AN ABSTRACT OF THE THESIS OF

L. Andrew Jansky for the degree of Master of Science in Civil Engineering presented on December 17, 1993.

Title: Three Dimensional Dynamic Video Position Sensing

Redacted for Privacy

Abstract Approved: [illegible]

Charles K. Sollitt

A comprehensive system to locate and track objects in two or three dimensional space, using non-contact video sensing techniques is described. The need exists to be able to quantify range and proximity of objects that would be difficult or impossible to measure using standard contact based sensor technology. Available video technology is surveyed and classified. Then, a hardware system is assembled that fulfills the project goal, within given budgetary constraints. The individual components of the system are described in detail.

The theoretical solution for single camera, 2-D positioning, is developed. A device dependent computer algorithm is developed to perform the object location. An accurate multi-camera, 3-D positioning algorithm is also developed. A method to calibrate the cameras is also described and applied. Computer algorithms to perform calibration and solve the multiple view, 3-D location geometry are presented. The theoretical equations and most of the algorithms are transferable, not hardware specific.

Examples using the 2-D model are presented. The first test is a submerged, single degree of freedom model that was subjected to wave action. Video tracking data is compared with available positioning data from string potentiometers. The second test is a surface float application where contact sensing methods were not possible.

The 3-D algorithm is demonstrated in an above water test. The longitudinal motion of a linear constrained target is measured with a string potentiometer and compared with a two-camera, 3-D video interpretation of the motion. The calibration method is verified with the 3-D algorithm.

Three Dimensional Dynamic Video Position Sensing

By

L. Andrew Jansky

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed December 17, 1993

Commencement June 1994

APPROVED:

Redacted for Privacy

Associate Professor of Civil Engineering in charge of major

Redacted for Privacy

Head of Department of Civil Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented: December 17, 1993

Typed by the author for: L. Andrew Jansky

ACKNOWLEDGEMENT

It has been four years since the inception of this work, and during that time technology in the image processing field has progressed rapidly. Fortunately a majority of the work done is not dependent on the current state of technology. The ideas and computer code can be adapted to current hardware with little effort.

With that said I would like to express my sincere gratitude to Professor Charles Sollitt for presenting me with the original idea of tracking objects with video cameras. His unwavering faith, support and guidance were invaluable.

The members of the O.H. Hinsdale Wave Research Laboratory staff, Bill Hollings, and Terry Dibble were extremely helpful in the experimental setup and testing of the system. I would like to thank Cheryl Zedwick for help formatting and preparing the document to meet the high standards of the Oregon State University Graduate School.

I would also like to thank the Office of Naval Research for their financial support under contract number N00014-86K0687, without whose support this work may not have been possible.

Finally I would like to thank my family and wife Susie for the continual prodding to get THAT THESIS DONE......

TABLE OF CONTENTS

TABLE OF CONTENTS (cont)

LIST OF FIGURES

LIST OF FIGURES (cont)

LIST OF TABLES

LIST OF SYMBOLS

a1, a2, ...	Intermediate variable
A	Resultant matrix
B	Resultant matrix
BL	Base line, distance between parallel optic axes
C_x, C_y	Number of pixels from frame buffer corner to image center
d_x, d_y	Distance between adjacent sensor elements (dimensional)
D	Distance from camera image plane to target
f	Focal length of camera
f_i	Focal length
f_1, f_2	Focal lengths
h	Height of area from target
I_i	Potential target location identifier
n	Matrix element index
$\mathbf{P}_w$	Point location vector, relative to the world coordinate frame
r1,2,...9	Rotation matrix elements
R	Rotation matrix
S_x	Horizontal image scale factor
T	Translation matrix
T_{x_i}	Translation matrix element, x direction
T_{y_i}	Translation matrix element, y direction
T_{z_i}	Translation matrix element, z direction

x, y, z	True world coordinates
x_1	Computer image distance, camera one
x_2	Computer image distance, camera two
x_{world}	World location
y_{world}	World location
z_{world}	World location
X, Y	Computer screen coordinates
$\bar{X}, \bar{Y}$	Relative screen centroid coordinates
X_d	Distorted image coordinate
Y_d	Distorted image coordinate
X_{f_i}, Y_{f_i}	Frame buffer coordinates
X_{imag}	Computer image location
Y_{imag}	Computer image location
X_i, Y_i	Image coordinates
X_u, Y_u	Undistorted image coordinates
θ_1, θ_2	Angle from P between target and image
κ_1	Radial lens distortion coefficient

Three Dimensional Dynamic Video Position Sensing

1.0 INTRODUCTION

1.1 Motivation

Many applications exist for non-contact video measurement techniques. There is a specific need to sense position of rigid and deformable bodies used in experiments at the O.H. Hinsdale Wave Research Laboratory, Oregon State University. Example applications include observations of compliant structures, neutrally or positively buoyant targets in a water continuum and measurent of surface displacement of erodible materals.

Existing measurement methods work well with single degree of freedom systems, but additional degrees of freedom greatly increase the complexity of setup and analysis. Traditional methods also impose limits on translational motion superimposed on periodic motions. The non-contact method is preferred because there is no influence on the object being observed. This is especially important when the object inertia is of the same order of magnitude as the force imposed by the measurement device.

The hardware system for this application is constrained by a $30,000 budget. The software system is required to be simple to use and easy to upgrade. The video system is intended to supplement existing measurement methods and provide additional capabilities to the Wave Research Lab.

1.2 Theoretical Overview

A computer vision, object location and tracking system has two unique theoretical bases: two dimensional, single camera theory and three dimensional, multi-camera theory. Calibration is different for each theory, but both theories share a common principle.

The fundamental principle in the computer vision system is the pinhole camera model. This model assumes that cameras behave as a box with a pinhole in one end and a projection plane on the other end. Light from an object enters the hole and is projected on the plane. The light rays are not bent by the pinhole in the process. The three dimensional object is mapped on to a two dimensional plane by the perspective transformation.

The one camera, two dimensional algorithm uses screen and world geometry to obtain position of objects. A simple scale ratio is calculated in the calibration setup. This is used to convert screen locations into world locations. The single camera algorithm imposes some significant limitations where as the multiple camera algorithm does not.

The multiple camera, three dimensional algorithm is based on projecting a line from the image plane through the lens center into three dimensional space. A line is similarly projected from the other cameras and the point in space where the lines intersect is the three dimensional location of the object or point. A set of simultaneous equations is solved using a least squares approach to get the exact location.

The calibration theory for the three dimensional model determines the relationship between the cameras two dimensional image coordinate system and the three dimensional world coordinate system. A set of simultaneous equations is formed by locating calibration points both on screen and in absolute world locations. The set of equations is solved using a least squares procedure. The resulting values are used in the location algorithm to determine absolute position of a chosen object, within the calibration volume.

1.3 Scope

This project develops a method to sense position and orientation of bodies used in scaled experiments. Extensive product research led to the assembly of a computer vision system that can achieve the project goals within budget. Hardware components include: a real-time image processor, PC 486-DX host computer, two video recorders, computer-video recorder interface, two remote controlable video cameras, two time code generators/readers and two time base correctors.

The system has two operating modes, two dimensional and three dimensional. The two dimensional mode uses a single camera, but all motions must occur in the two dimensional plane parallel to the camera image plane and lens distortion effects can be significant. No out of plane measurements can be determined in this mode.

The three dimensional mode was developed from an existing algorithm. It is based on epipolar geometry and relies on an accurate calibration. The calibration algorithm allows lens distortion to be calculated and by using multiple cameras the third dimension is resolvable with great accuracy.

The three dimensional algorithm and the calibration algorithm are solved by evaluating several redundant sets of simultaneous equations using a least squares optimizations approach. This allows the overdetermined set of equations to be solved.

Target indentification software, calibrations procedures and postion algorithims are programmed in Quick C and operate on a PC 486-DX computer system.

Several example test applications of the two dimensional system are performed. One test allows for verification of the results. Another test demonstrates qualitative performance.

A test of the three dimensional system is developed that quantifies the motions of an object and allows direct comparison to existing measurement techniques. A calibration is performed and results verified. Several runs are performed and the resulting target motions tracked. The data from the runs are compared to the traditional measurement procedure to verify the accuracy of the video measurement system.

2.0 SYSTEM DESIGN AND DEVELOPMENT

2.1 Design Considerations

The main operational requirements for the tracking system are; 1) to track rigid bodies with six degrees of freedom and, 2) to track non-rigid body motions such as water waves running up on structures and beaches. These considerations eliminate many existing systems that can not be modified by the end user. For example, systems that track rigid bodies often use special targets or lights attached to the body. Deformable bodies, like water waves, can not be tracked in this manner. Most of these systems are also turn-key systems and can not be modified for future applications, since no source code is provided. Cost is also a major factor in eliminating many options. Costs over budget ranged from $15,000 to $300,000 for a complete system in 1989.

In addition to the operational requirements of the tracking system, several other features are important. The use of standard hardware is desired so that initial costs and replacement costs are minimized. The ability to easily upgrade hardware is important. Access to software source code is essential. However, the most important factor is the ability to satisfy the operational requirements.

Standard hardware is desired for several reasons. The cost of variable scan rate cameras and other specialized hardware is the major limiting factor. Most standard hardware is easily operated and does not require special training. Standard video equipment can also be used as a simple video production and editing system for qualitative documentation for clients.

Standard video and computer hardware allows for simple upgrades. Numerous video suppliers are available with continued improvements in products. Video is also the standard input format for most image processing systems. The image processor selected should be easily upgraded for higher resolution, speed and additional functions.

By assembling a custom software system, access to source code is assured. Most specialized turnkey systems do not provide source code for future in-house modifications. This algorithm source code access is a critical factor because of the various applications anticipated at the Wave Research Laboratory.

All of these considerations were taken into account when seeking the solution to the proposed object tracking problem. Cost and the ability to achieve the project goal received the most weight in the decision.

2.2 System Development

The system was developed in several stages. The initial concept was defined and video was selected as the appropriate media. The method of timing, or frame synchronization, was also selected. Frame accurate computer VCR control was not economically feasible at that time, however. A basic one camera system was initially acquired. This was composed of; 1) a shuttered CCD camera, 2) S-VHS VCR, 3) SMPTE time code reader-generator, 4) RGB video monitor, 5) Time Base Corrector (TBC), 6) host computer, 7) image processor.

Familiarity with the system was gained and simple programs were developed. Programs to manipulate images and perform image processing tasks were written. The ability to discriminate and locate an object in the image was also developed. Other support programs such as image transferring routines, from host to image processor, were written as well.

These image transfer and object location programs demonstrated the significance of host speed. The original host computer, an IBM AT, was slowed significantly by the image transfer process. Because of slow speed and limited memory a faster host with more mass storage was acquired. Though real time image transfers were not possible, host-image processor communications times were greatly reduced.

The new 486-DX host allowed reasonable tracking of an object with a 6 sample per second (Hz) rate. Many of the same programs from the original host were used. A user friendly, windowed image processing environment was developed to allow others to use the system easily.

The 6 Hz. sampling rate is not as fast as the 30 Hz video rate which was the goal. By performing tests with other hosts it was determined that the image processor itself could not sample the video at 30 Hz in real time.

Having that knowledge, a second VCR for the 2 cameras system was researched. It had to be computer controllable, frame accurate, S-VHS, and have SMPTE time code capability. A VCR with these features had become available, it also had a built-in TBC and several other useful features. This was acquired along with the interface to allow complete remote computer control. The interface allowed commands to be given to the VCR, then status and position were returned to the host computer. With the controllable VCR and interface, automatic object tracking became possible.

The remaining hardware was then acquired. This consisted of a second CCD video camera, and a monitor. Two channels of video could now be recorded and viewed simultaneously. They were also synchronized together with a common time source.

The final task of algorithm development was then begun. Calibration and 2-D location theory was developed and accompanying programs written. Because of the complexity, the algorithms were written in a simple but limited language, and the results of each step were checked. After satisfactory results were obtained from the basic algorithms the final C programs were written. Intermediate results for each C program developed were verified with the previous method results.

The system was used in several two dimensional tests and the results were verified with alternative measurement methods. An experiment was devised to calibrate and test the three dimensional algorithm. The experiment was performed and measurements made. An existing measurement method was used to verify the results and evaluate the magnitude of the errors. These tests are described in chapters 5 and 6 of this thesis.

3.0 HARDWARE DESCRIPTION

3.1 Overview

The object tracking system used at the O.H. Hinsdale Wave Research Laboratory can be separated into three main categories: 1) image processing hardware, 2) video hardware and 3) host computer. Each of these categories have their own vocabulary, components specifications and purpose in the system.

The image processing system in the configuration tested has the ability to track moving objects in real-time at a maximum rate of 10 Hz. However, this is a highly idealized condition. A more realistic rate is approximately 4-6 Hz., depending on the camera settings, lighting conditions and complexity of the target recognition algorithm. The maximum sampling rate is 30 Hz, at which speed NTSC RS-170 video operates. This is done by processing the video tape frame by frame. The computer is capable of controlling the speed of the VCR and processes a single frame, then advances the tape and processes the next frame. This is automatic after the initial target is located with a mouse operated cursor box. The exact time of each frame is also acquired.

Although more than one object can be tracked, each must remain sufficiently separated so that the system can determine which pixel group is associated with each object. This is the only limit to the number of objects that can be tracked, in the pseudo real-time mode.

The image processor has a resolution of 512 pixel elements in the horizonal and 512 in the vertical; however, only 480 pixels in the horizontal are useable due to the RS-170 video standard. Another consideration with NTSC RS-170 video is the 4H:3V aspect ratio. The horizontal pixel size is one-third larger than the vertical size. Each pixel can have a grey value between 0 and 255, where 0 is black and 255 is white. Input video is RS-170 compatible, consisting of 30 frames per second. Each picture, or frame, consists of two fields, even lines and odd lines. These fields are interlaced to form one frame.

Since the resolution is fixed at 512 x 480, the only variable is the field of view. The cameras have variable zoom lens in which magnification power can be varied from 1 to 8, this results in a field of view range from 44.3° to 6.1° in the horizontal. The vertical field is 39.9° at wide angle and 4.6° in telephoto. The vertical dimension is the limiting field size. With a small field of view, the motion is divided over more pixels, giving a better effective resolution. By using a wider angle setting there is a trade-off between the magnitude of motion and the resolution that can be measured; more motion is viewable, but each pixel represents a larger distance.

The system consists of eight major pieces of hardware:

1. image processor
2. host computer
3. video cameras
4. video recorders
5. time base correctors
6. time code readers/generators
7. video recorder computer control hardware
8. video monitors

A photo of the hardware system appears in Figure 3.1. The image processor is not shown, as it is in a separate room for sound isolation. The host computer is at the left, the monitors are stacked to the right of the host computer. The host computer monitor is in the center. Three video recorders appear to the right of the computer monitor. The computer controllable VCR is on the bottom, with two additional S-VHS VCR's stacked above. The remote control units for both cameras appear in the top right corner.

3.2 Image Processor

The image processor is a Series 151 from Imaging Technology Incorporated (ITI). Imaging Technology can be contacted at:

Imaging Technology Incorporated
600 West Cumming Park
Woburn, Ma 01801
1-800-532-3500

The image processor consists of individual processing boards or modules which are attached to a VME bus back-plane which is linked to a host computer (IBM or SUN) by a host translator card.

The IBM host translator card is based on the 16 bit, 8 Mhz Industry Standard Architecture (ISA) bus. This is the slowest link in the system. All data must pass through this bus, and large amounts of data cannot be passed at video frame rates. The SUN uses a modified translator card. It is a 32 bit connection and has a faster data transfer rate. When benchmark tests were performed on both

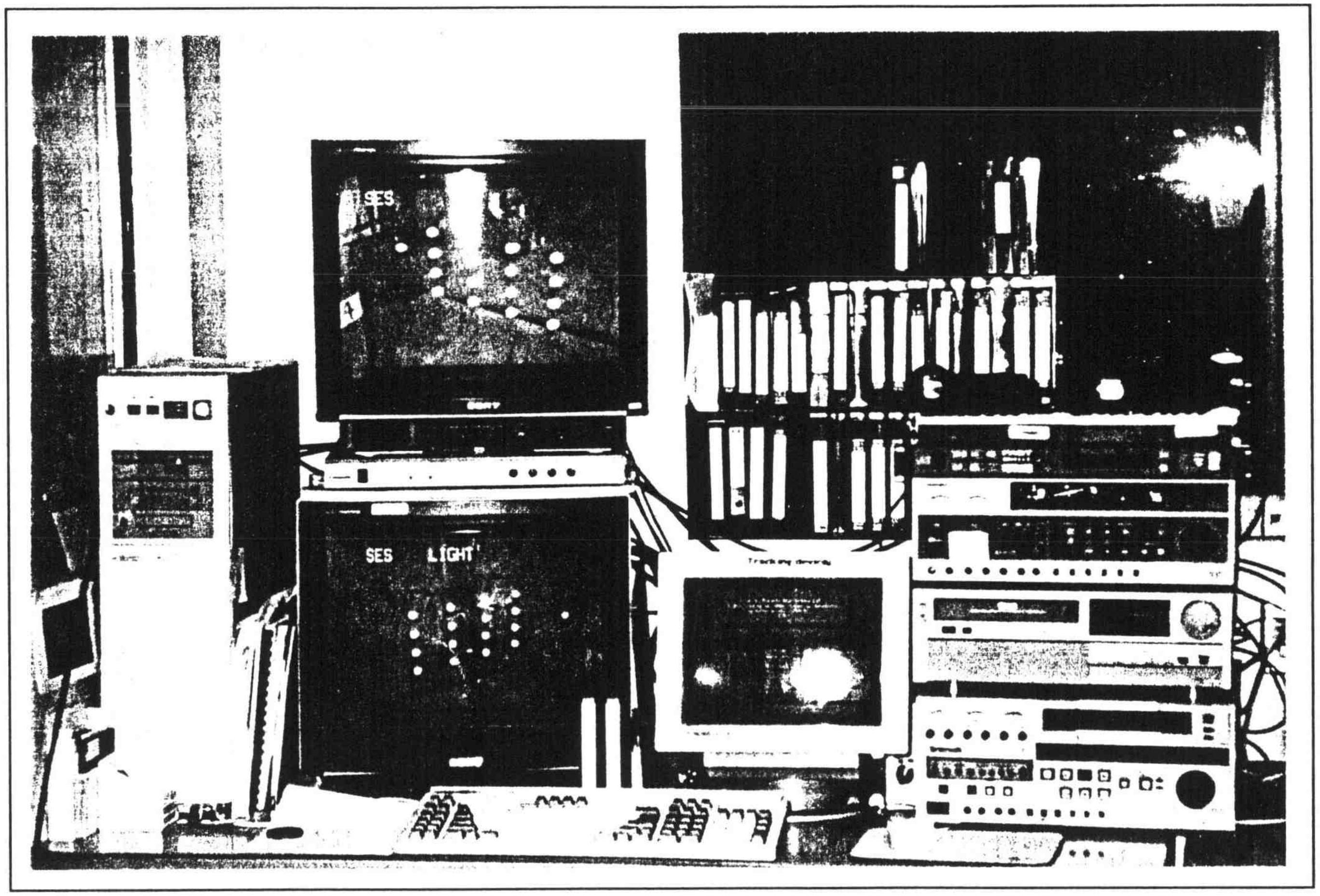


Figure 3.1 Selected hardware components of video image tracking system

systems, the SUN produced faster times but real time video transfer was still not possible.

