MACHINE VISION FOR SPACE ROBOTIC APPLICATIONS

by

Ender St. John - Olcayto

Bachelor of Engineering in Aerospace Engineering
University of Glasgow, Scotland
1988

Submitted in Partial Fulfillment of
the Requirements for the Degree of

MASTER OF SCIENCE IN
AERONAUTICS AND ASTRONAUTICS

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUNE 1990

Signature of Author [illegible]
Department of Aeronautics and Astronautics
May 11, 1990

Certified by [illegible]
Professor Harold Alexander
Thesis Supervisor
Department of Aeronautics and Astronautics

Accepted by [illegible]
Professor Harold Y. Wachman
Chairman, Departmental Graduate Committee
Department of Aeronautics and Astronautics

Machine Vision For Space Robotic Applications

by

Ender St. John - Olcayto

Submitted to the Department of Aeronautics and Astronautics
on May 11, 1990 in partial fulfillment of the requirements for
the degree of Master of Science in Aeronautics and Astronautics

Abstract

A vision system that locates and tracks the corners of a rectangle image has been designed. The system forms the framework upon which machine vision algorithms suitable for use on a space robot will be used. An attempt has been made to implement two algorithms to determine the pose of the rectangle from its two dimensional image. The algorithms attempted to use the screen locations of the corners. The corner location algorithm performed satisfactorily but was occasionally prone to misplacing the corners for certain image configurations. Tracking was achieved using three different methods. The simplest, in which the search for a corner proceeds from the its last known screen location was shown to perform better than the other two, which attempted to extrapolate the trajectories of the corners. The pose determination algorithms implemented the inverse perspective transformation (IPT) and the inverse orthographic transformation (IOT). The IPT requires all four corners of the image to calculate the pose of the rectangle but the IOT requires only three. The IPT assumes that the image arises from a perspective view of the target whereas with the IOT, it is assumed that it is formed by orthographically projecting the target onto the image plane. Neither the IPT or the IOT as currently implemented produced satisfactory results. However, the performance of the IOT is considerably ambiguous since the rectangle pose angles that it returns are with respect to the image and not to the camera. Future study is therefore required to relate the results from the IOT to the camera angles by knowing the accurate camera geometry. Since the IOT requires only three corners to calculate the pose, there exists the possibility of using a least squares procedure on the four sets of results that may be obtained from a rectangle image to improve the accuracy of the results.

Thesis Supervisor : Harold L. Alexander
Charles Stark Draper Assistant Professor
of Aeronautics and Astronautics

Acknowledgements

During my two year career as a graduate student, which could be described as virtually an instantaneous event by some MIT collegians, I have learned such wealth that I would not have thought possible in such a short period of time. This was possible, not only due to the overall scholarly atmosphere of the Institute but also because I had the fortune of working in a laboratory where there is always a source of help or advice on apparently any subject. Not only did my colleagues provide wisdom when it was most needed but also were also a source of wit at those crucial times when there was apparently little to laugh about.

I would first like to thank Terry Fong(e), who had the dubious pleasure of not only sharing an office with me but also an apartment. His uncompromising willingness to help in all matters technical and otherwise deserves my sincerest gratitude. I would especially like to extend my thanks to those software gurus Rob Sanner; whose suggestions where of immeasurable value (curse those pointers!), and Sam Druker; whose eagle eye spotted the most intractable of my programming errors. Ali, the man in search of an image but who found FMAID3 instead, should also be thanked for his patience with me, even though I must appeared quite unreasonable in my requests for help. My thesis advisor Professor Harold Alexander, who first suggested my research topic and came to my aid during certain critical occasions (of which there were many!) cannot be thanked enough for taking on board a wide eyed and enthusiastic graduate.

Since my stay here, I have made friends who have helped me in ways too numerous to mention. Firstly John Doran, the other half (some would say more) of the dynamic duo, with whom I first explored Boston. It was he who introduced me to expressions such as *"Up the yard!"* and who kept calling me *"A real lad"* for reasons that I am not entirely sure. Together, our dry wit and sense of cynicism would leave others completely in the dark as to what we were talking about. In this important *"stage of my life"*, I have had the great pleasure of meeting Eilish Doheny. She has helped me in many ways by her continual encouragement and her ability to always make me smile, even when the tasks ahead of me looked insurmountable.

Most though, I would like to thank my parents, whose love and encouragement through the years has given me the strength to pursue my childhood dreams. I will always be indebted to my Father, who still appears to be able to do anything, and to my Mother, whose patience and kind words in hours of need helped me through the most difficult of times. I can only hope to return a fraction of what they have given to me.

I would like to dedicate this work to my grandmother (Babaanne - my Father's Mother),
who died in February, 1990.

Chapter 1

Introduction

1.1 General Laboratory Activities

For the past decade, the MIT Space Systems Laboratory (SSL) has conducted research directed toward creating techniques and tools with which the industrialization of space will require. Orbital productivity, achieved by extra-vehicular activity (EVA); telerobots and autonomous vehicles, has been studied by SSL with such applications in mind as spacecraft and space station construction and orbital infrastructure support.

A major effort has been directed toward tele-operator and robot research to support orbital productivity studies. EVA is a particularly hazardous task. For the missions that are envisioned by todays space engineers and scientists, astronauts may be required to work in conditions which are as dangerous, if not more so, than at the bottom of the ocean. The hazards of EVA are numerous. Astronauts will be subjected to high radiation dosages and unique task related stresses, both physical, since movement is retarded by a cumbersome space suit, and psychological, caused by a number of factors but including the nature of working in a hostile environment. It is for these reasons that research into reducing the amount of human effort in space by the use of robots and tele-operated vehicles has been pursued in SSL. Work within the space tele-robotic field has included developing novel control strategies, the integration of multi-sensor systems within a control loop, and human/robot interaction. These strategies are tested on neutrally buoyant space vehicles within the MIT Alumni Swimming Pool to simulate their response in an environment that approximates the near weightless condition of orbital flight.

1.2 Focus of Research

The laboratory has relied on a variety of conventional navigational sensors for telerobotic operations. For example, the vehicle MPOD (Multi-mode Proximity Operations Device) employs :

1) An inclinometer system consisting of three pendula, each of which is constrained to rotate about one of three mutually perpendicular reference axes. The rotation of the pendula about their respective axes indicates the angles that the gravity vector makes with these axes. There are several important drawbacks to this system. Firstly, since the pendula bearings must be damped such that they do not oscillate about their equilibria, their speed of response is limited. Even a damping which corresponds to the fastest response time results in some oscillation. This damping also varies over time as the material suffers wear. Secondly, when the vehicle is in a configuration such that the gravity vector is parallel to one of the pendulum axes, that pendulum will adopt some arbitrary angle, making it difficult to calculate the gravity vector orientation. It may also be prone to large swings due to small angular displacements from this singular condition.

2) A rate-gyroscope package where each gyroscope returns an angular rate about one of three reference axes. The output signals of these devices are usually very noisy, corrupting the measurements.

3) 3-DAPS (Three-Dimensional Acoustic Positioning Device). This system uses a system of fixed ultrasonic emitters and vehicle mounted hydrophones. By sequencing sound pulses and measuring the time elapsed from the start of the pulse to its detection by the hydrophones, the distances to the emitters can be calculated from which the vehicle orientation may be inferred [Rowley, V.M. 1989]. Problems associated with 3-DAPS are its low update rate of around one and a half seconds, loss of data associated with the vehicle body or some other object blocking the path between a hydrophone and an emitter, and data corruption due to other acoustic sources within the test vicinity.

The main thrust of this study was to investigate the use of machine vision, to circumvent many of the problems associated with the current laboratory navigation techniques, some of which have been detailed above. Machine vision does not rely on any moving components and hence will not suffer from deterioration. Moreover, vision is not susceptible to the electrical noise which affects several of the sensors currently in use. Additionally, like 3-DAPS, it is possible to obtain both range and orientation with respect to some reference frame in a self contained system.

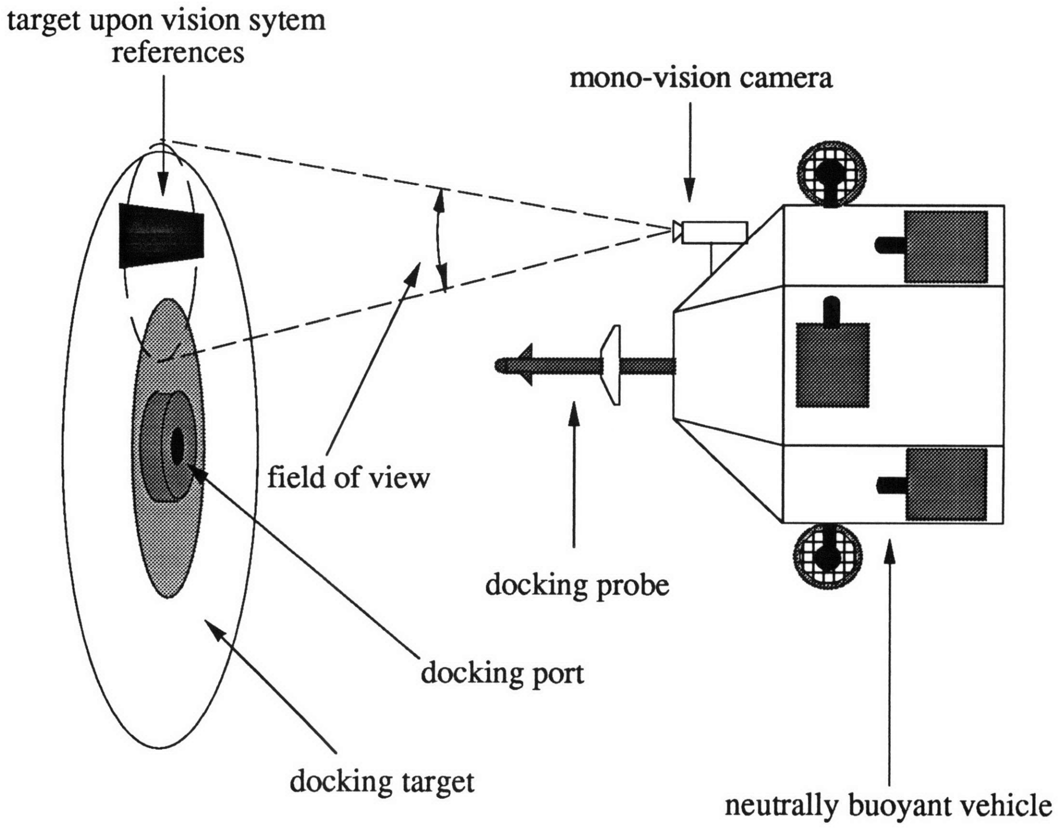


Figure 1.2-1) A neutrally buoyant space robot simulator using a vision system as a sensor for a feedback control in a docking operation

However, machine vision has a real application in a space environment since systems that rely on some medium to propagate sound waves, like 3-DAPS, cannot operate in space. A primary aim of this project was to create a system that could be incorporated, with as little modification as possible, into one of the current neutrally buoyant vehicles such that a variety of vehicle control algorithms can easily be integrated. A secondary aim was to investigate the possibility of machine vision as a means of feedback for an automatic control system in an autonomous robot and as a means of operator feedback for manual control of a space tele-robot. The system could potentially be incorporated within a feedback control loop on a vehicle in order to assist in such tasks as docking (see figure 1.2-1)), path following, and station keeping.

1.3 Scope of Project

Research is oriented around creating a system that has the ability to implement a certain class of image understanding algorithms. The algorithms to which particular attention was paid were those that are capable of extracting 3-D information from a single 2-D view of a planar object. A planar object is used since such shapes are likely to be common in a real space infrastructure. For example, such shapes may compose purposely laid paths for robot vehicles to follow, or marked on docking targets or even the structural elements of orbital assemblies. Planar objects also have simple geometries that can easily be modeled within a computer so that many aspects of machine vision like image recognition and feature point labeling are greatly simplified. Edge detection is likely to be easier than with more complex shapes due to a roughly even illumination over the surface. The use of non-planar targets , with their increased geometric complexity, may cause range and orientation determination difficulties due to such factors as increased internal modeling requirements and increased algorithm size. This would create increased computer memory and speed requirements.

The use of a single 2-D image for navigation purposes has several advantages over stereo vision. These include reduced hardware requirements since only one camera and a video digitizer are required. This is a pertinent issue for space operations since the less equipment need to complete a given task is highly desirable due to the expense of transporting materials to orbit. There is also the reduced risk of system breakdown since there are fewer components that may fail. Many algorithms exist for the 2-D image understanding, two of which will be investigated as possible techniques for vehicle navigation.

1.4 Image Processing

1.4.1 Background

Image processing can be classified into four main areas [Lim, J.S. 1990]:

1) Image enhancement, to improve image quality or intelligibility for human and machine processing,

2) Image restoration, to reduce the degradation of an image that has been corrupted by noise or some other agent,

3) Image coding, to compress the relevant information to as few bits as possible,

4) Image understanding, to extract useful information from an image.

It is the last step with which this project will mainly be concerned. However, various aspects of the other steps may be required for this last step to take place. The last step differs from the others in that the output from an image understanding algorithm is a symbolic representation of that image, whereas for the other steps the output is also an image [Lim, J.S. 1990]. This last step usually entails locating various feature points in an image, where an image feature is defined to be an attribute which can be used in an image understanding algorithm and a feature point is a feature that can be completely described by its image coordinates.

The determination of the orientation of a planar surface is a fundamental computer vision task . It is well known in the science of photogrammetry (the process of surveying, especially from aerial photographs) that if the 3-D coordinates and their corresponding 2-D perspective projections of an object are known, then the observer position as well as the look angles (the orientation of the observer with respect to the object) may be computed. Photogrammetry theory may be slightly extended to determine the pose of a target, provided that its geometry and its image coordinates are known [Haralick, R.M. 1980]. This project will be concerned with the particular case of determining the pose of a rectangular target with respect to a video camera. A rectangle is used since only four points, namely its vertices, are required to describe it fully. In order that the processed information may be used within a feedback control system, it is required that the configuration parameters be determined in real-time.

To achieve the desired goal, the algorithmic requirements of a vision system for real-time applications may be specified as follows :

1) Edge detection,

2) Locating feature points,

3) Feature point identification,

4) Tracking feature points as they move on the screen,

5) Determining the pose of the rectangle from the time varying image.

It is the last step that the next chapter will describe in detail.

Chapter 2

Theoretical Development

2.1 Inverse Image Transformations

The 3-D world is viewed as if rays from objects are projected onto an imaginary plane, called the image plane, which exists between the observer and the object being viewed. Image transformations relate the 3-D world to images on the image plane, whereas inverse transformations achieve the opposite. In order to determine the pose of a target from its image, inverse image transformations are used. In order for the inverse transformations to work, it is necessary to know some subset of its feature point locations. For the purpose of this project, where the target is a rectangle, the feature points are naturally the corners.

This chapter describes two kinds of inverse image transformations that may be-implemented in a vision system. The first is the inverse perspective transformation and the other is the inverse orthographic transformation.

2.2 Image Projection

The mathematics of projection are not new [Plastock, R.A., Kalley, G. 1986]. For hundreds of years, artists, engineers and architects have represented 3-D objects on a 2-D medium, for example the canvas upon which a scene is painted. There are two different kinds of projection : perspective and parallel (orthographic). Perspective projection represents a scene as it appears, whereas parallel projection retains the true size and shape of the object. Humans view objects through perspective projection. However, the parallel projection may be approximated as a perspective projection if a linear scale factor is included. This approximation becomes inaccurate if the object subtends a large angle in the direction of the field of view. This is due to the scale being unable to represent the entire object. The orthographic projection with a scale factor is termed a *"weak perspective projection"*. The inverse perspective and orthographic transformation will now be presented.

2.2.1 The Perspective Transformation and its Inverse

Perspective images are formed as rays from an object converge to a single point. This is the basis of the pin-hole camera , shown in figure 2.2.1-1) [Rummel, P. 1989].

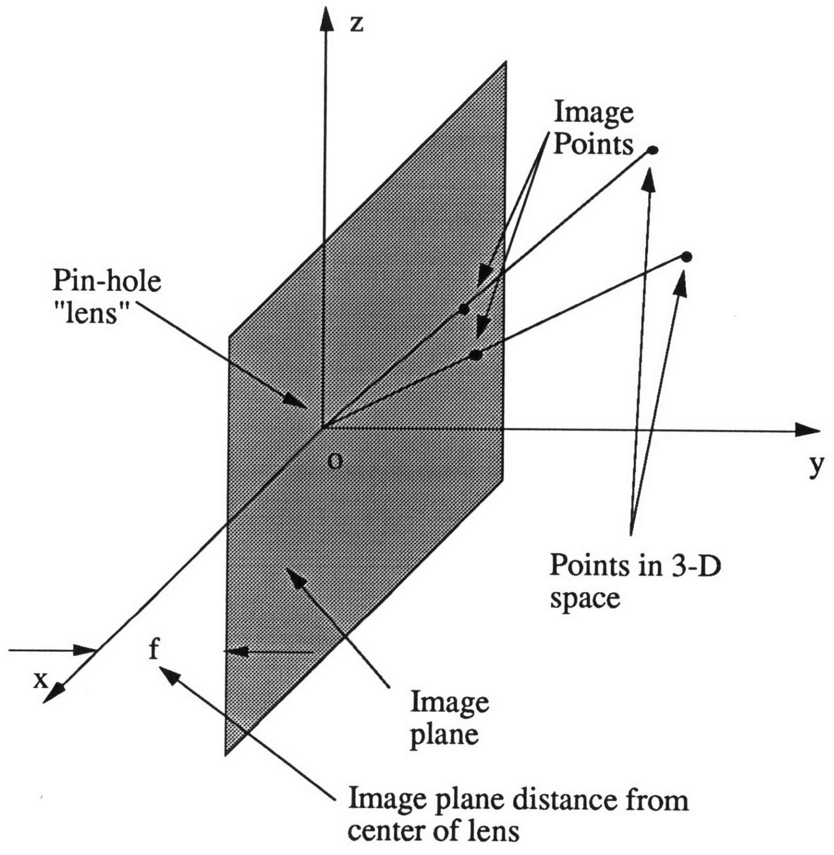


Figure 2.2.1-1) The pin-hole camera model for the perspective transformation

A perspective image of an object places an observer at the *center of projection* and the image appears on an *image plane* between the observer and the object (figure 2.2.1-1)). Perspective images are characterized by *perspective foreshortening* and *vanishing points* (figures 2.2.1-2) and 2.2.1-3)).

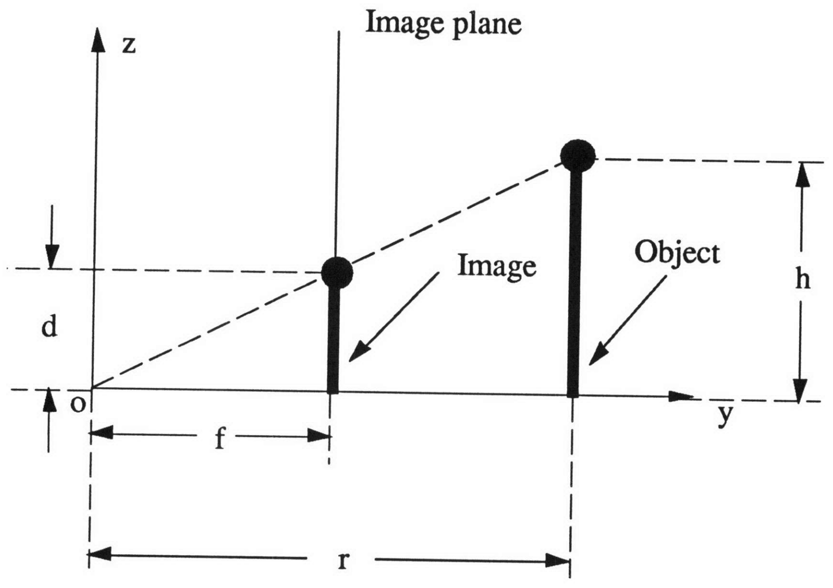


Figure 2.2.1-2) The effect of perspective foreshortening

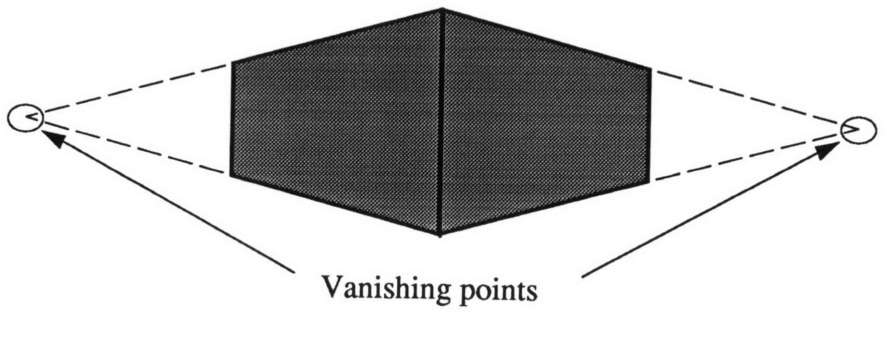


Figure 2.2.1-3) Vanishing points in a perspective image

It should be noted that the following development is carried out for a rectangle existing in a $z = z_0$ plane. A modification is then made to the theory to take into account that fact that the software assumes the target to lie in a $y = y_0$ plane.

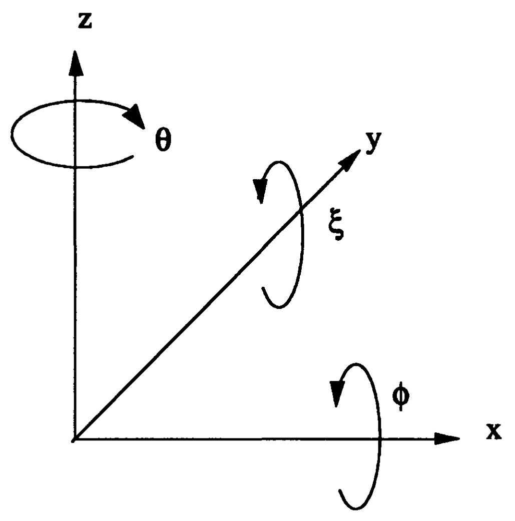


Figure 2.2.2-1) The tilt, swing and pan angles of a camera about the x, y and z axes

2.2.2 The Look-Angles from the Perspective Projection of a Rectangle

It is required to solve for the camera look angles i.e. the tilt; ϕ, swing; ξ, and pan; θ angles (figure 2.1.2-1)). Notice that camera convention dictate the tilt angle is defined to be positive clockwise. The perspective image of a rectangle parallel to the x-y plane is projected onto the image plane which is located at a distance f in front of the center of projection. The locations of the image corners are given by [Haralick, R.M. 1989]:

$$\begin{bmatrix} x_i^* \\ z_i^* \end{bmatrix} = \begin{bmatrix} \cos\xi & \sin\xi \\ -\sin\xi & \cos\xi \end{bmatrix} \begin{bmatrix} x_i' \\ z_i' \end{bmatrix}, \quad i = 1, 2, 3, 4 \qquad 2.1)$$

where

$$x' = f \frac{x \cos\theta + y \sin\theta}{-x \cos\phi \sin\theta + y \cos\phi \cos\theta + z \sin\phi} \qquad 2.2\ a)$$

$$z' = f \frac{x \sin\phi \sin\theta - y \sin\phi \cos\theta + z \cos\phi}{-x \cos\phi \sin\theta + y \cos\phi \cos\theta + z \sin\phi} \qquad 2.2\ b)$$

If the perspective projection (x^*, z^*) is known then the set of 3-D coordinates that could have produced this image point is given by :

$$\left\{ \begin{bmatrix} x \\ y \\ z \end{bmatrix} \middle| \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \lambda \begin{bmatrix} x' \cos\theta - f \sin\theta \cos\phi + z' \sin\theta \sin\phi \\ x' \sin\theta + f \cos\theta \cos\phi - z' \sin\phi \\ f \sin\phi + z' \cos\phi \end{bmatrix}, \begin{bmatrix} x' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\xi & -\sin\xi \\ \sin\xi & \cos\xi \end{bmatrix} \begin{bmatrix} x^* \\ z^* \end{bmatrix} \right\}$$

2.3)

for some λ. λ may take on any value since an infinite number of points that lie on the ray defined in Eq 2.3) above may have produced the image point.

