    static float reg;

    if (objectPTR[camera]->area > 0)
    {
        reg = (float)sqrt((double)objectPTR[camera]->area / (double)A1[camera]);
        XX[camera] = (float) ((float)objectPTR[camera]->xc - camera_org_x[camera])
            * x_gain_base[camera] / reg;
        YY[camera] = (float) ((float)objectPTR[camera]->yc - camera_org_y[camera])
            * y_gain_base[camera] / reg;
        delta_X[camera] = (float) ((double)-((float)objectPTR[camera]->xc -
            camera_org_x[camera]) * camera_delta_x[camera]);
    }
}

```

```

void send_serial_3Dx(void)
{
    static byte xc_h, xc_l, yc_h, yc_l, zc_h, zc_l;
    static byte theta_h, theta_l;
    static word ANGLE;
    static int X, Y, Z;
    static float reg, delta, Ax, Axb, subtotal;

    AA = objectPTR[CamID]->area;
    Ax = (float)AA;
    Axb = Ax*bb[CamID];
    delta = (float) sqrt((double)Axb*Axb - (double)4.0*Ax*(Ax*cc[CamID]-
        aa[CamID]));
    subtotal = -Axb + delta;
    if (subtotal < 0) subtotal = -Axb - delta;
    ZZ = CtoG[CamID] - (subtotal / (2.0*Ax));
    Z = (int)ZZ;

    reg = (float)sqrt((double)objectPTR[CamID]->area / (double)A1[CamID]);
    X = (int) ( (objectPTR[CamID]->xc - camera_org_x[CamID]) *
        x_gain_base[CamID] / reg);
    Y = (int) (- (objectPTR[CamID]->yc - camera_org_y[CamID]) *
        y_gain_base[CamID] / reg);
    ANGLE = track_orientation(objectPTR[CamID]->angle, CamID);

    xc_h = (X >> 8) & 0x00FF;
    xc_l = X & 0x00FF;
    yc_h = (Y >> 8) & 0x00FF;
    yc_l = Y & 0x00FF;
    zc_h = (Z >> 8) & 0x00FF;
    zc_l = Z & 0x00FF;
    theta_h = (ANGLE >> 8) & 0x00FF;
    theta_l = ANGLE & 0x00FF;

    out_char(0x1B);
    out_char(xc_h);
    out_char(xc_l);

```

```

    out_char(yc_h);
    out_char(yc_l);
    out_char(zc_h);
    out_char(zc_l);
    out_char(theta_h);
    out_char(theta_l);
    out_char(0x00);
}

void send_serial_2cam_3Dx(void)
{
    static byte xc_h, xc_l, yc_h, yc_l, zc_h, zc_l;
    static byte theta_h, theta_l;
    static word ANGLE;
    static int X, Y, Z;
    static float X1mBmX0, Z1, Ztemp;

    X1mBmX0 = delta_X[1] - Baseline - delta_X[0];
    Ztemp = FocalLen[0]*FocalLen[1]*X1mBmX0 + delta_X[0]*FocalLen[1]*delta_D;
    Z1 = Ztemp / (FocalLen[0]*delta_X[1] - FocalLen[1]*delta_X[0]);
    ZZ = Z1 + (delta_D*2) + Camera1_to_Ground;
    Z = (int) ZZ;
    Y = (int)-YY[0x00];
    X = (int) XX[0x00];
    ANGLE = track_orientation(objectPTR[0x00]->angle, 0x00);

    xc_h = (X >> 8) & 0x00FF;
    xc_l = X & 0x00FF;
    yc_h = (Y >> 8) & 0x00FF;
    yc_l = Y & 0x00FF;
    zc_h = (Z >> 8) & 0x00FF;
    zc_l = Z & 0x00FF;
    theta_h = (ANGLE >> 8) & 0x00FF;
    theta_l = ANGLE & 0x00FF;

    out_char(0x1B);
    out_char(xc_h);
    out_char(xc_l);
    out_char(yc_h);
    out_char(yc_l);
    out_char(zc_h);
    out_char(zc_l);
    out_char(theta_h);
    out_char(theta_l);
    out_char(0x00);
}

void cal_coeffs(byte camera)
{
    float A1D1, A1D1D1, A2D2, A2D2D2, A3D3, A3D3D3;

    // calculate a, b, c parameters from areas at three different points
    // that related to camera: AREA = a / (DISTANCE^2 + b*DISTANCE + c)

```

```

// Algorithm is created by: Sy-Hung Kuo      03/29/1993

A1D1 = A1[camera]*D1[camera];  A1D1D1 = A1D1*D1[camera];
A2D2 = A2[camera]*D2[camera];  A2D2D2 = A2D2*D2[camera];
A3D3 = A3[camera]*D3[camera];  A3D3D3 = A3D3*D3[camera];

bb[camera] = ((A3[camera]-A1[camera])*(A2D2D2-A1D1D1) -
              (A2[camera]-A1[camera])*(A3D3D3-A1D1D1)) /
              ((A3[camera]-A1[camera])*(A1D1-A2D2) -
              (A2[camera]-A1[camera])*(A1D1-A3D3));
cc[camera] = ((A1D1-A3D3)*bb[camera] - (A3D3D3-A1D1D1)) /
              (A3[camera]-A1[camera]);
aa[camera] = A1D1D1 + A1D1*bb[camera] + A1[camera]*cc[camera];
}

void default_setting(void)
{ int i;

  /*+++++*/
  /* START of: two cameras system's parameters */
  /*+++++*/

  // original point of Camera 0 := 0x00
  camera_org_x[0] = (float) 165.38;
  camera_org_y[0] = (float) 122.60;

  // original point of Camera 1 := 0x01
  camera_org_x[1] = (float) 127.10;
  camera_org_y[1] = (float) 113.10;

  Baseline = 20.0;
  delta_D = 2.5;
  FocalLen[0] = 1.25;
  FocalLen[1] = 1.25;

  // left bottom point of Camera 0 := 0x00
  xl_image_coord[0] = (float) 141.60;
  yb_image_coord[0] = (float) 141.43;
  xl_world_coord[0] = -10.0;
  yb_world_coord[0] = -10.0;
  // righth top point of Camera 0 := 0x00
  xr_image_coord[0] = (float) 188.90;
  yt_image_coord[0] = (float) 103.40;
  xr_world_coord[0] = +10.0;
  yt_world_coord[0] = +10.0;

  // left bottom point of Camera 1 := 0x01
  xl_image_coord[1] = (float) 103.44;
  yb_image_coord[1] = (float) 133.00;
  xl_world_coord[1] = -10.0;
  yb_world_coord[1] = -10.0;
  // righth top point of Camera 1 := 0x01
  xr_image_coord[1] = (float) 150.95;
  yt_image_coord[1] = (float) 92.90;
  xr_world_coord[1] = +10.0;

```

```

yt_world_coord[1] = +10.0;

// increment length per dot of camera 0's coordinate systems
camera_delta_x[0] = (FocalLen[0] * (xr_world_coord[0] - xl_world_coord[0]))
    / (Camera0_to_Ground * (xr_image_coord[0] - xl_image_coord[0]));
camera_delta_y[0] = (FocalLen[0] * (yt_world_coord[0] - yb_world_coord[0]))
    / (Camera0_to_Ground * (yt_image_coord[0] - yb_image_coord[0]));

// increment length per dot of camera 1's coordinate systems
camera_delta_x[1] = (FocalLen[1] * (xr_world_coord[1] - xl_world_coord[1]))
    / (Camera1_to_Ground * (xr_image_coord[1] - xl_image_coord[1]));
camera_delta_y[1] = (FocalLen[1] * (yt_world_coord[1] - yb_world_coord[1]))
    / (Camera1_to_Ground * (yt_image_coord[1] - yb_image_coord[1]));

/*+++++*/
/* END of: stereo system's parameters */
/*+++++*/

A1[0] = 135.0;
D1[0] = Camera0_to_Ground - 2.5;
A2[0] = 305.0;
D2[0] = Camera0_to_Ground - 102.5;
A3[0] = 1332.0;
D3[0] = Camera0_to_Ground - 202.5;
MaxArea[0] = A3[0] + A1[0];

A1[1] = 155.0;
D1[1] = Camera1_to_Ground - 2.5;
A2[1] = 380.0;
D2[1] = Camera1_to_Ground - 102.5;
A3[1] = 1512.0;
D3[1] = Camera1_to_Ground - 202.5;
MaxArea[1] = A3[1] + A1[1];

CtoG[0] = Camera0_to_Ground;
CtoG[1] = Camera1_to_Ground;
}

void cal_base_gain(void)
{ int i;

  for (i=0; i<CamNum; ++i)
  {
    x_gain_base[i] = (xr_world_coord[i] - xl_world_coord[i]) /
        (xr_image_coord[i] - xl_image_coord[i]);
    y_gain_base[i] = -(yt_world_coord[i] - yb_world_coord[i]) /
        (yt_image_coord[i] - yb_image_coord[i]);
  }
}

word track_orientation(int angle, byte camera)
{
  static int theta1[CamNum], theta2[CamNum];
  static int step[CamNum], direction[CamNum];

```



```

static long theta[CamNum];

// calculate the orientation (0 - 360 degrees) of object, based
// on the angle derived from central moments of the object.
// Algorithm is created by: Sy-Hung Kuo      05/22/1993

if (FirstCheck[camera] == First_YES) {
    FirstCheck[camera] = First_NO;
    theta1[camera] = angle;
    if (theta1[camera] < 0) theta[camera] = 18000 + theta1[camera];
    else theta[camera] = theta1[camera];
    return (theta[camera]);
}
else {
    theta2[camera] = angle;

    // determine the rotating direction
    if (theta1[camera] >= 0)
        if (theta2[camera] >= 0)
            if ((theta1[camera]-theta2[camera]) < 0)
                direction[camera] = CounterClockwise;
            else
                direction[camera] = Clockwise;
        else
            if ((theta1[camera] >= 4500) && (theta1[camera] <= 9000))
                direction[camera] = CounterClockwise;
            else
                direction[camera] = Clockwise;
    else
        if (theta2[camera] < 0)
            if ((theta1[camera]-theta2[camera]) < 0)
                direction[camera] = CounterClockwise;
            else
                direction[camera] = Clockwise;
        else
            if ((theta1[camera] >= -9000) && (theta1[camera] <= -4500))
                direction[camera] = Clockwise;
            else
                direction[camera] = CounterClockwise;

    if (direction[camera] == CounterClockwise)
        step[camera] = theta2[camera] - theta1[camera];
    else
        step[camera] = theta1[camera] - theta2[camera];
    if (step[camera] < 0) step[camera] += 18000;
    if (direction[camera] == CounterClockwise)
        theta[camera] += step[camera];
    else
        theta[camera] -= step[camera];
    if (theta[camera] >= 36000) theta[camera] -= 36000;
    if (theta[camera] < 0) theta[camera] += 36000;

    theta1[camera] = theta2[camera];
    return ((word) theta[camera]);
}
}

```

```

void init_UART_kuo(word COMp)
{
    COMx = COMp;
    outportb(COMx+3, 0x80);
    outportb(COMx+1, 0x00); /* MSB:----> set to 9600 baud rate */
    outportb(COMx+0, 0x0C); /* LSB:--^ (LSB = 0x06) set to 19200 baud */
    /* set the line control reg to mode : 8 data bits , 1 stop bit , no parity */
    outportb(COMx+3, 0x03);
    outportb(COMx+1, 0x00); /* disable all 4 classes of interupt */
}

void out_char(byte c)
{
    while((inportb(COMx+5) & 0x60) != 0x60);
    outportb(COMx, c);
}

void show_picture(int x, int y)
{
    unsigned long runner;
    int i, j;

    runner = 0;
    fgrabber(OFF);
    for (j=0; j<YMAX; ++j)
        for (i=0; i<XMAX; ++i)
            {
                if (runner > SegLimit) outportb(SegSwitch, ON); else outportb(SegSwitch,
                    OFF);
                putpixel(x+i, y+j, *(XPixel0 + runner++));
            }
    fgrabber(ON);
}

void show_one_camera_text(void)
{
    if (CamID == 0x00)
        {
            SandTextxy2(Monitor_x0a+Mon_OffsetX, Monitor_y0a-Mon_OffsetY, "Camera 0");
            SandTextxy4(Monitor_x1a+Mon_OffsetX, Monitor_y0a-Mon_OffsetY, "Camera 1");
        }
    else
        {
            SandTextxy2(Monitor_x1a+Mon_OffsetX, Monitor_y0a-Mon_OffsetY, "Camera 1");
            SandTextxy4(Monitor_x0a+Mon_OffsetX, Monitor_y0a-Mon_OffsetY, "Camera 0");
        }
}

void show_two_camera_text(void)
{

```

```
SandTextxy2(Monitor_x0a+Mon_OffsetX, Monitor_y0a-Mon_OffsetY, "Camera 0");  
SandTextxy2(Monitor_x1a+Mon_OffsetX, Monitor_y0a-Mon_OffsetY, "Camera 1");  
}
```