The Series 151 image processor has a serial data path for video data with host access to all boards individually. The video bus transfers digitized video images between the modules in a pipeline or daisy-chain fashion. The O.H. Hinsdale Wave Research Laboratory image processing system configuration is shown in Figure 3.2.

The video bus uses a proprietary connection to speed video transfers. The video bus is comprised of five individual data buses:

1.	VDI	Data input from the interface module
2.	VDA	Data from frame memory A
3.	VDB	Data from frame memory B
4.	VPI	Pipeline data bus
5.	OVR	Overlay and control data bus

The VDI bus carries digitized data and timing from the interface module to other modules. The interface module converts the input signal from analog to digital and it also inserts clock timing and synchronization for the entire system.

The VDA and VDB are data buses which transfer complete images from the frame memory to all other modules for processing. The image processor has three frame memories FRAMEA, B1 and B2. FRAMEA is a 16 bit image and B1 and B2 are 8 bit images. FRAMEA can be divided into two 8 bit images, ALOW and AHIGH.

The VPI bus connects the outputs of each module to the input of the next module and transfers either 8 bit or 16 bit data.

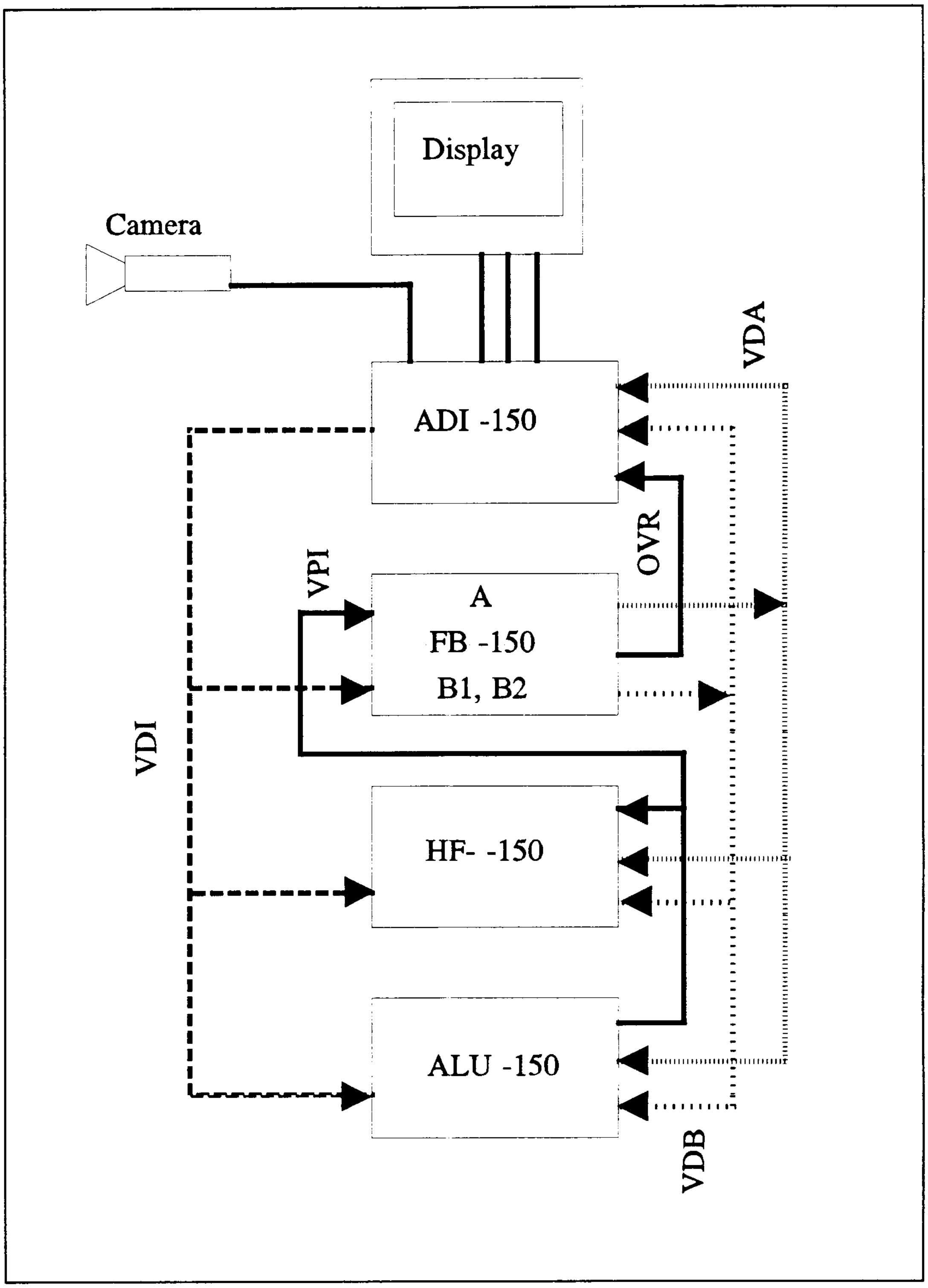


Figure 3.2 Video data pathway in the ITI-151

The OVR bus passes system control information and also enables color overlays on B2. When B2 is used for color overlays it is not able to display video images.

Timing and synchronization can come from the video source or can be user defined. Programmable timing allows the user to define a reduced Area-Of-Interest (AOI). This allows for a faster rate of data transfer and reduces the amount of data transferred. The AOI time base cannot be used for display, and any AOI processing must be completed then timing is returned to normal for display purposes.

The ITI-151 system is composed of four processing modules, with expansion room for seven more.

ADI-150	Analog/Digital Interface
FB-150	Frame Buffer
HF-150	Histogram/Feature extractor
ALU-150	Arithmetic Logic Unit

3.2.a ADI-150 Analog/Digital Interface

The ADI module accepts video input one of four video inputs. The module interfaces to RS-170 or RS-330 video cameras, video recorders and monitors. The module has user adjustable gain and offset circuitry to allow the signal to be adjusted to suit specific needs before processing. The gain amplifies or attenuates the incoming signal. The offset shifts the signal level from zero to minus one volt. The signal is then passed to the analog to digital converter (A/D) where it is digitized into a 8 bit image, or 256 possible gray scale levels. At this point the digitized image can be directed to the digital to analog converter (D/A) for unprocessed display, or passed to the next stage in the ADI processor.

The next process modifies the digitized image by mapping input values to output values by use of Look Up Tables (LUT). There are sixteen user modifiable input LUT 's. The LUT 's can be linear, inverse, logarithmic, exponential or user specified configurations. Through the use of look up tables, frame rate processing can be attained. Only simple modifications can be done with the LUT 's such as stretching the image to use all of the 256 values as opposed to only a small percentage of them. This enhances the contrast of the image. The image is then passed to the VDI video pipeline.

After the other system modules have processed the image, it is passed back to the ADI for D/A conversion to an analog display signal. Before it is converted to analog it is passed through output LUT 's. There are three output channels which each have 16 programmable LUT 's, however only one LUT can be used at a time.

If all three channels are modified in the same way, the display remains in a gray scale. If they are given different configurations, colors appear. These are pseudo-colors, not true colors. These colors are only displayed to the monitor and not to the image processor. These are useful for text overlays and other user operations.

The output image can be recorded by a video cassette recorder (VCR), but color is not available unless an RGB VCR is used. Video synchronization is supplied on the green channel and is also provided on a separate output.

3.2.b FB-150 Frame Buffer

The FB module provides image storage space. The image storage does not occupy host CPU space and allows real-time access to the image. The image is accessible to the host CPU, but is not retrievable in real-time. The 151 supports up to four FB modules for increased image storage.

Three frames can be stored at one time. These include one 16 bit image and two 8 bit images. All frame memories are 512 x 512. Due to RS-170 specifications only 512 horizontal x 480 vertical pixels can be displayed. The 16 bit frame memory, FRAMEA, functions as an image accumulator from frame subtractions and summations. These and some other operations yield results which are greater than eight bits. Frame store B2 is the only frame which can be used for overlays. If graphic overlays are needed B2 cannot be used for other processing.

The frame memories are addressed using a horizontal (X) and vertical (Y) convention. The upper left hand corner of the video screen is the origin (0,0). The FB stores video images in an interlaced format as it is transmitted by RS-170 video equipment. Interlacing is explained in Section 3.4.

Frame memories B1 and B2 receive input from the VDI bus. Either memory can be masked to prevent input. B1 and B2 output data onto the VDB bus, and connect to other modules for further processing or display. Either B1 or B2 can be displayed, or both can be multiplexed onto the bus.

Frame memory FRAMEA receives input from the VPI bus. This frame memory outputs data onto the VDA bus and is connected to all other modules for processing.

The FB controls the starting point of the frame memory being displayed or processed. The section of memory being processed or displayed is called the Active Video Window (AVW). The software adjustable pan and scroll values set the starting row and column of the AVW. FRAMEA, B1 and B2, each have a pan value and a scroll value. This also defines the starting point for AOI processing.

As video data is passed from one module to the next, a delay is introduced by processing. The FB can perform manual or automatic spin compensation for cumulative delay. Each horizontal line is shifted to the right a fixed amount of pixels corresponding to the spin or delay time.

3.2.c HF-150 Histogram/Feature Extractor

The HF module performs real-time analysis of pixel intensities on an image. It calculates histogram information describing brightness, contrast and dynamic range of the image which can be used for intensity stretching and lighting compensation. The other mode is feature extraction. These two modes are mutually exclusive and must be declared before a frame is acquired if a change is desired.

Feature information accelerates dimensional calculations and measurements. Sixteen features can be defined, where a feature is a single pixel value or a range in pixel values. The HF module can then extract 16,000 pixel (X, Y) locations which correspond to the desired value or range, from each image. This can be done in real-time. However, reading these values into the host CPU is done through the host bus. This is the sample limiting condition, determining the acquisition rate for the entire process.

Input to the HF is selectable, using software from any of the four video data buses. The VPI passes video through the board to the next module. Extracted data is accessed by the host computer. Two data buffers on the board allow one buffer to be acquiring data while the other is being read by the host.

In the feature extraction mode 16,000 features can be located but, due to IBM-AT memory addressing limitations, only 10,000 features can be accessed in one block. With the tested configuration, more than 1000 features slows down processing considerably. The best results are obtained when less than 200 pixels are located. A target can consist of one pixel, but more are preferable so there is less chance of losing the target. The centroid can be calculated and a single location for the center of the target obtained.

3.2.d ALU-150 Arithmetic Logic Unit

The ALU performs many common image processing functions with 16 bit accuracy. Image thresholding, subtraction, additions and averaging can be performed. Minimum and maximum pixel values can be determined and conditional processing can be performed. Some operations can be performed at frame rates. The ALU has two sets of eight programmable 256 byte LUT's.

Microsoft C was used as the software development language. Imaging Technology provided low level C libraries for register addressing and simple functions on the ITI-151.

3.3 Host computer

The host computer is an IBM compatible PC. It is a Northgate 25 Mhz 80486-DX operating under Microsoft DOS, version 5.0. Storage devices include 1.2 Mb and 1.44 Mb floppies, 16 Mb of RAM, 200 Mb hard disk, and a 160 Mb internal tape backup. Northgate can be contacted at:

Northgate Computer Systems
7075 Flying Cloud Drive
Eden Prairie, MN 55344
1-(800)-548-1993

The image processor is interfaced to the host PC with a standard AT ISA bus card occupying one expansion slot. Two 64 wire cables connect the card to the image processor. The bus runs at 8 Mhz and has a 16 bit data path. The image processor is capable of a 32 bit data path, with a SUN computer.

3.4 Video Cameras

Two Panasonic color video cameras are used with the system. Camera one is a model WV-3260, camera two is a model WV-D5100, both operate in a similar manner, age being the significant difference. Panasonic can be reached at:

Panasonic Industrial Company
1200 Westlake Ave., N.
Seattle, WA 98109
(206)-285-8883

Each camera has a low distortion Charged Coupled Device (CCD) image sensor. The image sensor is 2/3" type, and has 286,000 picture elements and has 460 lines of horizontal resolution. Vertical lines of resolution are not reported in NTSC cameras. Both cameras have a strobe effect shutter, which is used to give blur free recording of high speed action.

The camera outputs standard NTSC composite video and stereo sound. The video records 30 frames per second. Each frame is the combination of 30 even lines and 30 odd lines which are interlaced to form a single picture. Since the odd lines and the even lines are not scanned at the same time, this causes blurring unless a shutter is used. The shutter darkens the image by reducing the available lighting.

Both cameras have 8X zoom lens, with an automatic or manual iris. The focus is also either automatic or manual. The cameras can be mounted on a pan and tilt head. This head tilts 30 degrees from the horizontal and can pan 350 degrees and all of these functions can be analog controlled remotely up to 300 feet away.

3.5 Video Cassette Recorder

Two Panasonic video tape recorders (VTR) are used to record the output from the cameras. Both are SuperVHS, which have 480 lines of resolution. They can be used with standard VHS tapes but the resolution drops to 290 lines.

The simpler VTR is an AG-7300 which is primarily used for recording of events. Since the image processor can only use one input at a time, tapes must be analyzed sequentially, rather than simultaneously. This VTR is not computer controllable.

The AG-7750 has more features than the AG-7300. It has a built in Time Base Corrector (TBC), a built in time code reader/generator, is frame accurate and can be computer controlled. This allows for automatic processing of tapes and exact timing information to be extracted for multiple tape correlation.

Most image processing cannot be accomplished at video rates (30 Hz). Video tape recorders do not give a clear jitter free picture in slow motion picture. The image processor cannot lock on to these pictures and give acceptable results.

3.6 Time Base Corrector

A Time Base Corrector (TBC) repairs a damaged video signal by removing noise, jitter and image distortion from a VTR source. This is achieved by inserting a new synchronization signal on the video. The TBC for the AG-7300 is a Panasonic model TBC-100 and is external. The AG-7750 has a built in TBC. Both Time Base Correctors work in a similar manner and provide 16 lines of correction. The first 16 lines of the frame are adjusted and this is where image distortion and timing is critical. Full frame correctors are significantly more expensive.

3.7 Time Code Generator and Reader

All video tapes used are encoded with the Society of Motion Picture Time Encoding (SMPTE) process. This gives each of the 30 frames per second its own time stamp. The format provides for hours, minutes, seconds, and frames (1/30 of a second increments). This time can be synchronized with WWV national atomic time standard with a special hardware adaptor.
One time code generator/reader is a Fast Forward model F22, the other is built into the Panasonic AG-7750. Fast Forward can be contacted at:

Fast Forward Video
18200-C West McDurmott
Irvine, CA 92714
(714) 852-8404

Longitudinal time code is used on both VTR's. There are two time code reader/generators that can be preprogrammed to begin at any specific time. However, they are not true real time clocks and lose or gain several seconds over a 24 hour period.

If one time code source is used for both cameras, they must be genlocked together to a common timing source to assure proper syncing. If the two cameras are not properly synchronized an offset of up one-half frame may occur.

3.8 Computer VTR Control

The VTR control hardware, V-Lan, is capable of controlling several video products by computer. It is manufactured by Videomedia

Videomedia
211 Weddell Drive
Sunnyvale, CA 94089
(408) 745-6721

The V-lan hardware translates computer instructions into VTR compatible instructions which allows commands to be sent between the computer and the VTR. Most commands are processed in less than 10 milliseconds. Commands can be simple such as play, fast forward, stop, review, shuttle, or complex operations such as going to a specific frame.

With the ability to read time code into the computer, multiple pass pseudo real-time automatic tracking is possible. The tape can be ejected at the end of the process and the user prompted to insert the next tape.

3.9 Video Monitors

Two Sony model PVM-1910, 19 inch color monitors complete the system. Each monitor has multiple inputs which can accept two composite signals, an RGB signal and a computer signal. Since the input can be selected on the front panel, both cameras can be monitored simultaneously or the processed image and the original image can be viewed. Sony can be contacted at:

Sony Corporation of America
Sony Drive
Park Ridge, NJ 07656
(201) 930-1000

4.0 THEORETICAL FORMULATION

4.1 Overview

Computer vision systems are modeled after the human vision system. The human vision system is composed of two cameras and a complex computer. Its function is to extract information from the surroundings. This occurs in a non-contact manner; no physical contact is required. It is because of this, that some consider sight the most powerful human sense.

As with any model, simplifications must be imposed and limitations accepted. A computer vision system cannot duplicate human vision for versatility and adaptability. On the other hand, computer vision can exceed human vision in some aspects such as measurement accuracy, numerical precision and speed at specific tasks.

In the following sections, the theoretical basis of the vision system is explained. Monocular and binocular vision are discussed along with calibration. The computer in the human vision system is extremely complex and is capable of using one camera or two, but it must be calibrated. This takes about 5 years of running into and falling on things. It can take another 10-15 years to fine tune the human vision system. The computer system can be calibrated in a few seconds, but is limited to a stationary position. Computer vision shares many basic principles with human vision.

A fundamental principle in computer vision is the pinhole camera model. The assumption is that the camera is represented by a box with a pinhole in one end and a flat plane at the other. Light enters the hole and hits the plane on the other end of the box. Light rays are not bent or distorted by the pinhole. The light rays are then projected on the plane. This is referred to as the perspective transformation.

The perspective transformation, or imaging transformation is the manner in which light rays from three dimensional objects are projected onto the image plane; it is the mapping of three dimensional points into two dimensional space. This creates a problem since information is lost in the mapping. The inverse mapping, from 2-D to 3-D, cannot be performed due to the loss of information. Depth, or distance to object is not resolvable without additional information.

The perspective transformation allows light rays, or points to be mapped from one coordinate space to another. This introduces the next problem, relating one space to another. In a vision system there are several coordinate systems that must be related by transformation and rotation parameters. Each camera has its own coordinate system, the image processor has at least two of its own coordinate systems, and the world has a separate coordinate system. A mathematical relation has to be determined to allow point location to be expressed in any of these coordinate systems. This is done predominately in the calibration process.

As with human vision, computer vision can be performed with one or two cameras. When the human vision system uses only one camera, it can compensate for the loss of information using familiarity with the current situation. This is something a simple computer system cannot do, a simple graphic example is to close one eye and try to function; accurate depth perception is lost. The same is true for computer vision using one camera.

When both cameras are used, accurate depth perception is available, because of the parallax that exists between the two different views. A stereo vision system uses two cameras and a 3-D camera model to facilitate depth perception. Two cameras permit accurate 3-D world measurements without the need for additional information from the environment.