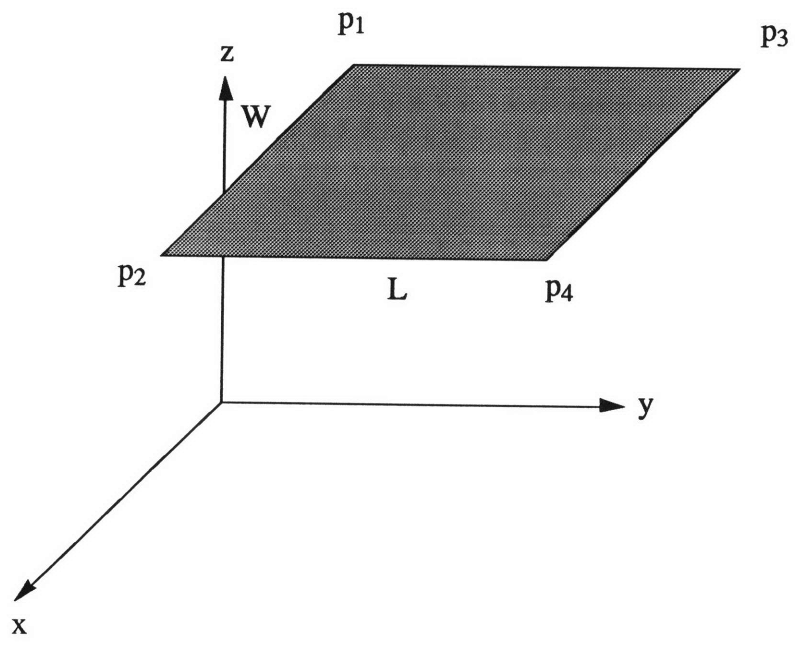


Figure 2.2.2-1) The target rectangle with respect to the world reference frame

The set of points defined in Eq. 2.3) may be restricted by constraining the all four image points to have arisen as perspective projections of the corners of a rectangle. If the rectangle is as shown in figure 2.2.2-1), the vertices are given by :

$$\mathbf{p}_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}, \qquad \mathbf{p}_2 = \begin{bmatrix} x_1 + W \\ y_1 \\ z_1 \end{bmatrix},$$
$$\mathbf{p}_3 = \begin{bmatrix} x_1 \\ y_1 + L \\ z_1 \end{bmatrix}, \qquad \mathbf{p}_4 = \begin{bmatrix} x_1 + W \\ y_1 + L \\ z_1 \end{bmatrix} \tag{2.4)}$$

where $y_1 > f$. Their corresponding image points are :

$$\mathbf{p}_i^* = \begin{bmatrix} x_i^* \\ z_i^* \end{bmatrix}, i = 1, 2, 3, 4 \qquad 2.5)$$

The observer look-angles may be obtained from Eqs. 2.3) and 2.4) [Haralick, R.M. 1989] :

$$\xi = \tan^{-1} \frac{[\, A(z_1^* - z_3^*) - B(z_2^* - z_4^*) - C(z_1^* - z_2^*) + D(z_3^* - z_4^*)]}{[A(x_1^* - x_3^*) - B(x_2^* - x_4^*) - C(x_1^* - x_2^*) + D(x_3^* - x_4^*)]} \qquad 2.6)$$

where

$$A = \frac{(x_2^*\, z_4^* - x_4^*\, z_2^*)}{E} \qquad 2.7\text{ a)}$$

$$B = \frac{(x_1^*\, z_3^* - x_3^*\, z_1^*)}{E} \qquad 2.7\text{ b)}$$

$$C = \frac{(x_3^*\, z_4^* - x_4^*\, z_3^*)}{F} \qquad 2.7\text{ c)}$$

$$D = \frac{(x_1^*\, z_2^* - x_2^*\, z_1^*)}{F} \qquad 2.7\text{ d)}$$

$$E = f\,[\; (x_1^* - x_3^*)(z_2^* - z_4^*) - (x_2^* - x_4^*)(z_1^* - z_3^*) \;] \qquad 2.7\text{ e)}$$

$$F = f\,[\; (x_1^* - x_2^*)(z_3^* - z_4^*) - (x_3^* - x_4^*)(z_1^* - z_2^*) \;] \qquad 2.7\text{ f)}$$

$$\phi = \tan^{-1} \frac{(x_2'\, z_4' - x_4'\, z_2')(z_1' - z_3') - (x_1'\, z_3' - x_3'\, z_1')(z_2' - z_4')}{f\,[(x_1' - x_3')(z_2' - z_4') - (x_2' - x_4')(z_1' - z_3')} \qquad 2.8)$$

$$\theta = \tan^{-1}\frac{\cos\phi\,(x_1' z_3' - x_3' z_1')(x_2' - x_4') - (x_2' z_4' - x_4' z_2')(x_1' - x_3')}{f\,[(x_1' - x_3')(z_2' - z_4') - (x_2' - x_4')(z_1' - z_3')} \qquad 2.9)$$

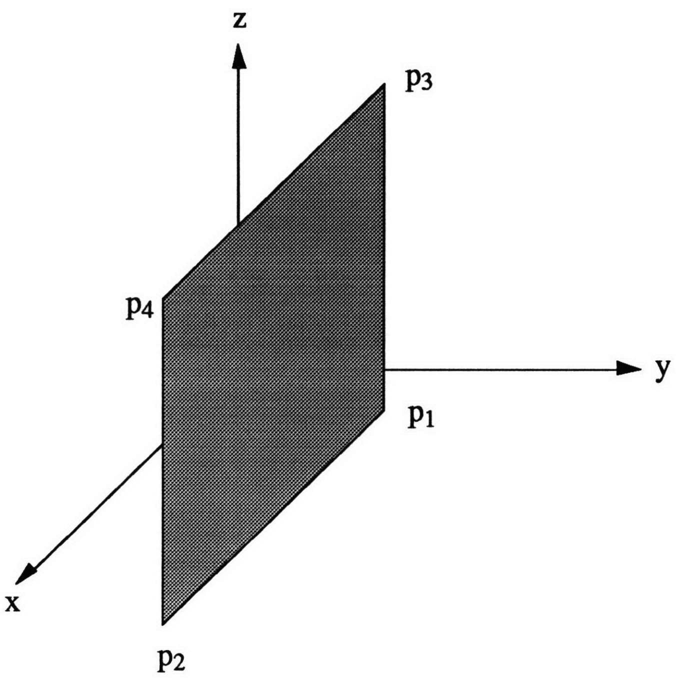


Figure 2.2.2-2) The orientation of the target used in implementing the inverse perspective transformation

For the purpose of this particular analysis, the rectangle is assumed to lie in the $y = y_1$ ($y_1 > f$) as shown in figure 2.2.2-2). From equation A1-21) in appendix A.1, the look angles are recalculated as :

$$\phi = \sin^{-1}(\cos\phi' \cos\theta') \qquad 2.10)$$

$$\theta = \tan^{-1}\frac{\cos\phi' \cos\theta}{-\sin\phi'} \qquad 2.11)$$

$$\xi = \tan^{-1}\frac{(\cos \xi' \sin \phi' - \sin \xi' \sin \phi' \cos \theta')}{(\sin \xi' \sin \theta' - \cos \xi' \sin \phi' \cos \theta')} \qquad 2.12)$$

where the dashed angles are those calculated earlier.

As can be seen from Eqs 2.10) - 2.12), the camera orientation may be determined without knowledge of the dimensions of the target. If the dimensions are known, however, the range may be also be calculated. The coordinates of the bottom left hand corner can be found as :

$$y_1 = \frac{W(x_2' \sin \theta + f \cos \theta \cos \phi - z_2' \cos \theta \sin \phi)(x_1' \sin \theta + f \cos \theta \cos \phi - z_1' \cos \theta \sin \phi)}{(x_2' - x_1') f \cos \phi + (x_1' z_2' - x_2' z_1') \sin \phi} \qquad 2.13)$$

$$x_1 = y_1 \frac{x_1' \cos \theta - f \sin \theta \cos \phi + z_1' \sin \theta \sin \phi}{x_1' \sin \theta - f \cos \theta \cos \phi + z_1' \cos \theta \sin \phi} \qquad 2.14)$$

$$z_1 = y_1 \frac{f \sin \phi + z_1' \cos \phi}{x_1' \sin \theta - f \cos \theta \cos \phi + z_1' \cos \theta \sin \phi} \qquad 2.15)$$

2.2.3 The Scaled Orthographic Transformation and its Inverse (*The Weak Perspective Transformation)*

Unlike the perspective transformation where all rays from the object converge to a single point (the center of projection), under orthographic projection, rays emanating from the target are parallel to each other, preserving the dimensions of the target under the projection. The scaled inverse orthographic projection attempts to find the rotations and scaling upon which the orthographic projection of the target, considered as a plane that can be rotated in 3-D space, must be operated such that it matches the image. The target projection must therefore be translated in the 2-D plane, rotated in 3-D space and then appropriately scaled such that its own orthographic projection coincides with that of the target's.

The transformation that achieves the above operations is called a *similarity transform..* A 3-D similarity transform T may be represented thus :

$$T(\mathbf{x}) = s\,\mathbf{R}\,\mathbf{x} + \mathbf{t} \qquad \mathbf{R}, \mathbf{t}, \mathbf{x} \in \Re^3, s \in \Re \qquad 2.16)$$

2.2.4 Evaluating the Transformation

The rotation matrix and the scale factor can be computed in a series of simple steps [Huttenlocher D. P. 1988], given three points of a planar target in 3-D space and their corresponding image points (see Appendix A.1) :

$$\mathbf{p}_{m_i} = \begin{bmatrix} x_{m_i} \\ y_{m_i} \\ z_{m_i} \end{bmatrix} i = 1, 2, 3 \qquad 2.17)$$

and the corresponding screen points :

$$\mathbf{p}_{s_i} = \begin{bmatrix} x_{s_i} \\ y_{s_i} \end{bmatrix} i = 1, 2, 3 \qquad 2.18)$$

1) Rotate and translate the target such that $\mathbf{p}_{m_1}$ is at the origin (0, 0, 0) and $\mathbf{p}_{m_2}$ and $\mathbf{p}_{m_3}$ are in the x-y plane,

2) Define the translation vector $\mathbf{b} = -\mathbf{p}_{m_1}$ and translate the screen points by $\mathbf{b}$ so that $\mathbf{p}_{m_1}$ is at the origin, $\mathbf{p}_{m_2}$ is at $\mathbf{p}_{m_2} + \mathbf{b}$ and $\mathbf{p}_{m_3}$ is at $\mathbf{p}_{m_3} + \mathbf{b}$,

3) Solve for the linear transformation :

$$\mathbf{L} = \begin{bmatrix} l_{11}\, l_{12} \\ l_{21}\, l_{22} \end{bmatrix}$$

given by the two pairs of equations :

$$l_{11}x_{m_2} + l_{12}y_{m_2} = x_{s} \qquad 2.19\text{ a)}$$
$$l_{11}x_{m_3} + l_{12}y_{m_3} = x_{s} \qquad 2.19\text{ b)}$$

$$l_{21}x_{m_2} + l_{22}y_{m_2} = x_{s} \qquad 2.20\text{ a)}$$
$$l_{21}x_{m_3} + l_{22}y_{m_3} = x_{s} \qquad 2.20\text{ b)}$$

4) Solve for c_1 and c_2 using :

$$c_1 = \pm\sqrt{\frac{1}{2}\left(w + \sqrt{w^2 + 4\,q^2}\right)} \qquad 2.21)$$

and

$$c_2 = \frac{-q}{c_1} \qquad 2.22)$$

where

$$w = l_{12}^2 + l_{22}^2 - (\,l_{11}^2 + l_{21}^2\,)$$

and

$$q = l_{11}l_{12} + l_{21}l_{2}$$

5) The scaled rotation matrix is given by :

$$s\,\mathbf{R} = \begin{bmatrix} l_{11} & l_{12} & \dfrac{(\,c_2\, l_{21} - c_1\, l_{22}\,)}{s} \\ l_{21} & l_{22} & \dfrac{(\,c_1\, l_{12} - c_2\, l_{11}\,)}{s} \\ c_1 & c_2 & \dfrac{(\,l_{11}\, l_{22} - l_{21}\, l_{12}\,)}{s} \end{bmatrix} \qquad 2.23)$$

where

$$s = \sqrt{l_{11}^2 + l_{11}^2 + c_1^2}.$$

The angles by which the target projection must be rotated may now be obtained from the rotation matrix. If the rotations are assumed to have arisen from consecutive rotations about the x, y and z axes respectively i.e. :

$$\mathbf{R} = \mathrm{Rot}\,(z, \psi)\ \mathrm{Rot}\,(\,y, \theta)\ \mathrm{Rot}\,(x, \phi) \qquad 2.24)$$

or more explicitly :

$$\mathbf{R} = \begin{bmatrix} c_\psi\, c_\theta & c_\psi\, s_\theta\, s_\phi + s_\psi\, c_\phi & -c_\psi\, s_\theta\, c_\phi + s_\psi\, s_\phi \\ -s_\psi\, s_\theta & -s_\psi\, s_\theta\, s_\phi + c_\psi\, c_\phi & s_\psi\, s_\theta\, c_\phi + c_\psi\, s_\phi \\ s_\theta & -c_\theta\, s_\phi & c_\theta\, c_\phi \end{bmatrix} \qquad 2.25)$$

the angles are therefore given by :

$$\begin{aligned} \theta &= \mathrm{atan2}(\,\mathbf{R}_{31},\ c_\psi\mathbf{R}_{11}\ -\ s_\psi\mathbf{R}_{21}\,) \\ &= \mathrm{atan2}(\,-\mathbf{R}_{31},\ c_\phi\mathbf{R}_{33}\ -\ s_\phi\mathbf{R}_{33}\,) \end{aligned} \qquad 2.26\ a)$$

$$\psi = \mathrm{atan2}(\,-\mathbf{R}_{21},\ \mathbf{R}_{11}\,) \qquad 2.26\ b)$$

$$\phi = \mathrm{atan2}(\,-\mathbf{R}_{32},\ \mathbf{R}_{33}\,) \qquad 2.26\ c)$$

where the atan2() function accepts two arguments and returns a value between $\pm\,\pi$.

There exists a singularity problem with Euler angles when $\theta = \pm\,90^c$ i.e. when $\mathbf{R}_{31} = \pm\,1$. This means that ψ and ϕ cannot be solved for independently. The singularity problem is overcome by fixing either ψ or ϕ at its previous value [Spofford, J. 1986] and solving for the other. Fixing ψ, we have :

$$\begin{aligned} \psi &= \psi_{\mathrm{old}} \\ \phi &= \mathrm{atan2}(\,s_\psi\mathbf{R}_{13} + c_\psi\mathbf{R}_{23},\ s_\psi\mathbf{R}_{12} + c_\psi\mathbf{R}_{22}) \end{aligned} \qquad 2.27)$$

The scale factor s provides a simple way of determining the range of the target. Since under an orthographic projection of an object, all dimensions are retained, the scale factor indicates the amount by which the *target* must scaled to obtain the image dimensions. Using the fact that all images are really produced by perspective projection, the similar triangles property may be used to obtain the range.

Referring to figure 2.1.2-2), if the scale factor gives the amount by which the orthographic projection of the target onto the image plane must be multiplied i.e.

$$hs = d \tag{2.28)}$$

then, knowing the image plane distance from the from the observer at the origin as f, the range r is given by :

$$r = \frac{f}{s} \tag{2.29)}$$

Chapter 3

Implementation Requirements

3.1 Introduction

This chapter details the needs of a vision system that is to implement algorithms that are based on knowledge of the feature points of the target. Feature point detection based on a binary search is first discussed. Three different techniques for feature point tracking using are then proposed.

3.2 Feature Point Location

The perspective and alignment transformations outlined in the previous chapter, assume knowledge of the four corner locations of the rectangle. Before the transformations can be applied therefore, some feature point extractor is required to obtain the corners. For real-time use, it is both infeasible and wasteful to use a global screen search for each new frame. A more realistic method would be a local search based upon past corner positions.

The feature point extractor used makes use of the fact that each corner is at the intersection of two edges, where each edge is a straight line. Since two points completely define a line, the extractor finds two points on each edge adjacent to the corner and solves for their intersection.

The target, which is a planar surface, has a uniform intensity. The image is thresholded such that everything above a defined intensity u will be treated as pure white and everything below as pure black. This permits a binary search to be performed across an edge,which now appears as a step change in intensity.

3.2.1 Locating Edge Boundary Crossings

An algorithm that searches a circle in the vicinity of a feature point has been developed. There are two distinct sections to the search; an approximate search which locates the region in which an edge exists, and a fine search which locates an edge point to within a precision ε.

Referring to figure 3.2.1-1) the following operations are performed :

1) A vector **r** is projected from a point in the vicinity of a corner,

2) The coordinates, denoted by position vector $\mathbf{p}_0$ and intensity $u(\mathbf{p}_0)$, of the vector end point are noted,

3) The vector is rotated by an amount $\Delta\theta$,

4) The intensity at the end of the vector is again checked. If it is different from before, it is known that an edge lies between the current and previous points. If the intensity is the same as the last point, the vector is again rotated until a point of different intensity is found.

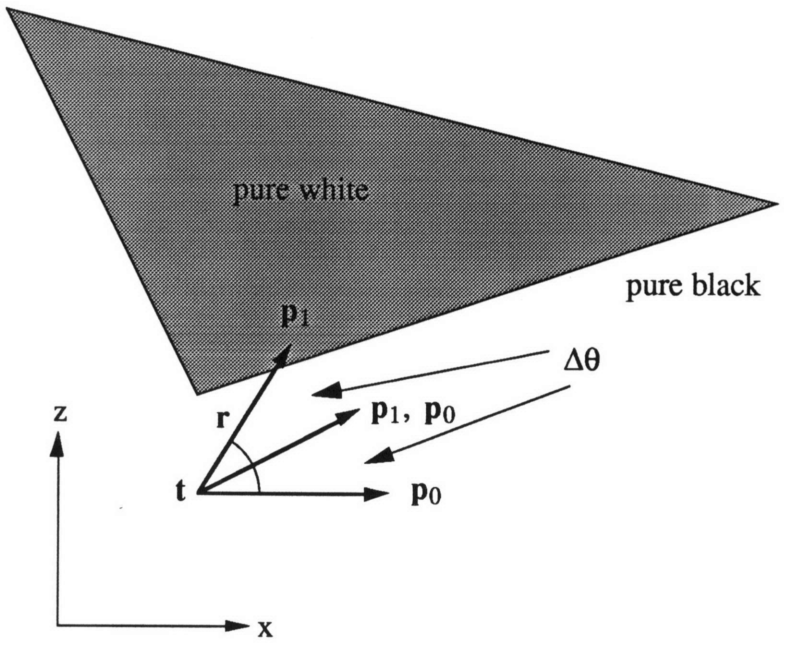


Figure 3.2.1-1) Projection of a search vector **r** from the initial search point **t** to find a brightness change

Algorithm *"Roughsearch"* :

```
    project vector r from initial search
    point

    get position vector $\mathbf{p}_0$ and intensi-
    ty $u(\mathbf{p}_0)$ of search vector endpoint

    $\mathbf{p}_1 \leftarrow \mathbf{p}_0$
    $u(\mathbf{p}_1) \leftarrow u(\mathbf{p}_0)$

    iterate :

        $\mathbf{p}_0 \leftarrow \mathbf{p}_1$
        $u(\mathbf{p}_0) \leftarrow u(\mathbf{p}_1)$

        rotate r by an amount $\Delta\theta$

        get position vector $\mathbf{p}_1$ and
        intensity $u(\mathbf{p}_1)$ of current
        vector endpoints.
    while u($\mathbf{p}_1$) = u($\mathbf{p}_0$)
    store $\mathbf{p}_1$
```

An accurate location of an edge boundary is found using the algorithm *"Finesearch"*, whose description follows :

1) The line joining the recently checked point whose intensity is different from the previous point i.e. $\mathbf{p}_1$ and $\mathbf{p}_0$ is bisected by a point $\mathbf{p}_2$.

2) Get intensity u($\mathbf{p}_2$) of $\mathbf{p}_2$,

3) If the intensities u($\mathbf{p}_1$) and u($\mathbf{p}_2$) are not equal, then $\mathbf{p}_2$ lies on the other side of the edge separating it from $\mathbf{p}_1$. In this case, $\mathbf{p}_2$ becomes the reference point whose intensity is opposite to $\mathbf{p}_1$ i.e. $\mathbf{p}_0 \leftarrow \mathbf{p}_2$, otherwise $\mathbf{p}_1 \leftarrow \mathbf{p}_2$,

4) d, the square of the Cartesian distance between the points that represent opposite intensities is calculated,

5) If d is greater than a specified tolerance value, then repeat the process, otherwise, the edge boundary is taken to be the point defined by position vector $\mathbf{p}_1$.

Algorithm *"Finesearch"*

```
    iterate :
        $\mathbf{p}_2 = \frac{(\mathbf{p}_1 + \mathbf{p}_0)}{2}$
        Get intensity u($\mathbf{p}_2$) of $\mathbf{p}_2$
        if $u(\mathbf{p}_2) \neq u(\mathbf{p}_1)$ :
            $\mathbf{p}_0 = \mathbf{p}_2$
        else :
            $\mathbf{p}_1 = \mathbf{p}_2$

        $d = \| \mathbf{p}_1 - \mathbf{p}_0 \|^2$

    while $d > \varepsilon^2$
```

Figure 3.2.1-2) shows the sequence of operations for two passes of the *finesearch* algorithm.

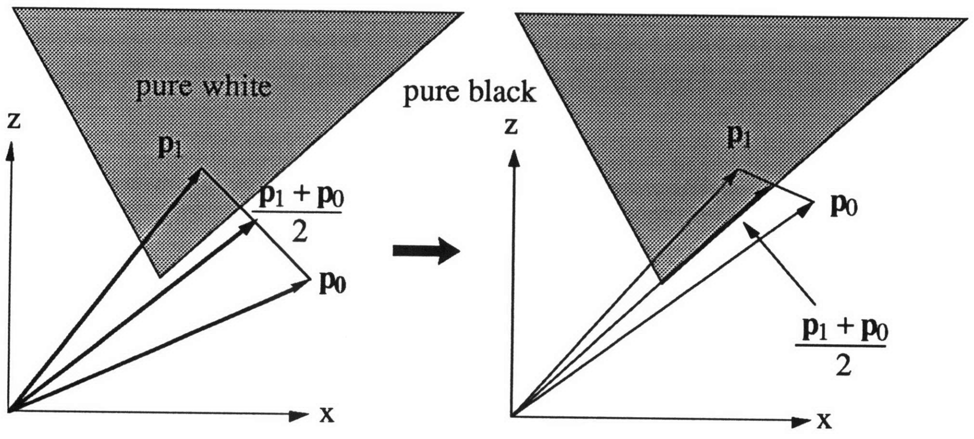


Figure 3.2.1-2) The fine binary search operation to locate an edge boundary

This process is repeated beginning with a new *Roughsearch* from the point where *Roughsearch* originally found a brightness change. This results in two edge points at a distance which is equal to the magnitude of **r** from the initial search point. The magnitude of **r** is decreased and the entire process is repeated. This results in four edge points, two on each edge adjacent to the vertex. The labeling scheme used is shown in figure 3.2.1-3).

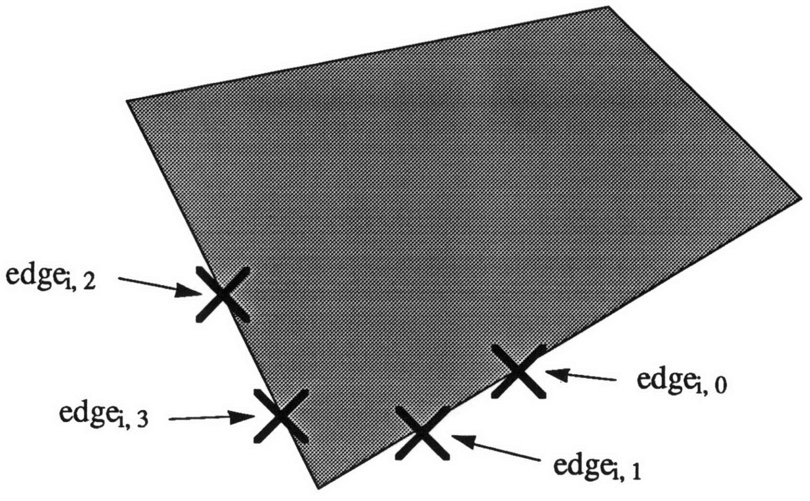


Figure 3.2.1-3) The convention for edge boundary crossing labeling

The first subscript denotes the current corner and the second denotes the edge boundary crossing.

3.2.2 Edge Gradient Determination

Given two points on a line, the gradient of that line can be determined. Furthermore, given two points on two different lines, the point of intersection may be solved. This simple observation allows the determination of the locations of the corners of the rectangle.

It should be noted that the equations of the edges are independent on the labeling of the crossings. Referring to figure 3.2.1-3), the gradient of $edge_{i_1}$ is given by :

$$\mathrm{grad}(edge_{i_1}) = \frac{Z(edge_{i,0}) - Z(edge_{i,1})}{X(edge_{i,0}) - X(edge_{i,1})} \qquad 3\text{-}1)$$

where the X and Z symbols denote the x and z components of the points as their arguments.

3.2.3 Feature Point Solving

Given the gradients and boundary locations of the edges adjacent to the feature points, the point of intersection of the edges may now be solved. Given the equations of the edges :

$$z - b_1 = m_1(x - c_1) \tag{3-3a)}$$

$$z - b_2 = m_2(x - c_2) \tag{3-3b)}$$

where

$b_1 = Z(\mathrm{edge}_{i,1})$ $\qquad$ $b_2 = Z(\mathrm{edge}_{i,3})$

$c_1 = X(\mathrm{edge}_{i,1})$ $\qquad$ $c_2 = X(\mathrm{edge}_{i,3})$

$m_1 = \mathrm{grad}(\mathrm{edge}_{i_1})$ $\qquad$ $m_2 = \mathrm{grad}(\mathrm{edge}_{i_2})$

Solving Eqs.3-3a) and 3-3b) :

$$X(\mathbf{p}_i) = \frac{m_1c_1 - m_2c_2 + b_2 - b_1}{m_1 - m_2} \tag{3-4a)}$$

$$Z(\mathbf{p}_i) = m_1(X(\mathbf{p}_i) - c_1) + b_1 \tag{3-4b)}$$

However, a problem exists when one of the edges has infinite gradient. The above development must be modified to cater for this condition. A line with infinite gradient is given by :

$$x = K \tag{3-5)}$$

where K is a constant. The two possible solutions for $\mathrm{edge}_{i_1} = \infty$ or $\mathrm{edge}_{i_2} = \infty$ are given below :

if $m_1 = \infty$

$$X(\mathbf{p}_i) = X(edge_{i,1})$$

$$Z(\mathbf{p}_i) = m_2(X(\mathbf{p}_i) - X(edge_{i,3})) + Z(edge_{i,3})$$

if $m_2 = \infty$

$$X(\mathbf{p}_i) = X(edge_{i,3})$$

$$Z(\mathbf{p}_i) = m_1(X(\mathbf{p}_i) - X(edge_{i,1})) + Z(edge_{i,1})$$

3.3 Point Tracking

In order that the pose be determined for a time varying image, the rectangle corners need to be tracked over time. The tracking algorithm should be robust enough to track the motions of the points that may be generated by the motion of an underwater vehicle.

The anticipated feature points positions are determined using the history of their positions in the past and extrapolating forward in time. Presented first is a feature point update routine which anticipates the next location to be the current feature point. This is essentially a method which assumes that the motions are so small that the corners are approximated as being stationary. Zero motion is of course incorrect but it is hoped that the method places the point sufficiently near the true location that the local search, outlined in section 3.2, will allow the true location to be detected. Two other methods that use extrapolation techniques are compared with each other and with the simple update method outlined above.