```
void warning_sound(void)  
{  
  sound(800); delay(20); nosound();  
}
```

```

/*****
/* Computer Vision System: */
/* */
/* by: Sy-Hung Kuo */
/*-----*/
/* Filename: KUO_8514.C */
/*-----*/
/* This file contains a set of 3D graphics subroutines. */
/* */
/*.....*/
/* */
/* Last Modified : April 1, 1994 */
/*****

```

```

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include <mem.h>
#include <dos.h>
#include <io.h>
#include <bios.h>
#include <graphics.h>

```

```

#include "monitor.h"
#include "bioskey.h"
#include "kuo_def.h"

```

```

#ifdef MAXCOLORS
  #undef MAXCOLORS
  #define MAXCOLORS 255
#endif

```

```

#define VGA_WIDTH 1024
#define VGA_HEIGHT 768
#define xOffset 60
#define yOffset 50

```

```

#define PIC_Column 320
#define StartRow 10
#define EndRow 214
#define PIC_Row (EndRow-StartRow)
#define PIC_Length PIC_Column*PIC_Row
#define ImageHeader 4
#define Bound_x1 0
#define Bound_y1 0
#define Bound_x2 Bound_x1+PIC_Column-1
#define Bound_y2 Bound_y1+PIC_Row-1
#define Window_X 432
#define Window_Y 0
#define Line_Y 324
#define INFO_Y 269

```

```

#define INFO_Color      220
#define ERASE_Color    99
#define GrayLevel      256

enum ReturnCode {ReadSuccess, WriteSuccess, ReadFail, WriteFail, MemNotEnough};
enum RGB_ID {RGB_RED, RGB_GREEN, RGB_BLUE, RGB_SIZE};

typedef struct
{
    char id[2];
    long filesize;
    int reserved[2];
    long headersize;
    long infoSize;
    long width;
    long depth;
    int bitPlanes;
    int bits;
    long fileCompression;
    long byteSizeImage;
    long XPixelsPerMeter;
    long YPixelsPerMeter;
    long colorUsed;
    long colorImportant;
} BmpHeader;

byte Palette256[256][3];

/* Function Prototypes */
int writeBMP(byte *palette, int X0, int Y0, int XGA_WIDTH, int XGA_HEIGHT);
void setDefaultPalette(byte palette[256][3]);
void setup_3D_screen(void);
void load_stroked_fonts(void);
void draw_3D_screen(void);
void draw_monitors_frame(void);
void draw_frame(int x0, int y0, int x1, int y1);
void load_PIC_file(char *filename, FILE *fp, byte *buffer);
void show_PIC_file(byte *buffer);
int huge DetectVGA256(void);
void setvgapalette(byte *PalBuf);
void checkerrors(void);
void set_default_font(int size);
void set_triplex_font(int size);
void set_sans_serif_font(int size);
void set_small_font(int size);
void SandTextxy(int x, int y, char *str);
void SandTextxy2(int x, int y, char *str);
void SandTextxy3(int x, int y, char *str);
void Erase_SandTextxy(int x, int y, char *str);
void SandLine(int x1, int y1, int x2, int y2);
void SandLine2(int x1, int y1, int x2, int y2);
void SandDashedLine(int x1, int y1, int x2, int y2);
void ExpandTextxy(int x, int y, char *str);
int input_an_integer(int x, int y, char *str);

```

```

/* write a Windows' BMP file */
int writeBMP(byte *palette, int X0, int Y0, int XGA_WIDTH, int XGA_HEIGHT)
{
    struct viewporttype vp;
    static BmpHeader BMP;
    char filename[15];
    FILE *fp;
    int k, linelen, x, y;

    linelen=XGA_WIDTH;

    if (linelen & 0x0003)
    {
        linelen |= 0x0003;
        ++linelen;
    }

    /* write the header */
    memset((char *)&BMP, 0, sizeof(BmpHeader));
    memcpy(BMP.id, "BM", 2);
    BMP.headersize=1078L;
    BMP.filesize=BMP.headersize+(long)linelen*(long)XGA_HEIGHT;
    BMP.width=(long)XGA_WIDTH;
    BMP.depth=(long)XGA_HEIGHT;
    BMP.infoSize=0x28L;
    BMP.bits=8;
    BMP.bitPlanes=1;
    BMP.fileCompression=0L;

    /* figure out an available filename */
    for (k=0; k<=999; ++k)
    {
        sprintf(filename, "IMAGE%d.BMP", k);
        fp=fopen(filename, "r+");
        if (fp == NULL) break;
        else fclose(fp);
    }
    fp=fopen(filename, "w+b");
    fwrite((char *)&BMP, 1, sizeof(BmpHeader), fp);

    /* write the palette */
    for (k=0; k<256; k++)
    {
        fputc(palette[k*RGB_SIZE+RGB_BLUE], fp);
        fputc(palette[k*RGB_SIZE+RGB_GREEN], fp);
        fputc(palette[k*RGB_SIZE+RGB_RED], fp);
        fputc(0, fp);
    }

    /* write the bitmap */
    getviewsettings(&vp);
    setviewport(0, 0, getmaxx(), getmaxy(), 0);
    for (y=(Y0+XGA_HEIGHT-1); y>=Y0; y--)
        for (x=X0; x<(X0+XGA_WIDTH); x++)

```

```

        fputc((byte)getpixel(x, y), fp);

fclose(fp);
setviewport(vp.left, vp.top, vp.right, vp.bottom, vp.clip);

if (ferror(fp)) return(WriteFail);
else return(WriteSuccess);
}

```

```

void setDefaultPalette(byte palette[256][3])
{ int c;

  // create Default VGA Palette
  for (c=0; c<256; c++)
    palette[c][0] = palette[c][1] = palette[c][2] = 0;

  // BLACK,
  palette[BLACK][0] = 0;
  palette[BLACK][1] = 0;
  palette[BLACK][2] = 0;

  // BLUE,
  palette[BLUE][0] = 0;
  palette[BLUE][1] = 0;
  palette[BLUE][2] = 255;

  // GREEN,
  palette[GREEN][0] = 0;
  palette[GREEN][1] = 255;
  palette[GREEN][2] = 0;

  // CYAN,
  palette[CYAN][0] = 0;
  palette[CYAN][1] = 255;
  palette[CYAN][2] = 255;

  // RED,
  palette[RED][0] = 255;
  palette[RED][1] = 0;
  palette[RED][2] = 0;

  // MAGENTA,
  palette[MAGENTA][0] = 255;
  palette[MAGENTA][1] = 0;
  palette[MAGENTA][2] = 255;

  // BROWN,
  palette[BROWN][0] = 128;
  palette[BROWN][1] = 64;
  palette[BROWN][2] = 0;

  // LIGHTGRAY,
  palette[LIGHTGRAY][0] = 192;
  palette[LIGHTGRAY][1] = 192;
  palette[LIGHTGRAY][2] = 192;
}

```

```

// DARKGRAY,
palette[DARKGRAY][0] = 128;
palette[DARKGRAY][1] = 128;
palette[DARKGRAY][2] = 128;

// LIGHTBLUE,
palette[LIGHTBLUE][0] = 0;
palette[LIGHTBLUE][1] = 128;
palette[LIGHTBLUE][2] = 255;

// LIGHTGREEN,
palette[LIGHTGREEN][0] = 0;
palette[LIGHTGREEN][1] = 255;
palette[LIGHTGREEN][2] = 128;

// LIGHTCYAN,
palette[LIGHTCYAN][0] = 128;
palette[LIGHTCYAN][1] = 255;
palette[LIGHTCYAN][2] = 255;

// LIGHTRED,
palette[LIGHTRED][0] = 255;
palette[LIGHTRED][1] = 64;
palette[LIGHTRED][2] = 64;

// LIGHTMAGENTA,
palette[LIGHTMAGENTA][0] = 255;
palette[LIGHTMAGENTA][1] = 128;
palette[LIGHTMAGENTA][2] = 255;

// YELLOW,
palette[YELLOW][0] = 255;
palette[YELLOW][1] = 255;
palette[YELLOW][2] = 0;

// WHITE
palette[WHITE][0] = 255;
palette[WHITE][1] = 255;
palette[WHITE][2] = 255;
}

void setup_3D_screen(void)
{
    int c, INPUT;
    char outstr[30];
    int Gdriver = IBM8514;
    int Gmode = IBM8514HI;

    // initialize video adapter
    registerbgidriver(IBM8514_driver);
    checkerrors();
    load_stroked_fonts();
    initgraph(&Gdriver, &Gmode, "");
    checkerrors();
}

```



```

// setup 256 gray levels
for (c=0; c<=MAXCOLORS; c++)
{
    Palette256[c][0] = Palette256[c][1] = Palette256[c][2] = c;
    setrgbpalette(c, c, c, c);
}
draw_3D_screen();
}

void load_stroked_fonts(void)
{
    registerbgifont(triplex_font);
    checkerrors();
    registerbgifont(sansserif_font);
    checkerrors();
    registerbgifont(small_font);
    checkerrors();
}

void draw_3D_screen(void)
{ int i;

    setcolor(255);
    setwritemode(COPY_PUT);
    // draw 3D windows
    setfillstyle(SOLID_FILL, 180);
    bar(0, 0, VGA_WIDTH-1, VGA_HEIGHT-1);
    setfillstyle(SOLID_FILL, ERASE_Color);
    bar(xOffset, yOffset, VGA_WIDTH-xOffset, VGA_HEIGHT-yOffset);
    for (i=0; i<6; i++)
    {
        setcolor(245);
        line(i, i, VGA_WIDTH-1-i, i);
        line(xOffset-i, VGA_HEIGHT-yOffset+i, VGA_WIDTH-xOffset+i, VGA_HEIGHT-
            yOffset+i);
        setcolor(210);
        line(i, i, i, VGA_HEIGHT-1-i);
        line(VGA_WIDTH-xOffset+i, VGA_HEIGHT-yOffset+i, VGA_WIDTH-xOffset+i,
            yOffset-i);
        setcolor(100);
        line(i, VGA_HEIGHT-1-i, VGA_WIDTH-1-i, VGA_HEIGHT-1-i);
        setcolor(90); // test
        line(xOffset-i, yOffset-i, VGA_WIDTH-xOffset+i, yOffset-i);
        setcolor(135);
        line(VGA_WIDTH-1-i, VGA_HEIGHT-1-i, VGA_WIDTH-1-i, i);
        line(xOffset-i, yOffset-i, xOffset-i, VGA_HEIGHT-yOffset+i);
    }
    set_triplex_font(4);
    SandTextxy3(407, 3, "STEREOSCOPE");
    SandTextxy(369, 719, "by: Sy-Hung Kuo");
    setviewport(80, 60, VGA_WIDTH-80, VGA_HEIGHT-60, 1);
    set_default_font(1);
}

```