The one camera, 2-D model, uses simple geometry and information about the world to get absolute position of objects. The one camera model is also simple in application since only one camera and support hardware are required. Only one view must be processed to get position information. The model is excellent for motion of objects that are planar, such that a camera can be positioned with its image plane parallel to the object motion. Calibration of this camera is also simple.

There are limitations with the 2-D model, however. Inverse mapping of points from 2-D to 3-D coordinates is difficult or impossible. True world positions can be obtained if something is known about the environment combined with camera positions. Algorithms exist to perform this type of vision and world positioning. The algorithms use multiple pictures from the same camera and a technique called optical flow. Another approach, developed by Holman (1990), uses one camera and other knowledge or approximations about the world geometry to place objects in 3-D space.

The two camera, 3-D model allows for better accuracy and does not require information about the world after calibration, although it uses more equipment and more complex geometry. There are two common approaches to 3-D vision, traditional photogrametry and modified photogrametry for image processing applications.

Traditional photogrametry uses photos taken from a plane flying at constant elevation. The distance between the photos and the elevation is known. Since the plane flew level, the two photos are in the same elevation plane. Extensive knowledge of the camera, such as focal length, lens distortion and location of fiducial marks are available. With this knowledge and a stereo plotter, accurate elevations can be determined.

Most of these conditions cannot be satisfied with a vision, or image processing system. Accurate focal lengths are hard to determine using variable zoom lenses and most video cameras do not have fiducial marks. It is also difficult to determine the exact distance between cameras and to place the cameras so their image planes are parallel. It is for these reasons that a modification of the traditional photogrametric process was explored.

An algorithm was developed to allow unconstrained placement of the cameras, with no requirement for parallel optic axes. The algorithm also accepts standard video cameras with variable focus and zoom lenses. This allows for more flexibility in the experimental environment. A 3-D vision algorithm was available to use two camera views for stereo vision. An algorithm was also available to calibrate the cameras to determine an effective focal length, camera orientation parameters, and other necessary information.

The two camera, 3-D placement algorithm is based on a line projected from the image plane through the lens center into 3-D space. The same procedure is applied to the other camera. The point in space where the lines intersect gives the 3-D location of the object or point. The lines do not always intersect exactly, and the amount of skew is one source of error.

An algorithm suggested by Ayach and Lustman (Ayach, 1991) suggests using a third camera to reduce the error by incorporating redundancy. One disadvantage with this approach is the need for an additional camera to be setup and calibrated and for the additional view to be analyzed. If calibration is performed carefully and accurately the third camera view is not necessary for acceptable results.

Calibration must be performed for both the one camera and the two camera model. Calibration of the cameras is the most crucial step in the vision system. From calibration, a relationship is established between the camera's 2-D image coordinate system and the desired reference frame, usually the 3-D world coordinate system. Transformation parameters are determined by calibration, which express the distances between the camera coordinate center and the world coordinate center. A rotation matrix is determined which is a function of the tilt, roll, and pan angles of the camera, relative to the world coordinate system. In addition, effective focal length, lens distortion, and camera-image processor scan error are determined.

Calibration is achieved by knowing the location of a set of points in the world coordinate frame. The location of the same points in the 2-D camera coordinate frame are also known. From this, the rotation matrix and translation matrix are determined along with the other camera system parameters.

4.2 Two Dimensional Positioning Theory

The 2-D positioning theory is based on simple geometry. The target is located on the computer screen, and the (X,Y) location is extracted. Using a predetermined calibration coefficient, these (X,Y) values are scaled to determine the true world (x,y) location.

Target location is performed by the image processor, and the software residing on the host computer. The video picture is digitized, first in time, into 30 frames per second, then in size, 512 pixels horizontal by 480 pixels vertical and into 256 grey levels.

The image processor thresholds the image, or turns it into a binary black and white image. This is displayed for the operator, since humans have difficulty differentiating more than 32 grey levels. After the image is thresholded, the target is selected by placing a mask or box around it. Any pixels that are above the threshold intensity within the box are considered part of that particular object.

Target pixels in the box are counted. Their screen X and Y locations are summed and their relative screen centroids are determined according to the standard formula:

$$\bar{X} = \frac{\sum_{i=1}^{n} X_i}{\text{total pixels composing target}} \qquad (4.2.1)$$

$$\bar{Y} = \frac{\sum_{i=1}^{n} Y_i}{\text{total pixels composing target}} \qquad (4.2.2)$$

The raw value for the centroid of each target is reported. The number of pixels is reported, and can be used for accurately assessing the centroid, detecting out of plane motion, or discovering lighting inconsistencies. A pixel representation of sphere and the method to calculate the centroid is seen in Figure 4.1. Sub-pixel centroiding is noted in the Y direction, since the centroid is located at a pixel intersection

The X values reported are off by a factor of 1.25, a problem introduced by the image processor. This discrepancy is caused by non-square pixels, due to the image processor resolution and to the aspect ratio of television. The factor is determined according to:

$$\frac{480 \text{ vertical pixels}}{512 \text{ horizontal pixels}} \qquad (4.2.3)$$

$$\frac{4}{3} = \text{Television Aspect Ratio} \qquad (4.2.4)$$

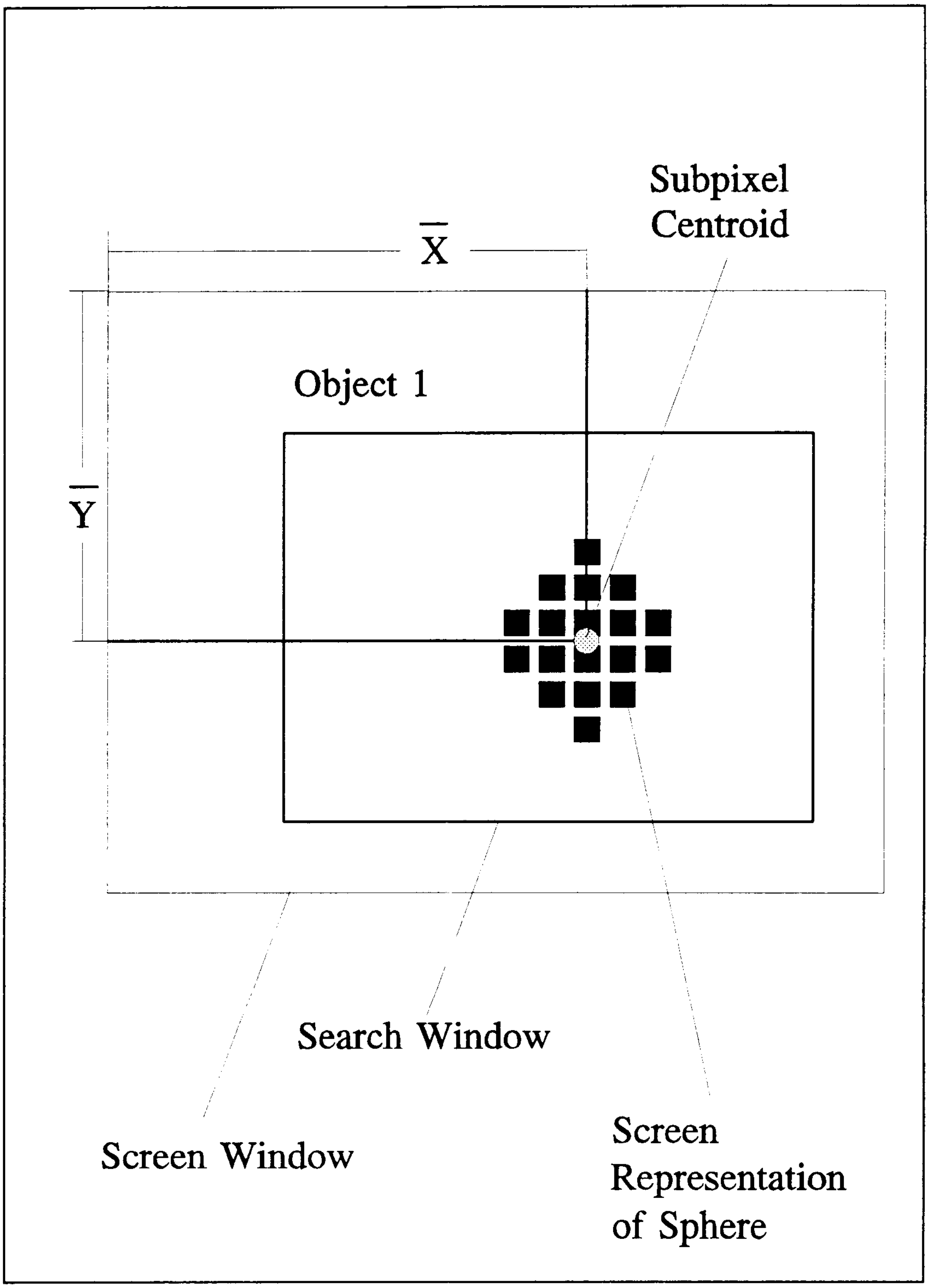


Figure 4.1 Two dimensional representation of a sphere, and sub-pixel centroid location

$$\left(\frac{480}{512}\right)\cdot\left(\frac{4}{3}\right) = 1.25 \qquad (4.2.5)$$

When these two ratios are combined the horizontal adjustment factor, 1.25, results. All raw X values are multiplied by 1.25 in order to compensate for the non-square pixel and aspect ratio problem. This is not the case in the 3-D model, since the algorithm compensates for it.

The (X,Y) screen centroid locations are transferred to 3-D world coordinates by a simple scale factor. The simple scale factor does not include lens distortion or other optical sources of error. It is found by determining the number of pixels spanning an object of known dimension and position in the field of view.

If better accuracy is required, a number of scale factors could be determined, which are a function of screen location. Lens distortion would be correctable in this way. This would require a more complex calibration process but would not compensate for motion out of the plane parallel to the image plane. The theory for the 2-D model is based on traditional photogrametric concepts. The Scheimplug condition is observed "image plane, object plane, lens plane all must intersect along a common line. If image and object planes are parallel, the lens plane must also be parallel, and they have a line of intersection at infinity" (Wolf, 1974). The Scheimplug condition is illustrated in Figure 4.2. This condition must be enforced in the rectification of tilted photos or images. It also must be enforced in true vertical images.

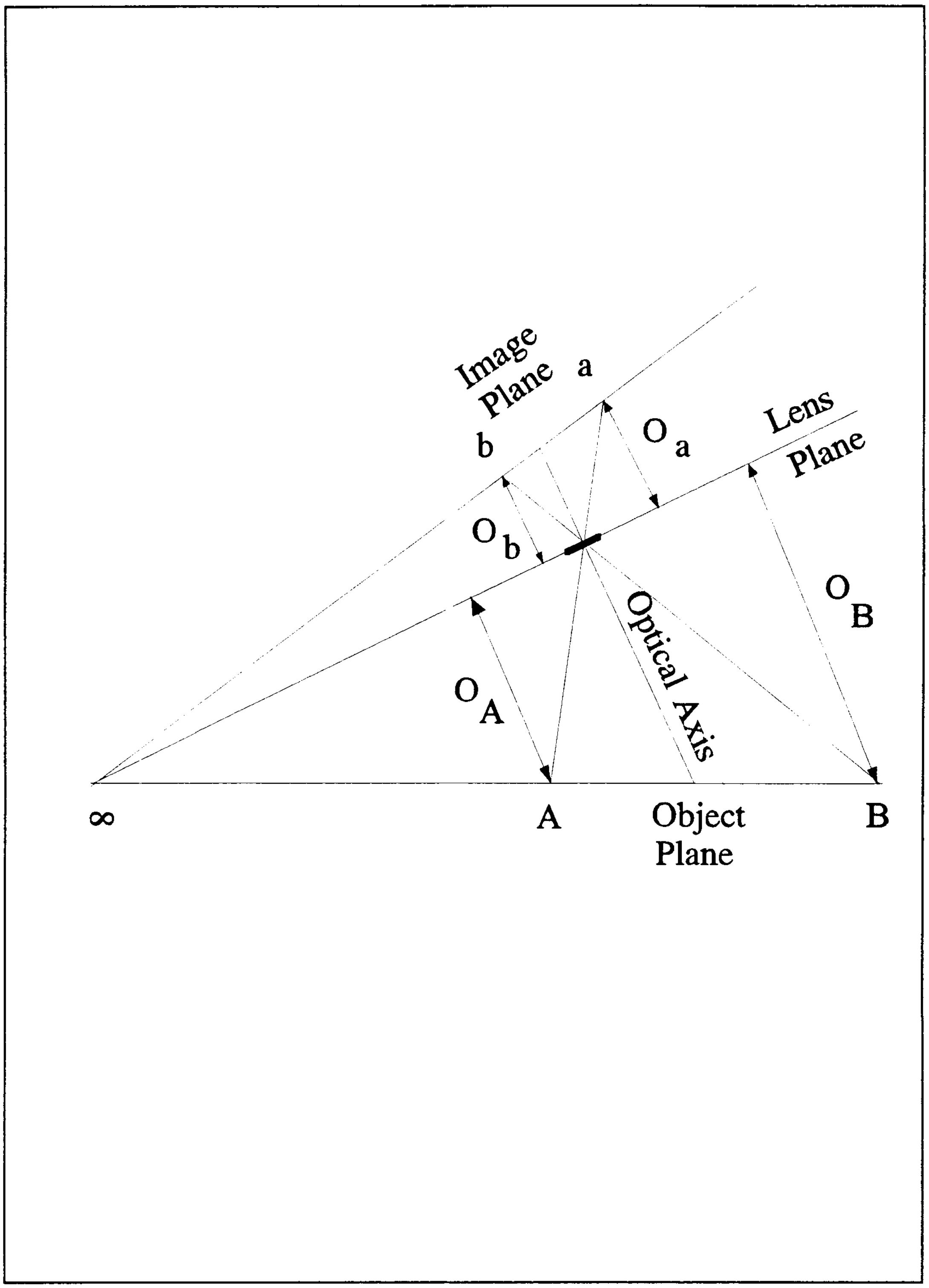


Figure 4.2 The Scheimplug Condition; image, lens, and object planes intersect along common line

When a tilted photo or image is used it must be rectified, or artificially adjusted so that it appears parallel to the image plane. Information is lost due to the adjustment, and the degree of resolution is not constant throughout the photo. Vertical or near vertical camera placement is used, if constant resolution throughout out the image is required.

An image is considered near vertical if the angle between the image plane and object plane (plane of interest) is less than 3 degrees (Wolf, 1974). Figure 4.3 indicates the relative angle of tilt. When the degree of tilt is larger than 3 degrees the image must be rectified. The rectified image can then be processed as a vertical image.

Several requirements must be satisfied before the photogrametric equations can be applied: (1) the image coordinates are corrected for lens aberrations and distortions, (2) the photo coordinate axis system is centered at the image principal point, (3) the focal length of the camera is known, (4) all points of interest are at the same relative distance from the image plane, (5) all point location and target motion is in the plane that is parallel to the image plane. Only targets 3 and 6 satisfy requirements (4) and (5) in Figure 4.4.

If a target is placed at any one of the locations 1-5 in Figure 4.4, its centroid will appear at the same pixel location, I_{1-5}, on the screen. If the target is at 6, in Figure 4.4., its centroid will appear at I_6 which also illustrates the effect of discrete resolution. Because the image is discrete the centroid can only be reported at finite locations. If sub-pixel location methods are used, continuous target locations are reportable.