Method 1 is merely a simple update and is given by :

$$\mathbf{p}_{i+1} = \mathbf{p}_i \qquad 3\text{-}6)$$

Method 2 uses the current position and velocity of the feature point, where velocity is defined to be the vector difference in the current and previous feature point position. It is given by :

$$\mathbf{p}_{i+1} = \mathbf{p}_i + (\mathbf{p}_i - \mathbf{p}_{i-1}) \qquad 3\text{-}7)$$

This may be written as :

$$\begin{bmatrix} \mathbf{p}_{i+1} \\ \mathbf{p}_i \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p}_i \\ \mathbf{p}_{i-1} \end{bmatrix} \qquad 3\text{-}8)$$

Method 3 extends the above further and also uses the acceleration of the feature point. The acceleration is defined to be the difference between the current and previous velocities. The anticipated feature point location is therefore given by :

$$\mathbf{p}_{i+1} = \mathbf{p}_i + (\mathbf{p}_i - \mathbf{p}_{i-1}) + \{(\mathbf{p}_i - \mathbf{p}_{i-1}) - (\mathbf{p}_i - \mathbf{p}_{i-1})\} \qquad 3\text{-}9)$$

This may be written as :

$$\begin{bmatrix} \mathbf{p}_{i+1} \\ \mathbf{p}_i \\ \mathbf{p}_{i-1} \end{bmatrix} = \begin{bmatrix} 3 & 3 & -2 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_i \\ \mathbf{p}_{i-1} \\ \mathbf{p}_{i-2} \end{bmatrix} \qquad 3\text{-}10)$$

The position of the corners as computed using the above methods and the point finding algorithm will be compared by attempting to track the corners of an image of a rectangle as it moves on the screen. The moving image was achieved by changing the camera angles.

The above methods are relatively crude ways of tracking feature points in time varying images. More complex methods of point tracking are available, for example ones in which a set of possible trajectories for each feature point is calculated and the most likely one chosen [Hwang, V.S.S. 1989]. However, what is sacrificed in terms of complexity is gained in computational speed.

Chapter 4

Calibration of Vision Software

4.1 Determining the Image Plane Distance to the Camera

The use of perspective transformations to determine the pose of an object requires knowing the distance of the image plane from the observer origin i.e. the center of projection. Knowing the dimensions of an object and those of its image and the distance of the target from the lens, an experiment can be carried out to determine the location of the plane with respect to the lens.

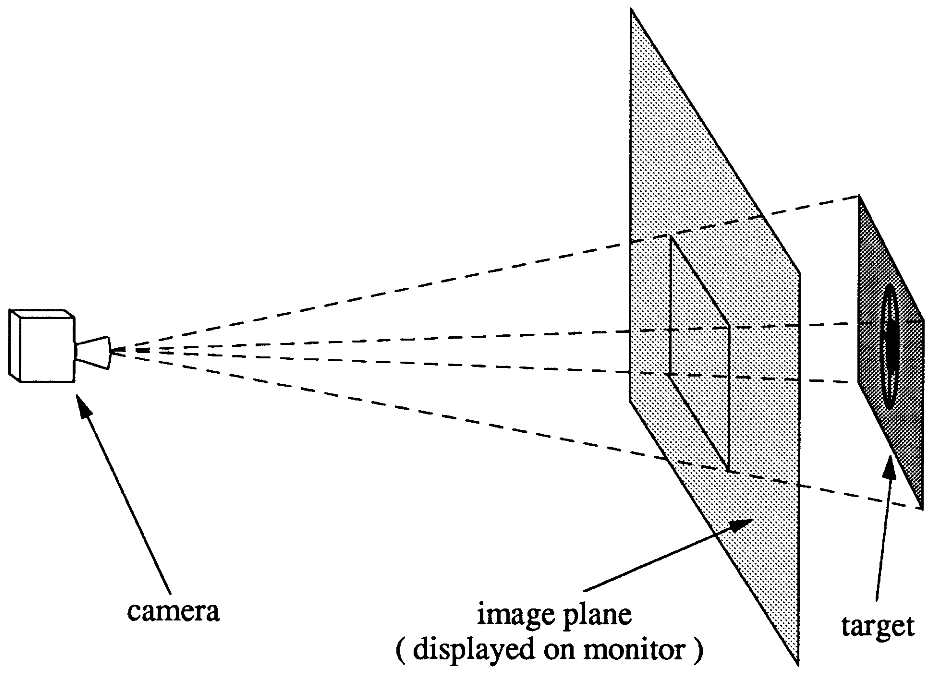


Figure 4.1-1) The camera arranged such that the optic axis is perpendicular to the image plane

Figure 4.1-1) shows a schematic of the camera/image plane/target setup. The image plane is at a distance f from the center of the lens. The camera was auto-focussed for a range of 1.2 metres. This means that an object placed at a distance of 1.2 m from the lens causes the virtual image to coincide with the CCD array of the camera. It is not important that the camera focus is a constant value, since the resultant change in the image plane distance is negligible compared to the range of the target. A circular reference target, shown in figure 4.1-2), with graduations marked on its horizontal and vertical diameters was placed in front of the camera.

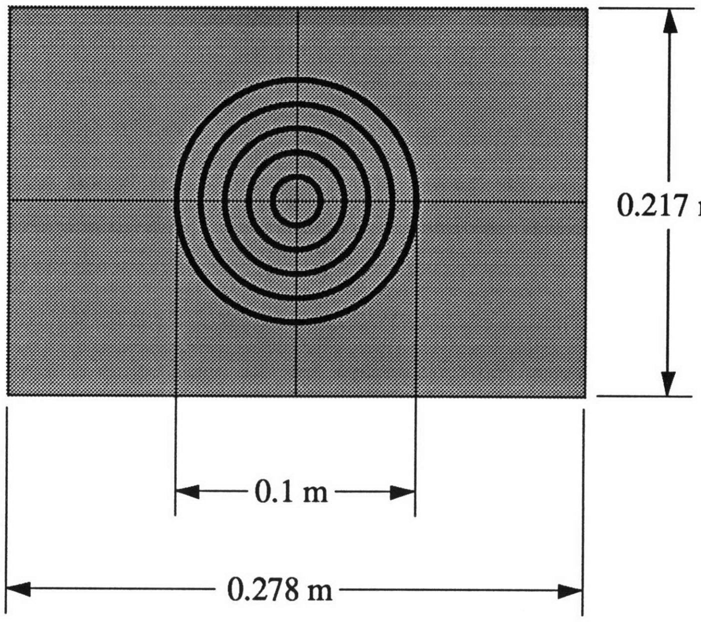


Figure 4.1-2) The target used for determining various operating parameters

The image gradation sizes were then measured. Using three different measurements to decrease the probability of measurement error, similar triangles can be used to determine the distance of the image plane. The camera was placed at a distance r away from the target. The range was measured as the distance from the point at which a plumb bob, hung from the front of the camera, made contact with the ground to the wall on which the target was mounted. For an object of size h, range r and corresponding image size d (see figure 2.1.2-2), the image plane distance f may be calculated :

$$f = \frac{dr}{h} \qquad 4.1)$$

The target was placed at a distance of 1.025m i.e. r = 1.025 m and three sets of measurements, shown in the Table 4.1, were taken

Table 4.1 Data for Calculating Image Plane Distance

h (m)	d (m)	f (m)
0.100	0.08	0.824
0.080	0.07	0.901
0.060	0.05	0.858

From the above results, an average can be found to give the image plane distance :

$$f = 0.86 \text{ m}$$

4.2 Determining Pixel Dimensions

In order that the system work with real linear measurement units i.e. metres, feet etc., instead of pixel coordinates, it is necessary to discover the dimensions of one pixel. Once this is obtained, the x and z scale factors can be calculated i.e. the amount by which the x and z pixel coordinates must be multiplied to obtain the position in terms of metres.

It should be noted that the RS-170 video standard results in 480 lines of digitized information per frame. However, the frame grabber is organized in a 512 by 512 array. This means that there are 32 lines in the frame grabber memory which are not used in the digitizing process. However, this offset in the z direction need not be taken into account within the software.

One method would be to use knowledge of the monitor dimensions. With the x and z dimensions and the number of pixels for each direction known, it is relatively easy to find the pixel dimensions. Another technique would be to draw a square on the screen and to measure the sides to discover the aspect ratio for each pixel. Knowing the number of pixels for each direction, it is relatively easy to find the x and z dimensions of a pixel. An alternative is to place a target such that it lies in the image plane. Knowing that the image of an object lying in the image plane will have all the same dimensions as the object, the number of pixels can be counted in the x and z directions to find how many pixels represent each dimension.

The last two methods will be used; the former, firstly to give the pixel dimensions and the latter to verify the distance of the image plane to the center of the lens. The first method is unsatisfactory since the screen dimension is not a reliable method of measuring the monitor aspect ratio since this may be changed by the manual vertical and horizontal controls.

4.2.1 Method 1

A square of side 200 pixels was plotted on the screen and the linear dimensions of the resulting rectangle were measured directly from the screen. The x and z dimensions were measured as :

$$x_d = 0.093 \text{ m}$$

and

$$z_d = 0.073 \text{ m}$$

Dividing these quantities by 200 gives the size of one pixel :

$$x_{pd_1} = 4.65 * 10^{-4} \text{ m}$$

and

$$z_{pd_1} = 3.65 * 10^{-4} \text{ m}$$

4.2.2 Method 2

The target shown in figure 4.1-2) was placed such that it coincided with the image plane i.e. image dimensions were the same as real dimensions. The pixel locations of the center, the topmost and rightmost gradations were noted :

$$p_{c_1} = \begin{bmatrix} 277 \\ 259 \end{bmatrix}, \quad p_{c_2} = \begin{bmatrix} 277 \\ 394 \end{bmatrix}, \quad p_{c_3} = \begin{bmatrix} 385 \\ 259 \end{bmatrix}$$

The length of the horizontal radius r_h in pixels is given by :

$$\begin{aligned} r_h &= 385 - 277 \\ &= 108 \end{aligned}$$

and the vertical radius by :

$$\begin{aligned} r_v &= 394 - 259 \\ &= 135 \end{aligned}$$

The corresponding pixel dimensions are found by dividing the lengths that the above radii represent. They are given by :

$$\begin{aligned} x_{pd_2} &= \frac{0.05}{108} \\ &= 4.63 * 10^{-4} \text{ m} \end{aligned}$$

and

$$z_{pd_2} = \frac{0.05}{135} = 3.70 * 10^{-4}\ \text{m}$$

These quantities are very similar to those obtained using method one. This verifies the image plane distance f of 0.86 metres. The average of the quantities are found thus to reduce the measurement uncertainty further :

$$x_{pd} = \frac{x_{pd_1} + x_{pd_2}}{2} = \frac{(4.65 + 4.63) * 10^{-4}}{2} = 4.64 * 10^{-4}\ \text{m}$$

$$z_{pd} = \frac{z_{pd_1} + z_{pd_2}}{2} = \frac{(3.65 + 3.70) * 10^{-4}}{2} = 3.675 * 10^{-4}\ \text{m}$$

The quantities x_{pd} and z_{pd} are incorporated as scaling factors in the vision software to obtain the image coordinates in metres.

Chapter 5

Computer Implementation

5.1 Introduction

This chapter describes the implementation of the algorithms outlined in the previous chapters and how various programming issues constrained their application. The various problems associated with implementation and how they have been overcome are presented. All software was written in 'C' and compiled using the Microsoft 5.1 compiler on an IBM PC-AT compatible microcomputer.

5.2 General Implementation Issues

5.2.1 Introduction to Frame Grabber Operation

The video source for the frame grabber is the RS-170 standard video signal. The signal is composed of digital timing pulses and analog video information. An interlacing scheme where every odd-numbered horizontal scan line is first displayed i.e. the first line then the third etc, followed by the even-numbered lines.

An image is digitized with a resolution of 512x512 pixels and 8-bit accuracy resulting in the image being represented by a total of 256 gray levels, a total of 256K of memory is required. Only 64K of the frame grabber memory are addressable at any one time with each block representing one of four quadrants of the screen. A point on the screen is therefore referenced by its quadrant number and local quadrant coordinate with respect to the local origin, located at the top left hand corner of the current block. Knowing the location of a point with respect to the screen coordinate frame, the appropriate 64K memory block may be chosen and local coordinates determined. Once this is achieved the memory location corresponding to the point may be accessed. The memory block is chosen by placing the appropriate value in the frame memory block select register (Appendix A.2).

5.2.2 Image Thresholding

The target, being essentially planar, is assumed to be evenly lit. The image is thresholded so that everything above a critical brightness is taken to be pure white and everything below to be pure black. Even though this information is not translated to the display, the routines that access the FG memory only return values of pure black or pure white. The critical brightness is determined by looking at the brightness across an edge and taking the mean gray level across it. The gray level variation over a target edge is plotted in figure 5.2.2-1).

As can be seen from figure 5.2.2-1), the edge is very well defined. The mean intensity is seen to be around 160. This value is taken to be the threshold value about which the binary search returns either "bright" or "dark".

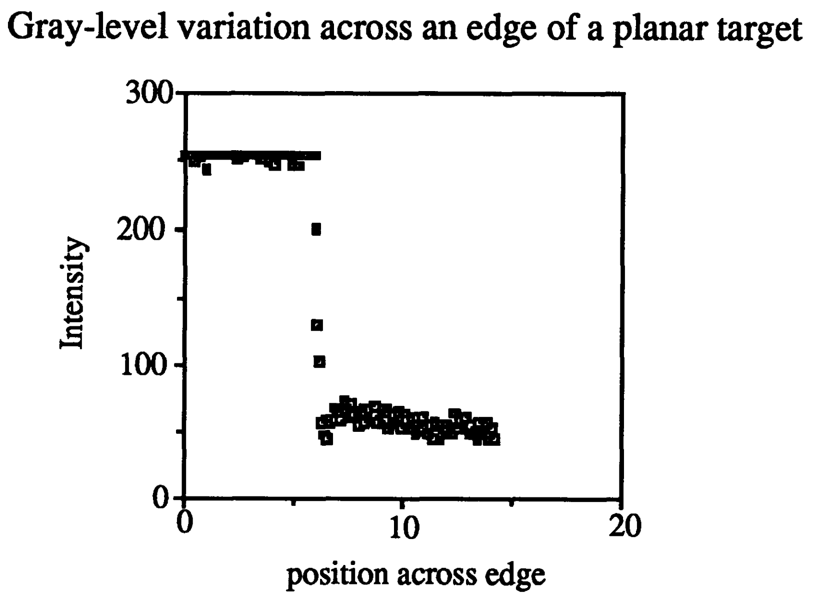


Figure 5.2.2-1) Gray-level variation across an edge of a planar target

5.3 Implementation Details

5.3.1 The Binary Search

The binary search is implemented almost identically to the algorithms *"Roughsearch"* and *"Finesearch"*. The important differences are made in an effort to increase speed and to prevent problems that could occur if the discrete nature of the image is neglected. In *finesearch* , the line joining the points of differing brightness was divided in two. Since division is computationally expensive and noting that the points always have integer components, a bit-wise shift right operation (which is a division by two in binary notation) can be carried out. This is a much more efficient operation than division. The edge crossing routine is performed by the function "findedge" located in file findedge.c (Appendix A.3).

There is a limit to the accuracy that a binary search can achieve when operating on discrete data. For the case when the points of differing binary intensity is only one pixel apart, the binary search will cause a limit cycle to ensue about the edge boundary. It is therefore necessary to terminate the search when the points are two pixels apart. This limits the accuracy of the edge boundary finding algorithm.

5.3.2 Gradient Finding and Feature Point Detection

When solving for the equations of the edges to find their point of intersection, it is necessary retain intermediate variables as floating point type, even though the result will be an integer quantity. If the intermediate results are typecast as integers, a high degree of accuracy will be lost and the feature point will be poorly located.

5.4 Image Transformation Implementation Issues

The perspective and orthographic transformations were implemented in a straightforward fashion and were a direct translation of the governing equations equations. However, with respect to perspective transformation, the target is assumed to lie in a $y = y_0$ plane. This necessitates slight modifications to the equations which are detailed in

Appendix A.1.

The alignment transformation needs only three feature points to process. Since a rectangle is used as the target, a degree of redundancy is available. However, the basic transform was implemented without using any redundancy, hence only three corner positions are required. The redundant case is discussed in Chapter 8.

In order permit large angular deviations of the camera but maintain the image on the screen, it was necessary to relocate the origin at the center of the screen. This translates the origin to the position (255, 255), in pixel coordinates.

5.5 Overall Software System

The requirements, hardware constraints and the operating environment of the vision system have been defined in chapters four and five. The process of software development is achieved by dividing the tasks defined in chapter four into functional units and defining the required data paths that connect each unit with one or more neighbors. Figure 5.5.1-1) shows the architecture of the system.

A description of the system as outlined in figure 5.5.1.1) follows. It should be noted that the communication paths denoted by solid lines indicate routes that are always used. The dashed path between FINDEDGE and PLACE need not be open for correct operation of the system since PLACE merely displays on the screen points that have been located using the other functions. Once the manager program POINTFIND is invoked, the user supplies initial information to the SETUP function. This data consists of the initial search points that the algorithms finesearch and roughsearch require. Once this data is supplied, POINTFIND calls FINDEDGE, which implements the binary search and edge labeling algorithm.

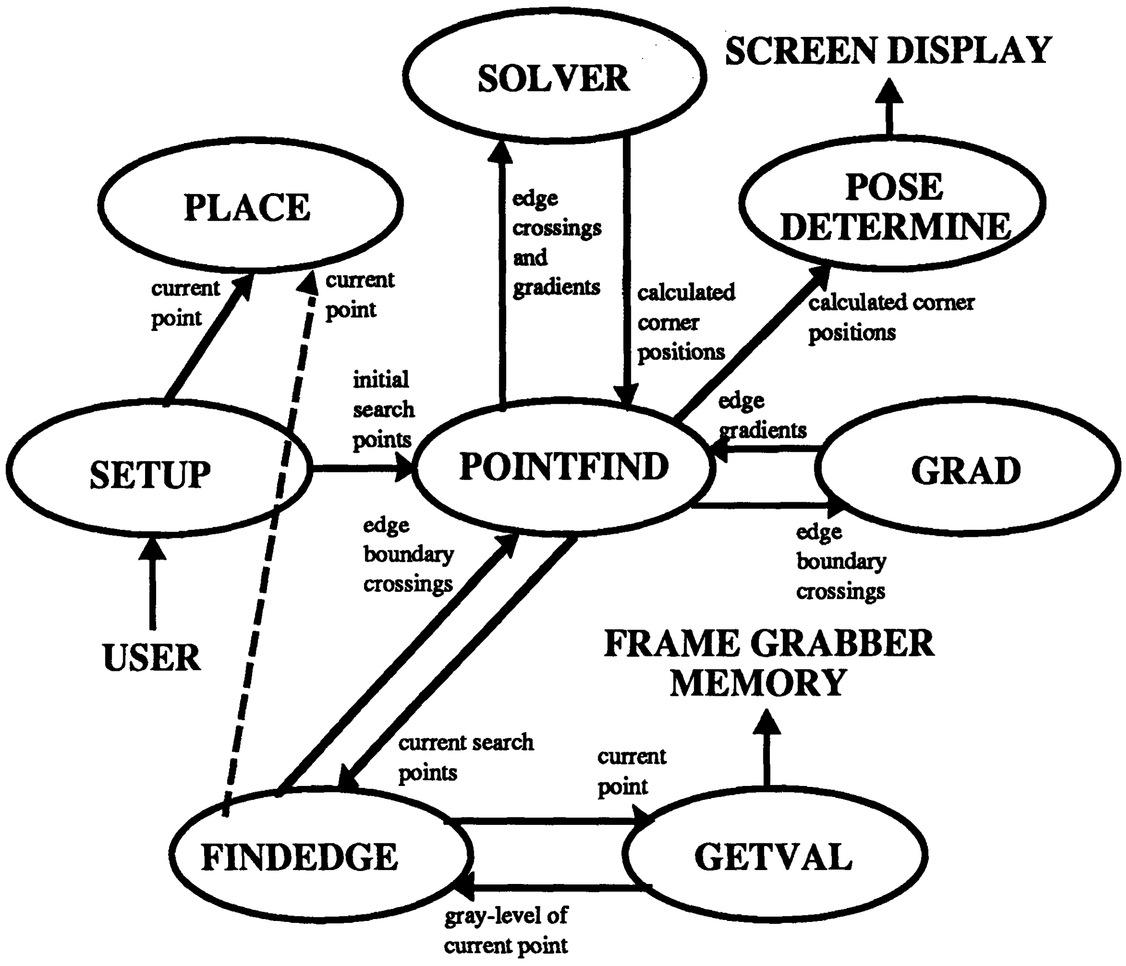


Figure 5.5.1.1) The functional units and the data paths used in the vision software

FINDEDGE returns the boundary crossings of the image edges which are passed via POINTFIND to the edge gradient finding function GRAD. SOLVER then utilizes the data provided by FINDEDGE and GRAD to solve the equations of the lines specified by the gradients and edge crossings. Assuming that the equations are the ones that describe the edges i.e. provided that the edge labeling was correct, the locations of the corners of the image will be returned. These points are then input into a selected pose determining routine.

Chapter 6

System Experiments

6.1 Introduction

This chapter first presents the performance of the basic vision system, i.e. the edge crossing, feature point locating, and feature point tracking algorithms detailed in chapter three. The chapter then describes the performance of the pose determination algorithms described in chapter two. Performance is evaluated using two versions of the software : one in which the edge crossing, feature point determination and pose determination algorithms are tested on a static image and another in which all but the pose determination algorithms are tested on a time varying image. Time varying images were generated by varying the camera's look angles. The calculation of pose was not carried out using the dynamic images due to difficulties in measuring the look angles when the camera was being moved.

For the static image tests, the performance was assessed using a series of images which were stored on disk. Dynamic tests were carried out using a series of test video images of the target and also by manually rotating the camera about the pan, tilt and swing axes.

6.2 Feature Location System Evaluation

6.2.1 Static Tests

6.2.1.1 Edge Finding Performance

The edge crossing algorithms *"Roughsearch"* and *"Finesearch"*, implemented in function *"findedge"* (see Appendix A.3) performed satisfactorily. For the static tests, square markers were placed on the screen at the calculated locations of the edge crossings so that they could be seen by the operator. Given initial search points that are located at distances less than the magnitude of the smaller search vector $\mathbf{r}_2$, *findedge* always finds four edge crossings in the vicinity of each of the feature points. The function was very fast and located the required number of points for the line solving algorithm within 0.03 seconds. However, problems occurred when the target was in certain configurations.

When the image was as shown in figure 6.2-1), the algorithm labeled the edge crossings incorrectly. This problem arose when the end of the search vector $\mathbf{r}_1$ was initially projected into an area of opposite intensity to the area to which the end of $\mathbf{r}_2$ was projected. For this case, the labeling for the corner was as shown in figure 6.2-2).

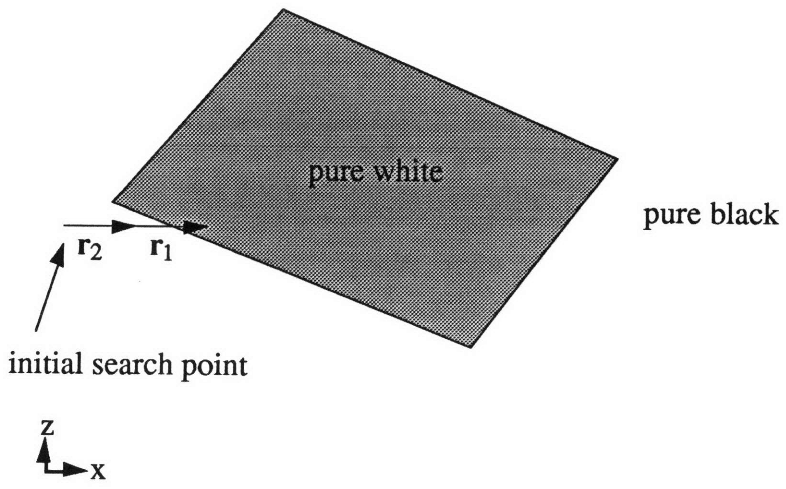


Figure 6.2-1) A configuration susceptible to incorrect labeling of edge boundaries

This problem could easily be solved by appropriately placing the point from which the search vectors are projected. This may be achieved by choosing the initial search point such that the intensities at the end of the vectors are equal.

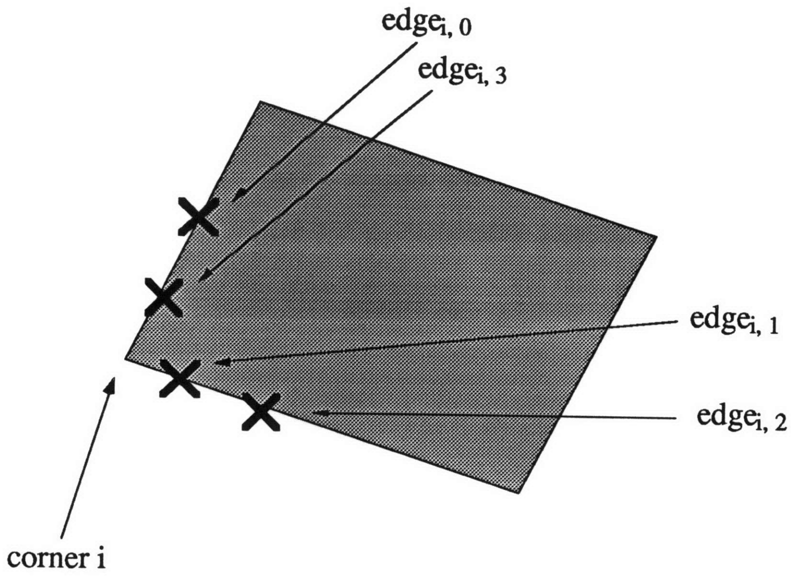


Figure 6.2-2) Incorrect labeling as a result of the image configuration as shown in figure 6.2-1)

6.2.1.2 Feature Point Location

When the edge crossings are labeled correctly, the program *pointfind* always located the feature points to within a few pixels of the corners, as perceived on the screen by the operator. However, when the function *findedge* used the incorrect labeling due to the problem outlined in section 6.2.1.1, the feature point was placed incorrectly but always in the vicinity of the correct corner. This problem occurred due to the line solving algorithm solving for the incorrect pair of lines. This case is shown in figure 6.2-3).