```

void draw_monitors_frame(void)
{
    /* Left hand Monitor for Camera 0x00 */
    set_triplex_font(4);
    SandTextxy2(Monitor_x0a+Mon_OffsetX, Monitor_y0a-Mon_OffsetY, "Camera 0");
    draw_frame(Monitor_x0a, Monitor_y0a, Monitor_x0b, Monitor_y0b);

    /* Right hand Monitor for Camera 0x01 */
    set_triplex_font(4);
    SandTextxy2(Monitor_x1a+Mon_OffsetX, Monitor_y0a-Mon_OffsetY, "Camera 1");
    draw_frame(Monitor_x1a, Monitor_y0a, Monitor_x1b, Monitor_y0b);

    set_default_font(1);
}

```

```

void draw_frame(int x0, int y0, int x1, int y1)
{ int i;

```

```

    x0--; x1++; y0--; y1++;
    for (i=4; i<=5; ++i)
    {
        setcolor(220);
        line(x0-i, y0-i, x1+i, y0-i);
        setcolor(200);
        line(x0-i, y0-i, x0-i, y1+i);
        setcolor(140);
        line(x0-i, y1+i, x1+i, y1+i);
        setcolor(140);
        line(x1+i, y0-i, x1+i, y1+i);
    }
    i=3;
    setcolor(170);
    line(x0-i, y0-i, x1+i, y0-i);
    line(x0-i, y0-i, x0-i, y1+i);
    line(x0-i, y1+i, x1+i, y1+i);
    line(x1+i, y0-i, x1+i, y1+i);
    for (i=1; i<=2; ++i)
    {
        setcolor(140);
        line(x0-i, y0-i, x1+i, y0-i);
        setcolor(140);
        line(x0-i, y0-i, x0-i, y1+i);
        setcolor(210);
        line(x0-i, y1+i, x1+i, y1+i);
        setcolor(190);
        line(x1+i, y0-i, x1+i, y1+i);
    }
}

```

```

void load_PIC_file(char *filename, FILE *fp, byte *buffer)
{ char outstr[30];

```

```

set_sans_serif_font(3);
setcolor(255);
sprintf(outstr, "%s is loading ...", filename);
outtextxy(Bound_x1+10, Bound_y1+150, outstr);

setcolor(INFO_Color);
set_sans_serif_font(4);
SandTextxy(Bound_x1+55, Bound_y1+INFO_Y, "Original Image");
set_default_font(1);
SandLine(0, Line_Y, VGA_WIDTH-1, Line_Y);

fseek(fp, (long)(StartRow*PIC_Column), SEEK_SET);
fread(buffer, (long)PIC_Length, sizeof(byte), fp);
}

void show_PIC_file(byte *buffer)
{ int x, y;

  /* show picture */
  for (y=0; y<PIC_Row; y++)
    for (x=0; x<PIC_Column; x++)
      putpixel(Bound_x1+x, Bound_y1+y, buffer[y*PIC_Column+x]);
}

/* detects SUPER VGA cards */
int huge DetectVGA256(void)
{
  // Returns Value: -> SuperVGA Mode:
  //      0          -> 320x200x256
  //      1          -> 640x400x256
  //      2          -> 640x480x256
  //      3          -> 800x600x256
  //      4          -> 1024x768x256
  return 4;
}

/* Setvgapalette sets the entire 256 color palette */
/* PalBuf contains RGB values for all 256 colors */
/* R,G,B values range from 0 to 63 */
void setvgapalette(byte *PalBuf)
{
  union REGS r;
  int i;

  /* set palette of SVGA adapter */
  outp(0x3C6, 0xFF);
  for (i=0; i<256; i++)
  {
    outp(0x3C8, i);
    outp(0x3C9, (*PalBuf++) >> 2);
    outp(0x3C9, (*PalBuf++) >> 2);
    outp(0x3C9, (*PalBuf++) >> 2);
  }
}

```

```

    r.x.ax = 0x1001;
    r.h.bh = 0x00;
    int86(0x10, &r, &r);
}

/* check for and report any graphics errors */
void checkerrors(void)
{
    int errorcode;

    /* read result of last graphics operation */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to exit:");
        getch();
        exit(1); /* terminate with an error code */
    }
}

void set_default_font(int size)
{
    setttextstyle(DEFAULT_FONT, HORIZ_DIR, size);
}

void set_triplex_font(int size)
{
    setttextstyle(TRIPLEX_FONT, HORIZ_DIR, size);
}

void set_sans_serif_font(int size)
{
    setttextstyle(SANS_SERIF_FONT, HORIZ_DIR, size);
}

void set_small_font(int size)
{
    setttextstyle(SMALL_FONT, HORIZ_DIR, size);
}

void SandTextxy(int x, int y, char *str)
{
    setcolor(255);
    outtextxy(x+2, y+2, str);
    setcolor(0);
    outtextxy(x, y, str);
    setcolor(150);
    outtextxy(x+1, y+1, str);
}

```

```

void SandTextxy2(int x, int y, char *str)
{
    setcolor(0);
    ExpandTextxy(x+2, y+2, str);
    setcolor(255);
    ExpandTextxy(x, y, str);
    setcolor(150);
    ExpandTextxy(x+1, y+1, str);
}

void SandTextxy3(int x, int y, char *str)
{
    setcolor(255);
    ExpandTextxy(x+2, y+2, str);
    setcolor(150);
    ExpandTextxy(x, y, str);
    setcolor(0);
    ExpandTextxy(x+1, y+1, str);
}

void SandTextxy4(int x, int y, char *str)
{
    setcolor(150);
    ExpandTextxy(x+2, y+2, str);
    setcolor(0);
    ExpandTextxy(x, y, str);
    setcolor(70);
    ExpandTextxy(x+1, y+1, str);
}

void Erase_SandTextxy(int x, int y, char *str)
{
    setcolor(ERASE_Color);
    ExpandTextxy(x+2, y+2, str);
    setcolor(ERASE_Color);
    ExpandTextxy(x, y, str);
    setcolor(ERASE_Color);
    ExpandTextxy(x+1, y+1, str);
}

void SandLine(int x1, int y1, int x2, int y2)
{
    setcolor(0);
    line(x1+2, y1+2, x2+2, y2+2);
    setcolor(255);
    line(x1, y1, x2, y2);
    setcolor(160);
    line(x1+1, y1+1, x2+1, y2+1);
}

```

```

void SandLine2(int x1, int y1, int x2, int y2)
{
    setcolor(0);
    line(x1+1, y1+1, x2+1, y2+1);
    setcolor(255);
    line(x1-1, y1-1, x2-1, y2-1);
    setcolor(160);
    line(x1, y1, x2, y2);
}

void SandDashedLine(int x1, int y1, int x2, int y2)
{
    setlinestyle(DASHED_LINE, 0, 1);
    setcolor(0);
    line(x1+1, y1+1, x2+1, y2+1);
    setcolor(255);
    line(x1-1, y1-1, x2-1, y2-1);
    setcolor(160);
    line(x1, y1, x2, y2);
    setlinestyle(SOLID_LINE, 0, 1);
}

void ExpandTextxy(int x, int y, char *str)
{
    outtextxy(x, y, str);
    outtextxy(x+1, y, str);
    outtextxy(x, y+1, str);
}

#define text_Width      6
#define text_Height    9

int input_an_integer(int x, int y, char *str)
{
    struct textsettingstype textinfo;
    struct fillsettingstype fillinfo;
    char input, outstr[2]; int i, color;

    gettextsettings(&textinfo);
    getfillsettings(&fillinfo);
    color = getcolor();
    setwritemode(XOR_PUT);
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
    setfillstyle(SOLID_FILL, 255);
    setcolor(255);

    for (;;)
    {
        while (1) // draw cursor
        {
            bar(x, y, x+text_Width, y+text_Height);
            for (i=0; i<100; ++i) delay(1);

```

```

    bar(x, y, x+text_Width, y+text_Height);
    for (i=0; i<60; ++i) delay(1);
    if (kbhit()) break;
}
input = getch();
if ((input >= 0x30) && (input <= 0x39))
{
    *str++ = input;
    sprintf(outstr, "%c", input);
    outtextxy(x, y, outstr);
    x += text_Width;
    y += text_Height;
}
if (input == 0x0D) break;
if (input == 0x08)
{
    setcolor(ERASE_Color);
    sprintf(outstr, "%c", *--str);
    x -= text_Width;
    y -= text_Height;
    outtextxy(x, y, outstr);
}
}
*str = '\0';

// restore original setting
setwritemode(COPY_PUT);
settextstyle(textinfo.font, textinfo.direction, textinfo.charsize);
setfillstyle(fillinfo.pattern, fillinfo.color);
setcolor(color);
return(atoi(str));
}

```

```

/*****
/* Computer Vision System: */
/* */
/* by: Sy-Hung Kuo */
/*-----*/
/* Filename: KUO_LOOP.C */
/*-----*/
/* This file contains the user interface. */
/* */
/*.....*/
/* */
/* Last Modified : April 1, 1994 */
/*****

```

```

#include <stdio.h>
#include <io.h>
#include <alloc.h>
#include <fcntl.h>
#include <dos.h>
#include <math.h>
#include <string.h>
#include <process.h>

```

```

#include "bioskey.h"
#include "monitor.h"
#include "kuo_def.h"

```

```

#define CUR_HGAIN 10
#define CUR_LGAIN 1

```

```

extern byte Palette256[256][3];

```

```

extern struct region
{
    int xc, yc, ax, by, angle, thres;
    float fxc, fyc;
    word area;
};

```

```

extern struct region *WindowPTR[CamNum], Window[CamNum];
extern struct region *objectPTR[CamNum], object[CamNum];
extern word his_bin[];
extern byte far *XPixel, far *XPixel0, far *ImageBuffer;
extern byte dig_buf[], dis_buf[];
extern byte CamID;

```

```

/* Function Prototypes */
void central_Moment_Loop(void);
void get_target_info(byte camera);
void accept_window(byte camera);

```