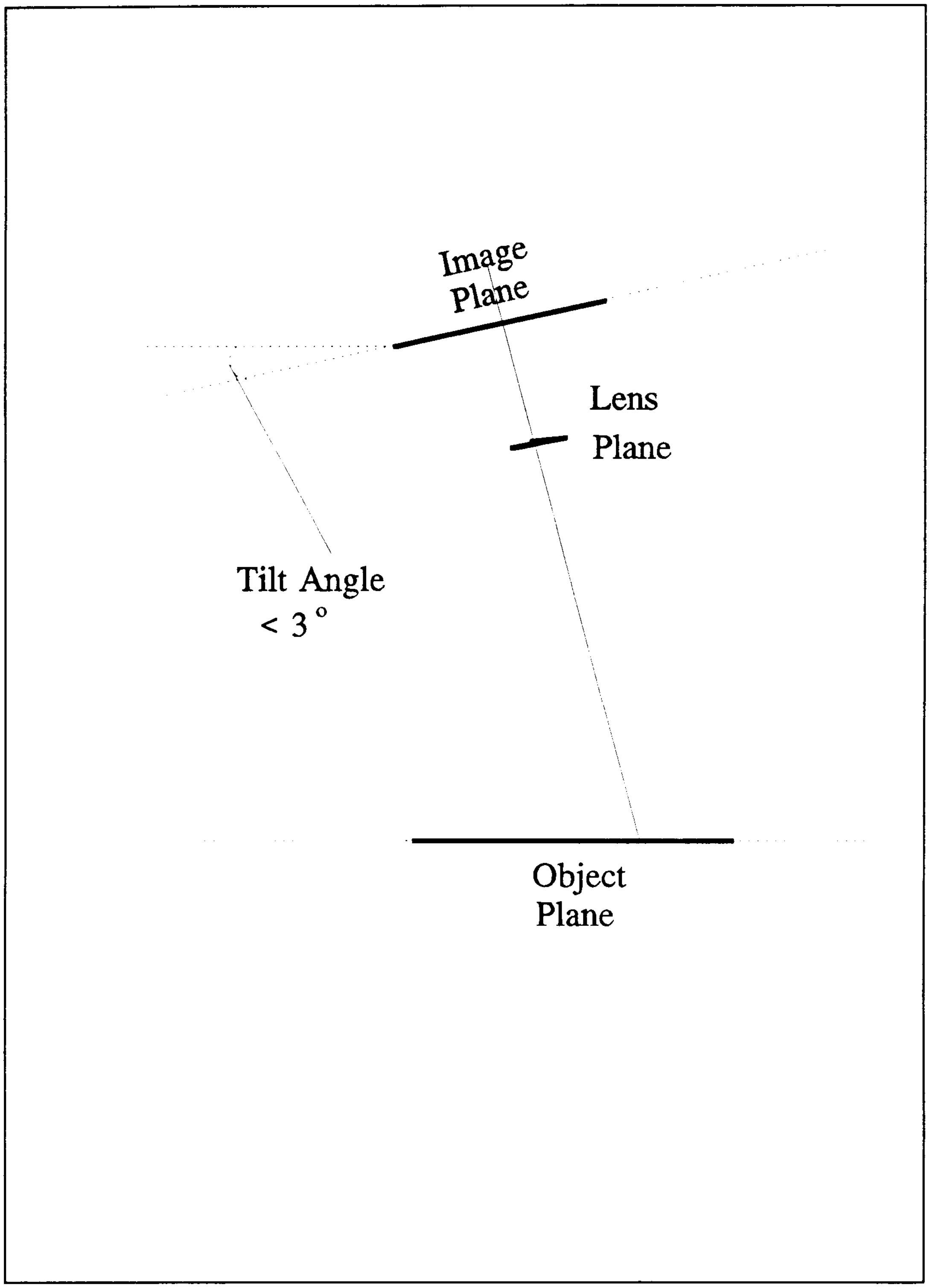


Figure 4.3 Relative angle of tilt between image plane and object plane

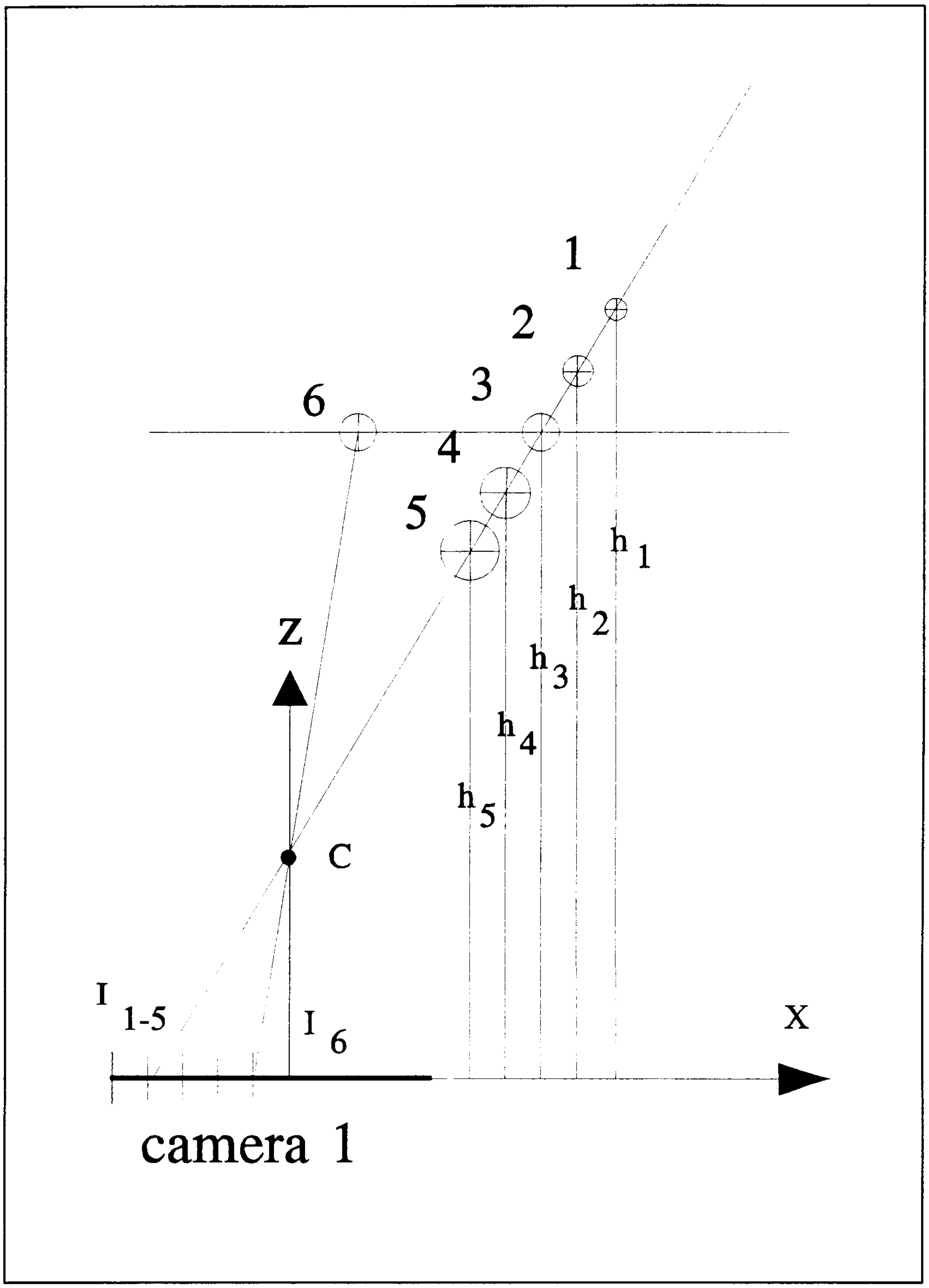


Figure 4.4 Significance of co-planar target position in single camera theory

Another concept is also illustrated in Figure 4.4. Using the target at 1, as a reference size, and moving it along the ray $I_{1\text{-}5}$-C, the apparent diameter of the target on the screen will *increase* as the distance h is decreased. Depth measurement would be possible using a modification of this idea.

If the size of the target is *decreased* as h is decreased the apparent diameter of the target on the screen will remain the same. If the target centroid remains on the ray, its screen location remains the same and its world location is not recoverable.

When these limitations are understood and the conditions are met, the simple formula for the target world location can be applied to the target (X,Y) image locations,

$$x_{world} = X_{image} \cdot \left(\frac{\text{Height (h)}}{\text{Focal (f)}} \right) \qquad (4.2.6)$$

$$y_{world} = Y_{image} \cdot \left(\frac{\text{Height (h)}}{\text{Focal (f)}} \right) \qquad (4.2.7)$$

where Height (h) and Focal (f) are defined in Figure 4.5.

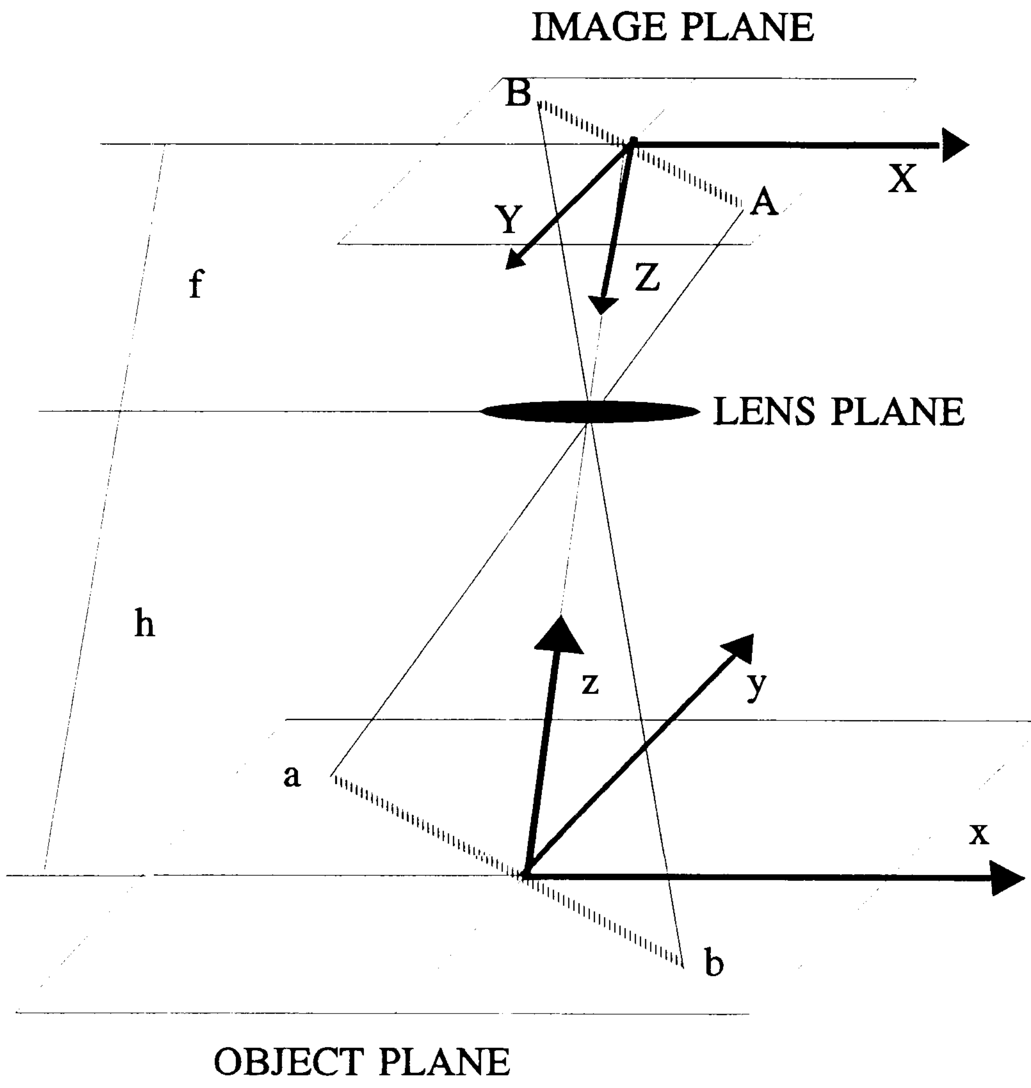
When an image processor is used, some of these requirements cannot be satisfied. The focal length is not known since the relative image plane location is a function of the image sensor size, image processor resolution and the monitor size. Only an effective focal length can be determined. This is complicated by video cameras that have variable zoom and focus ability. Lens distortions and aberrations are usually not given in the specifications of standard video cameras.

The lens center, image sensor center, and the image processor frame buffer center rarely coincide.

These problems require an alternate method of calibration and a modification to the 2-D object location theory. Focal length and camera elevation are not required in this approach. A scale factor is determined, which is equivalent to the height-focal length ratio. This is accomplished by placing an object of known dimensions (ab) in the field of view. The number of pixels that represent the object is determined (AB). The ratio of its length to number of pixels is calculated. This simple formula is equivalent to the traditional height-focal length scaling ratio. Figure 4.5 shows the equivalency of the two methods.

The X screen coordinate is adjusted, first by the 1.25 aspect ratio correction, then adjusted by the scaling ratio. The Y screen coordinates are adjusted by the scaling ratio only. The results are world x,y coordinates.

As mentioned previously, this simple scale factor does not account for radial or tangential lens distortion. It also restricts all motion to a plane that is parallel to the image plane. A complete mapping of the lens distortion could be performed by moving the calibration object to various positions in the field of view. This would reduce the error due to distortion, but would not solve the out of plane motion problem. It is because of these limitations, that a two camera, 3-D model was developed.



$$\text{Scale} = \frac{ab}{AB} = \frac{h}{f}$$

Figure 4.5 Single camera scale determination (calibration)

4.3 Three Dimensional Positioning Theory

The 3-D model uses the same object screen location algorithm as the 2-D model, however, the geometry solution technique and calibration procedure differ. No correction for the 1.25 aspect ratio is applied, since the 3-D model compensates for it internally. No scale factor calculation is required. Lens distortions are accounted for and depth perception is quantifiable.

For accurate 3-D positioning, careful calibration must be performed on the cameras and their relative location. The calibration determines needed information about the camera focal length, distortion parameter and other system parameters that are required for 3-D vision. The accuracy of the 3-D positioning is directly dependent on the calibration which is discussed fully in Sec. 4.4.

The fundamental problem in 3-D stereo vision is multi-camera point correspondence. The same object, or point, must be located in each camera view. This is complicated since the object is not represented the same in each camera, unless the cameras are at approximately the same location. If the object or target is a sphere and illuminated equally from each direction viewed, then it would appear the same from multiple views. Once the object is identified in both images, two lines are projected from the image plane through the lens center. The intersection of the lines in space define the object location in the world coordinate system. The world point, P, and its image representations I_1 and I_2 are shown in Figure 4.6.

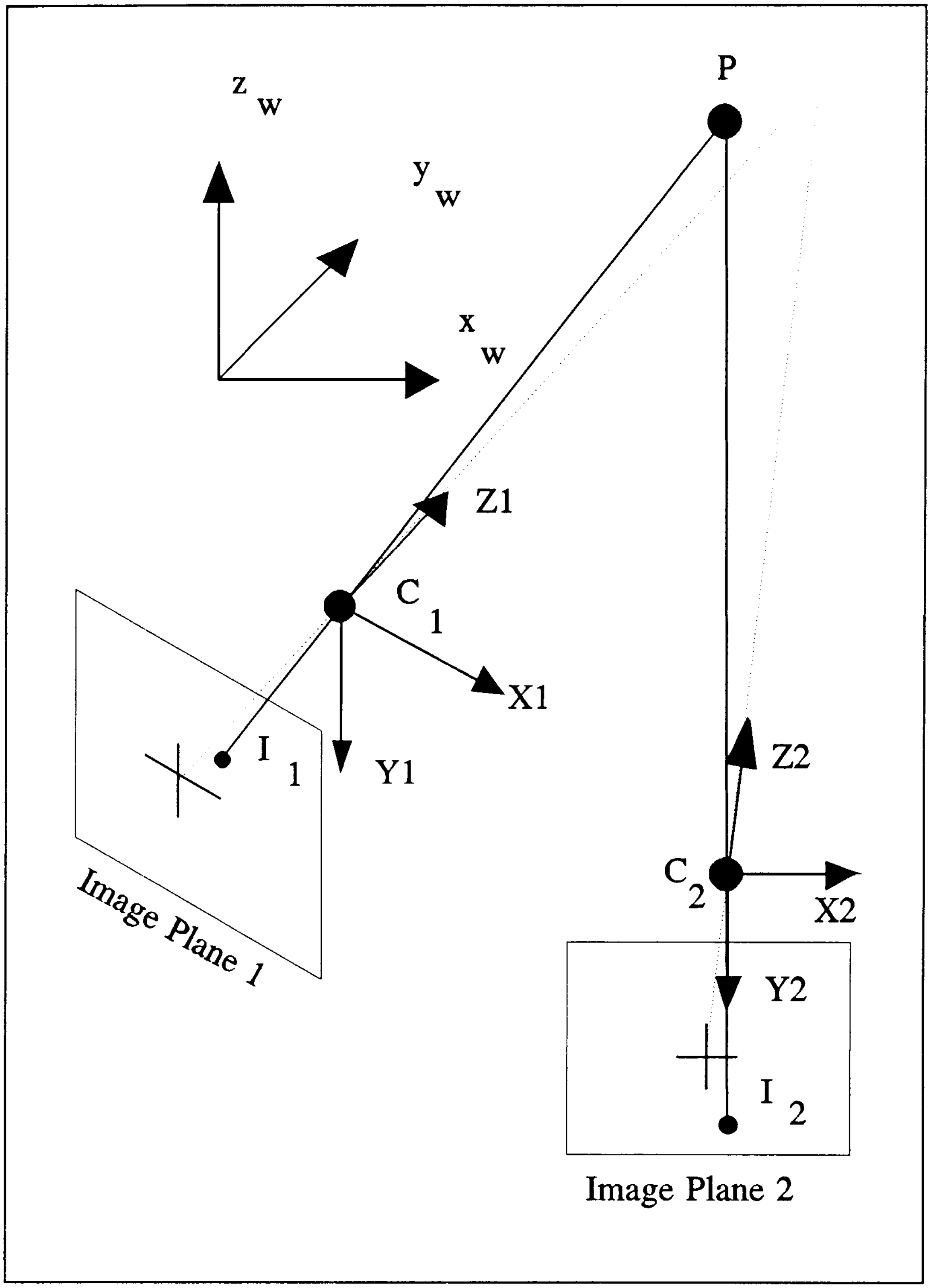


Figure 4.6 Two dimensional image representation and location of 3-D world point

Another factor in 3-D vision is the location of the cameras relative to each other. When the cameras are close together, matching a point in both images is simplified. However, unless precisely parallel optic axes are used the accuracy is reduced. As the angle between the cameras optical axes is increased, the accuracy of the depth measurement increases (EPST, 1987). As the angle increases, point correspondence between cameras becomes difficult to impossible for more and more points due to shading of one point by another. Trinocular vision with three cameras increases the probability that at least two views will see the same target.

In traditional stereography the cameras are set up with parallel optical axes, then simplified trigonometry allows depth to object to be calculated by solving a similar triangle problem. Referring to Figure 4.7:

$$D = \left[BL + x_1 + x_2 - \frac{D \cdot x_1}{f_1} \right] \cdot \tan \Theta_2 \tag{4.3.1}$$

$$0 = BL \cdot \tan \Theta_2 + x_1 \cdot \tan \Theta_2 + x_2 \cdot \tan \Theta_2 - \left[\frac{D \cdot x_1}{f_1} \right] \cdot \tan \Theta_2 - D \tag{4.3.2}$$

$$(BL + x_1 + x_2) \cdot \tan \Theta_2 = D \cdot \left[\frac{x_1}{f_1} \cdot \tan \Theta_2 + 1 \right] \tag{4.3.3}$$

$$\Theta_2 = \mathrm{ATAN} \left[\frac{f_2}{x_2} \right] \tag{4.3.4}$$

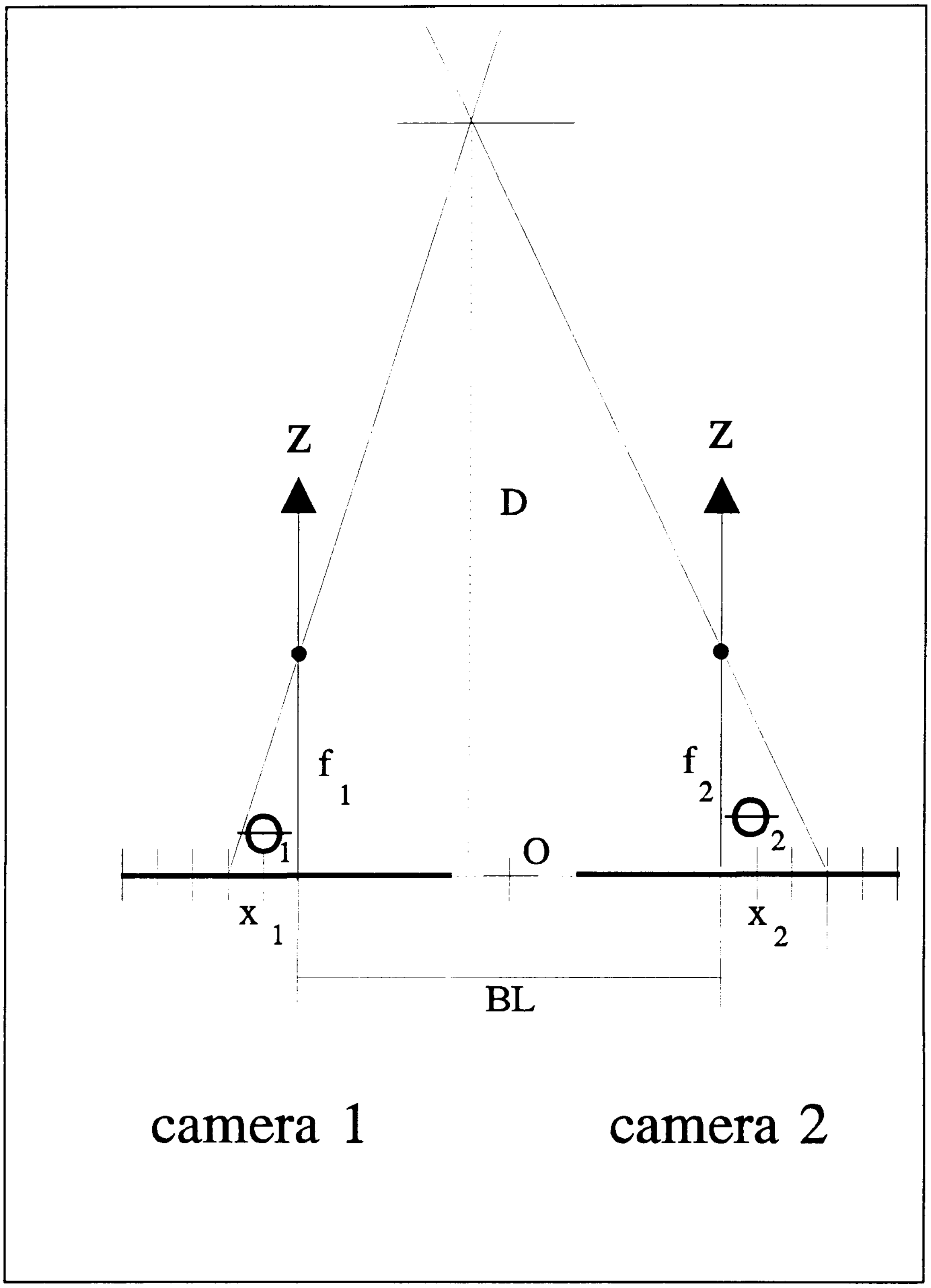


Figure 4.7 Traditional parallel camera axis stereographic geometry

$$D = \frac{(BL + x_2 + x_1)}{\left[\frac{x_1}{f_1} + \frac{x_2}{f_2}\right]} \tag{4.3.5}$$

The notations and orientations are defined in Figure 4.7. The focal lengths of both cameras and the baseline, BL, between the two cameras must be known.