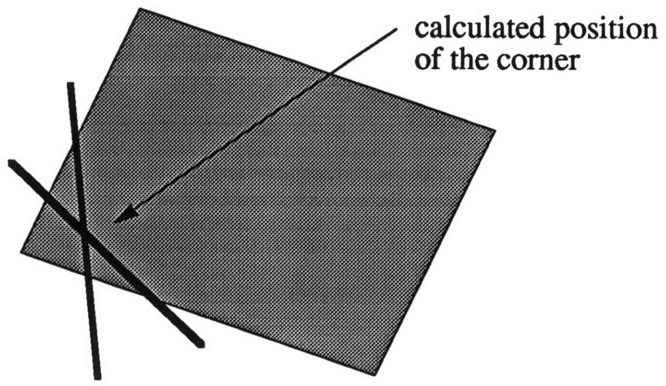


Figure 6.2-3) Calculated position of corner due to labeling as shown in figure 6.2-2)

Another problem of point placement occurred even when the crossings were labeled correctly. When integer arithmetic was used all the variables in the line intersection solving routine, accuracy was lost. It was observed that the algorithm in this case calculated the correct value in the x direction (figure 6.2-4) but the incorrect z position. This occurred because the z position was derived from the previously calculated x position. It was found that even though the coordinates of the corner should be integer values, since they refer to the pixel position, it is necessary to use high precision variables for all the intermediate calculations and to convert to integer values only when the final answers are required. In future applications, floating point variables should be used throughout only until a final result is required.

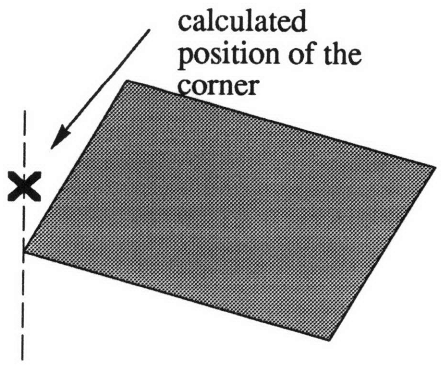


Figure 6.2-4) Misplacement of a corner in the z-direction if integer arithmetic is used in the intermediate calculations

6.2.2 Dynamic Tests

6.2.2.1 Edge Crossing and Feature Point Detection

When feature point related information was evaluated, the edge crossing were not displayed as for the static tests. Since writing to the screen takes a significant amount of time, causing a significant reduction in frame update rate, only the calculated positions of the feature points were displayed. Therefore, the ability to find and correctly label the edge crossings was indirectly assessed by observing the calculated positions of the corners.

6.2.2.2 Stationary Camera Tests

For most configurations, the feature points were placed in the correct vicinity of the corners. A slight wandering of the calculated positions was observed however, even though no target or camera angles were changed.

A time-varying "weave" pattern is observed on digitized images. Upon detailed inspection of the weave effect, it was observed that it causes calculated feature point positions to wander. It was also noted that the interference caused feature positions to change mainly in the z-direction. The wavering of the calculated feature points was probably due to changes on the order of a few pixels in the edge crossing positions. This could cause the calculated position of the feature points to change from frame to frame. This effect was examined in depth since it could conceivably cause instabilities in a pose determining algorithm.

The effect of the interference was examined for two particular images : one where the software was expected to locate the feature points correctly (figure 6.2-5a)) and another where it was expected to fail intermittently (figure 6.2-5b)). The software is expected to work correctly for the image type shown in figure 6.2-5a) when the initial search points are at the corners of the image, which was the case after the first frame. This was guaranteed when the images of the target were updated but the target pose and camera angles were unchanged. For both cases, the x and z positions of point $\mathbf{p}_0$ as calculated by the system were examined. Data from one hundred frames was examined with the results displayed in figures 6.2-6a) - 6.2-7b).

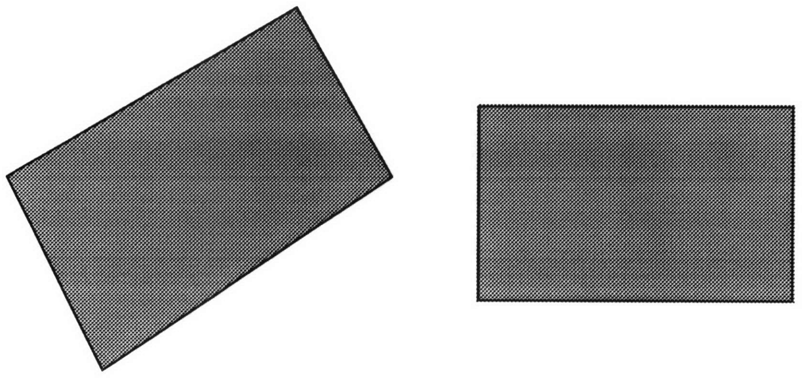


Figure 6.2-5a), 6.2-5b) Favorable and unfavorable orientations for the edge labeling algorithm

Configuration 1

Figure 6.2-6a) shows the x position variation in the calculated corner position when the image is as shown in figure 6.2-5a). It can be seen that for 42% of the frames there was a one-pixel variation about an average of 136 where 64% had this value, with some isolated points (4%) having an x-component with a two pixels deviation from the average.

Figure 6.2-6b) shows the z position variation in the calculated corner position for the same case. The percentage of points one pixel away from the average z position (39%) was less than that for the corresponding x average position. There was also a greater percentage of points located two pixels from the average (17%).

Configuration 2

Figure 6.2-7a) shows the x position variation when the image is as shown in figure 6.2-5b). It is immediately observed that there is no one pixel variation. However, 19% of the points are misplaced by 26 pixels. Since there is no "one pixel noise" apparent on the graph, it appears as though a one-pixel deviation caused the feature point to be grossly miscalculated.

Figure 6.2-7b) shows the z position for the same case as above. The one-pixel deviation is again exhibited. Even though the average value for this case was 377, the deviations occurred around the 376 value. It appears as though that the z component is more susceptible to this noise than the x component.

When the static version of the point finding program was run several times using a stored image, the same same coordinates were returned every time. The "wandering" effect caused by the "weave noise" was therefore probably due to the interaction of the camera and the frame grabber, since it was only observed when the frame grabber was being continually updated with a fresh image of the stationary target.

6.2.2.2 Overall Feature Point Detection

In general, the corner detection algorithm worked satisfactorily. Overall the algorithm works very well for most configurations with only a small error between the calculated and actual positions of the image corners (as seen on the screen) but has difficulty in reproducing results when the image has edges that are parallel to the edge of the screen.

6.2.2.3 Point Tracking Performance

Since the feature point search algorithm is only able to find the corners in the vicinity of the initial search points, it is necessary to locate these points in the vicinity of the corners. Some method for point tracking is therefore required for a time varying image. In the absence of the knowledge of where the "true" location of the corners are, the ability of the the three point tracking methods to place the initial search points in close proximity to the feature points as calculated by the point finding algorithm is now examined. The performance was assessed for three kinds of motion of the target, achieved by varying the camera angles and the focal length of the lens :

1) Translation	-	achieved by rotating the camera about the pan axis,
2) Rotation	-	achieved by rotating the camera about the swing axis,
3) Zooming	-	achieved by changing the focal length of the lens.

For each of the three methods, the following graphs for each motion were plotted :

1) Calculated x-position as returned from function *solver* (using a tracking method to supply an initial search point) and estimated x-position versus the frame number,

2) Calculated z-position and estimated z-position versus frame number,

3) The magnitude of the error of the calculated and estimated positions.

Due to interference from the video player, the frame grabber contents were frequently corrupted, preventing the playing back of standard image sequences. This necessitated the motions to be performed on line with the camera for each of the tests.

Data Analysis

The results for the various runs appear in the following figures :

method 1	-	figures 6.2-8a) - 6.2-8i)
method 2	-	figures 6.2-9a) - 6.2-9i)
method 3	-	figures 6.2-10a) - 6.2-10i)

Method 1 Results

Figures 6.2-8a) - f) show that the position as calculated by *solver* are tracked well. This is not surprising since the estimated position of the point is merely the previous position of that point. The trace of the estimated position is therefore just a one frame lag version of the calculated position.

The performance of the tracking is better analyzed by observing the tracking error plots, figures 6.2-8g) - 6.2-8i). A maximum tracking error of approximately 40 pixels occurs when the camera is rotated about its optic axis. This is due to *solver* misplacing the feature point.

Method 2 Results

Figures 6.2-9-1a) - 6.2-9f)) appear to show that update method two performs well at feature point tracking. The two peaks displayed in graphs figure 6.2-9a) and 6.2-9b) around the frame number 10 and 20 positions are caused by *solver* failing, therefore causing a large tracking error to occur. However, tracking is reestablished within a few frames. Even though the feature point was misplaced, causing a tracking error of 70 pixels, the error was usually under 10 pixels as Figures 6.2-9g) - 6.2-9i) show.

Method 3 Results

The feature point is again misplaced for the case of the zoom operation. The tracking is generally acceptable, except for the case of the misplacement. The error is around 16 pixels for translation and rotation and 100 pixels for zooming. This large error was again caused by the point location routine failing rather than the point tracking method being grossly inaccurate.

Overall Point Tracking Results

It was found that for search radii magnitudes of 80 and 40 pixels, all three point tracking techniques performed similarly. This should be the case if the tracking errors are less than the size of the smaller search radius. None of the update routines perform adequately when step changes in motion are input. The incorrect positions for the corners are sometimes calculated, with a few frames required for the calculation to settle, and sometimes the point placement routines completely fail to find any of the feature points. This can occur if the image moves by more than the size of the smaller search radii. This problem could be alleviated if the magnitudes of the search radii were changed as the speed of the feature points on the screen changes.

It was also discovered that since the point placement routines did not change the size of the radii according to the size of the target image, *findedge* failed to label the edge crossings correctly, thereby resulting in false positioning of the feature points and sometimes in complete loss of the target. This may easily be solved by positioning the initial search point such that the search vectors are projecting into areas of equal intensity. This will insure that the line intersection solved for is a corner of the rectangle.

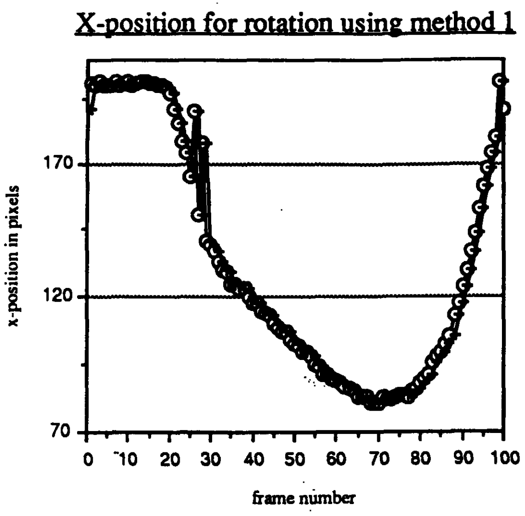


figure 6.2-8a)

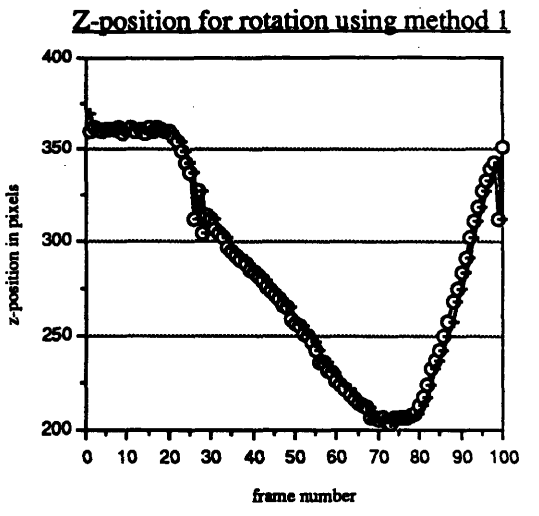


figure 6.2-8b)

figures 6.2-8 a) - 6.2-8d)

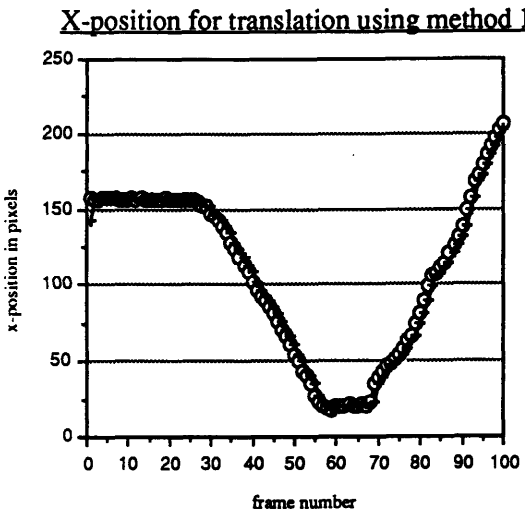


figure 6.2-8c)

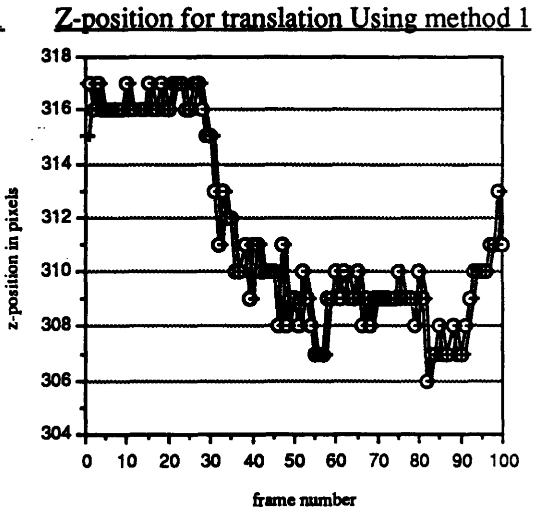


figure 6.2-8d)

Legend

—o— position calculated using feature point finding

—+— position calculated using point tracking

figures 6.2-8e) - f)

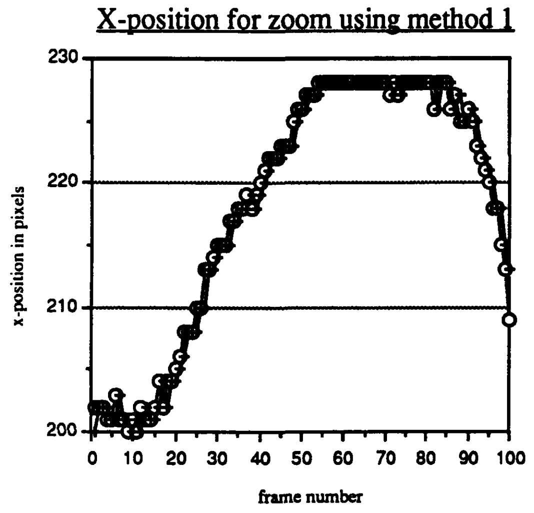


figure 6.2-8e)

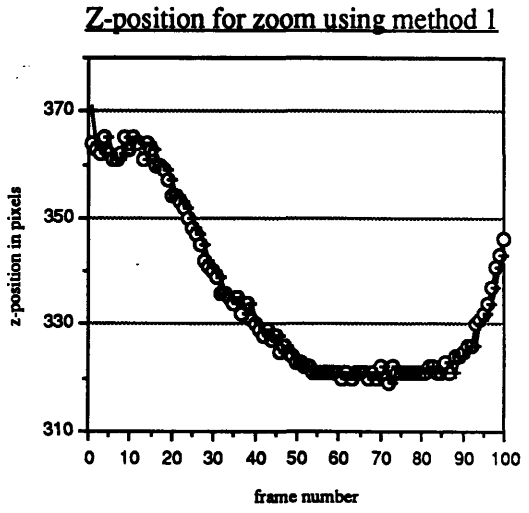


figure 6.2-8f)

Legend

—o— position calculated using feature point finding

—+— position calculated using point tracking

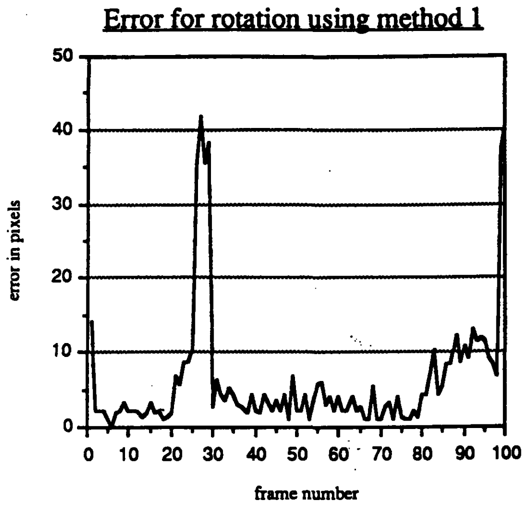


figure 6.2-8g)

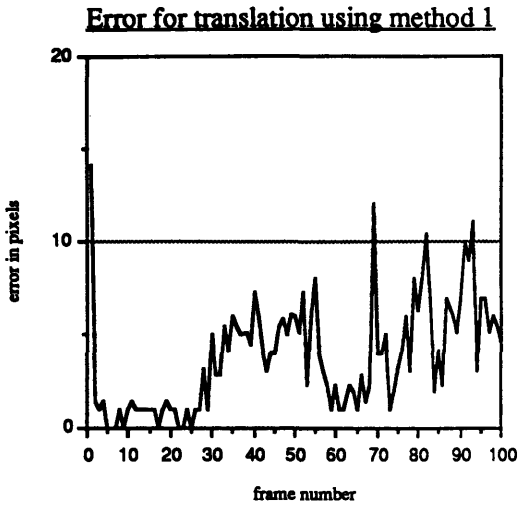


figure 6.2-8h)

figures 6.2-8g) - i)

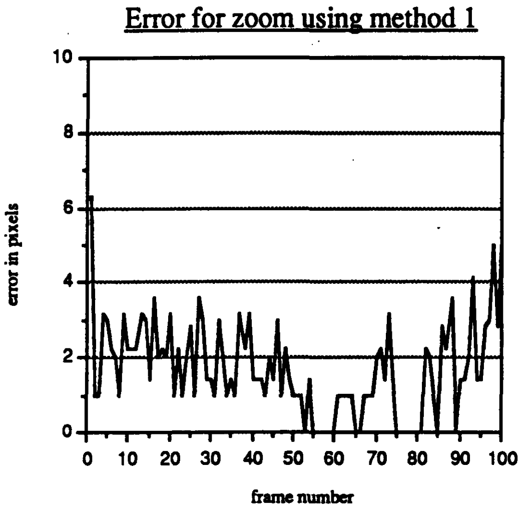


figure 6.2-8i)

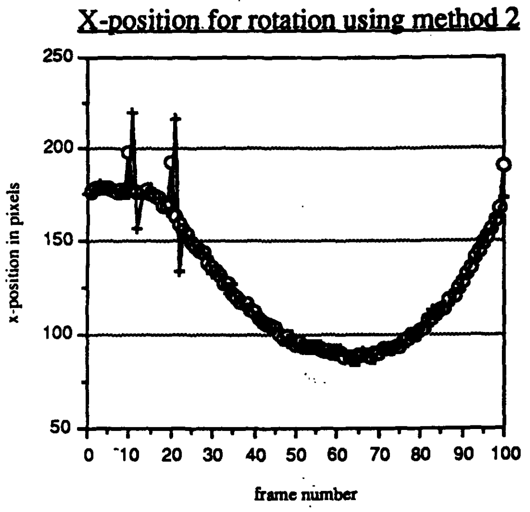


figure 6.2-9a)

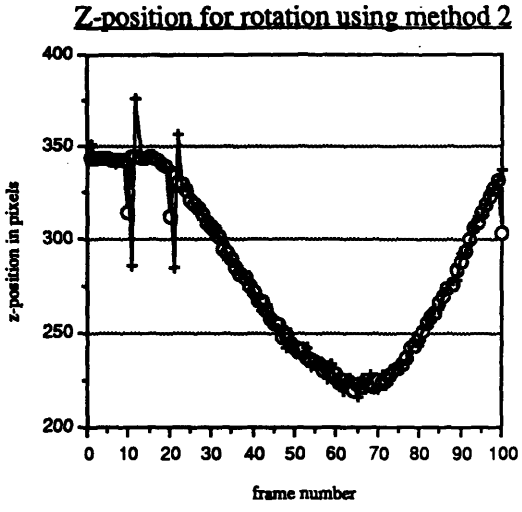


figure 6.2-9b)

figures 6.2-9a) - d)

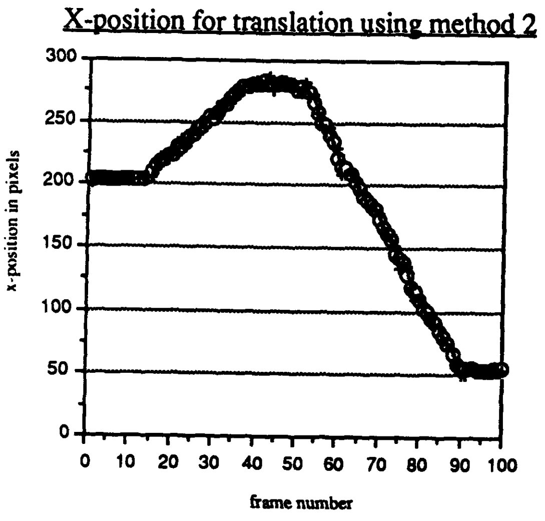


figure 6.2-9c)

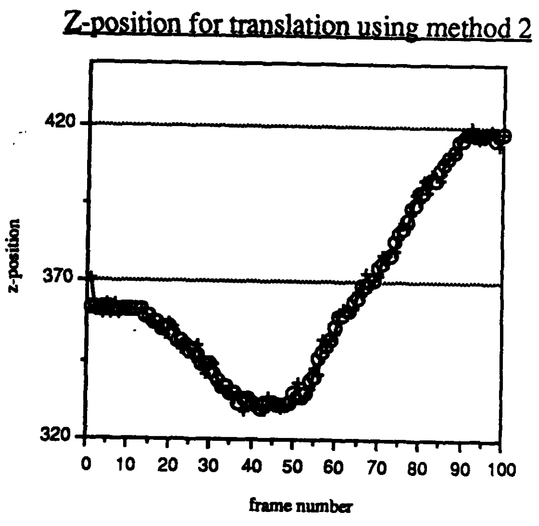


figure 6.2-9d)

Legend

—o— position calculated using feature point finding

—+— position calculated using point tracking

figures6.2.2.3e) - f)

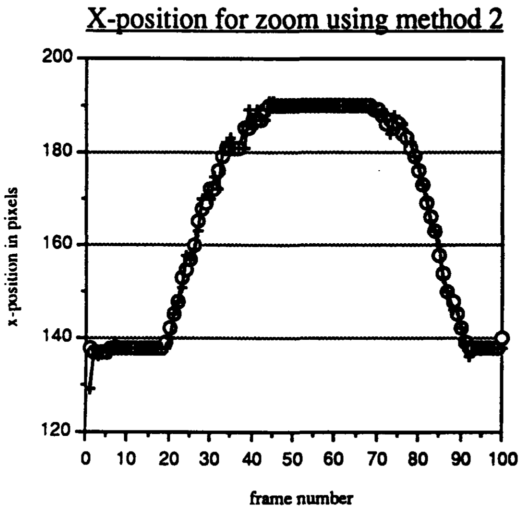


figure 6.2-9e)

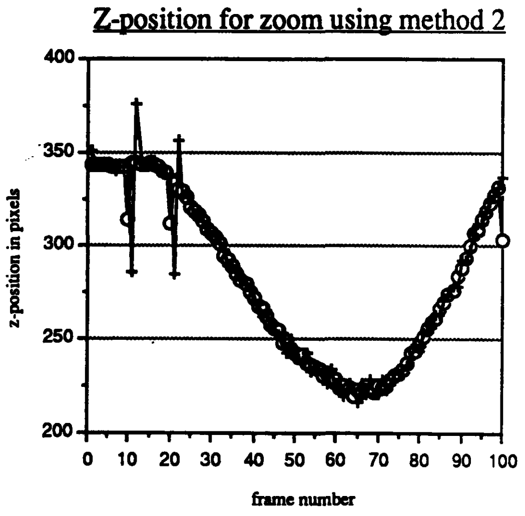


figure 6.2-9f)

Legend

—o— position calculated using feature point finding

—+— position calculated using point tracking

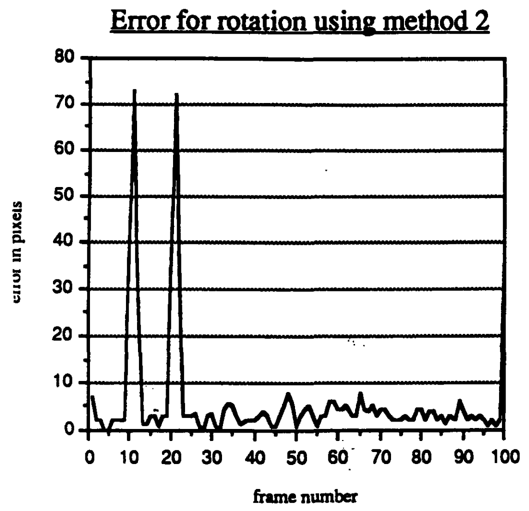


figure 6.2-9g)

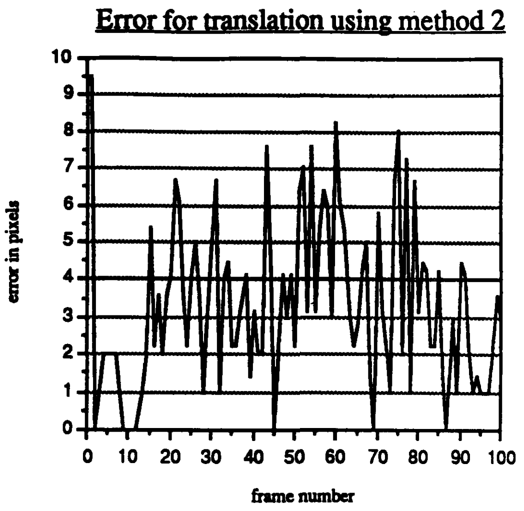


figure 6.2-9h)

figures 6.2.2.3g) - i)

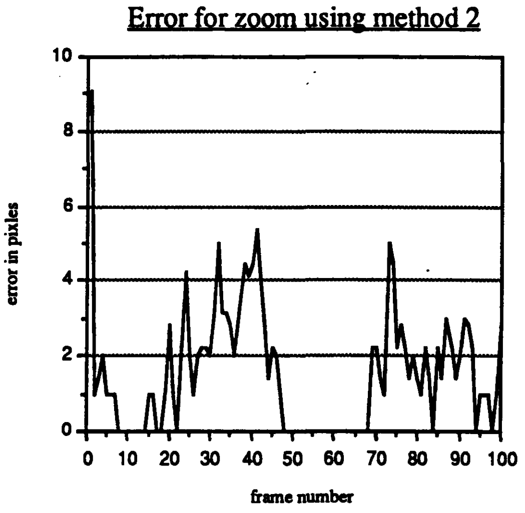


figure 6.2-9i)