```

void central_Moment_Loop(void)
{
    int i, MonX, INPUT;
    byte RegID;

    for (i=0; i<CamNum; ++i)
        objectPTR[i] = (struct region *) &object[i];

    for (;;) {
        if ((INPUT = bioskey(0)) == KEY_ESC) break;
        switch(INPUT) {
            case KEY_s : accept_window(CamID); one_camera_tracking_float(); break;
            case KEY_S : accept_window(CamID); one_camera_tracking_int(); break;
            case KEY_d : accept_window(CamID);
                if (CamID) RegID = 0x00;
                else RegID = 0x01;
                next_picture(RegID); accept_window(RegID);
                two_camera_tracking_float(); break;
            case KEY_D : accept_window(CamID);
                if (CamID) RegID = 0x00;
                else RegID = 0x01;
                next_picture(RegID); accept_window(RegID);
                two_camera_tracking_float2(); break;
            case KEY_r : // refresh windows
            case KEY_R : next_picture(0x01); show_picture(Monitor_x1a,Monitor_y0a);
                next_picture(0x00); show_picture(Monitor_x0a,Monitor_y0a);
                next_picture(CamID); break;
            case KEY_e : // equalize windows
            case KEY_E : next_picture(0x00); equalize_picture(Monitor_x0a,
                Monitor_y0a, 0x00, 0);
                next_picture(0x01); equalize_picture(Monitor_x1a,
                Monitor_y0a, 0x01, 0);
                next_picture(CamID); break;
            case KEY_u : undo_window(CamID); break;
            case KEY_c :
            case KEY_C : centroid_float((struct region *)&Window[CamID], 1, CamID);
                break;
            case KEY_t : threshold((struct region *)&Window[CamID], OFF); break;
            case KEY_T : threshold((struct region *)&Window[CamID], ON); break;
            case KEY_h : if (CamID == 0x00) MonX = Monitor_x0a;
                else MonX = Monitor_x1a;
                equalize_picture(MonX, Monitor_y0a, CamID, 0);
                draw_window_histogram((struct region *)&Window[CamID],
                CamID, 1, 1);
                break;
            case KEY_H : if (CamID == 0x00) MonX = Monitor_x0a;
                else MonX = Monitor_x1a;
                equalize_picture(MonX, Monitor_y0a, CamID, 0);
                draw_window_histogram((struct region *)&Window[CamID],
                CamID, 1, 0);
                break;
            case KEY_i : invert_video(); break;
        }
    }
}

```

```

    case KEY_ENTER      : accept_window(CamID); break;
    case KEY_SPACE      : next_picture(CamID); break; /* digitize picture */
    case KEY_0          : CamID=0x00; next_picture(CamID); break;
    case KEY_1          : CamID=0x01; next_picture(CamID); break;
    case KEY_NumUP      : window_left(CUR_HGAIN, CamID); break;
    case KEY_NumDOWN    : window_right(CUR_HGAIN, CamID); break;
    case KEY_NumLEFT    : window_down(CUR_HGAIN, CamID); break;
    case KEY_NumRIGHT   : window_up(CUR_HGAIN, CamID); break;
    case KEY_UP         : window_left(CUR_LGAIN, CamID); break;
    case KEY_DOWN       : window_right(CUR_LGAIN, CamID); break;
    case KEY_LEFT       : window_down(CUR_LGAIN, CamID); break;
    case KEY_RIGHT      : window_up(CUR_LGAIN, CamID); break;
    case Alt_C          : continuous_sample(); break;
    case Alt_T          : calibration_test(CamID); break;
    case Alt_S          : writeBMP((byte *)Palette256, 0, 0, VGA_WIDTH,
                                VGA_HEIGHT); break;
    case SYM_GREATER    : window_enlarge(5, CamID); break;
    case SYM_LESS       : window_shrink(5, CamID); break;
    case SYM_COMMA      : window_enlarge(1, CamID); break;
    case SYM_PERIOD     : window_shrink(1, CamID); break;
    case SYM_QUESTION   :
    case SYM_SLASH      : accept_window(CamID); get_target_info(CamID);
}
}
}
}

```

```
void get_target_info(byte camera)
```

```

{
    char c; int i;
    static char outstr[80] = "";

    setcolor(ERASE_Color);
    outtextxy(70, 600, outstr);

    objectPTR[camera] = (struct region *)&object[camera];

    sprintf(outstr, "Camera %d: xc=%d yc=%d, ax=%d, by=%d, angle=%d",
            camera, objectPTR[camera]->xc, objectPTR[camera]->yc,
            objectPTR[camera]->ax, objectPTR[camera]->by,
            objectPTR[camera]->angle);
    setcolor(255);
    outtextxy(70, 600, outstr);
}

```

```
void accept_window(byte camera)
```

```

{
    WindowPTR[camera] = (struct region *)&Window[camera];
}

```

```

/*****
/*  Computer Vision System:                               */
/*                                                         */
/*                                     by: Sy-Hung Kuo      */
/*-----*/
/*  Filename: KUO_WINS.C                                  */
/*-----*/
/*  This file contains:                                   */
/*                                                         */
/*  +  the subroutines to manually move and adjust windows. */
/*                                                         */
/*.....*/
/*                                                         */
/*  Last Modified :  April 1, 1994                        */
/*-----*/

```

```

#include <stdio.h>
#include <io.h>
#include <alloc.h>
#include <fcntl.h>
#include <dos.h>
#include <math.h>
#include <string.h>
#include <graphics.h>
#include <time.h>

```

```

#include "kuo_def.h"

```

```

extern struct region
{
    int xc, yc, ax, by, angle, thres;
    float fxc, fyc;
    word area;
};

```

```

extern struct region *WindowPTR[CamNum], Window[CamNum];
extern byte far *XPixel, far *XPixel0;
extern byte dig_buf[], dis_buf[];

```

```

/* Function Prototypes */
void window_tilt_right(byte camera);
void window_tilt_left(byte camera);
void window_horizontal(int gain, byte camera);
void window_vertical(int gain, byte camera);
void window_enlarge(int gain, byte camera);
void window_shrink(int gain, byte camera);
void window_up(int gain, byte camera);
void window_down(int gain, byte camera);
void window_left(int gain, byte camera);
void window_right(int gain, byte camera);

```

```

void window_horizontal(int gain, byte camera)
{
    undo_window_region((struct region *)&Window[camera]);
    WindowPTR[camera]->ax += gain;
    window_region((struct region *)&Window[camera]);
}

void window_vertical(int gain, byte camera)
{
    undo_window_region((struct region *)&Window[camera]);
    WindowPTR[camera]->by += gain;
    window_region((struct region *)&Window[camera]);
}

void window_enlarge(int gain, byte camera)
{
    undo_window_region((struct region *)&Window[camera]);
    WindowPTR[camera]->ax += gain;
    WindowPTR[camera]->by += gain;
    window_region((struct region *)&Window[camera]);
}

void window_shrink(int gain, byte camera)
{
    undo_window_region((struct region *)&Window[camera]);
    WindowPTR[camera]->ax -= gain;
    WindowPTR[camera]->by -= gain;
    window_region((struct region *)&Window[camera]);
}

void window_up(int gain, byte camera)
{
    undo_window_region((struct region *)&Window[camera]);
    WindowPTR[camera]->yc -= gain;
    window_region((struct region *)&Window[camera]);
}

void window_down(int gain, byte camera)
{
    undo_window_region((struct region *)&Window[camera]);
    WindowPTR[camera]->yc += gain;
    window_region((struct region *)&Window[camera]);
}

void window_left(int gain, byte camera)
{
    undo_window_region((struct region *)&Window[camera]);
    WindowPTR[camera]->xc -= gain;
    window_region((struct region *)&Window[camera]);
}

```

```
}  
  
void window_right(int gain, byte camera)  
{  
    undo_window_region((struct region *)&Window[camera]);  
    WindowPTR[camera]->xc += gain;  
    window_region((struct region *)&Window[camera]);  
}
```

```

/*****
/* Computer Vision System: */
/* */
/* by: Sy-Hung Kuo */
/*-----*/
/* Filename: KUO_REGN.C */
/*-----*/
/* This file contains: */
/* */
/* + the subroutines to define or undo the windows' region. */
/* */
/*.....*/
/* */
/* Last Modified : April 1, 1994 */
/*****

```

```

#include <stdio.h>
#include <io.h>
#include <alloc.h>
#include <fcntl.h>
#include <dos.h>
#include <math.h>
#include <string.h>

```

```

#include "kuo_def.h"

```

```

extern struct region
{
    int xc, yc, ax, by, angle, thres;
    float fxc, fyc;
    word area;
};

```

```

extern struct region Window[CamNum];
extern byte far *LineBuffer0, far *LineBuffer;

```

```

/* Function Prototypes */
void undo_window(byte camera);
void undo_window_region(struct region *pnt);
void do_thick_cross(struct region *pnt);
void window_region(struct region *pnt);

```

```

void undo_window(byte camera)
{
    undo_window_region((struct region *)&Window[camera]);
}

```

```

void undo_window_region(struct region *pnt)
{
    int b_x, b_y, a_x, a_y;
    int x, y;
    int x1, y1, x2, y2, x3, y3, x4, y4;
    double theta;
    float ctheta, stheta;

    x = pnt->xc;
    y = pnt->yc;
    LineBuffer = LineBuffer0;

    theta = (double)(.1*pnt->angle)*PI180;
    ctheta = (float) cos(theta);
    stheta = (float) sin(theta);

    b_y = (int)((float)(pnt->by) * ctheta);
    b_x = (int)((float)(pnt->by) * stheta);
    a_y = (int)((float)(pnt->ax) * stheta);
    a_x = (int)((float)(pnt->ax) * ctheta);

    x1 = (x - b_x - a_x);
    y1 = (y + b_y - a_y);
    x2 = (x + b_x - a_x);
    y2 = (y - b_y - a_y);
    x3 = (x + b_x + a_x);
    y3 = (y - b_y + a_y);
    x4 = (x - b_x + a_x);
    y4 = (y + b_y + a_y);

    fgrabber(OFF);
    undo_line(x1, y1, x2, y2);
    undo_line(x2, y2, x3, y3);
    undo_line(x3, y3, x4, y4);
    undo_line(x4, y4, x1, y1);
    fgrabber(ON);
}

void do_thick_cross(struct region *pnt) // for video 11-9-91
{
    int b_x, b_y, a_x, a_y;
    int x, y;
    int x1, y1, x2, y2, x3, y3, x4, y4;
    double theta;
    float ctheta, stheta;

    x = pnt->xc;
    y = pnt->yc;
    LineBuffer = LineBuffer0;

    theta = (double)pnt->angle * RADIAN * 0.01;
    ctheta = (float) cos(theta);
    stheta = (float) sin(theta);
}

```

```

    a_x = (int)((float)(pnt->ax) * ctheta);
    a_y = (int)((float)(pnt->ax) * VIDEO_PAR * stheta);
    b_x = (int)((float)(pnt->by) * stheta);
    b_y = (int)((float)(pnt->by) * VIDEO_PAR * ctheta);

    x1 = (x - b_x); /*NW*/
    y1 = (y - b_y);
    x2 = (x + b_x); /*SE*/
    y2 = (y + b_y);
    x3 = (x - a_x); /*SW*/
    y3 = (y + a_y);
    x4 = (x + a_x); /*NE*/
    y4 = (y - a_y);

    fg_line(x1, y1, x2, y2);
    fg_line(x3, y3, x4, y4);
}

```

```

void window_region(struct region *pnt)
{
    int b_x, b_y, a_x, a_y;
    int x, y;
    int x1, y1, x2, y2, x3, y3, x4, y4;
    double theta;
    float ctheta, stheta;

    x = pnt->xc;
    y = pnt->yc;
    LineBuffer = LineBuffer0;

    theta = (double)(.01*pnt->angle)*PI180;
    ctheta = (float) cos(theta);
    stheta = (float) sin(theta);

    b_y = (int)((float)(pnt->by) * ctheta);
    b_x = (int)((float)(pnt->by) * stheta);
    a_y = (int)((float)(pnt->ax) * stheta);
    a_x = (int)((float)(pnt->ax) * ctheta);

    x1 = (x - b_x - a_x);
    y1 = (y + b_y - a_y);
    x2 = (x + b_x - a_x);
    y2 = (y - b_y - a_y);
    x3 = (x + b_x + a_x);
    y3 = (y - b_y + a_y);
    x4 = (x - b_x + a_x);
    y4 = (y + b_y + a_y);

    fgrabber(OFF);
    save_line(x1, y1, x2, y2);
    save_line(x2, y2, x3, y3);
    save_line(x3, y3, x4, y4);
    save_line(x4, y4, x1, y1);
    fg_line(x1, y1, x2, y2);
    fg_line(x2, y2, x3, y3);
}

```