As noted earlier, depth measurement is difficult or impossible using one camera. The second camera provides the additional information required for 3-D positioning. The parallel axis camera configuration is seen in Figure 4.8. Camera one locates the centroid of each target 1-5 at the same screen location while camera 2 gives each target a unique location. Another observation illustrated in Figure 4.8 is that points 1-5 are evenly spaced along the ray, but they are not linearly mapped to the image plane of camera 2. This is due to the pinhole effect on oblique lines or surfaces.

If the orthonormal restriction on the optical axes is removed, an alternate and more flexible method for 3-D vision is possible. It is composed of a three step process; calibration, correlation, and 3-D positioning. "The direction of the two cameras is critical; the more orthogonal they are, the better the accuracy of location" (Saraga, 1977). Parallel optical axes simplify the geometry, but the useable field is limited to the area of overlap of the cameras and obtaining alignment of cameras is difficult.

The 3-D positioning is based on simple trigonometry, with the correspondence problem solved by epipolar geometry. The basic 3-D vision

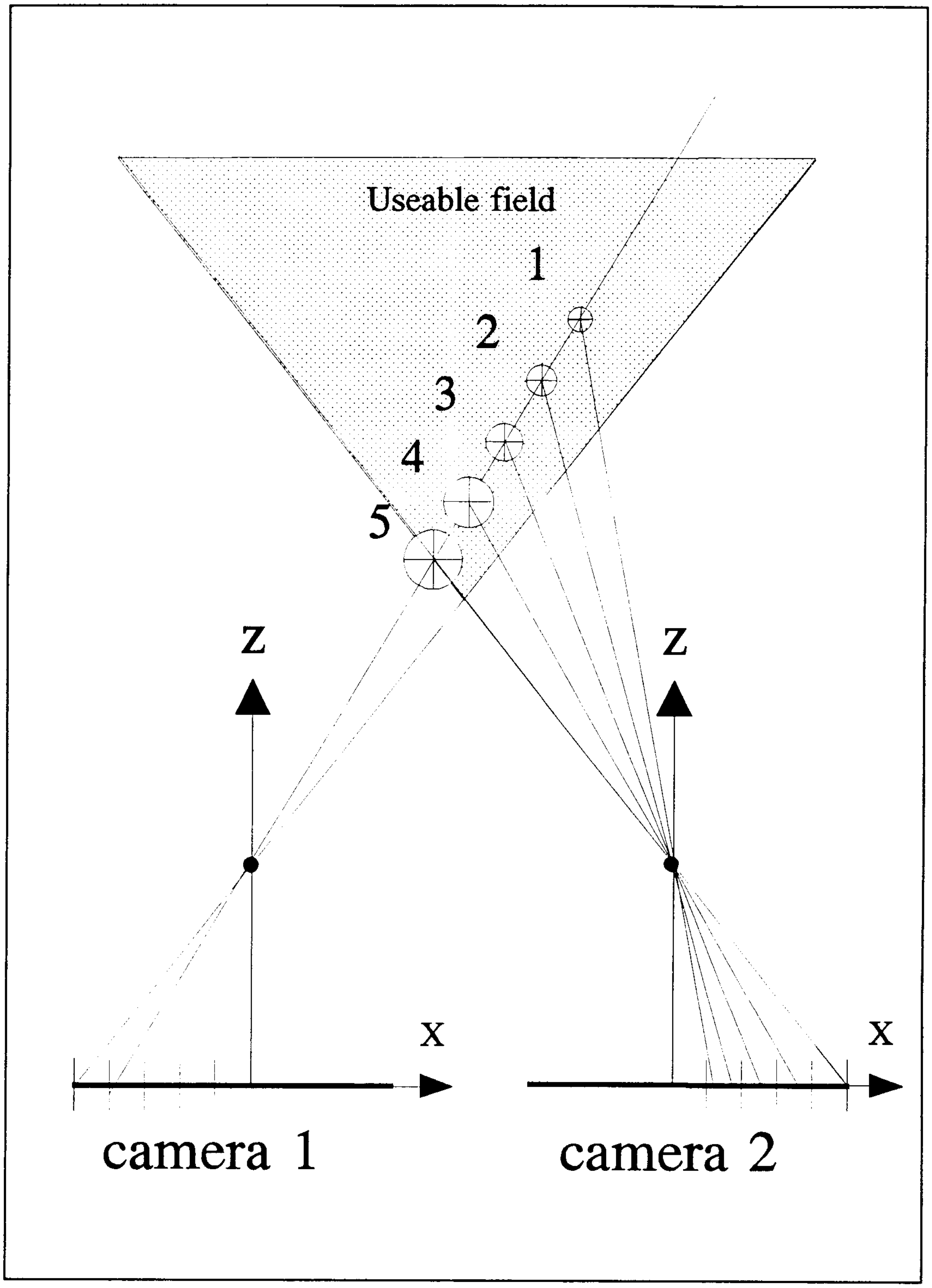


Figure 4.8 Limitation of parallel camera axis stereography

algorithm is adapted from Jen Fen Lee (1987). It was modified to utilize two or more camera views and optimized using a least squares technique.

The cameras are first calibrated individually. The calibration determines rotation R_i, and translation T_i, matrices of the camera with respect to the world coordinate system and the camera interior parameters. The coordinate systems are defined in Figure 4.9. Each camera has a right handed coordinate system centered at the optical center of the lens. The world coordinates system can be at any convenient location, although its origin should not be too near the optic axis (Z) of any of the cameras.

From two images the single 3-D point location is calculated by finding one epipolar, or matching line, in the first image which is then transformed into the second camera coordinate frame. The perspective transform is applied to find the corresponding epipolar line. Using epipolar geometry the correspondence difficulty is reduced. Once the target, or point of interest, is found in the first image the possible location of the same point in the second image is reduced to a line. These lines are referred to as conjugate epipolar lines. Location of conjugate points is accelerated, and potential for error is reduced. This is seen in Figure 4.10, which shows the epipolar plane formed by the triplet of the point of interest, and the center of each camera lens.

Once the 2-D, dimensionless image coordinates of the same point of interest are found in all images, X_{fi}, Y_{fi}, the 3-D world coordinates of the point are calculated by the following procedure:

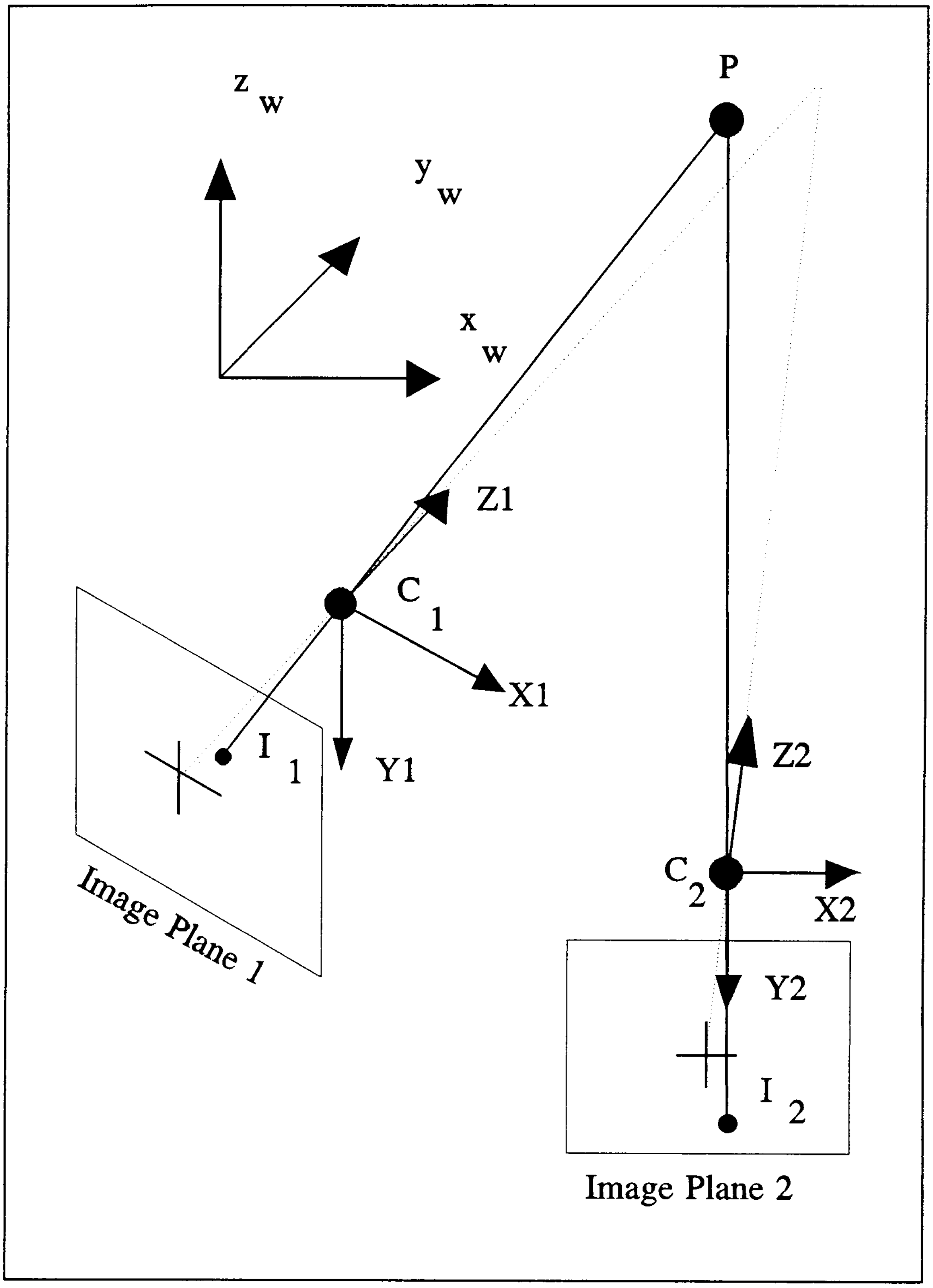


Figure 4.9 World and camera coordinate systems for three dimensional point calibration and location

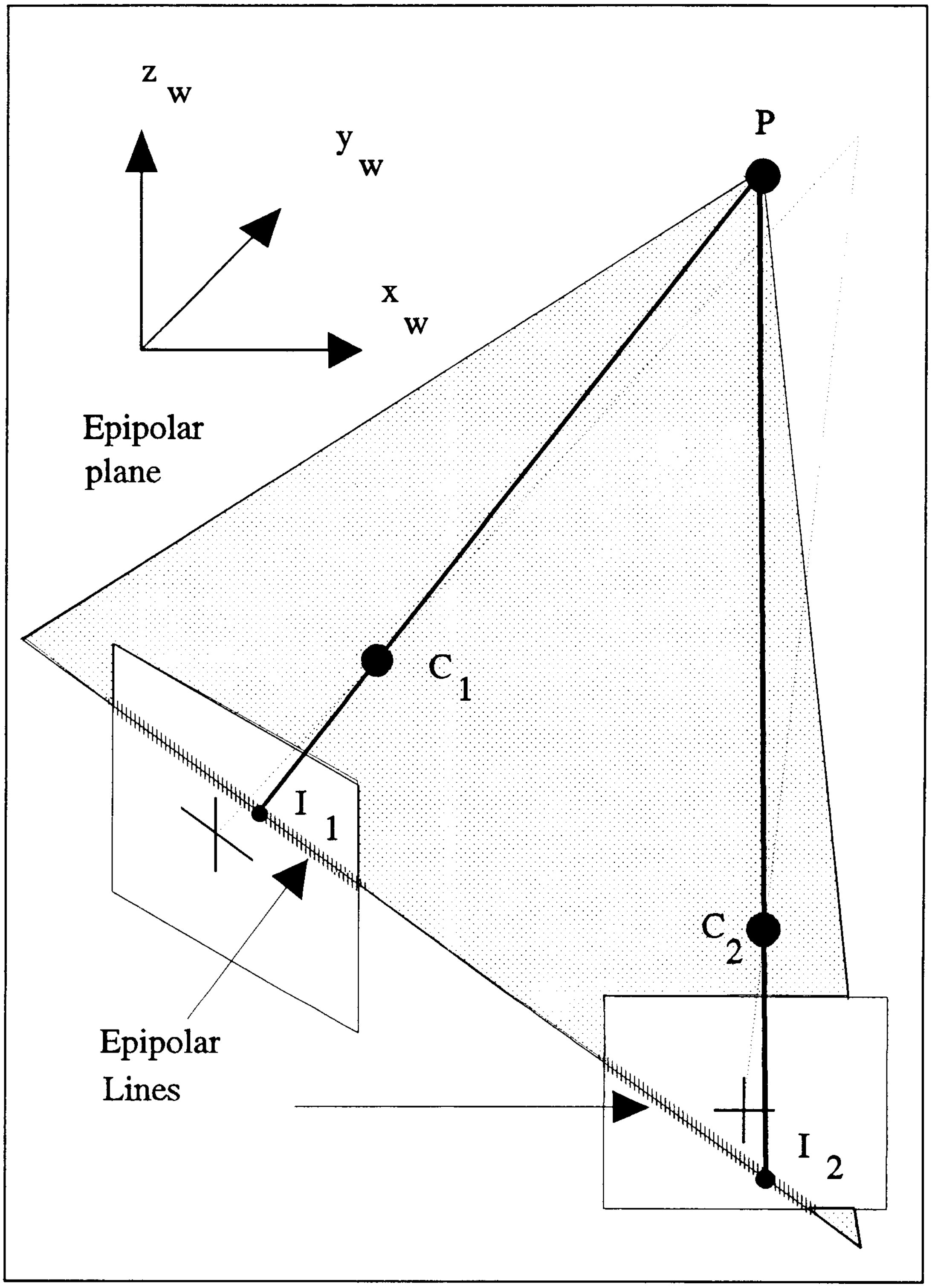


Figure 4.10 Epi-polar plane formed by the triplet of the point of interest and the center of each camera lens

$$X_d = \frac{d_x}{S_x} \cdot (X_{f_i} - C_x) \tag{4.3.6}$$

$$Y_d = d_y \cdot (Y_{f_i} - C_y) \tag{4.3.7}$$

where Sx is the horizontal image scale factor, dx and dy are the distance between adjacent sensor elements in the x and y image directions respectively. C_x and C_y translate the frame buffer origin to the center of the image.

The following two equations are calculated for each of the i cameras, (Tsai,87)

$$\left[r1_i - \frac{X_i}{f_i} \cdot r7_i\right] \cdot x_w + \left[r2_i - \frac{X_i}{f_i} \cdot r8_i\right] \cdot y_w + \left[r3_i - \frac{X_i}{f_i} \cdot r9_i\right] \cdot z_w = \frac{X_i}{f_i} \cdot T_{z_i} - T_{x_i} \tag{4.3.8}$$

$$\left[r4_i - \frac{Y_i}{f_i} \cdot r7_i\right] \cdot x_w + \left[r5_i - \frac{Y_i}{f_i} \cdot r8_i\right] \cdot y_w + \left[r6_i - \frac{Y_i}{f_i} \cdot r9_i\right] \cdot z_w = \frac{Y_i}{f_i} \cdot T_{z_i} - T_{y_i} \tag{4.3.9}$$

where rn_i is the element in the rotational matrix and Tx_i, Ty_i, and Tz_i are the elements in the translational matrix. The focal lengths of each camera, f_i, are calibrated and (X_i, Y_i) are previously calculated and i is the camera number. The following matrix is formed using two equations from each camera (Tsai,1987)

$$\mathbf{A} \cdot \mathbf{P}_w = \mathbf{B} \tag{4.3.10}$$

where

$$\mathbf{A} = \begin{bmatrix} r1_1 - \frac{X_1}{f_1} \cdot r7_1 & r2_1 - \frac{X_1}{f_1} \cdot r8_1 & r3_1 - \frac{X_1}{f_1} \cdot r9_1 \\ r4_1 - \frac{Y_1}{f_1} \cdot r7_1 & r5_1 - \frac{Y_1}{f_1} \cdot r8_1 & r6_1 - \frac{Y_1}{f_1} \cdot r9_1 \\ r1_2 - \frac{X_2}{f_2} \cdot r7_2 & r2_2 - \frac{X_2}{f_2} \cdot r8_2 & r3_2 - \frac{X_2}{f_2} \cdot r9_2 \\ r4_2 - \frac{Y_2}{f_2} \cdot r7_2 & r5_2 - \frac{Y_2}{f_2} \cdot r8_2 & r6_2 - \frac{Y_2}{f_2} \cdot r9_2 \\ . & . & . \\ . & . & . \\ . & . & . \\ r1_i - \frac{X_i}{f_i} \cdot r7_i & r2_i - \frac{X_i}{f_i} \cdot r8_i & r3_i - \frac{X_i}{f_i} \cdot r9_i \\ r4_i - \frac{Y_i}{f_i} \cdot r7_i & r5_i - \frac{Y_i}{f_i} \cdot r8_i & r6_i - \frac{Y_i}{f_i} \cdot r9_i \end{bmatrix} \tag{4.3.11}$$

$i = 1 \ldots n$ where $n \geq 2$ cameras.

and

$$\mathbf{B} = \begin{bmatrix} \frac{X_1}{f_1} \cdot T_{z_1} - T_{x_1} \\ \frac{Y_1}{f_1} \cdot T_{z_1} - T_{y_1} \\ \frac{X_2}{f_2} \cdot T_{z_2} - T_{x_2} \\ \frac{Y_2}{f_2} \cdot T_{z_2} - T_{y_2} \\ \cdot \\ \cdot \\ \frac{X_i}{f_i} \cdot T_{z_i} - T_{x_i} \\ \frac{Y_i}{f_i} \cdot T_{z_i} - T_{y_i} \end{bmatrix} \qquad (4.3.12)$$

The over determined set of equations is solved using the least square method (Ljung,1987)

$$\mathbf{P_w} = (\mathbf{A}^T \cdot \mathbf{A})^{-1} \cdot \mathbf{A}^T \cdot \mathbf{B} \qquad (4.3.13)$$

then

$$\mathbf{P}_w = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} \qquad (4.3.14)$$

The point P_w is located at (x_w,y_w,z_w), which is the location relative to the world coordinate frame, in the same units as the image sensor measurements.