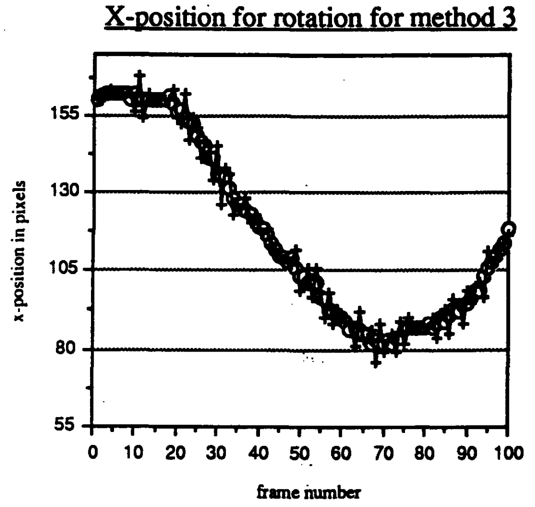


figure 6.2-10a)

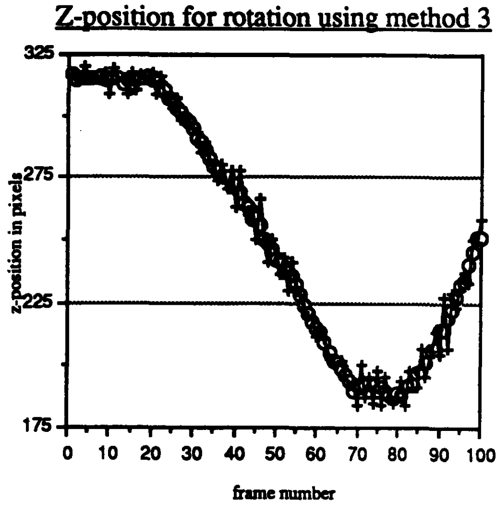


figure 6.2-10b)

figures 6.2-10a) - d)

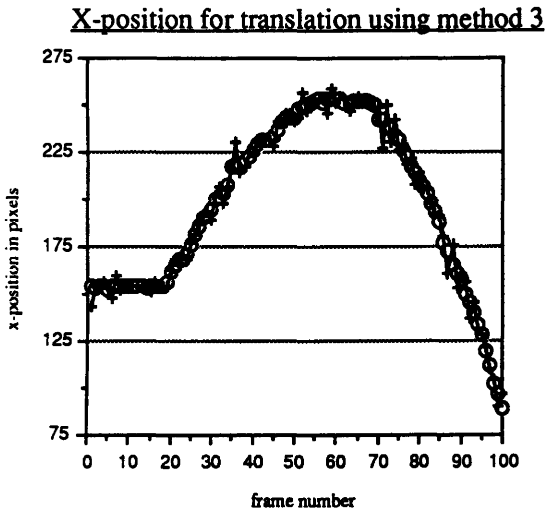


figure 6.2-10c)

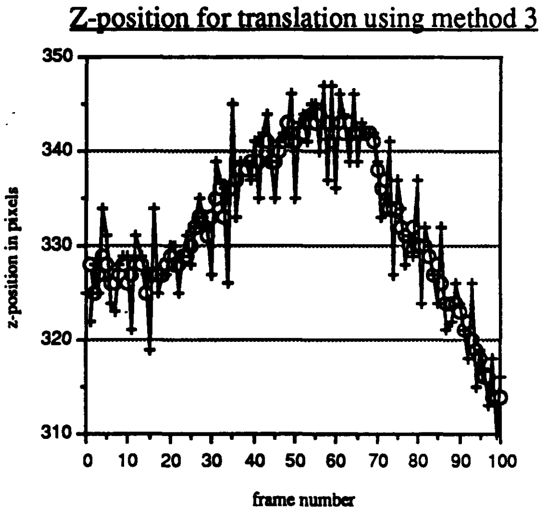


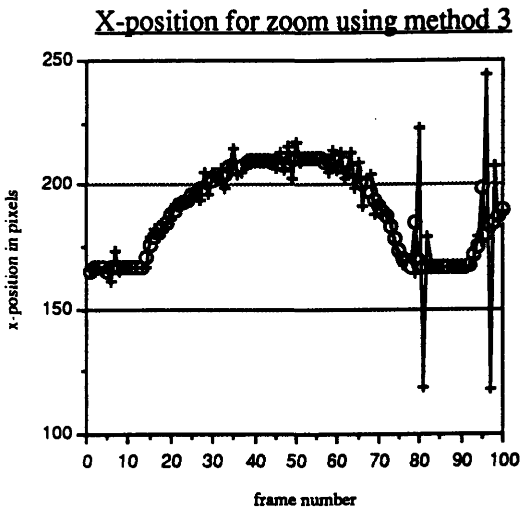
figure 6.2-10d)

Legend

—o— position calculated using feature point finding

—+— position calculated using point tracking

figures 6.2.2.3e) -f)



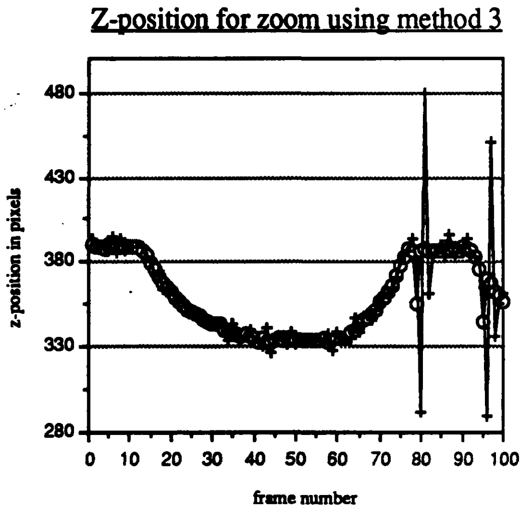


figure 6.2-10e) Legend figure 6.2-10f)

—o—	position calculated using feature point finding
—+—	position calculated using point tracking

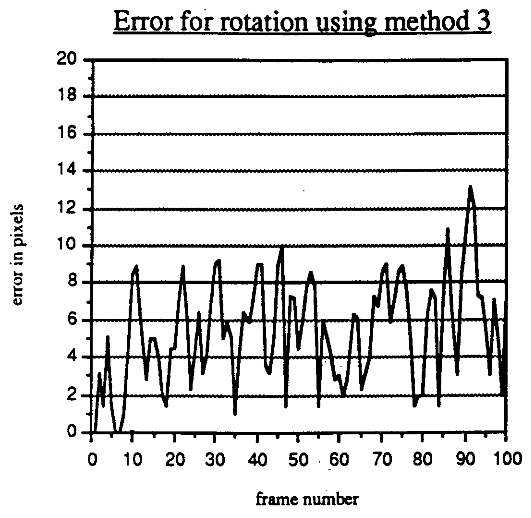


figure 6.2-10g)

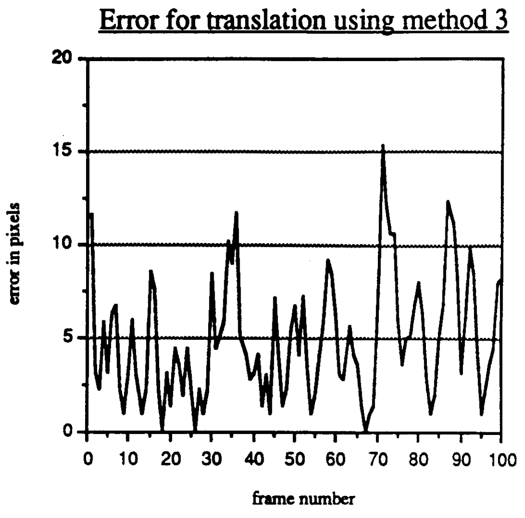


figure 6.2-10h)

figure6.2-10g) - i)

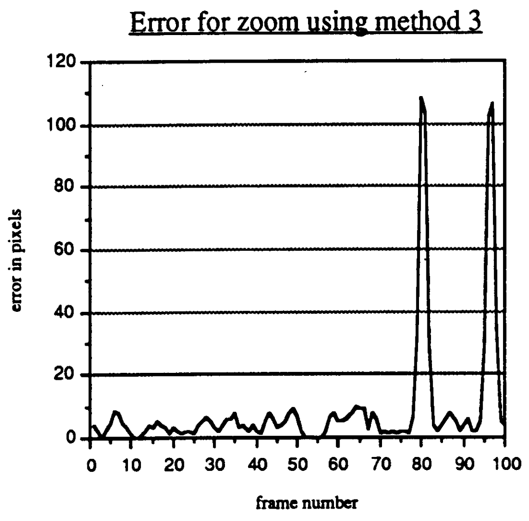


figure 6.2-10i)

6.3 Image Understanding Algorithm Performance

6.3.1. Procedure for Performance Assessment

The performance of the inverse image transformations was evaluated using four standard images as shown in figures 6.3.1-1) - 6.3.1-4). The inverse perspective transformation, which assumes the image is produced using the pin-hole camera model shown in figure 2.2.1-1), was tested first. The inverse orthographic transformation, which assumes that the image is produced by an orthographic projection of the target onto the image plane, was then tested.

The standard images were obtained by camera rotations of $(\phi, \xi, \theta) = (0^\circ, 0^\circ, 0^\circ), (0^\circ, -10^\circ, 0^\circ), (5^\circ, 0^\circ, 0^\circ)$ and $(5^\circ, -10^\circ, 0^\circ)$ (see figure 2.2.2-1)), with the target positioned 1.8 metres away from the camera. The reference target shown in figure 4.1-2) was used for the tests. Due to difficulties in accurate measurement of the desired camera rotations, the pan angle θ was held constant. The other angles could be measured by the deviation of a reference mark on the camera with respect to the horizontal. The was achieved by means of a plumb bob and protractors, mounted perpendicular to the camera's swing and tilt axes. There are two sets of results produced by both routines for each set of camera angles due to the inherent reflective ambiguity of the calculations. Table 6.1 gives the actual camera parameters with which the results from the inverse image transformations should be compared.

6.3.2 The Inverse Perspective Transformation

Obtaining meaningful results from the inverse perspective transformation (IPT) proved to be rather difficult. Even though the function that implemented the IPT was a direct implementation, the results did not appear to correlate to the input camera angles. The results for the four different scenes appear in Tables 6.2 and Table 6.3. These tables show the calculated camera angles, (ϕ, ξ, θ) and and the calculated position, (r_x, r_y, r_z), of the bottom left hand corner of the target, with respect to the camera. It should be noted that for each camera configuration, there are two possible sets of calculated camera angle and target range (see Appendix A.1).

The source of the apparent randomness of the results could be due to the algorithm. The IPT for a rectangle makes use of several division operations. When the image similar to that shown in figures 6.3.1-1) - 4), the divisors of those operations are likely to tend toward zero. Small changes in these numbers may translate into very large errors in the solution for the angles and hence the ranges.

Table 6.1 Actual Camera Parameters

	calculated camera angles			calculated ranges of bottom LH corner		
	ϕ	ξ	θ	r_x	r_y	r_z
image 1(figure 6.3.1-1)	0	0	0	0	1.8	0
image 2(figure 6.3.1-2)	0	-10	0	0	1.8	0
image 3(figure 6.3.1-3)	5	0	0	0	1.8	0
image 4(figure 6.3.1-4)	5	-10	0	0	1.8	0

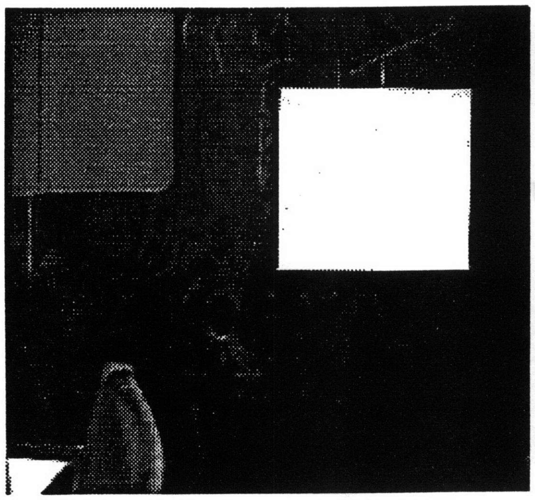


Figure 6.3.1-1: Target image with camera angles (0,0,0).

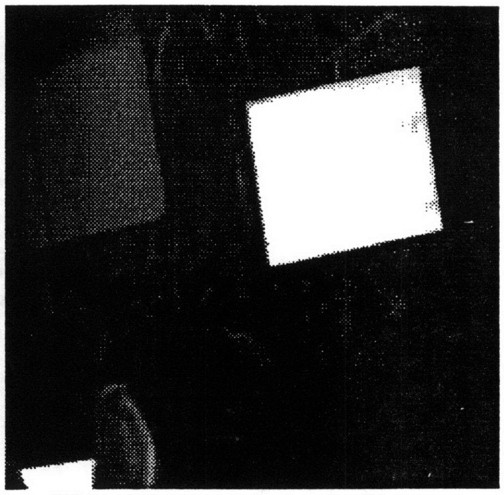


Figure 6.3.1-2: Target image with camera angles (0,-10,0).

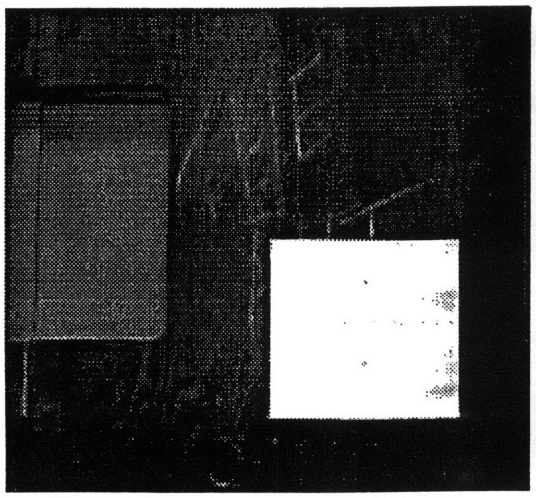


Figure 6.3.1-3: Target image with camera angles (5,0,0).

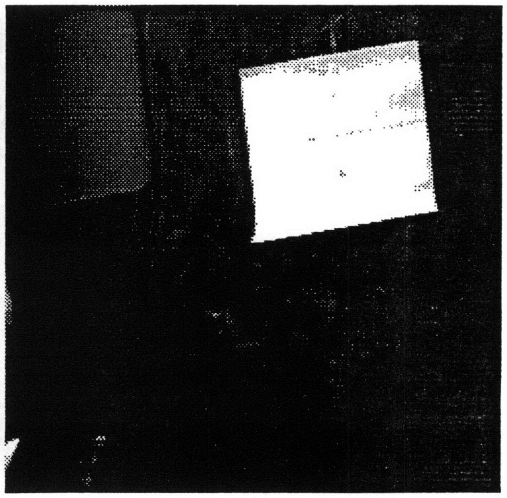


Figure 6.3.1-4: Target image with camera angles (5,-10,0).

6.3.3 The Inverse Orthographic Transformation

The results for the inverse orthographic transformation are also unsatisfactory. However, the results were more consistent than those for the IPT. Notice that the computed range was fairly consistent, with a mean value of 3.6m. This value was still incorrect since the actual range of the target from the lens was 1.8m.

Table 6.2 The Results for the Perspective Transformation Solution Set 1

	calculated camera angles			calculated ranges of bottom LH corner		
	ϕ	ξ	θ	r_x	r_y	r_z
image 1(figure 6.3.1-1)	-10.6	133.2	-40.4	2.0	2.3	-0.6
image 2(figure 6.3.1-2)	-9.1	69.4	-9.3	0.5	3.3	-0.5
image 3(figure 6.3.1-3)	-8.0	90.3	-20.5	1.0	2.5	-0.5
image 4(figure 6.3.1-4)	-7.2	131.2	-45.5	2.0	1.8	-0.5

Table 6.2 The Results for the Perspective Transformation Solution Set 2

	calculated camera angles			calculated ranges of bottom LH corner		
	ϕ	ξ	θ	r_x	r_y	r_z
image 1 (figure 6.3.1-1)	-10.9	-67.8	-32.8	1.5	2.3	-0.5
image 2 (figure 6.3.1-2)	-8.1	-46.2	-48.3	2.2	2.0	-0.4
image 3 (figure 6.3.1-3)	0.0	-89.0	176.3	-0.2	-2.8	0.2
image 4 (figure 6.3.1-4)	-0.5	-10.7	-55.7	2.79	2.1	-0.1

Table 6.3 The Results for the Orthographic Projection Solution Set 1

	calculated camera angles			range
	ϕ	ξ	θ	r
image 1 (figure 6.3.1-1)	12.3	0.5	7.1	3.6
image 2 (figure 6.3.1-2)	13.8	-10.0	3.0	3.4
image 3 (figure 6.3.1-3)	20.9	0.2	0.5	3.6
image 4 (figure 6.3.1-4)	-13.7	-8.9	5.8	3.6

Table 6.4 The Results for the Orthographic Projection Solution Set 2

	calculated camera angles			range
	ϕ	ξ	θ	r
image 1 (figure 6.3.1-1)	-10.2	0.5	-6.3	3.6
image 2 (figure 6.3.1-2)	-14.5	-10.2	-10.4	3.6
image 3 (figure 6.3.1-3)	-10.3	0.25	-0.9	3.6
image 4 (figure 6.3.1-4)	13.3	-8.9	-2.6	3.6

There appears to be some coupling of the computed angles since their values change, even when the corresponding camera angle was held constant. This is to be expected since the IOT does not return the camera look angles but rather the angles through which the target must be rotated through to produce a scaled, coplanar version of the image on the screen. The relationship between the target Euler angles and the camera look angles is a function of the geometry of the camera tripod mounting. This information was not available and hence the problem was not further pursued in this project.

Chapter 7

Conclusions

7.1 Discussion and Recommendations

A vision system has been developed that will serve as a framework for the implementation of various vision algorithms. The system was written in a highly modular form so that modifications could easily be made. Provision was made for different algorithms to be simply "plugged" into the system. However, the system is restricted to algorithms that are based on the feature points of an image of a planar target. The philosophy that drove this decision was that such shapes occur abundantly in man made environments. Since algorithms that require the feature points of an image were only explored, a rectangular target was chosen due to its well defined vertices. Many such algorithms only require three feature points for pose calculation, the use of a rectangle may therefore provide a degree of redundancy in the calculations.

The edge boundary finding routine is quite fast, taking on the order of 0.03 seconds. However, it is not able to to tell when the labeling is incorrect. This condition happens when the the ends of the search vectors are positioned within areas of opposite intensity. This results in the misplacement of the corners of the rectangle. For a time varying image, the system is able to recover the correct points as the target was rotated from an unfavorable to a favorable orientation, even though the points might be originally misplaced. Another deficiency in the algorithm is that if one or more of the points are grossly misplaced, possibly by excessive image motions, they may "migrate" to their incorrect corners. In this case, the program will report successful point placement, even though the points may be located at their incorrect positions.

Even though there are many drawbacks with the labeling and point finding algorithms, they are extremely robust. It is possible for part of the image to be out of the range of the screen and yet still have all the corners to be located.

The performance was affected by the quality of the image produced by the frame grabber. It produced a noisy image, probably due to a slight difference in synchronization between the camera and the monitor. For a static image this caused the calculated position of the corners to deviate from frame to frame.

The edge finding algorithm can be significantly improved. For example, the algorithm finds four points on each edge, even though only two are required to locate the corners. This excess number could be used in a least squares algorithm to find the best-fit line between the points. An alternative is to use only two of the edge boundary crossings. This certainly will increase the speed of operation by a factor of two over the current system but will not have the accuracy of the method.

For the dynamic versions of the software, the tracking system should use a simple update method where the beginning of the subsequent search is the current position of the feature point. The more complex methods of point update are more prone to error. This is possibly due to the fact that the time difference between frames is not constant. For cases where the inter-frame period is constant, anticipatory tracking schemes may be more successful. The frame rate variation is most likely caused by some target orientations being more favorable for the edge boundary crossing routine. A constant frame rate was attempted by accessing the computer clock. This method proved unreliable since the various operating system calls also access the clock. This prevented frames being updated at equal increments in time. It is suggested that a device dedicated to providing an adequate clock signal be used.

The image transformations tested did not perform satisfactorily in their current form. The inverse perspective transformation of the rectangle provides a closed form result enabling the calculation of the camera look angles and target position. However, to implement these equations within a program is clumsy and provides no level of redundancy for a possible least squares approximation to a set of possible solutions. It is suggested that this method not be used for future vision research for space robots.

Since the alignment method requires only three points and the target used was a rectangle, a degree of redundancy was available in the calculation of the rotation matrix

R. For a rectangle, there are four sets of three points that could each be used to calculate the components of **R**. The problem is solved for each set of points and a least squares routine is implemented to find the best fit for the matrix components.

The components of the linear transformation **L** may be solved for the following sets of points : $\{m_0, m_1, m_2\}$, $\{m_0, m_1, m_3\}$, $\{m_0, m_2, m_3\}$ and $\{m_1, m_2, m_3\}$. For each set, a solution $\mathbf{L}_i$ (i = 1, 2, 3, 4) is generated. A least squares solution is found in the following manner :

1) Given the values $l_{1,1_i}, l_{1,2_i}, l_{2,1_i}, l_{2,2_i}$ (i = 1, 2, 3, 4), a matrix A may be set up as :

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}$$

The overdetermined system of equations $\mathbf{Ax} = \mathbf{b}$ is to be solved. The vectors b are defined as :

$$\mathbf{b}_{1,1} = \begin{bmatrix} l_{1,1_1} \\ l_{1,1_2} \\ l_{1,1_3} \\ l_{1,1_4} \end{bmatrix}, \mathbf{b}_{1,2} = \begin{bmatrix} l_{1,2_1} \\ l_{1,2_2} \\ l_{1,2_3} \\ l_{1,2_4} \end{bmatrix}, \mathbf{b}_{2,1} = \begin{bmatrix} l_{2,1_1} \\ l_{2,1_2} \\ l_{2,1_3} \\ l_{2,1_4} \end{bmatrix},$$

$$\mathbf{b}_{2,2} = \begin{bmatrix} l_{2,2_1} \\ l_{2,2_2} \\ l_{2,2_3} \\ l_{2,2_4} \end{bmatrix}.$$

2) For each b vector, solve :

$$\mathbf{x} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{b}, \mathbf{x} \in \Re^2$$

The solutions should give equations of lines parallel to the ordinate axes.

3) In case there is a slight slope in the answers, the mid-point of the lines should chosen to be the components of **L** :

$$l = \mathbf{x}_{1,1} + 2.5\ \mathbf{x}_{2,1}.$$

Care should be taken with the above technique since two solutions are generated when the alignment transformation is applied. A means of identifying and applying the least squares to the two solutions generated by the alignment transformation should be developed.

It may also be possible to develop a pose determining algorithm that is not dependent on any particular image feature points. This would relieve the processor of complex edge labeling checks and point tracking thus enabling it to carry out other important tasks such as calculating control signals from the data generated by the vision software. It is not anticipated that the present system will be able to operate in unison with a control scheme on the same processor. Rather it is envisioned that the a processor will be devoted to vision tasks, updating the state of the environment (or plant) and another utilizing this information to calculate the appropriate control signals.

Immediate work should concentrate on obtaining the relevant data in order to implement the orthographic transformation correctly. For "dry land" tests, this involves obtaining a camera mounting such that the tilt, pan and swing angles may be changed independently and be accurately measured.

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

Haralick, R.M. 1989 Determining Camera Parameters from the Perspective Projection of a Rectangle Pattern Recognition, Vol. 22, No. 3, pp 225-230

Huttenlocher D. P. 1988 Three-Dimensional Recognition of Solid Objects from a Two-Dimensional Image, Ph.D Thesis, MIT Artificial Intelligence Laboratory, April 1988

Hwang, V.S.S. 1989 Tracking Feature Points in Time-Varying Images Using an Opportunistic Selection Approach, Pattern Recognition, Vol. 22, No. 3, pp. 247-256, 1989

Lim, J.S. 1990 Two dimensional Signal and Image Processing, Prentice Hall Signal Processing Series, Prentice Hall, 1989

PCVISION 1984 PCVISION Frame Grabber Manual 1984, Imaging Technology Incorporated

Plastock, R.A., Kalley, G. 1986 Theory and Problems of Computer Graphics, Schaum's Outline Series, McGraw-Hill 1986

Rowley, V.M. 1989 Effects of Stereovision and Graphics Overlay on a Teleoperator Docking Task, SM Thesis, MIT Department of Aeronautics, 1989

Rummel, P. 1989 Applied Robot Vision : Combining Workpiece Recognition and Inspection, Machine Vision for Inspection and Measurement, pp 203-221, Perspectives in Computing, Academic Press, 1989

Spofford, J. 1986 3-D Position and Attitude Measurement for Underwater Vehicles, MIT Space Systems Laboratory Report SSL#21-86, December 1986

Appendix A.1

Theoretical Background

A.1.1 Theoretical Development of the Perspective Transformation

The complete development of the pose determination of a rectangle using the perspective transformation is given below.