```
fg_line(x3, y3, x4, y4);  
fg_line(x4, y4, x1, y1);  
fgrabber(ON);  
}
```

```

/*****
/* Computer Vision System: */
/* */
/* by: Sy-Hung Kuo */
/*-----*/
/* Filename: KUO_FGRA.C */
/*-----*/
/* This file contains: */
/* */
/* + the subroutines to manipulate and initialize the frame grabber. */
/* */
/* + the subroutines to support up to six cameras. */
/* */
/*.....*/
/* */
/* Last Modified : April 1, 1994 */
/*****

```

```

#include <stdio.h>
#include <io.h>
#include <alloc.h>
#include <fcntl.h>
#include <dos.h>
#include <sys\stat.h>

```

```

#include "kuo_def.h"

```

```

extern unsigned long Image_Size;
extern word XMAX, YMAX;
extern byte far *XPixel, far *XPixel0, far *ImageBuffer;
extern byte dig_buf[], dis_buf[];

```

```

/* Function Prototypes */
void load_xcode_to_buf(int xsamp);
void continuous_sample();
void sample_picture(byte camera);
void init_fgrabber();
void load_xsam_dig();
void load_xsam_dis();
void invert_video();
void add_image(char *filename);
void frestore_data(char *filename);
void save_pic(char *filename);
void clr_fgrabber();
void fgrabber(char TurnON);

```

```

/* load xsam digitize & display codes into buffers */
void load_xcode_to_buf(int xsamp)
{
    char dig_filename[16], dis_filename[16];

```

```

word i;
int intfile;

if (xsamp == 0)
{
    strcpy(dig_filename, "dig_code.dat");
    strcpy(dis_filename, "dis_code.dat");
}

else if (xsamp == 1)
{
    strcpy(dig_filename, "digcode1.dat");
    strcpy(dis_filename, "discode1.dat");
}

else if (xsamp == 2)
{
    strcpy(dig_filename, "digcode2.dat");
    strcpy(dis_filename, "discode2.dat");
}

if ((intfile = open(dig_filename, O_BINARY, S_IREAD)) == -1)
{
    printf("\nFile \"%s\" not found\n", dis_filename);
    return;
}
read(intfile, dig_buf, XSAMSIZE); close (intfile);

if ((intfile = open(dis_filename, O_BINARY, S_IREAD)) == -1)
{
    printf("\nFile \"%s\" not found\n", dis_filename);
    return;
}
read(intfile, dis_buf, XSAMSIZE); close (intfile);
}

void continuous_sample() /* continuously sample picture */
{
    // program optimized by: Sy-Hung Kuo

    outportb(0x301, 0x21); /*disable video*/
    memcpy(XPixel0, dig_buf, XSAMSIZE);
    outportb(0x301, 0xA9); /* enable video */
}

#define _SVM_MEM      0xD0000L    /* Frame Grabber memory address */
#define _SVM_REG      0x300      /* Register base address */
#define _SVM_IRQ      3         /* Interrupt Request Number */
#define SVMLINC       0x0A      /* Active lines / 2 */
#define SVMLINT       0x0B      /* Top active video line / 2 */
#define VIDEOMUX      0x09      /* Video In MUX select register */

```

```

void sample_picture(byte camera) /* sample picture */
{
    // add Video MUX selection by: Sy-Hung Kuo    6/19/93
    outportb(_SVM_REG+VIDEOMUX, camera);

    /* blank & disable video*/
    outportb(0x301, 0x21);
    load_xsam_dig();

    /* wait for vertical sync */
    while ( (inportb(0x300) & 0x02) );
    while (!(inportb(0x300) & 0x02) );

    /* set up image memory address */
    outportb(0x30f, 0x00);
    outportb(0x30e, 0x00);
    outportb(0x30d, 0x00);
    outportb(0x308, 0x00);

    /* enable video */
    outportb(0x301, 0xA1);

    while ( (inportb(0x300) & 0x02) ); /* waiting for vertical dr */
    while (!(inportb(0x300) & 0x02) ); /* waiting for vertical dr */
    outportb(0x301, 0x21);           /* disable video */
    load_xsam_dis();                 /* display data on the video monitor */

    while ( (inportb(0x300) & 0x02) ); /* waiting for vertical dr */
    while (!(inportb(0x300) & 0x02) ); /* necessary for keeping the image clean */
    outportb(0x301, 0xA8);           /* otherwise turn it on */
    // program optimized by: Sy-Hung Kuo
}

/*
 * Initialize frame grabber
 * by An H. Nguyen
 * Date: 8-4-92
 * Comment: It took more than six years to figure out how to get
 * the frame grabber initialized!!!
 */
void init_fgrabber()
{
    /*
     * Load the line counters with values appropriate
     * for 240 lines in RS-170. Computation of these
     * values are described in the hardware documentation.
     * Note: The values are dependent upon the frame grabber model!
     */
    outportb(_SVM_REG+SVMLINT, 0xF4 /*0xF4*/ );
    outportb(_SVM_REG+SVMLINC, 0x87 /*0x87*/ ); //must be 0xFF for full screen
    // add Video MUX selection by: Sy-Hung Kuo    6/19/93
    outportb(_SVM_REG+VIDEOMUX, 0x00);
}

```

```

void load_xsam_dig() /* load digitize codes into xsam memory */
{
    // program optimized by: Sy-Hung Kuo

    memcpy(XPixel0, dig_buf, XSAMSIZE);
}

void load_xsam_dis() /* load display codes into xsam memory */
{
    // program optimized by: Sy-Hung Kuo

    memcpy(XPixel0, dis_buf, XSAMSIZE);
}

void invert_video()
{
    unsigned long i, j;
    byte data;
    byte far *temp;

    fgrabber(OFF);
    XPixel = XPixel0;
    temp = XPixel0;

    if (Image_Size > 0xFFFF)
    {
        for (i = 0; i < 0xFFFF; i++)
        {
            data = *XPixel++;
            *temp++ = 255 - data;
        }
        XPixel = XPixel0;
        temp = XPixel0;

        outportb(0x308, 0x01);
        for (i = 0; i < Image_Size - 0xFFFF; i++)
        {
            data = *XPixel++;
            *temp++ = 255 - data;
        }
    }
    else
    {
        for (i = 0; i < Image_Size; i++)
        {
            data = *XPixel++;
            *temp++ = 255 - data;
        }
    }

    outportb(0x308, 0x00);
    fgrabber(ON);
}

```

```

void add_image(char *filename)
{
    char data, far *temp;
    long i;
    FILE *ifstream;

    if ((ifstream = fopen(filename, "rb")) == NULL)
    {
        printf("\nFile \"%s\" not found.", filename);
        return;
    }

    XPixel = XPixel0;
    temp = ImageBuffer;

    fgrabber(OFF);

    if (Image_Size > 0xFFFF)
    {
        fread(ImageBuffer, sizeof(char), 0xFFFF, ifstream);
        for (i = 0; i < 0xFFFF; i++)
        {
            data = *temp++;
            if (*XPixel < data) *XPixel++ = data; else XPixel++;
        }

        fread(ImageBuffer, sizeof(char), Image_Size - 0xFFFF, ifstream);
        XPixel = XPixel0;
        temp = ImageBuffer;
        outportb(0x308, 0x01);
        for (i = 0; i < Image_Size - 0xFFFF; i++)
        {
            data = *temp++;
            if (*XPixel < data) *XPixel++ = data; else XPixel++;
        }
    }
    else
    {
        fread(ImageBuffer, sizeof(char), Image_Size, ifstream);
        for (i = 0; i < Image_Size; i++)
        {
            data = *temp++;
            if (*XPixel < data) *XPixel++ = data; else XPixel++;
        }
    }

    outportb(0x308, 0x00);
    fclose(ifstream);
    fgrabber(ON);
}

void frestore_data(char *filename)
{
    FILE *ifstream;

```

```

if ((ifstream = fopen(filename, "rb")) == NULL)
{
    printf("\nFile \"%s\" not found.", filename);
    return;
}

fgrabber(OFF);
if (Image_Size > 0xFFFF)
{
    fread(XPixel0, sizeof(char), 0xFFFF, ifstream);
    fread(XPixel0+0xFFFF, sizeof(char), 1, ifstream);
    outportb(0x308, 0x01);
    fread(XPixel0, sizeof(char), Image_Size-0x10000, ifstream);
}
else fread(XPixel0, sizeof(char), Image_Size, ifstream);
fclose(ifstream);
fgrabber(ON);
}

void save_pic(char *filename)
{
    word temp;
    FILE *ifstream;

    if ( (ifstream = fopen(filename, "wb")) == NULL)
    {
        printf("\nCannot save file \"%s\"", filename);
        return;
    }
    outportb(0x301, 0x28);

    if (Image_Size > 0xFFFF)
    {
        fwrite(XPixel0, sizeof(char), 0xFFFF, ifstream);
        fwrite(XPixel0+0xFFFF, sizeof(char), 1, ifstream);
        outportb(0x308, 0x01);
        fwrite(XPixel0, sizeof(char), Image_Size-0x10000, ifstream);
    }
    else fwrite(XPixel0, sizeof(char), Image_Size, ifstream);

    outportb(0x301, 0xA8);
    fclose(ifstream);
}

void clr_fgrabber()
{
    // program optimized by: Sy-Hung Kuo

    fgrabber(OFF);
    memset(XPixel0, 0x00, YMAX * XMAX);
    fgrabber(ON);
}

```

```
void fgrabber(char TurnON)
{
    /* waiting for vertical dr */
    while ( (inportb(0x300) & 2) );

    /* necessary for keeping the image clean */
    while (!(inportb(0x300) & 2) );

    if (TurnON) outportb(0x301, 0xA8); /* turn video on */
    else        outportb(0x301, 0x28); /* otherwise turn it off */
}
```



```

/*****
/* Computer Vision System: */
/* */
/* by: Sy-Hung Kuo */
/*-----*/
/* Filename: KUO_MOME.C */
/*-----*/
/* This file contains: */
/* + the subroutines to calculate the central moments. */
/* */
/*.....*/
/* Last Modified : April 1, 1994 */
/*****

#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <math.h>
#include <time.h>
#include <alloc.h>

#include "kuo_def.h"

#define AEr_X 125
#define AEr_Y 325

extern struct region
{
    int xc, yc, ax, by, angle, thres;
    float fxc, fyc;
    word area;
};

extern struct region *objectPTR[CamNum];
extern word THRESHOLD;
extern byte far *XPixel, far *XPixel0;
extern word XMAX, YMAX;
extern int MaxArea[CamNum];

/* Function Prototypes */
int an_cent_bi_gray_int(struct region *pnt, struct region *ctr, int lineskip);
int an_cent_bi_gray_float(struct region *pnt, struct region *ctr, int lineskip);
void centroid_float(struct region *pnt, int monflag, byte camera);
void centroid_int(struct region *pnt, int monflag, byte camera);
void Area_Error_Warning(void);

```