4.4 Three Dimensional Calibration Theory

Accurate and efficient calibration is critical in both 2-D and 3-D vision systems. Two dimensional vision calibration can be determined by a simple scaling procedure or a complex mapping scheme can be performed. Three dimensional calibration is more complex since multiple cameras must be related to each other, and also to a common reference system.

Three dimensional calibration can be divided into two methods, passive and active. Passive calibration involves measuring physical distance and angle of orientation of the camera relative to a world system. Active calibration calculates these values automatically by using the relative screen location of a set of known points. It is difficult to measure camera distances and angles precisely, without survey equipment. For this reason active calibration is used rather than passive calibration.

For 3-D measurements it is convenient to relate all camera coordinate systems to a single world coordinate system. This coordinate system can coincide with systems used by other concurrent measurement techniques.

Camera calibration establishes the relationship between the world coordinate system and the internal coordinate system associated with each camera. If a camera is moved, only its calibration relationship is modified, thus an efficient and autonomous calibration procedure is important (Tsai, 1986).

The calibration process determines the required camera model parameters and camera orientation parameters that are used when mapping points from a world coordinate system to the vision system pixel coordinate system. Two groups of parameters are required, intrinsic and extrinsic.

The intrinsic parameters include effective focal length, radial lens distortion, horizontal scale factor error, timing error, and image coordinate location. These parameters cannot be measured by passive means. The focal length of a combined video camera and image processor system is not physically measurable. Radial lens distortion is not commonly reported with consumer grade video cameras, and the timing error cannot be measured without sophisticated electronics (Lenz, 1987).

The extrinsic parameters are directly measurable. These measurements can be tedious and are prone to errors, however. The extrinsic parameters are the external position and orientation of the camera relative to the world system. They include roll, pitch and yaw of the camera relative to the world coordinate system. These form a rotation matrix. The distance (x,y,z) to the world coordinate system from the camera forms a translation matrix. The matrices are computed in the calibration process. The values can be checked by direct measurement if desired.

The calibration technique was adapted from R. Tsai (1986, 1987, 1988A, 1988B). The technique allows standard video cameras to be calibrated efficiently and accurately. The algorithm is a two step process to determine nine unique unknowns. The algorithm does not explicitly solve for roll, pitch and yaw; they are combined in the rotation matrix. When the elements of the matrix are included, 15 combinations of 9 unknowns are solved for.

The algorithm creates two sets of linear equations. The first set has seven unknowns, some of these unknowns are combinations of the 9 unique unknows; this is solved using the Moore-Penrose pseudo inverse least square technique (Ljung, 1987). The results from this are incorporated into the second set of linear equations, with three unknowns. This is also solved by a least square technique. No initial guesses or non-linear optimization schemes are required.

The calibration procedure is based on the standard pinhole camera model. Tsai introduced a Radial Alignment Constraint (RAC). This constraint is used to decouple the calibration parameters into two groups that can be solved easily. The RAC states "that the direction of the vector extending from the origin in the image plane to the image point (Oi to Pu in Figure 4.11) is radially aligned (or parallel) with the vector extending from the optical axis (or more precisely, the projection point of the object point on the optical axis) to the object point (POz to P in Figure 4.12)" (Tsai, 1988A). The algebraic equations established as a condition of the RAC are only a function of the extrinsic parameters. This is critical in eliminating non-linear optimization from the solution methodology. The first set of simultaneous equations are solved by the RAC, the second set of equations are solved with the projective perspective constraint equations (Tsai, 1988B).

The geometry of the camera model is shown in Figure 4.11. The 3-D object point P_w is at (x_w,y_w,z_w), in the world coordinate system. The 3-D object point P, is at (x,y,z) in the camera coordinate system, which has its origin at O, the optical lens center. The Z axis is aligned with the optical axis. The image plane coordinate system (X,Y), is aligned with (x,y) and is centered at O_i, a distance f, away from the optical lens center. The point $P_u(X_u,Y_u)$ is the image plane coordinate of P(x,y,z), if a perfect pinhole cameras is used. Radial lens distortion, however, moves the image projection of point P(x,y,z), to P_d. The undistorted projection is at P_u. The image processor coordinate system is centered at the upper left hand corner of the image plane. The image plane is expressed in frame buffer coordinates as $P(X_f,Y_f)$. Five coordinate systems are involved transforming from the initial world system to the final frame buffer system.

The complete coordinate system transformation from (x_w,y_w,z_w) to (X_f,Y_f), adapted from Tsai (Tsai, 1987), is listed in Figure 4.12. The purpose of calibration is to calculate the unknown transformation parameters.

The fundamental coordinated transformation equation is

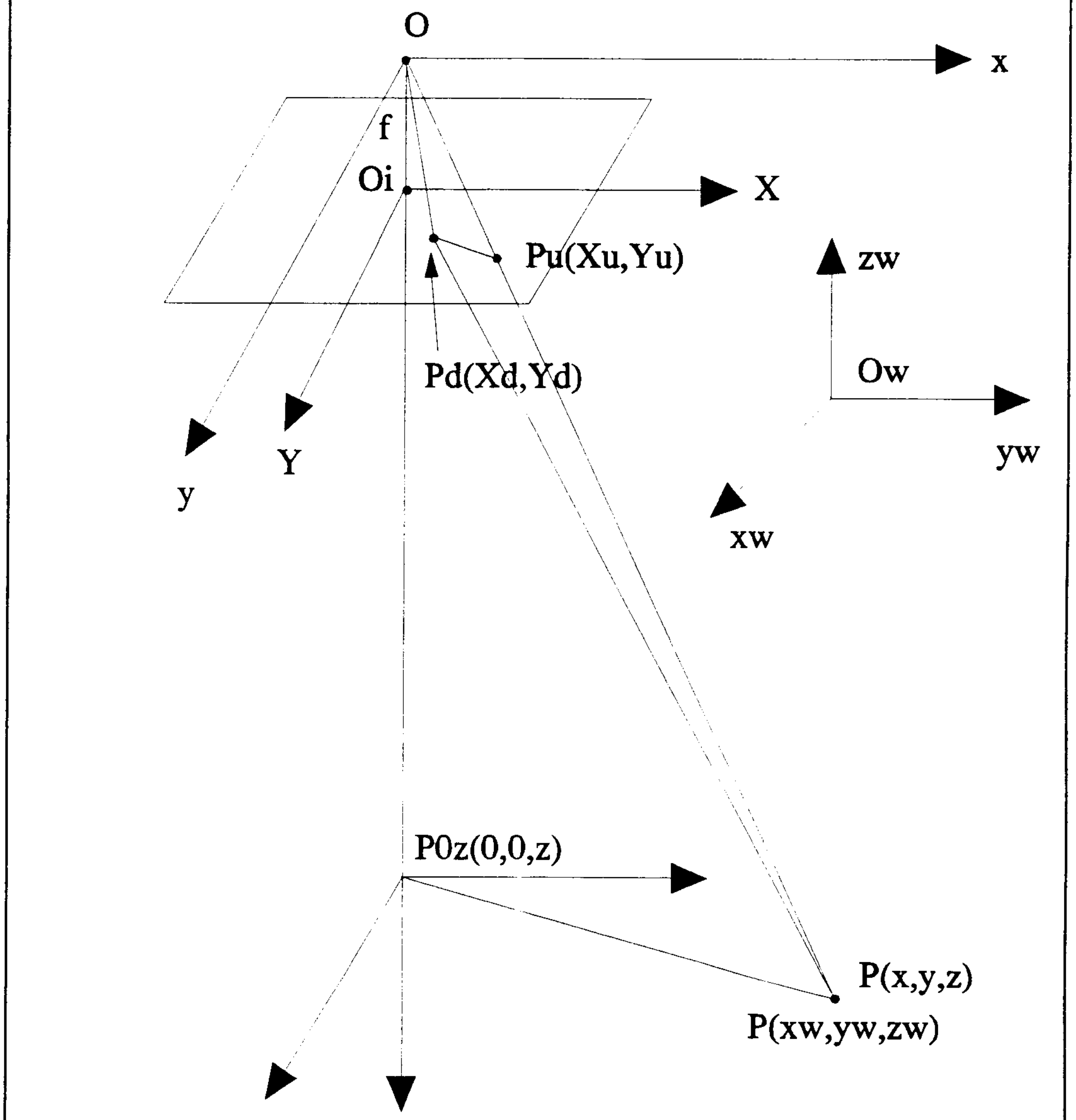


Figure 4.11 Geometry of three dimensional camera calibration model

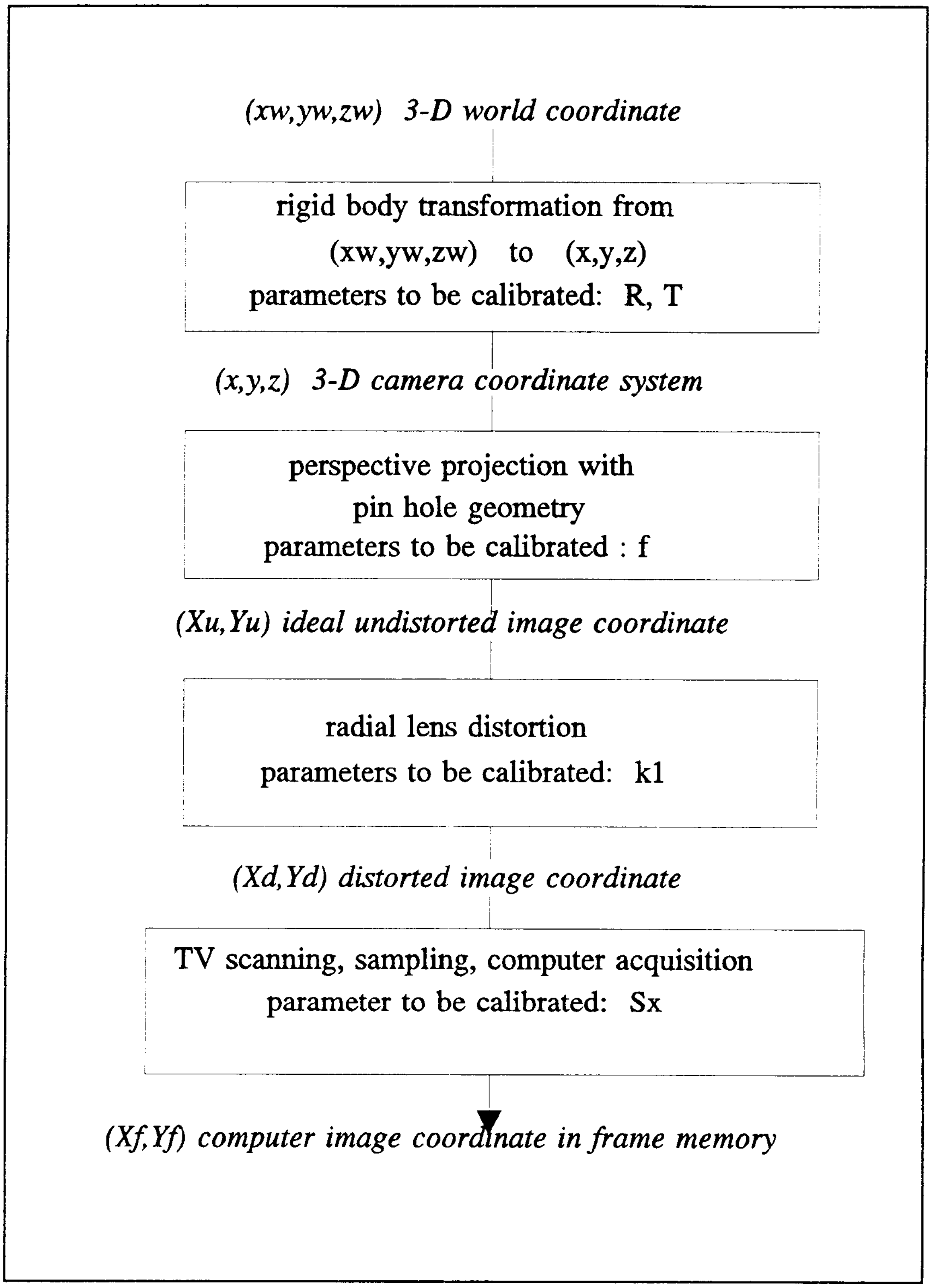

(xw,yw,zw) 3-D world coordinate
rigid body transformation from
(xw,yw,zw) to (x,y,z)
parameters to be calibrated: R, T
(x,y,z) 3-D camera coordinate system
perspective projection with
pin hole geometry
parameters to be calibrated : f
(Xu,Yu) ideal undistorted image coordinate
radial lens distortion
parameters to be calibrated: k1
(Xd,Yd) distorted image coordinate
TV scanning, sampling, computer acquisition
parameter to be calibrated: Sx
(Xf,Yf) computer image coordinate in frame memory

Figure 4.12 Complete coordinate system transformation process

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{R} \cdot \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + \mathbf{T} \qquad (4.4.1)$$

where

$$\mathbf{R} = \begin{bmatrix} r1 & r2 & r3 \\ r4 & r5 & r6 \\ r7 & r8 & r9 \end{bmatrix} \qquad (4.4.2)$$

and

$$\mathbf{T} = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} \qquad (4.4.3)$$

R is the rotation matrix, and T is the translation matrix. R is a 3 x 3 matrix, and is a function of the three euler angles tilt; pitch and yaw (Tsai, 1987). T is the three component translation vector.

The components of the R and T matrices are determined in the calibration process. In the Tsai method the order is crucial; the rotation must precede the translation. This order is opposite that of traditional methods.

The 3-D camera coordinate, (x,y,z) of P, is transformed to the undistorted image coordinate system, (X_u,Y_u), by using the pinhole camera perspective projection.

$$X_u = f \cdot \frac{x}{z} \qquad (4.4.4)$$

$$Y_u = f \cdot \frac{y}{z} \tag{4.4.5}$$

The effective focal length, f, is unknown and must be obtained via calibration. The undistorted image plane coordinate, $P_u(X_u,Y_u)$, is translated to the true image coordinate, (X_d,Y_d), by the following translation:

$$X_u = X_d + X_d \cdot \kappa_1 \cdot \left(X_d^2 + Y_d^2\right) \tag{4.4.6}$$

$$Y_u = Y_d + Y_d \cdot \kappa_1 \cdot \left(X_d^2 + Y_d^2\right) \tag{4.4.7}$$

The radial lens distortion, κ_1, is unknown and must be calibrated. The radial lens distortion can be expressed as an infinite sequence, but Tsai recommends only using the first term, κ_1, since more terms would not significantly improve the results, and could cause a numerical instability (Tsai, 1987).

The real image coordinates, $P_d(X_d,Y_d)$, are then transformed to the computer image frame coordinates, (X_f,Y_f):

$$X_f = \frac{S_x}{d_x} \cdot X_d + C_x \tag{4.4.8}$$

$$Y_f = \frac{1}{d_y} \cdot Y_d + C_y \tag{4.4.9}$$

The horizontal scale factor, Sx, is to be calibrated in this step. Cx, and Cy, are the image frame memory offset from the point where the optical axis intersects the image sensor to the image frame origin. This is not necessarily half of the resolution (Tsai, 1987). The sensor size parameters (CCD spacing), dx and dy are the center to center distance between adjacent sensor elements in the x (scan-line) and the y direction, respectively. The distance is measurable on a CCD camera and is also reported accurately by the camera manufacturer.

Equations 4.4.5 - 4.4.9 are combined, and the 3-D world coordinate is related to the 2-D computer image coordinate system:

$$S_x^{-1} \cdot d_x \cdot (X_f - C_x) + S_x^{-1} \cdot d_x \cdot (X_f - C_x) \cdot \kappa_1 \cdot \left[\left(S_x^{-1} \cdot d_x \cdot (X_f - C_x)\right)^2 + (d_y \cdot (Y_f - C_y))^2\right] = f \cdot \frac{x}{z} \qquad (4.4.10)$$

$$d_y \cdot (Y_f - C_y) + d_y \cdot (Y_f - C_y) \cdot \kappa_1 \cdot \left[\left(S_x^{-1} \cdot d_x \cdot (X_f - C_x)\right)^2 + (d_y \cdot (Y_f - C_y))^2\right] = f \cdot \frac{y}{z} \qquad (4.4.11)$$

Equations 4.4.10 and 4.4.11 are combined with 4.4.1, resulting in

$$S_x^{-1} \cdot d_x \cdot (X_f - C_x) + x_x^{-1} \cdot d_x \cdot (X_f - C_x) \cdot \kappa_1 \cdot \left[\left(S_x^{-1} \cdot d_x \cdot (X_f - C_x)\right)^2 + (d_y \cdot (Y_f - C_y))^2\right] = f \left[\frac{r1 \cdot x_w + r2 \cdot y_w + r3 \cdot z_w + T_x}{r7 \cdot x_w + r8 \cdot y_w + r9 \cdot z_w + T_z}\right] \qquad (4.4.12)$$

$$d_y \cdot (Y_f - C_y) + d_y \cdot (Y_f - C_y) \cdot \kappa_1 \cdot \left[\left(S_x^{-1} \cdot d_x \cdot (X_f - C_x)\right)^2 + (d_y \cdot (Y_f - C_y))^2\right] = f \left[\frac{r4 \cdot x_w + r5 \cdot y_w + r6 \cdot z_w + T_y}{r7 \cdot x_w + r8 \cdot y_w + r9 \cdot z_w + T_z}\right] \qquad (4.4.13)$$

The Tsai calibration algorithm can accommodate coplanar or multi-plane calibration points. The coplanar case cannot determine the image scale factor, Sx, however. It is also inconvenient to set up large scale precisely co-planar calibration points that fill the camera field of view. For these reasons, the more versatile multi-plane case is used. Using surveying ground control techniques, precise multi-plane point location is possible. The algorithm is presented in a computation oriented manner in Sec. 6.9.

The first step is to determine the frame coordinates of each calibration point. A single still frame or an average of several frames for greater accuracy is acquired. The row and column, (X_{fj},Y_{fj}), of each calibration point, j, is extracted from the frame. This is the only user intensive process, but the accuracy of the calibration and resulting 3-D measurement depend on accurate location of the extracted points. Tsai reports average 3-D measurement accuracy of one part in 4000, with one part in 5000 as an upper limit (Tsai 1987, 1988). Of course, 3-D video measurement accuracy cannot be better than the ground control point accuracy.