From Eqs. 2.2) and 2.4) repeated below :

$$\left\{ \begin{bmatrix} x \\ y \\ z \end{bmatrix} \middle| \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \lambda \begin{bmatrix} x' \cos\theta - f \sin\theta \cos\phi + z' \sin\theta \sin\phi \\ x' \sin\theta + f \cos\theta \cos\phi - z' \sin\phi \\ f \sin\phi + z' \cos\phi \end{bmatrix}, \begin{bmatrix} x' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\xi & -\sin\xi \\ \sin\xi & \cos\xi \end{bmatrix} \begin{bmatrix} x^* \\ z^* \end{bmatrix} \right\}$$

A1-1)

$$p_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}, \qquad p_2 = \begin{bmatrix} x_1 + W \\ y_1 \\ z_1 \end{bmatrix},$$

A1-2)

$$p_3 = \begin{bmatrix} x_1 \\ y_1 + L \\ z_1 \end{bmatrix}, \qquad p_4 = \begin{bmatrix} x_1 + W \\ y_1 + L \\ z_1 \end{bmatrix}$$

we have for some λ_1, λ_2, λ_3, λ_4 :

$$\lambda_1 \begin{bmatrix} x' \cos\theta - f \sin\theta \cos\phi + z' \sin\theta \sin\phi \\ x' \sin\theta + f \cos\theta \cos\phi - z' \sin\phi \\ f \sin\phi + z' \cos\phi \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \qquad \text{A1-3a)}$$

$$\lambda_2 \begin{bmatrix} x' \cos\theta - f \sin\theta \cos\phi + z' \sin\theta \sin\phi \\ x' \sin\theta + f \cos\theta \cos\phi - z' \sin\phi \\ f \sin\phi + z' \cos\phi \end{bmatrix} = \begin{bmatrix} x_1 + W \\ y_1 \\ z_1 \end{bmatrix} \qquad \text{A1-3b)}$$

$$\lambda_3 \begin{bmatrix} x' \cos\theta - f \sin\theta \cos\phi + z' \sin\theta \sin\phi \\ x' \sin\theta + f \cos\theta \cos\phi - z' \sin\phi \\ f \sin\phi + z' \cos\phi \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 + L \\ z_1 \end{bmatrix} \qquad \text{A1-3c)}$$

$$\lambda_4 \begin{bmatrix} x' \cos\theta - f \sin\theta \cos\phi + z' \sin\theta \sin\phi \\ x' \sin\theta + f \cos\theta \cos\phi - z' \sin\phi \\ f \sin\phi + z' \cos\phi \end{bmatrix} = \begin{bmatrix} x_1 + W \\ y_1 + L \\ z_1 \end{bmatrix} \qquad \text{A1-3d)}$$

Equations A1-3a)-d) can now be solved for θ, ϕ and ξ with λ_1, λ_2, λ_3, λ_4, x_1, y_1, z_1, W and L unknown. The first and third components of the right hand sides of Eqs. A1-3a), A1-3c) and A1-3d) are the same and can be used to show that

$$\cos\theta \sin\phi\, f\,(x_1' - x_3') + \sin\theta\, f\,(z_1' - z_3') + \cos\theta \cos\phi\,(x_1' z_3' - x_3' z_1') = 0 \qquad \text{A1-4)}$$

$$\cos\theta \sin\phi\, f\,(x_2' - x_4') + \sin\theta\, f\,(z_2' - z_4') + \cos\theta \cos\phi\,(x_2' z_4' - x_4' z_2') = 0 \qquad \text{A1-5)}$$

Multiplying A1-4) by $(z_2' - z_4')$ and A1-5) by $(z_1' - z_3')$ and subtracting yields :

$$\begin{aligned} &\cos\theta \sin\phi\, f\,[(x_1' - x_3')(z_2' - z_4') - (x_2' - x_4')(z_1' - z_3')] \\ &+ \cos\theta \cos\phi\,[(x_1' z_3' - x_3' z_1')(z_2' - z_4') \\ &- (x_2' z_4' - x_4' z_2')(z_1' - z_3')] = 0 \end{aligned} \qquad \text{A1-6)}$$

Dividing Eq.A1-6) by cos θ and solving for φ gives :

$$\phi = \tan^{-1}\frac{(x_2' z_4' - x_4' z_2')(z_1' - z_3') - (x_1' z_3' - x_3' z_1')(z_2' - z_4')}{f\,[(x_1' - x_3')(z_2' - z_4') - (x_2' - x_4')(z_1' - z_3')} \qquad \text{A1-7)}$$

Since φ has an ambiguity of 180°, the correct value can be chosen from that value between ± 90°. Values outside this range means the camera is looking at the hemisphere behind it.

Multiplying Eq.A1-4) by $(x_2' - x_4')$ and A1-5) by $(x_1' - x_3')$ and subtracting gives :

$$\begin{aligned} &\sin\theta\, f\,[(z_1' - z_3')(x_2' - x_4') - (z_2' - z_4')(x_1' - x_3')] \\ &+ \cos\theta \cos\phi\,[(x_1' z_3' - x_3' z_1')(x_2' - x_4') \\ &- (x_2' z_4' - x_4' z_2')(x_1' - x_3')] = 0 \end{aligned} \qquad \text{A1-8)}$$

Eq.A1-8) results in :

$$\theta = \tan^{-1}\frac{\cos\phi\,(x_1' z_3' - x_3' z_1')(x_2' - x_4') - (x_2' z_4' - x_4' z_2')(x_1' - x_3')}{f\,[(x_1' - x_3')(z_2' - z_4') - (x_2' - x_4')(z_1' - z_3')} \qquad \text{A1-9)}$$

Again, the value between ± 90°should be selected. Since

$$\begin{bmatrix} x_i^* \\ z_i^* \end{bmatrix} = \begin{bmatrix} \cos\xi & \sin\xi \\ -\sin\xi & \cos\xi \end{bmatrix} \begin{bmatrix} x_i' \\ z_i' \end{bmatrix}, \quad i = 1, 2, 3, 4 \qquad \text{A1-10)}$$

it can be seen from Eqs A1-7) and A1-9) that once ξ is known, θ and φ may be determined. As the second and third components of Eqs A1-3a) and A1-3b) are the same and the second and third components of Eqs A1-3c) and A1-3d) are the same, an alternative equation for φ may be found :

$$\phi = \tan^{-1}\frac{(x_1' z_2' - x_2' z_1')(z_3' - z_4') - (x_3' z_4' - x_4' z_3')(z_1' - z_2')}{f\,[(x_3' - x_4')(z_1' - z_2') - (x_1' - x_2')(z_3' - z_4')} \qquad \text{A1-11)}$$

Eqs A1-9) and A1-11) imply :

$$\frac{(x_2' z_4' - x_4' z_2')(z_1' - z_3') - (x_1' z_3' - x_3' z_1')(z_2' - z_4')}{f\,[(x_1' - x_3')(z_2' - z_4') - (x_2' - x_4')(z_1' - z_3')} = \frac{(x_1' z_2' - x_2' z_1')(z_3' - z_4') - (x_3' z_4' - x_4' z_3')(z_1' - z_2')}{f\,[(x_3' - x_4')(z_1' - z_2') - (x_1' - x_2')(z_3' - z_4')} \qquad \text{A1-12)}$$

From Eq.1), which is restated below

$$\begin{bmatrix} x_i^* \\ z_i^* \end{bmatrix} = \begin{bmatrix} \cos\xi & \sin\xi \\ -\sin\xi & \cos\xi \end{bmatrix} \begin{bmatrix} x_i' \\ z_i' \end{bmatrix}, \quad i = 1, 2, 3, 4 \qquad \text{A1-13)}$$

the following property is noticed :

$$\begin{aligned} x_2' z_4' - x_4' z_2' &= (x_2^* \cos\xi - z_2^* \sin\xi)(x_4^* \sin\xi + z_4^* \cos\xi) \\ &\quad - (x_4^* \cos\xi - z_4^* \sin\xi)(x_2^* \sin\xi + z_2^* \cos\xi) \\ &= (x_2^* z_4^* - x_4^* z_2^*)\cos^2\xi + (x_2^* z_4^* - x_4^* z_2^*)\sin^2\xi \\ &= (x_2^* z_4^* - x_4^* z_2^*) \end{aligned} \qquad \text{A1-14)}$$

The other denominators of Eq.A1-12 are similarly rotationally invariant. Eq.12) therefore becomes :

$$\frac{(x_2^*z_4^*-x_4^*z_2^*)[\sin\xi(x_1^*-x_3^*)+\cos\xi(z_1^*-z_3^*)-(x_1^*z_3^*-x_3^*z_1^*)\,[\sin\xi(x_2^*-x_4^*)+\cos\xi(z_2^*-z_4^*)]}{f\,[(x_1^*-x_3^*)(z_2^*-z_4^*)-(x_2^*-x_4^*)(z_1^*-z_3^*)} = \frac{(x_1^*z_2^*-x_2^*z_1^*)[\sin\xi(x_3^*-x_4^*)+\cos\xi(z_3^*-z_4^*)-(x_3^*z_4^*-x_4^*z_3^*)\,[\sin\xi\,(x_1^*-x_2^*)+\cos\xi(z_1^*-z_2^*)]}{f\,[(x_1^*-x_3^*)(z_2^*-z_4^*)-(x_2^*-x_4^*)(z_1^*-z_3^*)}$$

i.e. the above states that a linear combination of the sine and cosine of ξ. ξ is therefore given by:

$$\xi = \tan^{-1} \frac{[\,A(z_1^* - z_3^*) - B(z_2^* - z_4^*) - C(z_1^* - z_2^*) + D(z_3^* - z_4^*)]}{[A(x_1^* - x_3^*) - B(x_2^* - x_4^*) - C(x_1^* - x_2^*) + D(x_3^* - x_4^*)]} \qquad \text{A1-15)}$$

where

$$A = \frac{(x_2^* z_4^* - x_4^* z_2^*)}{E} \qquad \text{A1-16a)}$$

$$B = \frac{(x_1^* z_3^* - x_3^* z_1^*)}{E} \qquad \text{A1-16b)}$$

$$C = \frac{(x_3^* z_4^* - x_4^* z_3^*)}{F} \qquad \text{A1-17c)}$$

$$D = \frac{(x_1^* z_2^* - x_2^* z_1^*)}{F} \qquad \text{A1-18d)}$$

$$E = f\,[\,(x_1^* - x_3^*)(z_2^* - z_4^*) - (x_2^* - x_4^*)(z_1^* - z_3^*)\,] \qquad \text{A1-19e)}$$

$$F = f\,[\,(x_1^* - x_2^*)(z_3^* - z_4^*) - (x_3^* - x_4^*)(z_1^* - z_2^*)\,] \qquad \text{A1-20f)}$$

Throughout the project however, the rectangle has been assumed to lie in a $y = y_1$ plane. This is more appropriate since in the nominal orientation, i.e. when all the camera angles are zero, the rectangle is directly ahead. By solving the problem using the equations above but with the rectangle in a $y = y_1$ plane, the coordinate system for the original problem of the rectangle in a $z = z_1$ plane may be thought of as having a prior rotation of $\pm 90^\circ$. The choice of $+90^\circ$ or -90° is depends on keeping the corners of the rectangle in front of the image plane. The solution gives angles θ', ϕ', and ξ'. From these angles, new angles θ, ϕ, and ξ may be determined so that a rotation of θ about the z axis, followed by a rotation of ϕ about the x axis, followed by a rotation of ξ about the y axis gives the same effective rotation about the x axis of ±90°, followed by a rotation about the z axis of θ', followed by a rotation of ϕ' about the x axis, followed by a rotation of ξ' about the y axis.

This procedure reduces to equating the components of the two corresponding rotation matrices. For a prior rotation of +90° about the axis, the components of the following rotation matrices must be equal :

$$\begin{bmatrix} \cos\xi'\cos\theta' + \sin\xi'\sin\phi'\sin\theta' & -\sin\xi'\cos\phi' & \cos\xi'\sin\phi' - \sin\xi'\sin\phi'\cos\theta' \\ -\cos\phi'\sin\theta' & -\sin\phi' & \cos\phi'\cos\theta' \\ -\sin\xi'\cos\theta' + \cos\xi'\sin\phi'\sin\theta' & -\cos\xi'\cos\phi' & -\sin\xi'\sin\theta' - \cos\xi'\sin\phi'\cos\theta' \end{bmatrix}$$

$$= \begin{bmatrix} \cos\xi\cos\theta + \sin\xi\sin\phi\sin\theta & -\cos\xi\sin\theta - \sin\xi\sin\phi\cos\theta & \sin\xi\cos\phi \\ -\cos\phi\sin\theta & \cos\phi\cos\theta & \sin\phi \\ -\sin\xi\cos\theta + \cos\xi\sin\phi\sin\theta & -\sin\xi\sin\theta - \cos\xi\sin\phi\cos\theta & \cos\xi\cos\phi \end{bmatrix}$$

A1-21)

From this, we have :

$$\phi = \sin^{-1}(\cos\phi'\cos\theta') \qquad \text{A1-22a)}$$

$$\theta = \tan^{-1}\frac{\cos\phi'\sin\theta'}{-\sin\phi'} \qquad \text{A1-22b)}$$

$$\xi = \tan^{-1}\frac{\cos\xi'\sin\phi' - \sin\xi'\sin\phi'\cos\theta'}{-\sin\xi'\sin\theta' - \cos\xi'\sin\phi'\cos\theta'} \qquad \text{A1-22c)}$$

A.1.2 Theoretical Development and Calculation of the Inverse Orthographic Transformation

The following results [Huttenlocher D. P. 1988] show that given two sets of three points each in a different plane give the orientation of one plane with respect to another.

Lemma A.1.1 Given three non-colinear points p_{m_i} in the plane and three corresponding points p_{s_i} in the plane where i = 1, 2, 3 then there exists a unique affine transformation :

$$\mathbf{A}\ (\mathbf{x}) = \mathbf{Lx} + \mathbf{b}\ ,\ \mathbf{x} \in \Re \qquad \text{A1-21)}$$

where **L** is a linear transformation and **b** is a translation such that A (p_{m_i}) = p_{s_i}. An affine transformation of a plane is given by :

$$\begin{aligned} x' &= a_1x + b_1y + c_1 \\ y' &= a_2x + b_2y + c_2 \end{aligned} \qquad \text{A1-22)}$$

for any point (x, y). A transform relating the points p_{m_i} and p_{s_i} is given results in two sets of three equations in three unknowns :

$$x'_{s_i} = a_1x_{m_i} + b_1y_{m_i} + c_1 \quad i = 1, 2, 3 \qquad \text{A1-23a)}$$

$$y'_{s_i} = a_2x_{m_i} + b_2y_{m_i} + c_2 \quad i = 1, 2, 3 \qquad \text{A1-23b)}$$

Both sets of equations can be solved if :

$$\begin{vmatrix} x_{m_1} & y_{m_1} & 1 \\ x_{m_2} & y_{m_2} & 1 \\ x_{m_3} & y_{m_3} & 1 \end{vmatrix} \neq 0$$

This means that the points p_{m_i} must not be colinear.

Definition A.1-1

A transformation $\mathbf{T}$ is a similarity transform over a vector space $\mathbf{V}$ if

$$\begin{aligned} \|\mathbf{T}\mathbf{v}_1\| = \|\mathbf{T}\mathbf{v}_2\| &\Leftrightarrow \|\mathbf{v}_1\| = \|\mathbf{v}_2\| \\ \mathbf{T}\mathbf{v}_1 \cdot \mathbf{T}\mathbf{v}_2 = 0 &\Leftrightarrow \mathbf{v}_1 \cdot \mathbf{v}_2 = 0 \end{aligned} \qquad \text{A1-24)}$$

for any $\mathbf{v}_1$ and $\mathbf{v}_2$ in $\mathbf{V}$.

Theorem A.1.1

Given a linear transformation $\mathbf{L}$ of the plane, there exists a unique (up to a reflection) similarity transform $\mathbf{U}$ of space such that :

$$\mathbf{L}\mathbf{v} \cong \mathbf{U}\mathbf{v}^* \qquad \text{A1-25)}$$

for any 2-D vector $\mathbf{v}$, where $\mathbf{v}^* = (x, y, 0)$ for any $\mathbf{v} = (x, y)$ and $\mathbf{v} \cong \mathbf{w}$ iff $\mathbf{v} = (x, y)$ and $\mathbf{w} = (x, y, z)$. The relationship between the transformations is given by :

$$\begin{tikzcd} \mathbf{V}^2 \arrow[r, "\mathbf{L}"] \arrow[d, "*"] & \mathbf{V}^2 \arrow[d, shift left, "\cong"] \\ \mathbf{V}^3 \arrow[r, "\mathbf{U}"] & \mathbf{V}^2 \arrow[u, shift left] \end{tikzcd}$$

where $\mathbf{V}^2$ and $\mathbf{V}^3$ are 2-D and 3-D vector spaces respectively. $\mathbf{U}$ is interpreted as a rotation and scale of two basis vectors that define a plane such that their image under an orthographic projection is the same as applying $\mathbf{L}$ to the basis vectors.

Proof. Some 3-D transformation exists for any $\mathbf{L}$ since it may be embedded in the upper-left part of a 3 x 3 matrix with all the remaining entries zero. The following shows the existence of a unique (up to a reflection) similarity transformation $\mathbf{U}$.

Let $\mathbf{e}_1$ and $\mathbf{e}_2$ be orthonormal vectors in the plane, with

$$\begin{aligned} \mathbf{e}_{s_1} &= \mathbf{L}\mathbf{e}_{m_1} \\ \mathbf{e}_{s_2} &= \mathbf{L}\mathbf{e}_{m_2} \end{aligned} \qquad \text{A1-26)}$$

If $\mathbf{v}_1 = \mathbf{U}\mathbf{e}_1^*$ and $\mathbf{v}_2 = \mathbf{U}\mathbf{e}_2^*$, we have by the definition of $\mathbf{U}$:

$$\begin{aligned} \mathbf{v}_1 &= \mathbf{e}_{s_1} + c_1 z \\ \mathbf{v}_2 &= \mathbf{e}_{s_2} + c_2 z \end{aligned} \qquad \text{A1-27)}$$

where $z = (0, 0, 1)$ and c_1 and c_2 are constants.

U is a similarity transformation iff

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = 0 \qquad \text{A1-28a)}$$

$$||\mathbf{v}_1|| = ||\mathbf{v}_2|| \qquad \text{A1-28b)}$$

since $\mathbf{e}_1^*$ and $\mathbf{e}_2^*$ are orthogonal and of the same length. From Eqs. A-7) and A-8), we have :

$$\begin{aligned} (\mathbf{e}_{s_1} + c_1\mathbf{z}) \cdot (\mathbf{e}_{s_2} + c_2\mathbf{z}) &= 0, \\ \mathbf{e}_{s_1} \cdot \mathbf{e}_{s_2} + c_1 c_2 &= 0 \end{aligned} \qquad \text{A1-29)}$$

and hence

$$c_1 c_2 = -\mathbf{e}_{s_1} \cdot \mathbf{e}_{s_2} \qquad \text{A1-30)}$$

The right side of Eq A-10) can be calculated since $\mathbf{L}$ and $\mathbf{e}_1$ and $\mathbf{e}_2$ are known.

In order for A-8a) to hold,

$$||\mathbf{v}_1||^2 = ||\mathbf{v}_2||^2 \qquad \text{A-31a)}$$

$$||e_{s_1}||^2 + c_1^2 = ||e_{s_2}||^2 + c_2^2 \qquad \text{A-31b)}$$

resulting in

$$c_1^2 - c_2^2 = ||e_{s_2}||^2 - ||e_{s_1}||^2 \qquad \text{A-32)}$$

Calling the right side of Eqs. A-10) and A-12) k_1 and k_2 respectively, we have

$$c_1 c_2 = k_1 \qquad \text{A-33a)}$$

$$c_1^2 - c_2^2 = k_2 \qquad \text{A-33b)}$$

Eqs. A-13a) and A-13b) always have a solution that is unique up to a sign ambiguity. Substituting

$$c_2 = \frac{k_1}{c_1} \qquad \text{A-34)}$$

into Eq. A13b), we obtain

$$c_1^4 - k_2 c_1^2 - k_1 = 0 \qquad \text{A-35)}$$

Substituting x for c_1^2 and solving for x gives

$$x = \frac{1}{2}\left(k_2 \pm \sqrt{k_2^2 + 4\,k_1^2}\right) \qquad \text{A-36)}$$

Since $c_1 = \sqrt{x}$, only the positive solution of Eq. A-16) is required. Eq. A-16) can only have one positive solution as $4k_1^2 \geq 0$ and therefore the quantity inside the square root is $\geq k_2$. Two real solutions for c_1 corresponding to $\pm\sqrt{x}$ therefore exist. Since $c_1\, c_2 = k_1$ there are also two solutions for c_2.

Eq.A-16) does not have a solution for x when $c_1 = 0$ as the substitution given by Eq. A-14) does not hold. However, when $c_1 = 0$, Eq.A-15) gives $c_2 = \pm\sqrt{-k_2}$. There are always two solutions for c_2 as the following shows. Using Eq.A-11b) and substituting $c_1 = 0$ we have :

$$||e_{s_1}||^2 \geq ||e_{s_2}||^2 \tag{A-37)}$$

and hence

$$k_2 = ||e_{s_2}||^2 - ||e_{s_1}||^2 \leq 0 \tag{A-38)}$$

Since the argument under the square root is positive, c_2 has two values.

There are always two solutions for c_1 and c_2 differing in sign. Iff the similarity transformation **U** exists, the equations for c_1 and c_2 will have a solution, resulting in two possible solutions. These solutions correspond to the reflective ambiguity in **U**.

Theorem A.1-2

Given three non-colinear points p_{m_i} in the plane and three corresponding points p_{s_i} (i = 1, 2, 3), there exists a unique similarity transformation (up to a reflection), Q such that $Q(p^*_{m_i}) \cong p_{s_i}$, where $v^* = (x, y, 0)$ for any $v = (x, y)$ and $v \cong w$ iff $v = (x, y)$ and $w = (x, y, z)$. Q is a 3-D translation, rotation and a scale factor.

Proof. From Lemma A.1-1 there is a unique affine transformation such that $A\ (p_{m_i}) = p_{s_i}$. A consists of a translation vector **b** and a linear **L**. **b** can be chosen such that $\mathbf{b} = p_{m_1} - p_{s_1}$ and L to be the linear transformation such that $\mathbf{L}p_{m_i} - \mathbf{b} = p_{s_i}$ (i = 1, 2).

Given **L** by Theorem A1, there is a unique, up to reflection, rotation and scale **U** such that $\mathbf{U}\mathbf{v}^* \cong \mathbf{L}\mathbf{v}$ for all 2-D vectors **v**. Combining **b** and **U** specifies a unique similarity transformation Q consisting of a translation, rotation and scale such that $Q(p_{m_i}) = p_{s_i}$.

It should be noted that it is not necessary the 3-D transformation Q for planar models. The affine transformation A is sufficient to map points from the model plane to the image plane.

Appendix A.2

The PCVISION Frame Grabber and its Operation

Provided here is a brief guide to the operation of the frame grabber (FG) used for the project. The FG digitizes an RS-170 standard video signal with an accuracy of 8-bits, therefore being capable of representing the image with 256 levels of gray, and resolution of 512x512 pixels.

The frame is arranged into four blocks of memory, each being 256x256 bytes (figure A.2-2)). In order to address each pixel in terms of its row and column, it is first necessary to know the memory block in which the pixel is located. Once this is determined, the position in memory within that block is given by the high and low bytes of the memory address register. The frame memory starts from the top left hand corner of the block and ends at the bottom right hand corner. For the purposes of this project, the origin is located at the bottom, left hand corner of the frame. The FG is locatable to any 64K boundary in the memory space of the PC-AT but was placed located at location HD0000 for this project (the H denotes a hexadecimal number). The appropriate image block is selected by one of the control/status registers provided on the FG.

The user is provided with 7 registers [PCVISION 1984]for grabber status and control. These are located from memory location HFF00 to HFF06. The most commonly used is the control/status register. The available registers are shown in table A.2-1.

The status/control register is a 16 bit register composed of various bits which may be either be set by the user or to report the current FG state. It is located at memory location HF000 and HF001. The low bits are mainly used for control and the high bits for to control such operations as continual frame capture ("grabbing"), or freeze frame mode ("snapping").

Table A2.1 Frame Grabber Register Summary

(from PCVISION Frame Grabber Manual)

Register Base Address

+	
0	Control/Status Low
1	Control/Status High
2	Look-up Table (LUT) Address
3	LUT Data
4	Mask
5	Frame Memory Block Select
6	Reset Vertical Blank Interupt

A description of the control/status (low) register follows. Not all the bits in this register are of immmediate use to the user. The board select (BDSEL) bit is used to enable the frame grabber. It of use when more than one board is installed in PC-AT, for color image processing applications for example. Since only one board was required for this project, BDSEL should always be set. Bits LUTMD0 and LUTMD1 select the input look-up tables. The input LUTs transform the image brightness prior to storage in memory. These were not used in the project and were set to zero. The ACQMODE0 and ACQMODE1 bits control the image accessing. These bits should be set to 0 and 1 or 1 and 1 for freezing ("snapping") or grabbing a frame respectively. The CLOCK bit should be set to 1 in order that the FG uses the synchrization signal for timing.

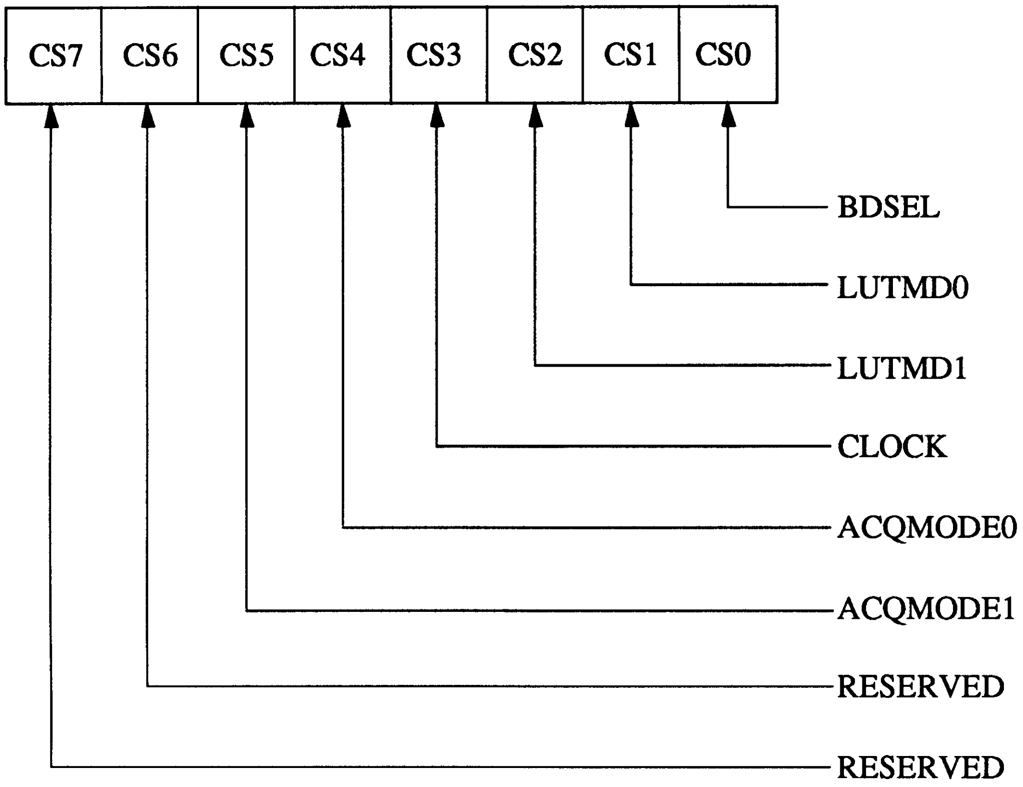


Figure A.2-1) The control/status (low) register (from Addendum to PCVISION Frame Grabber Manual)

The other important register is the frame memory block select register. Only the least two significant bits of this register are needed. This register selects the appropriate 64K block of memory, that contains the required screen position. The bits select the image quadrants (figure A.2.2) as shown below.