```

/*
  Integer Version: (SHK: 04/16/93)

  * Modified an_cent_bi_gray() to work with 320x240 images.
  * Without gray level option.  AHN & SHK: 4-16-93
  * Program optimized by: Sy-Hung Kuo
*/
int an_cent_bi_gray_int(struct region *pnt, struct region *ctr, int lineskip)
{
  static long  M00, M10, M20, M01, M02, M11;
  register word  column, row;
  unsigned long runner;
  static long  cMxx ,cMyy ,cMxy;
  static int  top, bottom, left, right;
  static float xcenter, ycenter, ftemp1, ftemp2, theta;
  static struct region xctr[CamNum];

  cMxx = cMyy = cMxy = 0;

  top    = pnt->yc - pnt->by;
  bottom = pnt->yc + pnt->by;
  left   = pnt->xc - pnt->ax;
  right  = pnt->xc + pnt->ax;

  M00 = M10 = M20 = M01 = M02 = M11 = 0;

  for (row = top+1; row < bottom; row += lineskip)
  {
    runner = ((unsigned long)row * XMAX) + left + 1;

    for (column = left+1; column < right; column++)
    {
      if (runner > SegLimit) outportb(SegSwitch, ON); else outportb(SegSwitch,
        OFF);
      XPixel = XPixel0 + runner++;
      if (*XPixel++ >= THRESHOLD) { // white targets
        M00++; // Area !
        M10 += column;
        M20 += (long)column*column;
        M01 += row;
        M02 += (long)row*row;
        M11 += (long)column*row;
      }
    }
  }

  if (M00 < AREA_MIN) { ctr->area = M00; return (0); }
  else {
    xcenter = ((float) M10 / (float) M00);
    ycenter = ((float) M01 / (float) M00);
    ctr->xc = (int)(xcenter+0.5);
    ctr->yc = (int)(ycenter+0.5);
    ctr->area = M00;
    /* central moments */
    cMxx = M20 - (long)(xcenter*(float)M10);
    cMyy = M02 - (long)(ycenter*(float)M01);
  }
}

```

```

    cMxy = M11 - (long) (ycenter*(float)M10);
    ftemp1 = -((float)cMyy / VIDEO_PAR) + ((float)cMxx * VIDEO_PAR);
    ftemp2 = -(float) (cMxy << 1);
    theta = (float) (atan2((double)ftemp2, (double)ftemp1) * DEGREEdiv2);
    ctr->angle = (int) (theta * 100.0);
    xctr->xc = ctr->xc;
    xctr->yc = ctr->yc;
    xctr->ax = CrossXY;
    xctr->by = CrossXY;
    xctr->angle = ctr->angle;
    do_thick_cross(xctr);
    return(1);
}
}

/*
Floating Point Version: (SHK: 05/15/93)

    * Modified an_cent_bi_gray() to work with 320x240 images.
    * Without gray level option.  AHN & SHK: 4-16-93
    * Program optimized by: Sy-Hung Kuo
*/
int an_cent_bi_gray_float(struct region *pnt, struct region *ctr, int lineskip)
{
    static long  M00, M10, M20, M01, M02, M11;
    register word  column, row;
    unsigned long runner;
    static float cMxx ,cMyy ,cMxy;
    static int top, bottom, left, right;
    static float xcenter, ycenter, ftemp1, ftemp2, theta;
    static struct region xctr[CamNum];

    cMxx = cMyy = cMxy = 0.;

    top    = pnt->yc - pnt->by;
    bottom = pnt->yc + pnt->by;
    left   = pnt->xc - pnt->ax;
    right  = pnt->xc + pnt->ax;

    M00 = M10 = M20 = M01 = M02 = M11 = 0;

    for (row = top+1; row < bottom; row += lineskip)
    {
        runner = ((unsigned long)row * XMAX) + left + 1;

        for (column = left+1; column < right; column++)
        {
            if (runner > SegLimit) outportb(SegSwitch, ON); else outportb(SegSwitch,
                OFF);
            XPixel = XPixel0 + runner++;
            if (*XPixel >= THRESHOLD) { // white targets
                M00++; // Area !
                M10 += column;
                M20 += (long)column*column;
                M01 += row;
            }
        }
    }
}

```

```

        M02 += (long)row*row;
        M11 += (long)column*row;
    }
}

if (M00 < AREA_MIN) { ctr->area = M00; return (0); }
else {
    ctr->fxc = xcenter = ((float) M10 / (float) M00);
    ctr->fyc = ycenter = ((float) M01 / (float) M00);
    ctr->xc = (int)(xcenter+0.5);
    ctr->yc = (int)(ycenter+0.5);
    ctr->area = M00;
    /* central moments */
    cMxx = (float)M20 - (xcenter*(float)M10);
    cMyy = (float)M02 - (ycenter*(float)M01);
    cMxy = (float)M11 - (ycenter*(float)M10);
    ftemp1 = -(cMyy / VIDEO_PAR) + (cMxx * VIDEO_PAR);
    ftemp2 = -(cMxy * 2);
    theta = (float) (atan2((double)ftemp2, (double)ftemp1) * DEGREEdiv2);
    ctr->angle = (int) (theta * 100.0);
    xctr->xc = ctr->xc;
    xctr->yc = ctr->yc;
    xctr->ax = CrossXY;
    xctr->by = CrossXY;
    xctr->angle = ctr->angle;
    do_thick_cross(xctr);
    return(1);
}
}

void centroid_float(struct region *pnt, int monflag, byte camera)
{
    static char outstr[2][80] = {"", ""};

    set_default_font(1);
    fgrabber(OFF);

    if ((an_cent_bi_gray_float(pnt, objectPTR[camera], 1))
    {
        if (objectPTR[camera]->area > MaxArea[camera])
            objectPTR[camera]->area = MaxArea[camera];
        setcolor(ERASE_Color);
        outtextxy(100, 470+camera*30, (char *)&outstr[camera][0]);
        setcolor(255);
        if (monflag)
        {
            sprintf((char *)&outstr[camera][0], "x%d=%5.2f, y%d=%5.2f, theta%d=%5.2f,
            A%d=%4d",
                camera, objectPTR[camera]->fxc,
                camera, objectPTR[camera]->fyc,
                camera, (float)objectPTR[camera]->angle/100.,
                camera, objectPTR[camera]->area);
            outtextxy(100, 470+camera*30, (char *)&outstr[camera][0]);
        }
    }
}

```

```

    }
    else
        Area_Error_Warning();

    fgrabber(ON);
}

void centroid_int(struct region *pnt, int monflag, byte camera)
{
    static char outstr[2][80] = {"", ""};

    set_default_font(1);
    fgrabber(OFF);

    if ((an_cent_bi_gray_int(pnt, objectPTR[camera], 1))
        {
            if (objectPTR[camera]->area > MaxArea[camera])
                objectPTR[camera]->area = MaxArea[camera];
            setcolor(ERASE_Color);
            outtextxy(100, 470+camera*30, (char *)&outstr[camera][0]);
            setcolor(255);
            if (monflag)
                {
                    sprintf((char *)&outstr[camera][0], "x%d=%3d, y%d=%3d, theta%d=%5.2f,
                        A%d=%4d",
                        camera, objectPTR[camera]->xc,
                        camera, objectPTR[camera]->yc,
                        camera, (float)objectPTR[camera]->angle/100.,
                        camera, objectPTR[camera]->area);
                    outtextxy(100, 470+camera*30, (char *)&outstr[camera][0]);
                }
        }
    else
        Area_Error_Warning();

    fgrabber(ON);
}

void Area_Error_Warning(void)
{ static char outstr[80] = "";

    set_default_font(2);
    // Area Error ! (Area = 0)
    sprintf(outstr, "Object is Missing !");
    setcolor(255);
    outtextxy(AEr_X, AEr_Y, outstr);
    sound(800); delay(30); nosound();
    delay(99);
    setcolor(ERASE_Color);
    outtextxy(AEr_X, AEr_Y, outstr);
    set_default_font(1);
}

```

```

/*****
/*  Computer Vision System:                               */
/*                                                         */
/*                                     by: Sy-Hung Kuo      */
/*-----*/
/*  Filename: KUO_LINE.C                                  */
/*-----*/
/*  This file contains:                                   */
/*                                                         */
/*  +  the subroutines to draw lines or pixels in the frame grabber.  */
/*                                                         */
/*.....*/
/*                                                         */
/*  Last Modified :  April 1, 1994                        */
/*****

```

```

#include <stdio.h>
#include <io.h>
#include <alloc.h>
#include <fcntl.h>
#include <dos.h>
#include <math.h>
#include <string.h>

```

```

#include "kuo_def.h"

```

```

#define line_Color    0xFF

```

```

extern struct region
{
    int xc, yc, ax, by, angle, thres;
    float fxc, fyc;
    word area;
};

```

```

extern word XMAX, YMAX;
extern byte far *XPixel0;
extern byte far *LineBuffer0, far *LineBuffer;

```

```

/* Function Prototypes */
void undo_line(int x1, int y1, int x2, int y2);
void save_line(int x1, int y1, int x2, int y2);
void fg_line(int x1, int y1, int x2, int y2);
void b_line(int x1, int y1, int x2, int y2);
void v_line(int x, int y1, int y2);
void h_line(int x1, int x2, int y);
void fg_pixel(int x, int y, int pixel_Color);

```

```

void undo_line(int x1, int y1, int x2, int y2)
{
    register int t, delta_d;
    int xerr = 0, yerr = 0, delta_x, delta_y;
    int incx, incy;
    unsigned long temp;

    delta_x = x2 - x1;
    delta_y = y2 - y1;

    if (delta_x > 0) incx = 1;
    else
        if (delta_x == 0) incx = 0;
        else incx = -1;

    if (delta_y > 0) incy = 1;
    else
        if (delta_y == 0) incy = 0;
        else incy = -1;

    delta_x = abs(delta_x);
    delta_y = abs(delta_y);
    if (delta_x > delta_y) delta_d = delta_x;
    else delta_d = delta_y;

    for (t = 0; t <= delta_d+1; t++)
    {
        temp = (long)y1 * (long)XMAX + (long)x1;
        if (temp > SegLimit) outportb(SegSwitch, ON); else outportb(SegSwitch, OFF);
        *(XPixel0 + temp) = *LineBuffer++;

        xerr += delta_x;
        yerr += delta_y;
        if (xerr > delta_d)
        {
            xerr -= delta_d;
            x1 += incx;
        }
        if (yerr > delta_d)
        {
            yerr -= delta_d;
            y1 += incy;
        }
    }
}

```