Using a simple mouse directed cross hair to locate the calibration points by eye results in +/- 1 pixel location accuracy. This will not give acceptable calibration accuracy; sub-pixel calibration point location is critical for accurate results. This is done by calculating the centroid of the calibration points and reporting a decimal pixel location. This was illustrated in Figure 4.1.

Spherical targets, represented by a large number of pixels, give a statistically more accurate centroid than a target that is represented by a single pixel. The spherical target also has the same 2-D planar representation from any view point. A sphere is the only shape that has this property.

The screen location of the calibration point, the image center offset (Cx,Cy) and the specific camera image sensor dimensions are used to compute the distorted image coordinate for each calibration point:

$$d_x = \left[\frac{\text{sensor size x direction}}{\text{number of x sensor elements}}\right] \cdot \left[\frac{\text{number of x sensor elements}}{\text{number of scan line pixels in frame memory}}\right] \tag{4.4.14}$$

$$d_y = \left[\frac{\text{sensor size y direction}}{\text{number of y sensor elements}}\right] \tag{4.4.15}$$

$$X_{d_j} = \frac{1}{S_x} \cdot d_x \cdot \left(X_{f_j} - C_x\right) \tag{4.4.16}$$

$$Y_{d_j} = d_y \cdot \left(Y_{f_j} - C_y\right) \tag{4.4.17}$$

for $j=1...n$, where n is the number of calibration points. S_x is assumed to be one and is absorbed into the unknowns for the linear equation. It is computed explicitly in Equation 4.4.26.

With the number of calibration points, n, larger than seven, an over-determined system of linear equations is established according to Equation 4.4.18

$$\begin{bmatrix} Y_{d_j} \cdot x_{w_j} & Y_{d_j} \cdot y_{w_j} & Y_{d_j} \cdot z_{w_j} & Y_{d_j} & -X_{d_j} \cdot x_{w_j} & -X_{d_j} \cdot y_{w_j} & -X_{d_j} \cdot z_{w_j} \end{bmatrix} \cdot$$

$$\begin{bmatrix} T_y^{-1} \cdot S_x \cdot r1 \\ T_y^{-1} \cdot S_x \cdot r2 \\ T_y^{-1} \cdot S_x \cdot r3 \\ T_y^{-1} \cdot S_x \cdot T_x \\ T_y^{-1} \cdot r4 \\ T_y^{-1} \cdot r5 \\ T_y^{-1} \cdot r6 \end{bmatrix} = \begin{bmatrix} X_{d_j} \end{bmatrix} \qquad (4.4.18)$$

where (x_{wj}, y_{wj}, z_{wj}) is the corresponding 3-D world coordinate for the modified image coordinate calculated in Equations 4.4.16 and 4.4.17. A least square technique is used to solve for the seven unknowns. A unique solution exists if all the columns are linearly independent, a detailed proof can be found in Tsai (1986, 1987).

Adopting the same notation as Tsai, let the seven unknowns determined in Equation 4.4.18 be defined as:

$$a_1 = T_y^{-1} \cdot S_x \cdot r\,1 \qquad (4.4.19)$$

$$a_2 = T_y^{-1} \cdot S_x \cdot r\,2 \qquad (4.4.20)$$

$$a_3 = T_y^{-1} \cdot S_x \cdot r3 \tag{4.4.21a}$$

$$a_4 = T_y^{-1} \cdot S_x \cdot T_x \tag{4.4.21b}$$

$$a_5 = T_y^{-1} \cdot r4 \tag{4.4.22}$$

$$a_6 = T_y^{-1} \cdot r5 \tag{4.4.23}$$

$$a_7 = T_y^{-1} \cdot r6 \tag{4.4.24}$$

then Ty is calculated as:

$$|T_y| = \sqrt{\left(a_5^2 + a_6^2 + a_7^2\right)} \tag{4.4.25}$$

Tsai uses the RAC to determine if T_y is positive or negative for the coplanar case. This gives mixed results for the noncoplanar case. The simplest method is to calculate f, with the positive sign of T_y. If f is negative then the sign of T_y is changed and f is re-calculated. This is an acceptable method since by definition only positive values of f and z are possible.

The scan error, S_x, is determined by

$$S_x = |T_y| \cdot \sqrt{a_1^2 + a_2^2 + a_3^2} \tag{4.4.26}$$

The first six elements of the rotation matrix, R, are determined using

$$r1 = a_1 \cdot \frac{T_y}{S_x} \quad (4.4.27)$$

$$r2 = a_2 \cdot \frac{T_y}{S_x} \quad (4.4.28)$$

$$r3 = a_3 \cdot \frac{T_y}{S_x} \quad (4.4.29)$$

$$r4 = a_5 \cdot T_y \quad (4.4.30)$$

$$r5 = a_6 \cdot T_y \quad (4.4.31)$$

$$r6 = a_7 \cdot T_y \quad (4.4.32)$$

and T_x by

$$T_x = a_4 \cdot T_y \quad (4.4.33)$$

This completes the first two rows of the rotation matrix. The third row is found by taking the cross product of the first two, and is based on the orthonormal property of the rotation matrix and the right hand rule. If the determinate of R is not one, then a problem exists with the input data.

The remaining unknowns, f, K_1, and T_z are found by solving a set of independent linear equations based upon the projective constraint, Equations 4.4.4 and 4.4.5 (Tsai, 1988A,1988B).

The calibration is repeated for all cameras. The results are used in the 3-D positioning algorithm. The calibration need only be performed once, unless the camera is moved, or the focal length changed.

This is the theoretical basis for the image processing system algorithm. The 2-D planar algorithm uses simple geometry for object location, but only motions parallel to the image plane are allowed. The 3-D positioning algorithm removes this restriction. Multiple cameras are allowed, and unconstrained motion viewed by at least two cameras are permitted. The 3-D algorithm has the potential for greater accuracy over the entire field of view due to multiple camera redundancy. The accuracy of each method is directly dependent upon the calibration procedure.

Calibration for the 2-D case is a simple scaling procedure, but a more complex mapping scheme is possible. The 3-D algorithm requires a structured calibration procedure, where points of known locations are found in each camera image. From these images and the point locations, parameters are calculated that relate each camera to the world coordinate system. Information about the individual camera settings is also determined. The calibration is verified by 3-D location of the original calibration points and projecting the locations back on to the image.

If the original calibration points are located with the accuracy required by the 3-D algorithm, then the calibration is considered successful. If the points are not located within the specified precision then an error has occurred.

The calculated world point location accuracy cannot be better than the original world point location accuracy. It is also possible for a error in world point location or image location of the respective point. Small errors in these two steps are magnified in the calibration.

An overview of the two dimensional and three dimensional theory is shown in Figures 4.13 and 4.14, respectively. In the two following chapters, the validity of the 2-D and 3-D methods and calibration algorithms are demonstrated. Application procedures and steps are explained and outlined and recommendations for best results are presented.

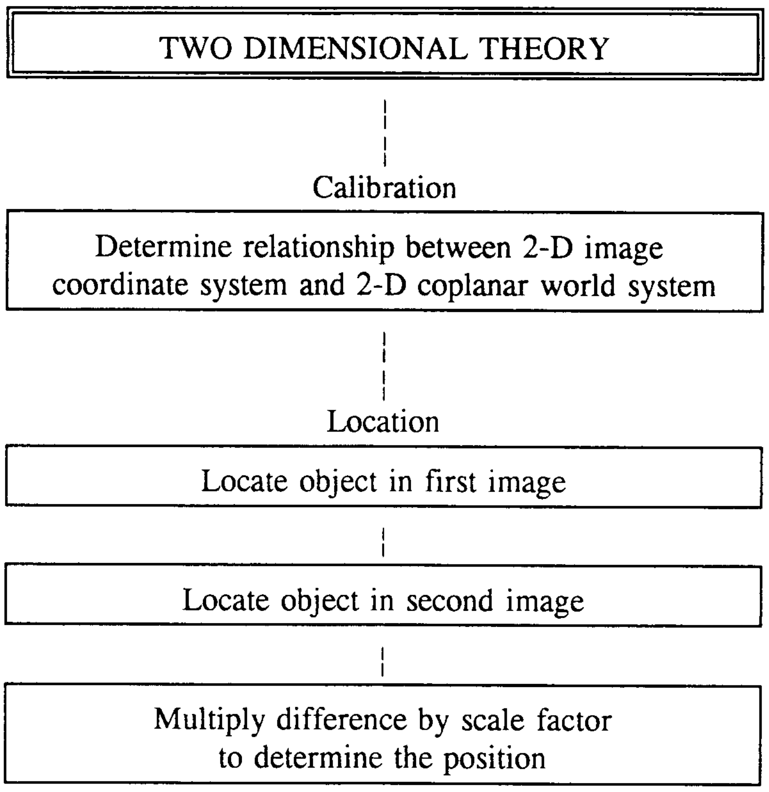

TWO DIMENSIONAL THEORY
Calibration
Determine relationship between 2-D image coordinate system and 2-D coplanar world system
Location
Locate object in first image
Locate object in second image
Multiply difference by scale factor to determine the position

Figure 4.13 Overview of 2-D theory

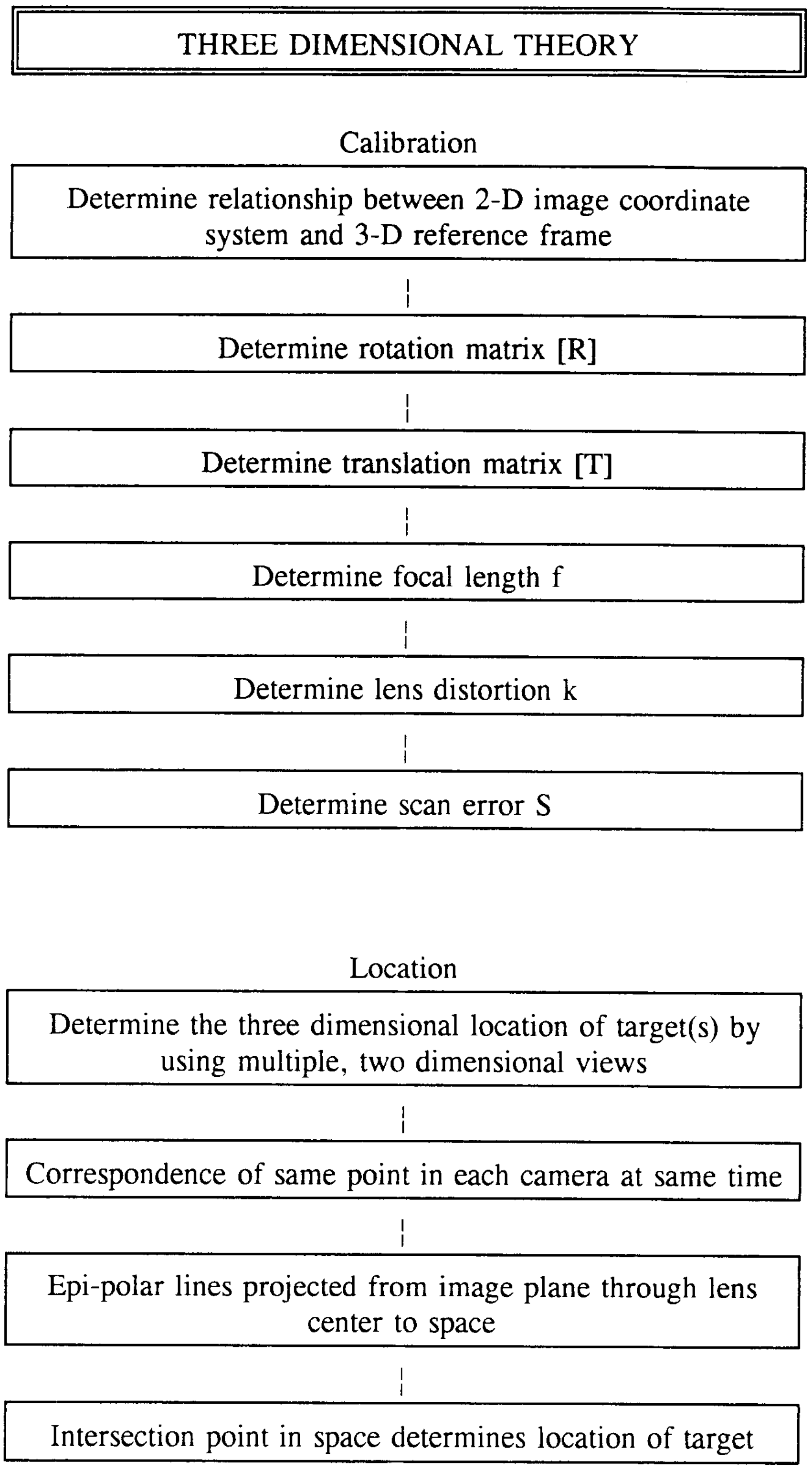

THREE DIMENSIONAL THEORY
Calibration
Determine relationship between 2-D image coordinate system and 3-D reference frame
Determine rotation matrix [R]
Determine translation matrix [T]
Determine focal length f
Determine lens distortion k
Determine scan error S
Location
Determine the three dimensional location of target(s) by using multiple, two dimensional views
Correspondence of same point in each camera at same time
Epi-polar lines projected from image plane through lens center to space
Intersection point in space determines location of target

Figure 4.14 Overview of 3-D theory

5.0 TWO DIMENSIONAL EXPERIMENTAL VERIFICATION

5.1 Overview

This chapter demonstrates implementation of the 2-D, one camera model. The validity of the model is verified by direct measurement comparisons. In a second example, a demonstration of target following is provided without scaling the motion.

Careful calibration is critical for accurate motion measurement. A simple scaling methodology for calibration gives acceptable results if the motions satisfy the single plane requirements. The motions must remain in a single plane, and that plane must be nearly parallel to the image plane. The angle between the planes must be less than three degrees.

The accuracy of the 2-D model is checked in a linear motion experiment with string potentiometer measurements to complement video data collection. If the calibration is performed carefully, excellent agreement between the two systems can be obtained.

As an example of the 2-D single camera tracking system, multiple small scale floating targets are tracked in a circular wave basin experiment. Alternative measurements using contact based sensor technology are not possible. If cable extension transducers (CET), such as string potentiometers are attached to the small targets, the target motions are influenced.

5.2 Random Mooring Study

Video motion measurement was used in a collinear constrained mooring, random motion study, at the O.H. Hinsdale Wave Research Laboratory. The experiment was supervised by Drs. Solomon Yim and Oded Gottlieb, under Office of Naval Research grant N00014-88K0729. The purpose of the experiment was to verify a non-linear mooring theory, under various wave loading conditions.

The experiment was performed with an 18 inch diameter float that was constrained to slide along a 1 inch square rod. The rod was anchored above the floor of the two dimensional wave channel at the Wave Research Laboratory. The channel is 342 feet long, 12 feet wide and 15 feet deep. Tension springs were attached to opposite sides of the float through a pulley system to a rigid mount above the water surface. String potentiometer CET sensors were attached to monitor float position. Figure 5.1 illustrates the setup.

The setup was subjected to various forcing conditions, with and without water. Pluck tests were performed in air and water and a decay response was calculated. The float was then subjected to wave forces. The CET sensors recorded instantaneous position information accurate to 1/16 (0.0625) inch.

This information is the basis for validation of the 2-D, single camera video positioning system. The test satisfies the requirements of the 2-D case. Motions are coplanar, and the motion plane is parallel to the image plane. The camera is oriented such that the motions are within the field of view and perpendicular to the optic axis, illustrated in Figure 5.2. Under these conditions the maximum video theoretical resolution is equal to the field of view divided by the number of