Bit 0	Bit 1	Block
0	0	0
0	1	1
1	0	2
1	1	3

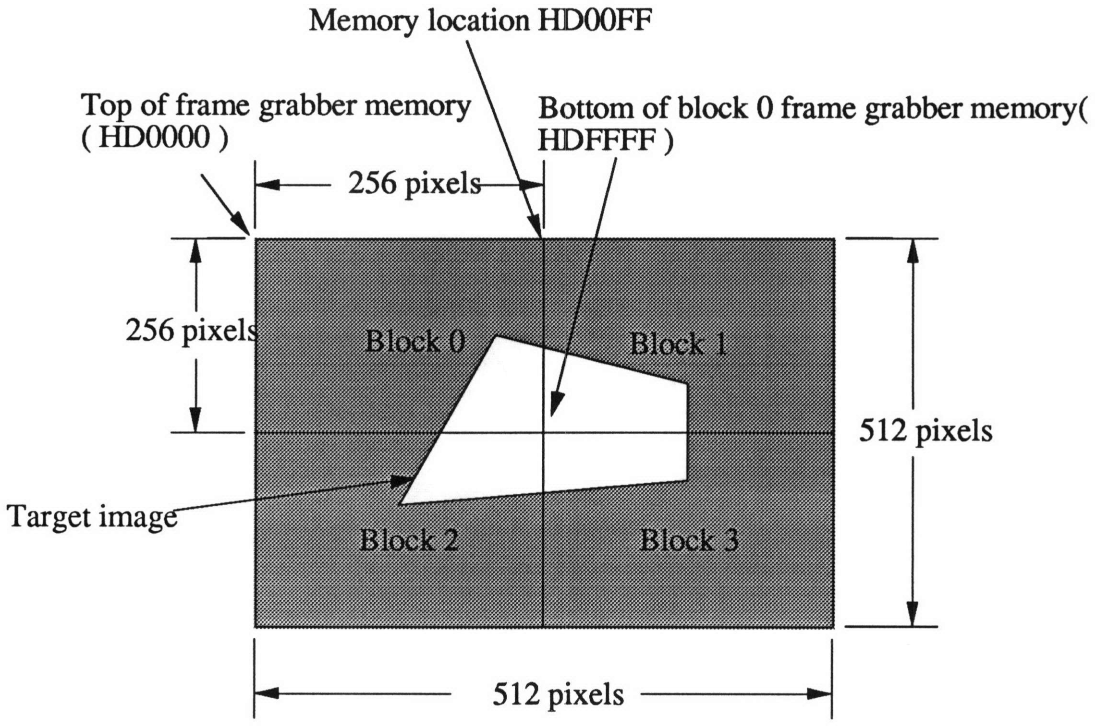


Figure A.2-2) FG block screen memory arrangement

Even though the FG produces a 512x512 image from a video signal, the RS-170 video standard is 480 lines. This means that 32 lines are unused and contain no useful image information. These lines are located at the bottom of each memory block and should not be visible to the user.

Since it is required to address the pixels with respect to the beginning of memory block, the following routine (*"glo-lo"*) is needed to convert screen point "global" into a "local" memory block coordinate with its respective origin at the top left hand corner of the block :

Algorithm *"glo-lo"* (global to local coordinates)

```
if X(global) < 256 AND X(global) ≥ 0 AND Z(global) <
    256 AND Z(global) ≥ 0
    X(local) = X(global)
    Z(local) = 255 - Z(global)
    select memory block 2
else if X(global) < 256 AND X(global) $\geq$ 0 AND Z(global) > 255 AND Z(global) < 512
    X(local) = X(global)
    Z(local) = 511 - Z(global)
    select memory block 0
else if X(global) > 255 AND X(global) < 512 AND Z(global) < 256 AND Z(global) $\geq$ 0
    X(local) = X(global) - 256
    Z(local) = 255 - Z(global)
    select memory block 3
else if X(global) > 255 AND X(global) < 512 AND Z(global) > 255 AND Z(global) < 512
    X(local) = X(global)
    Z(local) = 511 - Z(global)
    select memory block 1
else
    the point is out the range of the FG memory.
```

Appendix A.3

"ViSTA" Vision Software

A.3.1 Introduction

Presented here is ViSTA (Vision for Simulated Space Telerobotic Applications), a suite of programs for testing out machine vision algorithms. All software was written in C and compiled with the Microsoft C version 5.1 Compiler. Development was carried out on an CompuAdd AT compatible computer running at 12 MHz. Section A.3.2 gives the general instructions required for the running of the software. Section A.3.3 presents the software used for the static tests, with information on how it to modify it to allow for pose determination and section A.3.4 lists the various software modifications for the point tracking versions of the software. Information on modification of the software is provided in all sections and as comments within the programs.

A.3.2 General Operation of the Software

For all the versions of the software, the following sequence of operations are required to set up the frame grabber and to provide initial search points for the edge crossing and feature point finding algorithms.

1) Prior to the initial operation of the frame grabber, the output look-up table (OUTLUT) must be initialized. The tables are used to transform the pixel intensities prior to the display. The eight bits of pixel information coming from the frame memory are mapped on a one-one basis to intensity i.e. the OUTLUT is programmed as a ramp. This allows the full gray scale to be displayed on the monitor. The program INIT must

be run to program the output OUTLUT as a ramp. The frame grabber is now capable of either continuously acquiring or freezing a frame from a video source.

2) In order to perform tests, it is usually required to observe the target on the monitor before a test scene is selected. This is achieved by running the program GRAB, which continuously acquires a frame. Once a scene has been selected, it can be frozen by running the program SNAP. A desired frame will be held in memory when a key is pressed.

3) If the scene is desired to be stored on disk, the program ISAVE may be invoked. It will prompt the user for a filename. It is suggested that image files be saved as "filename.pic". Once a filename has been chosen and carriage return is hit, the frame will be stored in the current directory. As the image is being saved, the image on the screen will be inverted, pixel by pixel, to show how the save is progressing.

4) To recover an image from the current directory, the program RESTORE should be executed. Like ISAVE, RESTORE will prompt the user for a filename. Once carriage return is pressed, the image will be loaded into the frame grabber. The process takes about one minute.

The programs mentioned above appear on pages 91-97 except for their components place.h, place.c, getval.h and getval.c. The make file is also given in order that future modifications to these programs take minimum effort.

A.3.3 Static Test Programs

This set of programs is used to demonstrate the edge boundary crossing and feature point detection algorithms. They are given together with there associated 'make' file to ease program modification. They are intended to be used with a static image captured using the programs mentioned in section A.3.2. Once an image has been placed in the frame grabber memory, the following operations should be carried out :

1) The program pointfind should be run. The user will be prompted for a filename in which to store data such as edge crossing, edge gradients and feature point location. This information is also printed on the computer display as the program progresses.

2) The user should input the size of the search radii to be used in the search.

3) After the above information is supplied, four squares will appear on the image display. The bottom left hand pixels of these squares are the initial search points. These squares should be placed within the vicinity of the corners that the search algorithm is to locate. After locating each corner by means of the numeric keypad (8 = up, 2 = down, 6 = right, 4 = left) the '5' key should be pressed to confirm the desired location of the current search point.

4) After the last point is confirmed, pointfind embarks upon the binary searches required to locate the edge crossings. During various stages of the program run, the user will be prompted for a key press. After each prompt the program will display on the image screen the calculated location of the edge boundaries.

A.3.4 Dynamic Test Programs

There are three versions of the dynamic test software, each based on the static version. Operational setup is identical to the static version. POINT1,POINT2 and POINT3 implement update models one, two and three respectively. Input to these programs should be a video source since they implement a function version of SNAP to update the image display.

The image screen will show flashing squares to show the location of the corners as calculated by the *solver* routine. If a camera is being used as a video input, the position can now be changed to assess the ability of the algorithms to track the corners. To prevent the programs from being caught in an infinite loop if the search vectors are unable to find a corner, the program will exit the current loop that it is in an attempt the search again. The success of *findedge* is checked by the variable *success* , which if zero, means that the routine has failed to find any edges in the vicinity of the current search point.

The point tracking programs will record the position of the first feature point as calculated by *solver* in the file specified by the user. It is suggested that these programs not be run for more than about one minute in order to avoid excessively large data file sizes.

System Setup Software

```
/*      MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

        FILENAME            :       init.h

        AUTHOR              :       Ender St.John-Olcayto

        CREATED             :       11/22/89

        LAST MODIFIED       :       11/22/89

        DESCRIPTION         :       sets up the input and output LUTs of the grabber
*/

#include <stdio.h>
#include <conio.h>

#define     pfgreg      0xff00          /*Frame grabber register base address*/
#define     conl        pfgreg
#define     conh        pfgreg + 0x1
#define     luta        pfgreg + 0x2
#define     lutd        pfgreg + 0x3
#define     mask        pfgreg + 0x4
#define     fbb0        pfgreg + 0x5
#define     fbb1        pfgreg + 0x45
#define     fbb2        pfgreg + 0x65
#define     hf          0xf
#define     h10         0x10
```

```
/*      MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

        FILENAME                :               init.c

        AUTHOR                  :               Ender St.John-Olcayto

        CREATED                 :               11/22/89

        LAST MODIFIED           :               11/22/89

        DESCRIPTION             :               sets up the input and output LUTs of the grabber

*/

#include "init.h"

main() {

        int outlut, sel, a, b, pass;
        long int xluta, xlutd, xmask, xconh, xconl, xfbb0, conls;
        xluta=luta;
        xlutd=lutd;
        xmask=mask;
        xconh=conh;
        xconl=conl;
        xfbb0=fbb0;
        pass=1;
/*-------------------------------required for RGB displays i.e 3 boards----------------------------------*/

/*---------------------------------Initialise Output Look Up Tables--------------------------------------*/

        for(outlut=0; outlut<=4; outlut=outlut+2) {
/*------------------------------------------------init LUT's 0,1-------------------------------------------------*/
                outp(xconl,outlut);
                for(sel=0; sel<=2; sel=sel+2) {
                        outp(xconh,(sel*16));
/*-----------------------------------------------select next LUT------------------------------------------------*/
                        for(a=0;a<=255;a++) {
                                outp(xluta,a);
/*------------------------------------------------write address--------------------------------------------------*/
                                outp(xlutd,a);
/*--------------------------------------------------write data----------------------------------------------------*/
                        }
                }
                outp(xconh,32);         /*Set up LUT 1 for..*/
/*---------------------------------------------------overlays-----------------------------------------------------*/
                for(a=0;a<=255;a=a+2) {
/*----------------------------------zero forces output to max white---------------------------------------*/
                        outp(xluta,a);
                        outp(xlutd,255);
/*---------------------------------load every oddlocation with 255---------------------------------------*/
                }
/*----------------------------------set up LUT 3 for reverse video----------------------------------------*/
                outp(xconh,64);
/*----------------------------------------------select LUT3-------------------------------------------------*/
                b=255;
                for(a=0;a<=255;a++) {
/*------------------------------------load LUT with inverse ramp-----------------------------------------*/
                        outp(xluta,a);
                        outp(xlutd,b);
                        b=b-1;
                }
                outp(xconh,96);
/*-------------------------------------set LUT 3 for color bar---------------------------------------------*/
```

```
            b=0;
            for(a=0;a<=255;a++) {
                    outp(xluta,a);
                    outp(xlutd,0);
            }
            if(outlut==0)  {
                    for(a=0;a<=63;a++)   {
                            outp(xluta,a);
                            outp(xlutd,255);
                            outp(xluta,a+128);
                            outp(xlutd,255);
                    }
            }
            else if(outlut==4) {
                    for(a=0; a<=31; a++) {
                            outp(xluta,a);
                            outp(xlutd,255);
                            outp(xluta,a+64)        outp(xlutd,255);
                            outp(xluta,a+128);
                            outp(xlutd,255);
                            outp(xluta,a+192);
                            outp(xlutd,255);
                    }
            }
            else {
                    for(a=0; a<=127; a++) {
                            outp(xluta,a);
                            outp(lutd,255);
                    }
            }
    }
/*----------------------------------------initialize input Look Up Tables----------------------------------*/
    outp(xconl,6);
/*----------------------------------------initialize LUT0=linear--------------------------------------------*/
/*----------------------------------------   "      LUT1=inverse linear-----------------------------------*/
/*----------------------------------------   "      LUT2=threshold-----------------------------------------*/
/*----------------------------------------   "      LUT3=expansion----------------------------------------*/
    for(a=0;a<=255;a++) {
            outp(xluta,a);
            outp(xlutd,a);
    }
    outp(xconl,70);                                              /*Select LUT1*/
    for(a=0; a<=255; a++)          {
            outp(xluta,a);
            outp(xlutd,255-a);
    }
    for(sel=134; sel<=198; sel=sel+64)  {                    /*Select LUT2*/
            outp(xconl,sel);
            for(a=0;a<=255;a++) {
                    outp(xluta,a);
                    outp(xlutd,0);
            }
    }
    outp(xconl,134);                                             /*Select LUT2*/
    for(a=64; a<=196; a++)         {
            outp(xluta,a);
            outp(xlutd,255);
    }
    outp(xconl,246);                                             /*Select LUT3*/
    for(a=0; a<=127; a++)          {
/*-------------------------------------Expand from 0 to 128 in steps of two------------------------------*/
            outp(xluta,a);
            outp(xlutd,a*2);
    }
    outp(xmask,0);                                               /*Set Mask Register to zero*/
```

```
        outp(xconh,0);                                /*Set Control High to zero */
        outp(xconl,9);                                /*Set Control Control Low BDSEL and PLL*/
        outp(xfbb0,0);                                /*Set FBB Register to zero */
        conls = (inp(conl)&hf);
        outp(conl,conls+h10);
        printf("Completed");

}

/*      MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

        FILENAME            :           snap.c

        AUTHOR              :           Ender St.John-Olcayto

        CREATED             :           11/21/89

        LAST MODIFIED       :           11/21/89

        DESCRIPTION         :           Program to freeze a frame

*/

#include <stdio.h>
#include <conio.h>

#defineconl         0xFF00
#defineconh         0xFF01

main() {

        int     conls, conhs;
        char    s;

        while ((s = getch()) !='s') {
                conls = inp(conl)&(0xF);
                while (conhs!= 0)
                        conhs = inp(conh)&(0x4);
                        while (conhs == 0)
                                conhs = inp(conh)&(0x4);
                        outp(conl, conls+0x20);
                                while (conls!= 0)
                                        conls = inp(conl)&(0x30);
        }

}
```

```
/*      MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

        FILENAME            :           grab.c

        AUTHOR              :           Ender St.John-Olcayto

        CREATED             :           11/22/89

        LAST MODIFIED       :           11/22/89

        DESCRIPTION         :           Continually aquires images

*/

#include <stdio.h>
#include <conio.h>

main() {

        unsigned    port, value;

        port        =           0xFF00;
        value       =           0x49;

        outp(port,value);

}

/*      MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

        FILENAME            :           isave.h

        AUTHOR              :           Ender St.John-Olcayto

        CREATED             :           2/24/90

        LAST MODIFIED       :           2/24/90

        DESCRIPTION         :           Header file for isave.c

*/

#include <stdio.h>

typedef struct {

        int x, z;

} coord;

unsigned char                           getval(coord);
```

```
/*
        MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

        FILENAME            :           isave.c

        AUTHOR              :           Ender St.John-Olcayto

        CREATED             :           2/24/90

        LAST MODIFIED       :           2/24/90

        DESCRIPTION         :           Header file for isave.c

*/

#include "isave.h"

main() {

       FILE *picture;

       coord point;

       int c;
       char filename[15];

       printf("Filename = ");
       scanf("%s", filename);
       printf("Saving picture\n");

       picture = fopen(filename, "w");

       for (point.z = 0; point.z <= 511; point.z++) {
              for (point.x = 0; point.x <= 511; point.x++) {
                     c = (int) getval(point);

/*--------------------------if pixel intensity is the same as the EOF character-------------------------------------------------------
--jack intensity up by one value of gray to prevent----------------------------------------------------------------program from
premature termination-----------------------------------*/

                     if (c == 0x1a)
                            c = 0x1b;
                     fputc(c, picture);
              }
       }
       fclose(picture);
}
```

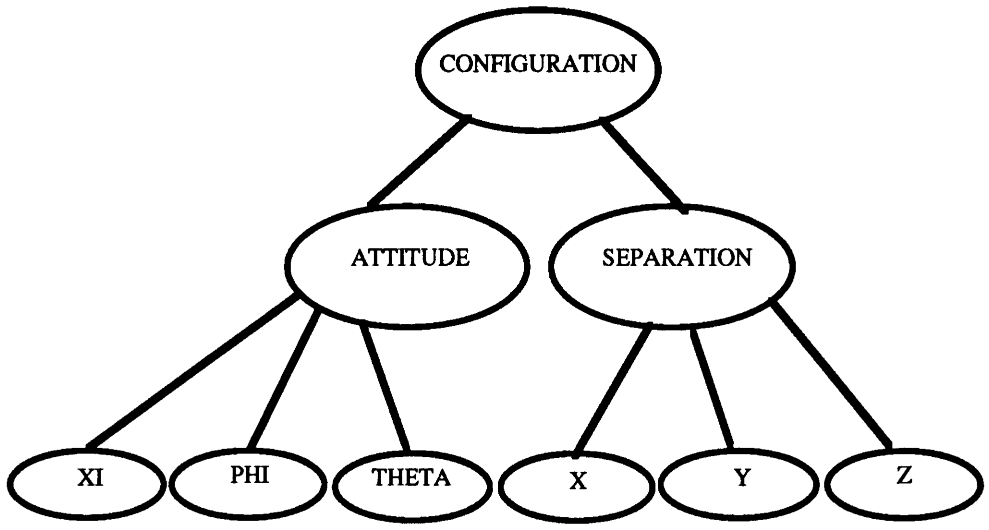
```
#
#       MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY
#
#       FILENAME                :               pix
#
#       AUTHOR                  :               Ender St.John-Olcayto
#
#       CREATED                 :               2/2/90
#
#       LAST MODIFIED           :               2/6/90
#
#       DESCRIPTION             :               MAKE file for image save (isave) program
#

# Macro definitions

model               = /As /FPi87                #small memory model, inline 8087 support
fenable             = /Ze                       #enable far pointers switch
uchars              = /J                        #unsigned chars switch
stdcomp             = /c /Od $(model)   #compile with above and no optimization
stdlink             =                           #print linker information
stdlibs             = slibc7                    #standard libs with coprocessor
objects             = isave+getval
progdep             = isave.obj getval.obj
program             = isave

#standard .C -> .OBJ inference rule

.c.obj                                  :
                    cl $(stdcomp) $*.c

#dependency block definitions for compile with above rule

isave.obj           :           isave.h     isave.c

#i.e. recompile if changes made to either header or source files

#compile the following as special cases

getval.obj          :           getval.h    getval.c
                    cl $(stdcomp) $(fenable) $(uchars) $*.c

#link result together

$(program)          :           $(progdep)
link $(stdlink) $(objects),$*,,$(stdlibs)
```

```
/*      MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

        FILENAME            :           restore.c

        AUTHOR              :           Ender St.John-Olcayto

        CREATED             :           2/24/90

        LAST MODIFIED       :           3/13/90

        DESCRIPTION         :           restore a 512*512 image starting from bottom LH
                                        corner and ending at top RH corner
*/

#include "isave.h"

main() {

        FILE *picture;

        coord point;
        unsigned char c;
        char filename[15];

        printf("Filename = ");
        scanf("%s", filename);
        printf("Loading picture\n");

        picture = fopen(filename, "r");

        if (!picture) exit(1);

        for (point.z = 0; point.z <= 511; point.z++) {

                        for (point.x = 0; point.x <= 511; point.x++) {

                                        c = (unsigned char) fgetc(picture);

                                        place(point, c);

                        }

        }

        fclose(picture);

}
```

A.3.5 Static Test Software Listings

The following shows the software that is used for both the edge finding performance tests and the pose determining algorithms evalations. The functional element POINTFIND is shown with the invese perspective transformation algorithm, CONDIT1, in place. This may be interchanged with the routine ALIGN that implements the non-redundant inverse orthographic transformation. If no pose information is required, the corresponding function is simply commented out in POINTFIND and VISION, the "make" file that compiles and links the files together. All occurances of variables that depend on results from CONDIT1 or ALIGN should be removed.

There are slight differences in the data structures that return information from the two different pose routines which should be taken into account. When function CONDIT1 is used, the data structure CONFIGURATION is :