```

void save_line(int x1, int y1, int x2, int y2)
{
    register int t, delta_d;
    int xerr = 0, yerr = 0, delta_x, delta_y;
    int incx, incy;
    unsigned long temp;

    delta_x = x2 - x1;
    delta_y = y2 - y1;

```

```

if (delta_x > 0) incx = 1;
else
    if (delta_x == 0) incx = 0;
    else incx = -1;

if (delta_y > 0) incy = 1;
else
    if (delta_y == 0) incy = 0;
    else incy = -1;

delta_x = abs(delta_x);
delta_y = abs(delta_y);
if (delta_x > delta_y) delta_d = delta_x;
else delta_d = delta_y;

for (t = 0; t <= delta_d+1; t++)
{
    temp = (long)y1 * (long)XMAX + (long)x1;
    if (temp > SegLimit) outportb(SegSwitch, ON); else outportb(SegSwitch, OFF);
    *LineBuffer++ = *(XPixel0 + temp);

    xerr += delta_x;
    yerr += delta_y;
    if (xerr > delta_d)
    {
        xerr -= delta_d;
        x1 += incx;
    }
    if (yerr > delta_d)
    {
        yerr -= delta_d;
        y1 += incy;
    }
}
}

void fg_line(int x1, int y1, int x2, int y2)
{
    if (x1 == x2) v_line(x1, y1, y2);
    if (y1 == y2) h_line(x1, x2, y1);
    b_line(x1, y1, x2, y2);
}

void b_line(int x1, int y1, int x2, int y2)
{
    register int t, delta_d;
    int xerr = 0, yerr = 0, delta_x, delta_y;
    int incx, incy;
    unsigned long temp;

    delta_x = x2 - x1;
    delta_y = y2 - y1;

```



```

if (delta_x > 0) incx = 1;
else
    if (!delta_x) incx = 0;
    else incx = -1;

if (delta_y > 0) incy = 1;
else
    if (!delta_y) incy = 0;
    else incy = -1;

delta_x = abs(delta_x);
delta_y = abs(delta_y);
if (delta_x > delta_y) delta_d = delta_x;
else delta_d = delta_y;

for (t = 0; t <= delta_d+1; t++)
{
    temp = (long)y1 * (long)XMAX + (long)x1;
    if (temp > SegLimit) outportb(SegSwitch, ON); else outportb(SegSwitch, OFF);
    *(XPixel0 + temp) = line_Color;

    xerr += delta_x;
    yerr += delta_y;
    if (xerr > delta_d)
    {
        xerr -= delta_d;
        x1 += incx;
    }
    if (yerr > delta_d)
    {
        yerr -= delta_d;
        y1 += incy;
    }
}
}

void v_line(int x, int y1, int y2)
{
    register int i;
    int reg;
    long temp;

    if (y1 > y2) { reg = y1; y1 = y2; y2 = reg; }

    temp = (long)y1 * (long)XMAX + (long)x;
    if (temp > SegLimit) outportb(SegSwitch, ON); else outportb(SegSwitch, OFF);
    *(XPixel0 + temp) = line_Color;

    for (i=1; i<=(y2-y1); ++i)
    {
        temp += (long)XMAX;
        if (temp > SegLimit) outportb(SegSwitch, ON); else outportb(SegSwitch, OFF);
        *(XPixel0 + temp) = line_Color;
    }
}

```

```

}

void h_line(int x1, int x2, int y)
{
    register int i;
    int reg;
    long temp;

    if (x1 > x2) { reg = x1; x1 = x2; x2 = reg; }

    temp = (long)y * (long)XMAX + (long)x1;
    if (temp > SegLimit) outportb(SegSwitch, ON); else outportb(SegSwitch, OFF);
    *(XPixel0 + temp) = line_Color;

    for (i=1; i<=(x2-x1); ++i)
    {
        ++temp;
        if (temp > SegLimit) outportb(SegSwitch, ON); else outportb(SegSwitch, OFF);
        *(XPixel0 + temp) = line_Color;
    }
}

void fg_pixel(int x, int y, int pixel_Color)
{
    long temp;

    temp = (long)y * (long)XMAX + (long)x;
    if (temp > SegLimit) outportb(SegSwitch, ON); else outportb(SegSwitch, OFF);
    *(XPixel0 + temp) = pixel_Color;
}

```

```

/*****
/* Computer Vision System: */
/* */
/* by: Sy-Hung Kuo */
/*-----*/
/* Filename: KUO_HIST.C */
/*-----*/
/* This file contains: */
/* */
/* + the subroutines to draw the histogram of an image. */
/* */
/* + the subroutine to equalize the histogram of an image. */
/* */
/*.....*/
/* */
/* Last Modified : April 1, 1994 */
/*****

```

```

#include <graphics.h>
#include <conio.h>

#include "kuo_def.h"

```

```

#define GrayLevel 256
#define Histo_X1 100
#define Histo_X2 488
#define Histo_Y 580
#define OverClean 88
#define OverDraw 30
#define Line_Y 330
#define TIMES 2000.
#define thr_len 60
#define thr_exp 10
#define thr_top 30
#define thr_x 50
#define thr_y 335

```

```

extern struct region
{
    int xc, yc, ax, by, angle, thres;
    float fxc, fyc;
    word area;
};

```

```

extern word THRESHOLD;
extern byte far *XPixel, far *XPixel0;
extern word XMAX, YMAX;

```

```

/* Function Prototypes */
void cleanup_display_board(void);

```

```

void cleanup_left_side(void);
void cleanup_right_side(void);
void draw_window_histogram(struct region *win, byte camera, int displayFlag, int
cleanupFlag);
void draw_window_histogram2(struct region *win, byte camera, int displayFlag);
int *histogram(byte camera, int displayFlag);
void equalize_picture(int x, int y, byte camera, int displayFlag);

void cleanup_display_board(void)
{
    setcolor(ERASE_Color);
    setfillstyle(SOLID_FILL, ERASE_Color);
    bar(2, Line_Y, 862, Histo_Y+OverClean);
}

void cleanup_left_side(void)
{
    setcolor(ERASE_Color);
    setfillstyle(SOLID_FILL, ERASE_Color);
    bar(2, Line_Y, 432, Histo_Y+OverClean);
}

void cleanup_right_side(void)
{
    setcolor(ERASE_Color);
    setfillstyle(SOLID_FILL, ERASE_Color);
    bar(432, Line_Y, 862, Histo_Y+OverClean);
}

void draw_window_histogram(struct region *win, byte camera, int displayFlag, int
cleanupFlag)
{ static int image[GrayLevel];
  int i, j, x, y, HiLimit, Hi;
  int top, bottom, left, right;
  word row, column;
  static char outstr[40];
  unsigned long runner;
  long TotalPixel;

  for (i=0; i<GrayLevel; i++) image[i] = 0;
  TotalPixel = 0L;

  top    = win->yc - win->by;
  bottom = win->yc + win->by;
  left   = win->xc - win->ax;
  right  = win->xc + win->ax;

  fgrabber(OFF);
  for (row=top+1; row<bottom; row++)
  {
    runner = ((unsigned long)row * XMAX) + left + 1;

```

```

    for (column=left+1; column<right; column++)
    {
        if (runner > SegLimit) outportb(SegSwitch, ON); else outportb(SegSwitch,
            OFF);
        XPixel = XPixel0 + runner++;
        image[(byte)*XPixel]++;
        TotalPixel++;
    }
}
fgrabber(ON);

if (cleanupFlag) cleanup_display_board();
else if (camera == 0x00) cleanup_left_side();
    else cleanup_right_side();
if (displayFlag)
{
    if (camera == 0x00) x = Histo_X1;
    else x = Histo_X2;
    SandLine(x, Histo_Y, x+GrayLevel, Histo_Y);
    setcolor(255);
    set_small_font(6);
    sprintf(outstr, "Histogram of Camera %d's Window", camera);
    outtextxy(x-10, Histo_Y+25, outstr);
    // set_default_font(1);
    outtextxy(x-3, Histo_Y+5, "0");
    outtextxy(x+GrayLevel-12, Histo_Y+5, "255");
    outtextxy(x+55, Histo_Y+5, "<- Gray Level ->");

    HiLimit = Histo_Y - Line_Y - OverDraw;
    for (i=0; i<GrayLevel; i++)
        if (image[i] != 0)
        {
            Hi = (int)(image[i]*TIMES/TotalPixel);
            if (Hi > HiLimit) Hi = HiLimit;
            line(x+i, Histo_Y-1, x+i, Histo_Y-1-Hi);
        }
    sprintf(outstr, "Current Threshold = %d", THRESHOLD);
    outtextxy(x+thr_x, thr_y, outstr);
    y = Line_Y + thr_top;
    SandDashedLine(x+THRESHOLD, y-10, x+THRESHOLD, Histo_Y-2);
}
}

void draw_window_histogram2(struct region *win, byte camera, int displayFlag)
{ static int image[GrayLevel], position[GrayLevel];
  int i, j, HiLimit, Hi;
  int top, bottom, left, right;
  word row, column;
  static float probability[GrayLevel], cumulateProb[GrayLevel];
  static char outstr[40];
  unsigned long runner;
  long TotalPixel;

  for (i=0; i<GrayLevel; i++) image[i] = 0;

```

```

TotalPixel = 0L;

top    = win->yc - win->by;
bottom = win->yc + win->by;
left   = win->xc - win->ax;
right  = win->xc + win->ax;

fgrabber(OFF);
for (row=top+1; row<bottom; row++)
{
    runner = ((unsigned long)row * XMAX) + left + 1;
    for (column=left+1; column<right; column++)
    {
        if (runner > SegLimit) outportb(SegSwitch, ON); else outportb(SegSwitch,
            OFF);
        XPixel = XPixel0 + runner++;
        image[(byte)*XPixel]++;
        TotalPixel++;
    }
}
fgrabber(ON);

if (displayFlag)
{
    cleanup_display_board();

    SandLine(Histo_X1, Histo_Y, Histo_X1+GrayLevel, Histo_Y);
    setcolor(255);
    set_small_font(6);
    sprintf(outstr, "Histogram of Window%d's Image", camera);
    outtextxy(Histo_X1, Histo_Y+25, outstr);
    // set_default_font(1);
    outtextxy(Histo_X1-3, Histo_Y+5, "0");
    outtextxy(Histo_X1+GrayLevel-12, Histo_Y+5, "255");
    outtextxy(Histo_X1+55, Histo_Y+5, "<- Gray Level ->");

    HiLimit = Histo_Y - Line_Y - OverDraw;
    for (i=0; i<GrayLevel; i++)
        if (image[i] != 0)
        {
            Hi = (int)(image[i]*TIMES/TotalPixel);
            if (Hi > HiLimit) Hi = HiLimit;
            line(Histo_X1+i, Histo_Y-1, Histo_X1+i, Histo_Y-1-Hi);
        }
}

for (i=0; i<GrayLevel; i++)
    probability[i] = ((float)image[i]) / ((float)TotalPixel);
for (i=0; i<GrayLevel; i++)
{
    cumulateProb[i] = 0;
    for (j=0; j<=i; j++)
        if (image[j] != 0) cumulateProb[i] += probability[j];
    position[i] = (int)((GrayLevel-1) * cumulateProb[i]);
}

```

```

if (displayFlag)
{
    SandLine(Histo_X2, Histo_Y, Histo_X2+GrayLevel, Histo_Y);
    setcolor(255);
    set_small_font(6);
    sprintf(outstr, "Histogram of Equalized Window%d", camera);
    outtextxy(Histo_X2, Histo_Y+25, outstr);
    // set_default_font(1);
    outtextxy(Histo_X2-3, Histo_Y+5, "0");
    outtextxy(Histo_X2+GrayLevel-12, Histo_Y+5, "255");
    outtextxy(Histo_X2+55, Histo_Y+5, "<- Gray Level ->");
    for (i=0; i<GrayLevel; i++)
        if (image[i] != 0)
            {
                Hi = (int)(image[i]*TIMES/TotalPixel);
                if (Hi > HiLimit) Hi = HiLimit;
                line(Histo_X2+position[i], Histo_Y-1, Histo_X2+position[i],
                    Histo_Y-1-Hi);
            }
    }
}

int *histogram(byte camera, int displayFlag)
{ static int image[GrayLevel], position[GrayLevel];
  int i, j, x, HiLimit, Hi;
  static float probability[GrayLevel], cumulateProb[GrayLevel];
  static char outstr[40];
  unsigned long runner;
  long TotalPixel;

  for (i=0; i<GrayLevel; i++) image[i] = 0;
  TotalPixel = 0L;

  fgrabber(OFF);
  runner = 0;
  for (j=0; j<YMAX; ++j)
    for (i=0; i<XMAX; ++i)
      {
        if (runner > SegLimit) outportb(SegSwitch, ON); else outportb(SegSwitch,
          OFF);
        XPixel = XPixel0 + runner++;
        image[(byte)*XPixel]++;
        TotalPixel++;
      }
  fgrabber(ON);

  if (displayFlag)
  {
    if (camera == 0x00) x = Histo_X1;
    else                x = Histo_X2;
    SandLine(x, Histo_Y, x+GrayLevel, Histo_Y);
    setcolor(255);
    set_small_font(6);
    sprintf(outstr, "Histogram of Camera %d's Image", camera);

```