When the allignment transformation is used, the SEPARATION branch has only one component, sincethe range is represented by a scaling factor of the image plane distance.

```
/*
        MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

        Filename            :               pointfind.h

        Author              :               Ender St. John-Olcayto

        Created             :               1/29/90

        Last modified       :               4/17/90

        Description         :               header file for pointfind.c
*/

#include <stdio.h>
#include <conio.h>

#define                     XSCALE          4.64E-4
#define                     ZSCALE          3.675E-4

/*------------------------------------------------Data Types-----------------------------------------------*/

typedef struct {
        int x, z;
} COORD;

typedef struct {
        double a;
        unsigned char b;
} LEVEL;

typedef struct {
        double ex, zed;
} DCOORD;

typedef struct {
        double XI, PHI, THETA;
} ATTITUDE;

typedef struct {
        double X, Y, Z;
} SEPARATION;

typedef struct {
        ATTITUDE orientation;
        SEPARATION distance;
} CONFIGURATION;

/*-----------------------------------FUNCTION DECLARATIONS-------------------------------------*/

COORD                                       *setup();

void                                        place( COORD );

void                                        findedge(COORD, COORD *, int, COORD *);

void                                        grad( COORD *, COORD *, LEVEL *);

void                                        condition(DCOORD *, DCOORD *, DCOORD *,
                                            DCOORD *, CONFIGURATION *);
```

```
/*
        SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

 File name                      :                 pointfind1.c

 Author                         :                 Ender St. John-Olcayto

 Created                        :                 7/1/89

 Last modified                  :                 4/17/90


 Description                    :                 static version to determine the pose using
                                                  the inverse perspective transformation

*/
#include "pointfind.h"
double                                            theta;
double                                            delta_theta = 0.3;
COORD                           temp;
FILE                                              *data;
main() {
/*----------------------------------------------variable list------------------------------------------------*/
        COORD                                     c[3], *p0, edge[4][4],
                                                  square[8][8], origin;

        DCOORD                                    hold[4], point0,
                                                  point1, point2, point3;

        LEVEL                                     m1[4], m2[4];

        CONFIGURATION                             status;

        ATTITUDE                                  rotations;

        SEPARATION                                proximity;

        double                                    xi, phi, thta,
                                                  rx, ry, rz,


        int                                       i, r1, r2, j, k;

        char                                      filename[15];
/*-------------------------------------------------data file setup------------------------------------------/*

        printf("Filename for the test results = ");
        scanf("%s", filename);
        data = fopen(filename, "w");
        fprintf(data, "STATIC POINT FINDING AND ORIENTATION DATA FILE\n");
        fprintf(data, "_______________________________________________\n");

        printf("r1 = ");
        scanf("%d", &r1);
        fprintf(data, "Major search radius r1\t=\t%d\n", r1);
        printf("\nr2 = ");
        scanf("%d", &r2);
        fprintf(data, "Minor search radius r2\t=\t%d\n", r2);

/*-------------------------------load user defined initial search points----------------------------------*/

        p0 = setup();

        fprintf(data, "Search centres are:\n");
        fprintf(data, "__________________\n");
        fprintf(data, "1.(%u, %u)\n2.(%u, %u)\n3.(%u, %u)\n4.(%u, %u)\n\n",
                                p0[0].x, p0[0].z, p0[1].x, p0[1].z,
                                p0[2].x, p0[2].z, p0[3].x, p0[3].z);

/*------------------------------------loop through all the corners-----------------------------------------*/

        for (i=0; i<=3; i++) {

                                theta = 0;

                                c[0].x = p0[i].x + r1;
                                c[0].z = p0[i].z;
                                c[1].x = 0;
                                c[1].z = 0;
                                c[2].x = 0;
                                c[2].z = 0;

                                findedge(p0[i], c, r1, &edge[i][0]);

                                c[0] = temp;
                                c[1].x = 0;
                                c[1].z = 0;
                                c[2].x = 0;
                                c[2].z = 0;

                                findedge(p0[i] ,c, r1, &edge[i][2]);

                                theta = 0;
                                c[0].x = p0[i].x + r2;
                                c[0].z = p0[i].z;
                                c[1].x = 0;
                                c[1].z = 0;
                                c[2].x = 0;
                                c[2].z = 0;

                                findedge(p0[i], c, r2, &edge[i][1]);

                                c[0] = temp;
                                c[1].x = 0;
                                c[1].z = 0;
                                c[2].x = 0;
                                c[2].z = 0;

                                findedge(p0[i], c, r2, &edge[i][3]);

        }

        printf("All Edge Boundaries Have Been Detected\n");
        printf("Hit Any Key To Continue\n\n");
        getch();

        for(i=0;i<=3;i++) {
                        printf("edge points are:\n");
                        printf("________________\n");
                        fprintf(data, "edge points are:\n");
                        fprintf(data, "________________\n");

                        printf("edge[%d][0].x = %u\tedge[%d][0].z = %u\n", i,
                                edge[i][0].x, i, edge[i][0].z);
                        fprintf(data,
                                "edge[%d][0].x = %u\tedge[%d][0].z = %u\n",
                                i, edge[i][0].x, i, edge[i][0].z);

                        printf("edge[%d][1].x = %u\tedge[%d][1].z = %u\n",
                                i, edge[i][1].x, i , edge[i][1].z);
                        fprintf(data,
                                "edge[%d][1].x = %u\tedge[%d][1].z = %u\n",
                                i, edge[i][1].x, i, edge[i][1].z);

                        printf("edge[%d][2].x = %u\tedge[%d][2].z = %u\n",
                                i,edge[i][2].x, i, edge[i][2].z);
                        fprintf(data, "edge[%d][2].x = %u\t edge[%d][2].z = %u\n",
                                i, edge[i][2].x, i, edge[i][2].z);


                        printf("edge[%d][3].x = %u\t edge[%d][3].z = %u\n\n",
                                i, edge[i][3].x, i, edge[i][3].z);
                        fprintf(data,"edge[%d][3].x = %u\t edge[%d][3].z = %u\n\n",
                                i, edge[i][3].x, i, edge[i][3].z);
                        for(j=0;j<=3;j++) {
                                printf("edge[%u][%u]\n", i, j);
                                for(t=0;t<=3;t++) {

                                        for(s=0;s<=3;s++) {

                                                square[t][s].x = edge[i][j].x + t;

                                                square[t][s].z = edge[i][j].z + s;

                                                place(square[t][s]);

                                        }
                                }
                        }
        }
/*-------------------------------------------Algorithm Solver -------------------------------------------*/

        for(i=0;i<=3;i++) {

                        grad(&edge[i][0], &edge[i][1], &m1[i]);
                        grad(&edge[i][2], &edge[i][3], &m2[i]);

                        if ((m1[i].b == 1) && (m2[i].b == 1)) {

                                hold[i].ex = ( ((m1[i].a)*edge[i][1].x -
                                (m2[i].a)*edge[i][3].x + edge[i][3].z -
                                edge[i][1].z)/((m1[i].a) - (m2[i].a)) );

                                p0[i].x = (int) hold[i].ex;

                                hold[i].zed = ((m1[i].a)*(hold[i].ex -
                                edge[i][1].x)+ edge[i][1].z );
```

```
                                        p0[i].z = (int) hold[i].zed;

                        }

                        else if (m1[i].b == 0) {

                                        p0[i].x = edge[i][1].x;
                                        p0[i].z = (int) ( (m2[i].a)*(p0[i].x
                                                - edge[i][3].x) + edge[i][3].z );

                        }

                        else {

                                        p0[i].x = edge[i][3].x;
                                        p0[i].z = (int) ( (m1[i].a)*(p0[i].x
                                                - edge[i][1].x) + edge[i][1].z );

                        }

        }

/*------------------------------reset the origin to the center of the screen------------------------------*/

        origin.x = 255;
        origin.z = 255;

/*--------------------------reference points with respect to the new origin----------------------------*/
/*------------------and scale such real dimensions are represented by the structures----------------*/

        point0.ex               = (double) (XSCALE * (p0[0].x-origin.x));
        point0.zed              = (double) (ZSCALE * (p0[0].z-origin.z));
        point1.ex               = (double) (XSCALE * (p0[1].x-origin.x));
        point1.zed              = (double) (ZSCALE * (p0[1].z-origin.z));
        point2.ex               = (double) (XSCALE * (p0[2].x-origin.x));
        point2.zed              = (double) (ZSCALE * (p0[2].z-origin.z));
        point3.ex               = (double) (XSCALE * (p0[3].x-origin.x));
        point3.zed              = (double) (ZSCALE * (p0[3].z-origin.z));


/*------------------------plug in required pose determining routine here------------------------------*/

        condition(&point0, &point1, &point3, &point2, &status);

/*--------------------------extract the range and orientaiton data structures----------------------------*/

        rotations = status.orientation;
        proximity = status.distance;

        xi                      = rotations.XI;
        phi                     = rotations.PHI;
        thta                    = rotations.THETA;

        rx                      = proximity.X;
        ry                      = proximity.Y;
        rz                      = proximity.Z;

        printf("Edge Gradients:\n");
        printf("_______________\n");
        fprintf(data, "Edge Gradients:\n");
        fprintf(data, "_______________\n");

        for(i=0;i<=3;i++) {
```

```
                        printf("m1[%d].a = %f\tm1[%d].b = %u\n",
                                    i, m1[i].a, i, m1[i].b);
                        printf("m2[%d].a = %f\tm2[%d].b = %u\n\n",
                                    i, m2[i].a, i, m2[i].b);

                        fprintf(data, "m1[%d].a = %f\tm1[%d].b = %u\n",
                                    i, m1[i].a, i, m1[i].b);
                        fprintf(data, "m2[%d].a = %f\tm2[%d].b = %u\n\n",
                                    i, m2[i].a, i, m2[i].b);

                        printf("Press Any Key To Continue\n");

                        getch();

                        for(j=0;j<=7;j++) {
                                    for(k=0;k<=7;k++) {
                                                square[k][j].x = p0[i].x + k;
                                                square[k][j].z = p0[i].z + j;
                                                place(square[k][j]);
                                    }
                        }
            }

            printf("\n\n");
            fprintf(data, "\n\n");

            for (i=0;i<=3;i++) {

                        printf("p0[%d].x = %d\t p0[%d].z = %d\n\n",
                                    i, p0[i].x, i, p0[i].z);
                        fprintf(data, "p0[%d].x = %d\t p0[%d].z = %d\n",
                                    i, p0[i].x, i, p0[i].z);

            }

            printf("\n\n");
            fprintf(data, "\n\n");

            printf("Camera Look Angles:\n");
            printf("___________________\n");
            printf("xi=%f\nphi=%f\nthta=%f\n\n", xi, phi, thta);
            fprintf(data, "Camera Look Angles:\n");
            fprintf(data, "___________________\n");
            fprintf(data, "xi=%f\nphi=%f\nthta=%f\n\n", xi, phi, thta);

            printf("Target Range:\n");
            printf("_____________\n");
            printf("rx=%f\nry=%f\nrz=%f\n", rx, ry, rz);
            fprintf(data, "Target Range:\n");
            fprintf(data, "_____________\n");
            fprintf(data, "\n\nrx=%f\nry=%f\nrz=%f\n", rx, ry, rz);

}
```

```
/*
        MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

        Filename            :           findedge.h

        Author              :           Ender St. John-Olcayto

        Created             :           1/30/90

        Last modified       :           4/2/90

        Description         :           header file for findedge.c
*/

#include <stdio.h>
#include <math.h>
#include <conio.h>

/*------------define the limit top which the binary search will continue to search-------------------*/

#define                     EPSILON         2

typedef struct {

        int x, z;

} COORD;

/*----------------------------------------function definitions----------------------------------------------*/

void                        findedge(COORD, COORD *, int, COORD *);

char                        getval(COORD);

/*-----------------------------------------global variables-------------------------------------------------*/

extern double               theta;

extern double               delta_theta;

extern COORD                temp;
```

```
/*
        MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

        Filename                :               findedge.c


        Author                  :               Ender St. John-Olcayto

        Created                 :               11/7/89


        Last modified           :               4/2/90



        Description             :               function to find 8 points on the edge of a
                                                rectangle such that the 4 corners can be found

*/

#include "findedge.h"

void findedge(COORD start, COORD *p, int radius, COORD *side) {

        char                                    val[3];
        unsigned int                            d = 0;

        val[0] = 0;
        val[1] = 0;
        val[2] = 0;

/*---------------------------------------------Algorithm roughsearch-----------------------------------------*/

        while (val[0] == val[1]) {

                        theta = theta + delta_theta;

                        p[1].x = (int) (start.x + radius*cos(theta));
                        p[1].z = (int) (start.z + radius*sin(theta));

                        val[0] = getval(p[0]);
                        val[1] = getval(p[1]);

                        if (val[0] == val[1]) {

                                p[0] = p[1];
                                val[0] = val[1];

                        }

        }

        temp = p[1];

/*---------------------------------------------Algorithm finesearch------------------------------------------*/

        do {

/*---------------divide the vector from point p[0] to p[1] by two by bit shifting right----------------*/

                        p[2].x = (p[1].x + p[0].x) >> 1;

                        p[2].z = (p[1].z + p[0].z) >> 1;

                        val[1] = getval(p[1]);
```

```
                                        val[2] = getval(p[2]);
                                        if (val[2] != val[1]) {
                                                        p[0] = p[2];
                                                        val[0] = val[2];
                                        }

                                        if (val[2] == val[1]) {
                                                        p[1] = p[2];
                                                        val[1] = val[2];
                                        }
                                        d = (unsigned int) ((p[1].x - p[0].x)*(p[1].x - p[0].x))
                                                        + ((p[1].z - p[0].z)*(p[1].z - p[0].z));

/*-----------keeping dividing vector into two until length is EPSILON (two pixels)--------------*/

        } while (d > EPSILON);

        side -> x = p[1].x;
        side -> z = p[1].z;

        return;
}


/*
        MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

 Filename                       :               getval.h

 Author                         :               Ender St. John-Olcayto

 Created                        :               2/24/90

 Last modified                  :               4/2/90

 Description                    :               header file for getval.c

*/

#include <stdio.h>
#include <conio.h>

#define BLOCK                                   0xFF05          /*block select register*/
#define                         TOP                     0xD000  /*top of frame grabber*/
#define                         threshold               127     /*image threshold*/
#define MK_FP(seg,ofs)  ((void far *) (((unsigned long)(seg) << 16) |
                                (unsigned)(ofs)))

typedef struct {
        int x, z;
} COORD;
```

```
/*
        MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

Filename                        :               getval.c

Author                          :               Ender St. John-Olcayto

Created                         :               1/17/90

Last modified                   :               3/28/90

Description                                     : Returns the grey-level of a screen point
                                                  specified in global coordinates
*/

#include "getval.h"

unsigned char getval(COORD global) {
        COORD                                   local;
        unsigned char far                       *s;
        unsigned char                           gl;
        unsigned int                            offset,
                                                segment = TOP;

        local.x = 0;
        local.z = 0;

/*------------------------------------------algorithm glo-lo------------------------------------------*/

        if (((global.x < 256) && (global.x >=0 ))
                                && ((global.z < 256) && (global.z >= 0))) {

                                local.x = global.x;
                                local.z = 255 - global.z;
                                outp(BLOCK, 2);

        }

        else if (((global.x < 256) && (global.x >= 0))
                                && ((global.z > 255) && (global.z < 512))) {

                                local.x = global.x;
                                local.z = 511 - global.z;
                                outp(BLOCK, 0);

        }

        else if (((global.x > 255) && (global.x < 512))
                                && ((global.z < 256) && (global.z >=0 ))) {

                                local.x = global.x - 256;
                                local.z = 255 - global.z;
                                outp(BLOCK, 3);

        }

        else if (((global.x > 255) && (global.x<512))
                                && ((global.z > 255) && (global.z<512))) {

                                local.x = global.x - 256;
                                local.z = 511 - global.z;
                                outp(BLOCK, 1);

        }
/*-----------------------------------else to catch out of screen excursions--------------------------------*/

          else {
/*----------------if COORDinates are off the screen, let the grey-level be pure black----------------*/

                              gl = 0;
                              return (gl);

          }

          offset = ((local.z << 8)|local.x);
          s =  MK_FP(segment, offset);
          gl = *s;
          if (gl<threshold)
                              gl = 0;
          else
                              gl = 255;
          return (gl);
}

unsigned char                                   getval(COORD);


/*
          MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

          Filename                              : place.h

          Author                                : Ender St. John-Olcayto

          Created                               : 1/29/90

          Last modified                         : 4/2/90

          Description                           : header file for place.c
*/

#include <stdio.h>
#include <conio.h>

#define   BLOCK                                 0xFF05
#define                         TOP                           0xD000
#define   MK_FP(seg,ofs)      ((void far *) (((unsigned long)(seg) << 16) |
          (unsigned)(ofs)))

typedef struct {
          int x, z;
} COORD;

void                                            place( COORD );
```

```
/*
        MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

        Filename            :           place.c

        Author              :           Ender St. John-Olcayto

        Created             :           1/28/90

        Last modified       :           3/27/90

        Description         :           places a point on the screen by NOTTING the cur-
                                        rent backgound color

*/

#include "place.h"


void place( COORD global ) {

        COORD                                   local;
        unsigned char far                       *s = 0;
        unsigned char                           gl;
        unsigned int                            offset,
                                                segment = TOP;


        local.x = 0;
        local.z = 0;

/*----------------------------------------Algorithm glo-lo-----------------------------------------------*/

        if (((global.x < 256) && (global.x >= 0 ))
                        && ((global.z < 256) && (global.z >= 0 ))) {
                        local.x = global.x;
                        local.z = 255 - global.z;
                        outp(BLOCK, 2);
        }

        else if (((global.x < 256) && (global.x >= 0))
                        && ((global.z > 255) && (global.z < 512 ))) {
                        local.x = global.x;
                        local.z = 511 - global.z;
                        outp(BLOCK, 0);
        }

        else if (((global.x >255) && (global.x < 512))
                        && ((global.z < 256) && (global.z >=0 ))) {
                        local.x = global.x - 256;
                        local.z = 255 - global.z;
                        outp(BLOCK, 3);
        }

        else if (((global.x > 255) && (global.x < 512))
                        && ((global.z > 255) && (global.z < 512))) {
                        local.x = global.x - 256;
                        local.z = 511 - global.z;
                        outp(BLOCK, 1);
        }

        else {
                        /* no operations yet - for future expansion */
        }
        offset = (( local.z << 8) | local.x);
        s =  MK_FP(segment, offset);
        gl = *s;
        *s = ~gl;
        return;

}


/*
        MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

        Filename            :               grad.h

        Author              :               Ender St. John-Olcayto

        Created             :               1/31/90

        Last modified       :               4/2/90

        Description         :               header file for grad.c

*/


#include <stdio.h>
#include <stdlib.h>

typedef struct {
        int x, z;
} COORD;

typedef struct {
        double a;
        unsigned char b;
} LEVEL;

void                                        grad( COORD *, COORD *, LEVEL *);

extern FILE                                 *data;


/*
        MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

        Filename            :               grad.c

        Author              :               Ender St. John-Olcayto

        Created             :               1/31/90

        Last modified       :               4/2/90

        Description         :               edge gradient finding function

*/

#include "grad.h"

void grad(COORD *point1, COORD *point2, LEVEL *m) {

        if( (point1 -> x) > (point2 -> x) ) {

                                m -> a = ((double) (point1->z - point2->z))/
                                                ((double) (point1->x - point2->x));
```

```
                    m -> b = 1;

        }

        else if ( (point1 -> x) < (point2 -> x) ) {
                    m -> a = ((double) (point2->z - point1->z))/
                                      ((double) (point2->x - point1->x));
                    m -> b = 1;
        }

        else {

                    m -> a = 0;
                    m -> b = 0;

        }

        return;

}


/*
        MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

        Filename            :           stp.h

        Author              :           Ender St. John-Olcayto

        Created             :           1/19/90

        Last modified       :           4/2/90

        Description         :           header file for stp.c

*/

#include <stdio.h>
#include <conio.h>
# include <malloc.h>
#include <stdlib.h>

#define                     NEXT                '5'     /*next point chosen by hitting 5*/

typedef struct {
        int x, z;
        } COORD;

COORD   *setup();

void                        place( COORD );
```

```
/*
        MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

        Filename            :            stp.c

        Author              :            Ender St. John-Olcayto

        Created             :            1/26/90

        Last modified       :            4/18/90

        Description         :            Asks for center for search vectors for each
                                         block and for the search radii.  Supplies info
                                         for pointfind
*/

#include "stp.h"


COORD *setup() {

        int                                      i, j, k,
                                                 a, b, key;

        COORD               pos, square[4][4][4],
                            *ptemp;

        char                *data = "Search centre = (%u, %u)\t gl = %u\n";

        const char          *message =
                            "\nmalloc failed for variable ptemp in function 'stp'- aborting";

        unsigned char       col;

/*--------------make sure that program can allocate the required chunk of memory-----------------*/

        if((ptemp = (COORD *) malloc(4*sizeof(COORD))) == NULL) {
                            printf(message);
                            exit(0);
        }

        for (k = 0; k <= 3; k++) {

                            switch (k) {

                                                 case 0: {
                                                         a = 0;
                                                         b = 0;
                                                         break;
                                                 }

                                                 case 1: {
                                                         a = 256
                                                         b = 0;
                                                         break;
                                                 }

                                                 case 2: {
                                                         a = 256;
                                                         b = 256;
                                                         break;
                                                 }

                                                 default : {
                                                         a = 0;
```

```
                                        b = 256;
                                }

                        }

                        pos.x = 128 + a;
                        pos.z = 128 + b;

                        for (j = 0; j <= 3; j++) {

                                for (i = 0; i <= 3; i++) {

                                        square[k][j][i].x = pos.x + i;
                                        square[k][j][i].z = pos.z + j;
                                        place(square[k][j][i]);

                                }

                        }

}

printf("Are these starting points OK?");
printf("\n");
printf("Use keypad to change and carriage return");
printf("\n");
printf("to fix search centres.\n");

for (k = 0; k <= 3; k++) {

                        while ( (key = getch() ) != NEXT) {

                                switch (key) {

                                        case '4' : {

                                                for (j = 0; j <= 3; j++) {

                                                        for (i = 0; i <= 3; i++) {


                                                                place(square[k][j][i]);
                                                                square[k][j][i].x =
                                                                square[k][j][i].x - 1;
                                                                place(square[k][j][i]);
                                                        }

                                                }
                                                col = getval(square[k][0][0]);
                                                printf(data, square[k][0][0].x,
                                                        square[k][0][0].z, ~col);

                                                break;

                                        }

                                        case '6' : {

                                                for (j = 0; j <= 3; j++) {

                                                        for (i = 0; i <= 3; i++) {

                                                                place(square[k][j][i]);
                                                                square[k][j][i].x =
                                                                square[k][j][i].x + 1;
                                place(square[k][j][i]);

                        }

                }

                col = getval(square[k][0][0]);
                printf(data, square[k][0][0].x,
                square[k][0][0].z, ~col);

                break;

        }

        case '8' : {

                for (j = 0; j <= 3; j++) {

                        for (i = 0; i <= 3; i++) {
                                place(square[k][j][i]);
                                square[k][j][i].z =
                                square[k][j][i].z + 1;

                                place(square[k][j][i]);

                        }

                }

                col = getval(square[k][0][0]);
                printf(data, square[k][0][0].x,
                        square[k][0][0].z, ~col);

                break;

        }

        case '2' : {

                for (j = 0; j <= 3; j++) {

                        for (i = 0; i <=3; i++) {

                                place(square[k][j][i]);
                                square[k][j][i].z =
                                square[k][j][i].z - 1;

                                place(square[k][j][i]);


                        }

                }

                col = getval(square[k][0][0]);
                printf(data, square[k][0][0].x,
                square[k][0][0].z, ~col);

        }

        default : {

                /* no operation */
```

```
                                                    break;
                                            }
                                    }
                            }
        }
        for (i = 0; i <=3; i++) {
                            ptemp[i] = square[i][0][0];
        }
        return (ptemp);
}
```

```
#
#       SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY
#
#       Filename                :               vision
#
#       Author                  :               Ender St. John-Olcayto
#
#       Created                 :               2/2/90
#
#       Last modified           :               4/4/90
#
#       Description             :               MAKE file for vision programs (for static test ver-
                                                sions)
#

# Macro definitions

model           =               /As /FPi87              #small memory model, inline 8087 support
fenable         =               /Ze                     #enable far pointers switch
uchars          =               /J                      #unsigned chars switch
stdcomp =       /c /Zi /Od $(model)     #compile with above and no optimization
stdlink         =               /co /li /map /noi       #print linker information
stdlibs         =               slibc7                  #standard libs with coprocessor
objects =       pointfind+stp+findedge+grad+place+getval+condit1
progdep =       pointfind.obj stp.obj findedge.obj grad.obj place.obj getval.obj\
                        condit1.obj
program =       pointfind

#standard .C -> .OBJ inference rule

.c.obj          :
cl $(stdcomp) $*.c

#dependency block definitions for compile with above rule

pointfind.obj   :               pointfind.h     pointfind.c

stp.obj         :               stp.h                   stp.c

grad.obj  :     grad.h                          grad.c

condit1.obj     :               condit1.h       condit1.c

#i.e. recompile if changes made to either header or source files
```

```
#compile the following as special cases

place.obj :        place.h                          place.c
           cl $(stdcomp) $(fenable) $(uchars) $*.c

getval.obj:        getval.h             getval.c
           cl $(stdcomp) $(fenable) $(uchars) $*.c

findedge.obj       :                    findedge.h       findedge.c
           cl $(stdcomp) $(uchars)      $*.c

#link result together

$(program)         :                    $(progdep)
link $(stdlink) /STACK:4000 $(objects),$*,,$(stdlibs)
```

A.3.6 Dynamaic Test Software Listings

The following software was used to test the feature point update tracking routines. Since the three versions are all very similar, only POINT3; which implements the more complex update routine will be shown. A functional version of the freeze frame routine is also listed, this being the function that updates the screen display one all operations have been completed on the current frame.

```
/*
        MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

        Filename             :            point3.c

        Author               :            Ender St. John-Olcayto

        Created              :            7/1/89

        Last modified        :            4/11/90

        Description          :            dynamic version of pointfind that uses position,
                                          velocity and acceleration estimates to extrapolate
                                          the next feature point position
*/

#include "pointfind.h"

double                      theta;

double                      delta_theta = 0.5;

COORD          temp;

FILE                        *data;

main() {

        COORD                              c[3], *p0, p01[4],
                                           p02[4], t0[4],
                                           edge[4][4], est[4],
                                        square[8][8];

        DCOORD                          hold[4];

        LEVEL                           m1[4], m2[4];

        int                             i, r1, r2, t1, t2, j, k;

        unsigned int                    flag = 0, success = 1;

        char                            filename[15];

        printf("Filename for the test results = ");
        scanf("%s", filename);
        data = fopen(filename, "w");

        printf("r1 = ");
        scanf("%d", &r1);
        printf("\nr2 = ");
        scanf("%d", &r2);

        p0 = setup();

        for (i=0; i<=3; i++) {

                t0[i] = p0[i];
                p01[i].x = 0;
                p01[i].z = 0;
                p02[i].x = 0;
                p02[i].z = 0;

        }

        do {

                snap();

                for (i=0; i<=3; i++) {

                                theta = 0;

                                c[0].x = t0[i].x + r1;
                                c[0].z = t0[i].z;
                                c[1].x = 0;
                                c[1].z = 0;
                                c[2].x = 0;
                                c[2].z = 0;

/*--the new version of pointfind checks to see if findedge is wasting time looking for a corner- */
/*---------------------------------by checking the status of the success flag----------------------------*/

                                success = findedge(p0[i], c, r1, &edge[i][0]);

                                if (!success) break;

                                c[0] = temp;
                                c[1].x = 0;
                                c[1].z = 0;
                                c[2].x = 0;
                                c[2].z = 0;

                                success = findedge(p0[i] ,c, r1, &edge[i][2]);

                                if (!success) break;
                theta = 0;
                c[0].x = t0[i].x + r2;
                c[0].z = t0[i].z;
                c[1].x = 0;
                c[1].z = 0;
                c[2].x = 0;
                c[2].z = 0;

                success = findedge(p0[i], c, r2, &edge[i][1]);

                if (!success) break;

                c[0] = temp;
                c[1].x = 0;
                c[1].z = 0;
                c[2].x = 0;
                c[2].z = 0;

                success = findedge(p0[i], c, r2, &edge[i][3]);

                if (!success) break;

}

for(i=0;i<=3;i++) {

                if (!success) break;

                grad(&edge[i][0], &edge[i][1], &m1[i]);
                grad(&edge[i][2], &edge[i][3], &m2[i]);

                if ((m1[i].b == 1) && (m2[i].b == 1)) {

                                hold[i].ex = ( ((m1[i].a)*edge[i][1].x -
                                (m2[i].a)*edge[i][3].x + edge[i][3].z -
                                edge[i][1].z)/((m1[i].a) - (m2[i].a)) );

                                p0[i].x = (int) hold[i].ex;

                                hold[i].zed = ((m1[i].a)*(hold[i].ex -
                                edge[i][1].x)+ edge[i][1].z );

                                p0[i].z = (int) hold[i].zed;

                }

                else if (m1[i].b == 0) {

                                p0[i].x = edge[i][1].x;

                                p0[i].z = (int) ( (m2[i].a)*(p0[i].x
                                          - edge[i][3].x) + edge[i][3].z );

                }

                else {

                                p0[i].x = edge[i][3].x;

                                p0[i].z = (int) ( (m1[i].a)*(p0[i].x
                                          - edge[i][1].x) + edge[i][1].z );

                }

                est[i] = t0[i];
```

```
                        if (flag < 2 ) {

                                        t0[i].x = p0[i].x;
                                        t0[i].z = p0[i].z;

                        }

                        else {

                                        t0[i].x = 3*p0[i].x - 3*p01[i].x + p02[i].x;
                                        t0[i].z = 3*p0[i].z - 3*p01[i].z + p02[i].z;

                        }

                        p02[i] = p01[i];

                        p01[i] = p0[i];

                }

                if (flag < 2)
                                        flag = flag + 1;

                for(i=0;i<=3;i++) {
                        for(j=0;j<=7;j++) {
                                        for(k=0;k<=7;k++) {
                                                square[k][j].x = p0[i].x + k;
                                                square[k][j].z = p0[i].z + j;
                                                place(square[k][j]);
                                        }
                        }
                }

                fprintf(data, "%d\t%d\t%d\t%d\n",
                        p0[0].x, p0[0].z, est[0].x, est[0].z);

/*------------------------------------------keep going until a key is hit------------------------------------*/

        } while (!kbhit());

        fclose(data);

}

/*
        MIT SPACE SYSTEMS LABORATORY, LAB OF ORBITAL PRODUCTIVITY

        Filename        :       snap.c

        Author          :       Ender St. John-Olcayto

        Created         :       11/21/89

        Last modified   :       4/4/90

        Description     :       Function to freeze a frame.
*/

#include <stdio.h>
#include <conio.h>

#define conl 0xFF00
#define conh 0xFF01
```

```
void snap() {

        int conls, conhs;
        char s;

        conls = inp(conl)&(0xF);

        while (conhs!=0)

                conhs = inp(conh)&(0x4);

        while (conhs==0)

                conhs = inp(conh)&(0x4);

        outp(conl,conls+0x20);

        while (conls!=0)

                conls = inp(conl)&(0x30);

        return;

}
```