```

    outtextxy(x-5, Histo_Y+25, outstr);
    // set_default_font(1);
    outtextxy(x-3, Histo_Y+5, "0");
    outtextxy(x+GrayLevel-12, Histo_Y+5, "255");
    outtextxy(x+55, Histo_Y+5, "<- Gray Level ->");

    HiLimit = Histo_Y - Line_Y - OverDraw;
    for (i=0; i<GrayLevel; i++)
        if (image[i] != 0)
            {
                Hi = (int)(image[i]*TIMES/TotalPixel);
                if (Hi > HiLimit) Hi = HiLimit;
                line(x+i, Histo_Y-1, x+i, Histo_Y-1-Hi);
            }
    }

    for (i=0; i<GrayLevel; i++)
        probability[i] = ((float)image[i]) / ((float)TotalPixel);
    for (i=0; i<GrayLevel; i++)
        {
            cumulateProb[i] = 0;
            for (j=0; j<=i; j++)
                if (image[j] != 0) cumulateProb[i] += probability[j];
            position[i] = (int)((GrayLevel-1) * cumulateProb[i]);
        }

    return ((int *)position);
}

void equalize_picture(int x, int y, byte camera, int displayFlag)
{ int *position;
  int i, j;
  unsigned long runner;

  position = histogram(camera, displayFlag);

  fgrabber(OFF);
  runner = 0;
  for (j=0; j<YMAX; ++j)
    for (i=0; i<XMAX; ++i)
      {
        if (runner > SegLimit) outportb(SegSwitch, ON); else outportb(SegSwitch,
          OFF);
        XPixel = XPixel0 + runner++;
        putpixel(x+i, y+j, position[(byte)*XPixel]);
      }
  fgrabber(ON);
}

```



```

/*****
/* Computer Vision System: */
/* */
/* by: Sy-Hung Kuo */
/*-----*/
/* Filename: KUO_THRE.C */
/*-----*/
/* This file contains: */
/* */
/* + the subroutines to define or change the threshold. */
/* */
/*.....*/
/* */
/* Last Modified : April 1, 1994 */
/*****

```

```

#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <graphics.h>

```

```

#include "kuo_def.h"

```

```

extern struct region
{
    int xc, yc, ax, by, angle, thres;
    float fxc, fyc;
    word area;
};

```

```

extern struct region Window[CamNum];
extern int XMAX, YMAX;
extern byte far *XPixel, far *XPixel0;
extern word THRESHOLD;

```

```

/* Function Prototypes */
void threshold(struct region *pnt, int BW_flag);
void threshold_window(struct region *pnt, word threshold, int BW_flag);
void change_threshold(void);
void next_picture(byte camera);
void calibration_test(byte camera);

```

```

void threshold(struct region *pnt, int BW_flag)
{
    fgrabber(OFF);
    threshold_window(pnt, THRESHOLD, BW_flag);
    fgrabber(ON);
}

```

```

void threshold_window(struct region *pnt, word threshold, int BW_flag)
{
    int i, j;
    int x, y, top, bottom, left, right;
    unsigned long reg, runner;

    x = pnt->xc;
    y = pnt->yc;

    top    = y - pnt->by;
    bottom = y + pnt->by;
    left   = x - pnt->ax;
    right  = x + pnt->ax;

    reg = (unsigned long)(top+1) * XMAX;

    for (i = top+1; i <= bottom-1; i++)
    {
        runner = reg + (left + 1);
        for (j = left+1; j <= right-1; j++)
        {
            if (runner > SegLimit) outportb(SegSwitch, ON); else outportb(SegSwitch,
                OFF);
            XPixel = XPixel0 + runner++;
            if (*XPixel >= threshold) *XPixel = 0xFF;
            else if (BW_flag) *XPixel = 0x00;
        }
        reg += XMAX;
    }
}

```

```

void change_threshold(void)
{
    struct textsettingstype textinfo;
    char str[30];

    gettextsettings(&textinfo);
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);

    sprintf( str, "Current Threshold = %d", THRESHOLD );
    outtextxy(100, 630, str);
    sprintf( str, "Please input the new threshold ? " );
    outtextxy(100, 640, str);

    THRESHOLD = (word) input_an_integer(400, 640, str);
    settextstyle(textinfo.font, textinfo.direction, textinfo.charsize);
}

```

```

void next_picture(byte camera)
{
    sample_picture(camera);
    window_region((struct region *)&Window[camera]);
}

```

```
}

void calibration_test(byte camera)
{ int cx, cy;

  cx = (int) (Xmax2 / 2);
  cy = (int) (Ymax / 2);
  sample_picture(camera);
  fgrabber(OFF);
  fg_pixel(cx, cy, 0x00);
  fg_pixel(cx+1, cy, 0x00);
  fg_pixel(cx-1, cy, 0x00);
  fg_pixel(cx, cy+1, 0x00);
  fg_pixel(cx, cy-1, 0x00);
  fg_pixel(cx+2, cy+2, 0x00);
  fg_pixel(cx+2, cy-2, 0x00);
  fg_pixel(cx-2, cy+2, 0x00);
  fg_pixel(cx-2, cy-2, 0x00);
  fg_pixel(cx+3, cy, 0x00);
  fg_pixel(cx-3, cy, 0x00);
  fg_pixel(cx, cy+3, 0x00);
  fg_pixel(cx, cy-3, 0x00);
  fg_pixel(cx+2, cy, 0xFF);
  fg_pixel(cx-2, cy, 0xFF);
  fg_pixel(cx, cy+2, 0xFF);
  fg_pixel(cx, cy-2, 0xFF);
  fgrabber(ON);
}
```

Bibliography

Ballard, D. H., and Brown, C. M. [1982]. *Computer Vision*, Prentice-Hall.

Block, J. R., and Yuker, H. E. [1989]. *Can You Believe Your Eyes?* Gardner Press, Inc.

Casasent, A., Cheatham, L., and Fetterly, D. [1982]. "Optical System to Compute Intensity Moments: Design." *Applied Optics*.

Castleman, K. R. [1979]. *Digital Image Processing*, Prentice-Hall.

Chaudhuri, B. B. [1983]. "A Note on Fast Algorithms for Spatial Domain Techniques in Image Processing." *IEEE Trans. Syst. Man Cyb.*, vol. SMC-13, no. 6, pp. 1166-1169.

Craig, J. J. [1989]. *Introduction to Robotics: Mechanics and Control*, Addison-Wesley.

Dudani, S., Breeding, K., and McGhee, R. [1977]. "Aircraft Identification by Moment Invariants." *IEEE Transactions on Computers*.

Equitz, W. H. [1989]. "A New Vector Quantization Clustering Algorithm." *IEEE Trans. Acous. Speech Signal Processing*, vol. ASSP-37, no. 10, pp. 1568-1575.

- Fu, K. S., Gonzalez, R. C., and Lee, C. S. G. [1987]. *Robotics: Control, Sensing, Vision, and Intelligence*, McGraw-Hill.
- Gonzalez, R. C., and Woods, R. E. [1992]. *Digital Image Processing*, Addison-Wesley.
- Graham, C. H. [1965]. *Vision and Visual Perception*, John Wiley & Sons.
- Green, W. B. [1983]. *Digital Image Processing—A Systems Approach*, Van Nostrand Reinhold.
- Grimson, W. E. L. [1981]. *From Images to Surfaces*, MIT Press, Cambridge.
- Hering E., Bridgeman B., and Stark L. [1977]. *The Theory of Binocular Vision*, Plenum Press, New York.
- Horn, B. K. P. [1986]. *Robot Vision*, McGraw-Hill.
- Hu, M. K. [1962]. "Visual Pattern Recognition by Moment Invariants." *IRE Trans. Inf. Theory*, IT-8.

- Jain, A. K. [1989]. *Fundamentals of Digital Image Processing*, Prentice-Hall.
- Kittler, J., and Ullingworth, J. [1985]. "On Threshold Selection Using Clustering Criteria." *IEEE Transactions on Systems, Man, and Cybernetics*. Vol. SMC-15.
- Lee, C. C. [1983]. "Elimination of Redundant Operations for a Fast Sobel Operator." *IEEE Trans. Syst. Man Cybern.*, vol. SMC-13, no. 3, pp. 242-245.
- McKeown, D. M., Harvey, W. A., and McDermott, J. [1985]. "Rule-Based Interpretation of Aerial Imagery." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-7, no. 5, pp. 570-585.
- Nguyen, A. H. [1992]. "Model Control of Image Processing for Telerobotics and Biomedical Instrumentation." *Ph.D. Thesis, UC Berkeley*.
- Nguyen, A. H., and Stark, L. [1993]. "Top-down Model Control of Image Processing: Pupillometry." *Computerized Medical Imaging and Graphics*.
- Otsu, N. [1979]. "A Threshold Selection Method from Gray-Level Histograms." *IEEE Transactions on Systems, Man, and Cybernetics*. Vol. SMC-9.

Perez, A., and Gonzalez, R. C. [1987]. "An Iterative Thresholding Algorithm for Image Segmentation." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-9, no. 6, pp. 742-751.

Pratt, W. K. [1991]. *Digital Image Processing*, John Wiley & Sons.

Schalkoff, R. J. [1989]. *Digital Image Processing and Computer Vision*, John Wiley & Sons.

Shariat, H., and Price, K. E. [1990]. "Motion Estimation with More Than Two Frames." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, no. 5, pp. 417-434.

Smith, F., and Wright, M. [1971]. "Automatic Ship Photo Interpretation by the Method of Moments." *IEEE Transactions on Computers*.

Sobel, I. [1970]. "Camera Models and Machine Perception." *Ph.D. Thesis, Stanford University*.

Stark, L., Mills, B., Nguyen, A. H., and Ngo, H. [1988]. "Instrumentation and Robotic Image Processing Using Top-down Model Control." *Robotics and Manufacturing*.

Tsai, R. Y. [1986]. "An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision." *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 364-374.

Weszka, J. S. [1978]. "A Survey of Threshold Selection Techniques." *Comput. Graphics Image Proc.*, vol. 7, pp. 259-265.

Wood, R. C. [1969]. "On Optimum Quantization." *IEEE Trans. Info. Theory*, vol. IT-15, pp. 248-252.

Wu, A. Y., and Hong, T. H. and Rosenfeld. [1982]. "Threshold Selection Using Quadrees." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-4.

Zahalak, G. I. [1981]. "A Distribution-Moment Approximation for Kinematic Theories of Muscular Contraction." *Math. Biosciences*.