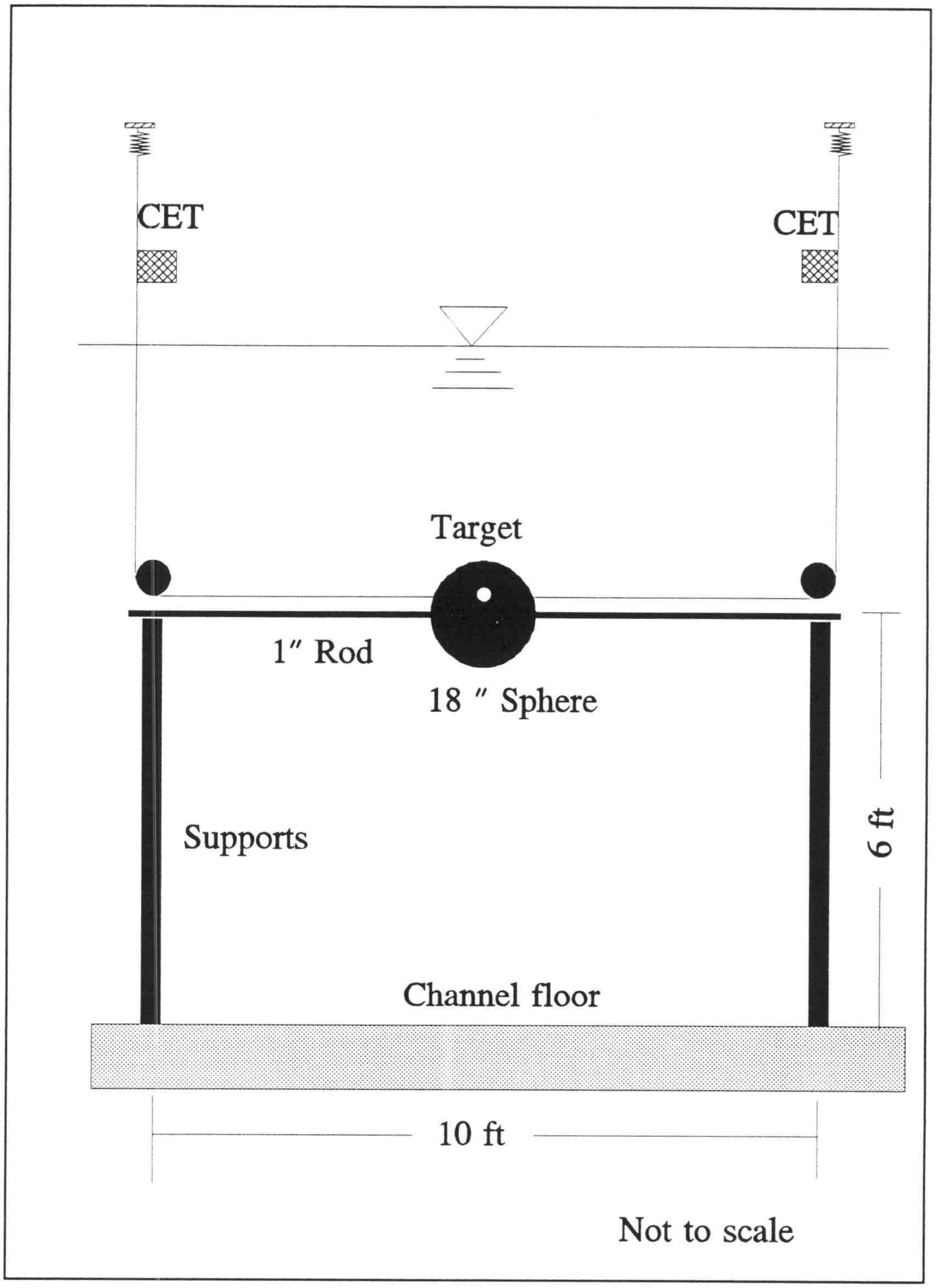


Figure 5.1 Experimental setup for translating sphere expermient, elevation view

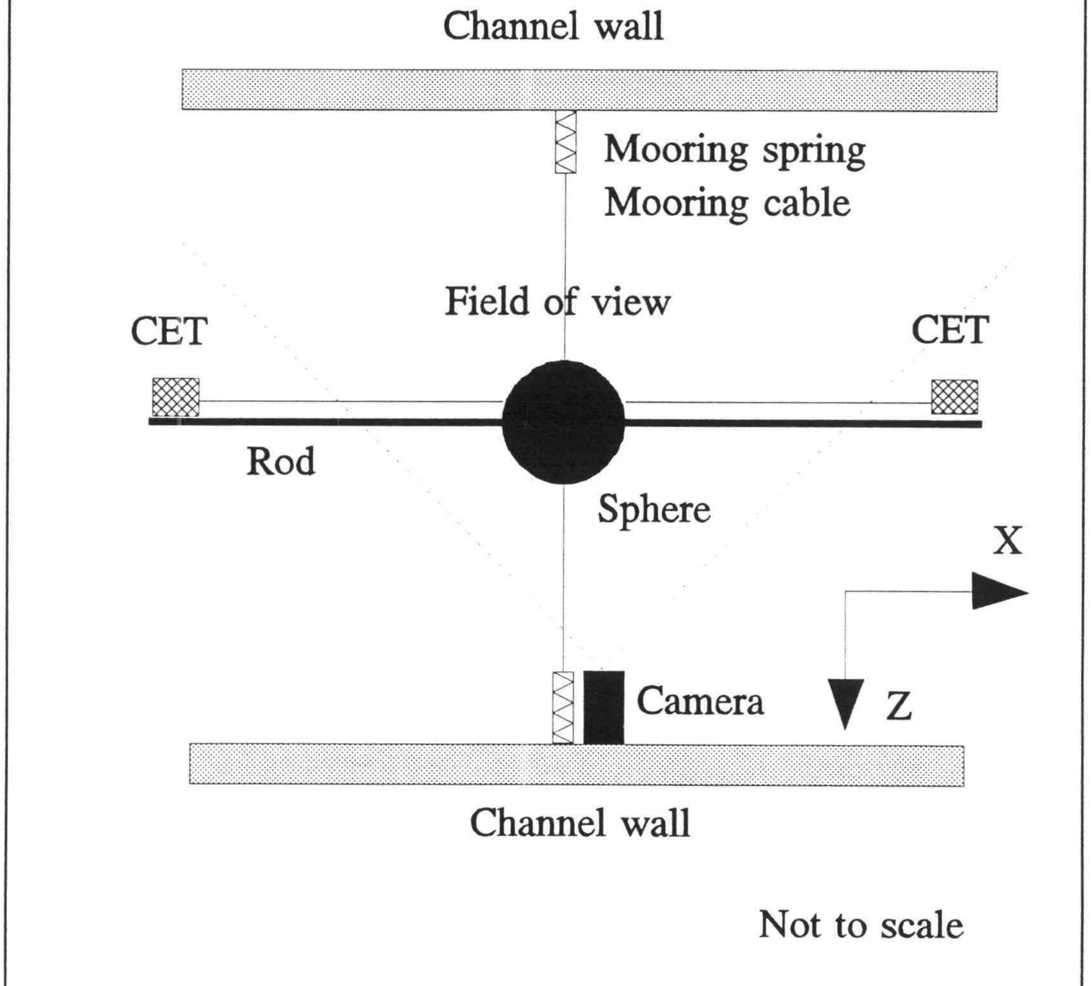


Figure 5.2 Expermimental setup for translating ball experiment showing camera field of view, plan view

horizontal pixels on the video monitor. For most tests this was (72 inches/512 pixels) = .14 inch/pixel, without resorting to sub-pixel location methods.

From the recorded video, the float position is determined by the X,Y screen location of a target. A white target is painted on the black float to provide maximum contrast. Everything else in the field of view is black or gray. The target is located on the screen using a moving box mask centroiding technique. A box is placed around the target on the screen and the computer searches for the target within the box and reports its centroid. If the target approaches the end of the box, the box is moved to a new location with the target at the center. The target is then located in the next frame. This process is transparent to the user. A time series, sampled at 30 Hz is created for the float position. This procedure was followed for several plucking tests and various wave conditions.

The calibration is performed by determining the screen location of a set of known points on the rod. The points are white dots spaced at one foot intervals. From these values an average pixel to distance linear ratio is determined. This value is used to convert the screen locations into real world locations. More sophisticated methods could have been used for determining this ratio but the two dimensional model was not the focus of research.

A sample of the video tracking data from pluck test P12-B is shown in Table 5.1. A corresponding time series from the CET data is shown in Table 5.2. The two data files are graphically represented in Figure 5.3. The CET data compares well with the video, the results degrade near the edges of the field of view. This can be seen at the larger peak excursions. The errors at the peaks

Table 5.1 Section of video tracking data from pluck test P12-B.

TIME CODE	PIXELS	
HH:MM:SS:FF	X	Y
10:00:59.15	232.75	254.50
10:00:59.14	233.00	254.00
10:00:59.13	232.75	254.50
10:00:59.12	233.00	254.00
10:00:59.11	233.00	254.00
10:00:59.10	232.80	254.80
10:00:59.09	233.00	254.00
10:00:59.08	232.80	254.80
10:00:59.07	232.75	254.50
10:00:59.06	232.33	254.67
10:00:59.05	231.75	254.50
10:00:59.04	231.33	254.67
10:00:59.03	230.50	255.00
10:00:59.02	230.00	254.50
10:00:59.01	229.50	255.00
10:00:59.00	228.80	254.80
10:00:58.29	227.80	254.80
10:00:58.28	226.80	254.80
10:00:58.27	225.80	254.80
10:00:58.26	225.20	254.80
10:00:58.25	224.50	255.00
10:00:58.24	223.50	255.00
10:00:58.23	222.50	255.00
10:00:58.22	222.50	255.00
10:00:58.21	220.80	254.80
10:00:58.20	220.20	254.80
10:00:58.19	219.50	255.00
10:00:58.18	218.80	254.80
10:00:58.17	218.50	255.00
10:00:58.16	217.80	254.80
10:00:58.15	217.50	255.00
10:00:58.14	217.20	254.80
10:00:58.13	217.20	254.80
10:00:58.12	216.80	254.80
10:00:58.11	217.50	255.00
10:00:58.10	217.80	254.80
10:00:58.09	218.50	255.00

Table 5.2 Time series from CET for pluck test P12-B

INCHES	SECONDS
-17.4524	36054.55
-17.4316	36054.56
-17.406	36054.56
-17.3779	36054.57
-17.3474	36054.57
-17.3022	36054.58
-17.2681	36054.58
-17.2168	36054.59
-17.1655	36054.59
-17.1082	36054.6
-17.0386	36054.6
-16.9678	36054.61
-16.8958	36054.61
-16.8152	36054.62
-16.7273	36054.62
-16.6443	36054.63
-16.5564	36054.63
-16.4563	36054.64
-16.3586	36054.64
-16.2561	36054.65
-16.145	36054.65
-16.0327	36054.66
-15.9167	36054.66
-15.791	36054.67
-15.6665	36054.67
-15.5322	36054.68
-15.3979	36054.68
-15.2612	36054.69
-15.1221	36054.69
-14.9792	36054.7
-14.8352	36054.7
-14.679	36054.71
-14.5276	36054.71
-14.3677	36054.72
-14.2065	36054.72
-14.0442	36054.73
-13.8745	36054.73
-13.7036	36054.74

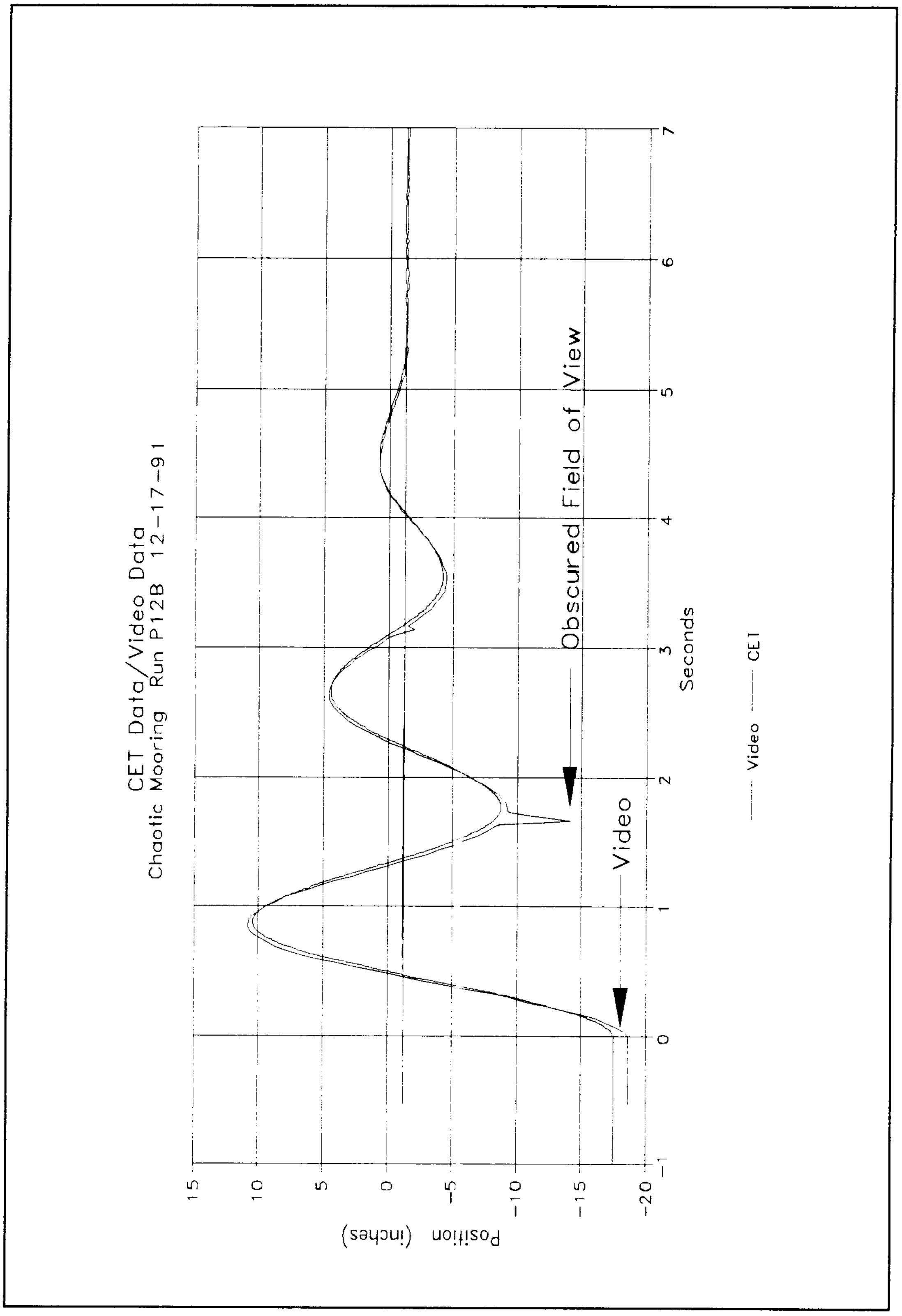


Figure 5.3 Position versus time for video image and CET data

(0.125 inches) can be reduced by adjusting the scaling factor, but this causes the smaller excursions to be under predicted. The underlying problem is using a constant linear scale factor for a nonlinear mapping.

The obscured field of view noted in Figure 5.3 is caused by a white rope entering the image field of the plucking test. This emphasizes how critical lighting and reflections from the other objects can be. The erroneous point can be avoided by using a black rope to eliminate the target confusion.

A graphical time series from run W14 is shown in Figure 5.4. This motion is induced by waves passing over the submerged sphere and mooring. The video agrees well with the CET data, with no significance errors. There is a time offset between the two time series. It is caused by delays introduced by CET digitization hardware operation and by image processing software execution. The delay is small (0.2 seconds) and identifiable but is not possible to eliminate. As with the pluck test, the video slightly over predicts the motion near the edges of the field.

The difference between the two methods of measurement is partially due to the use of a wide angle lens resulting in increased edge distortion. The errors are enhanced by using a single linear scale factor. In both cases P12-B and W14, a unique scale factor could have been determined for each segment of the field of view or a polynomial curve could be best fit to the calibration points. This would give more accurate target locations and could be used to create a distortion function for the lens, if that additional accuracy is desired.

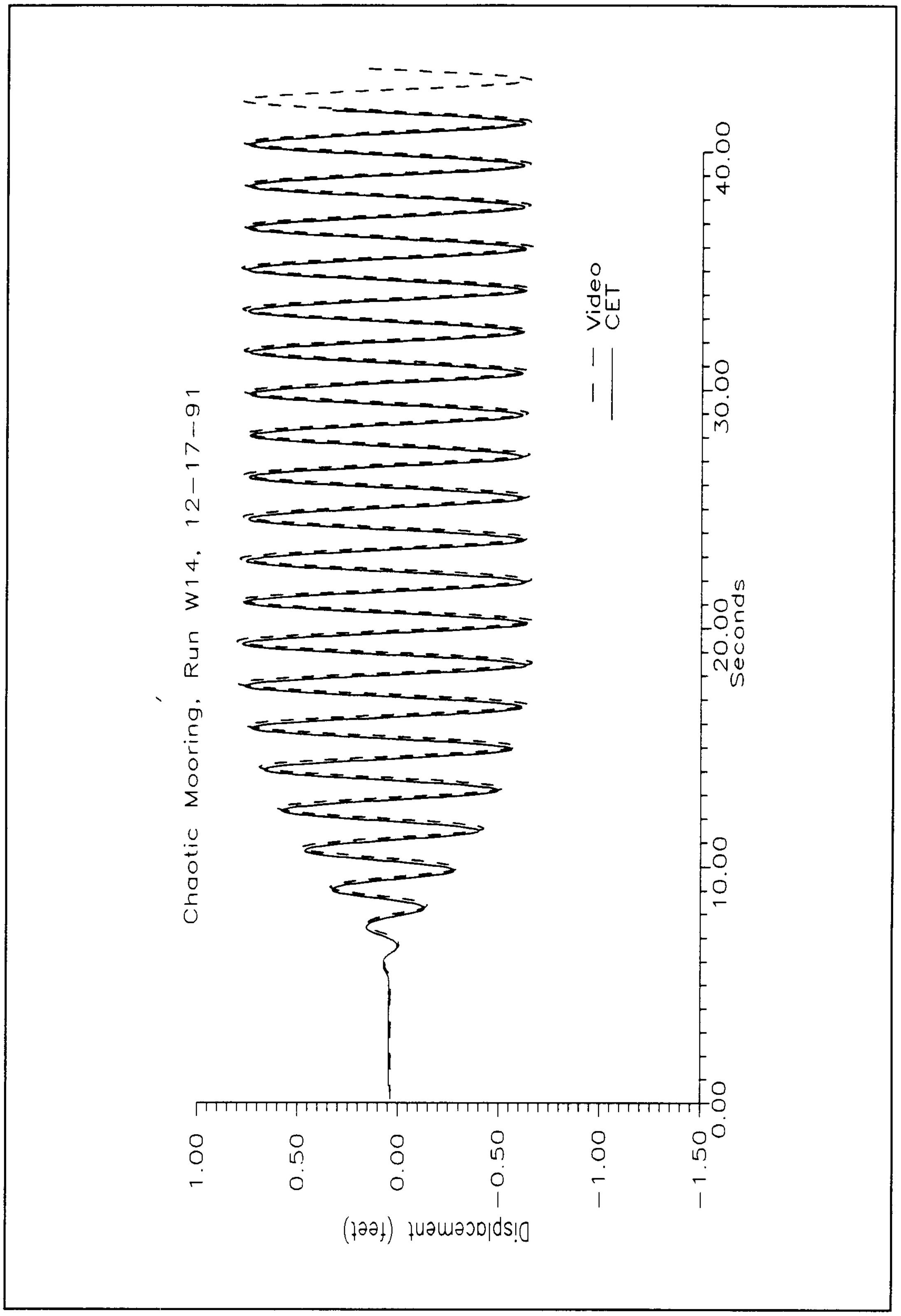


Figure 5.4 Comparison of video data with CET data from chaotic mooring study

5.3 Long Shore Current Study

The 2-D, single camera, case was used to collect position information for a long shore current modeling study. The study was conducted in 1991 by David Katzev (Katzev, 1991), under Office of Naval Research contract number 00014-86K0687, in the small circular wave basin at the O.H. Hinsdale Wave Research Laboratory.

A small circular wave basin, 15 feet in diameter, with a spiral wave maker, was used because of its ability to simulate an infinite beach. Long shore currents are formed when waves interact with the beach. These currents flow continuously around perimeter of the basin in the direction of the longshore wave component.

Information was desired about the current and wave induced velocities. Traditional data collection methods would involve sampling direction and strength of the current, and wave height. For a spatial representation, this data is required at many locations. It is not feasible to install many sensors throughout the wave field because the basin is relatively small and a large number of sensors would influence the currents and wave heights.

An alternate and tedious method would be to use a few sensors, and place them at specific locations. This would require repeating the same wave conditions, relocating the sensors and combining the individual location data to build a spatial representation of the wave field.

Instead, video data acquisition methods were used. A small floating ball was tracked as it followed the current and was subjected to wave excitation. The initial X,Y locations were subsequently converted into directional velocities and accelerations.

The X,Y locations are obtained by using a moving box mask and centroiding algorithm. The video tape is advanced frame by frame and each frame is processed. The screen X,Y locations are converted into real world x,y location by the use of a calibrated scale ratio. The ratio is determined by counting the number of pixels that represent a known distance in the field of view.

The small scale circular wave basin has a 15 foot diameter and a one foot depth. The beach shape is convex with a height proportional to the radius to the two-thirds power. The camera is placed 28 feet above the basin with a near vertical optical axis. The relative positions are presented in Figure 5.5.

A sample of the unscaled pixel data and scaled distances are listed in Table 5.3 and a trajectory of the pixel motion is shown in Figure 5.6. The scaled motion in feet is shown in Figure 5.7. The circular trajectory of the target is apparent in the figure as are the individual wave excursions. The location of the target and the time of each frame are known, from this kinematic properties are calculated using finite difference approximations (Katzev, 1991). The video tracking results are not directly verifiable. However, by overlaying the resolved trajectory upon the video recording of actual motion, excellent correlations are observed. Small errors are due to the fixed scaling coefficient and lens distortion.

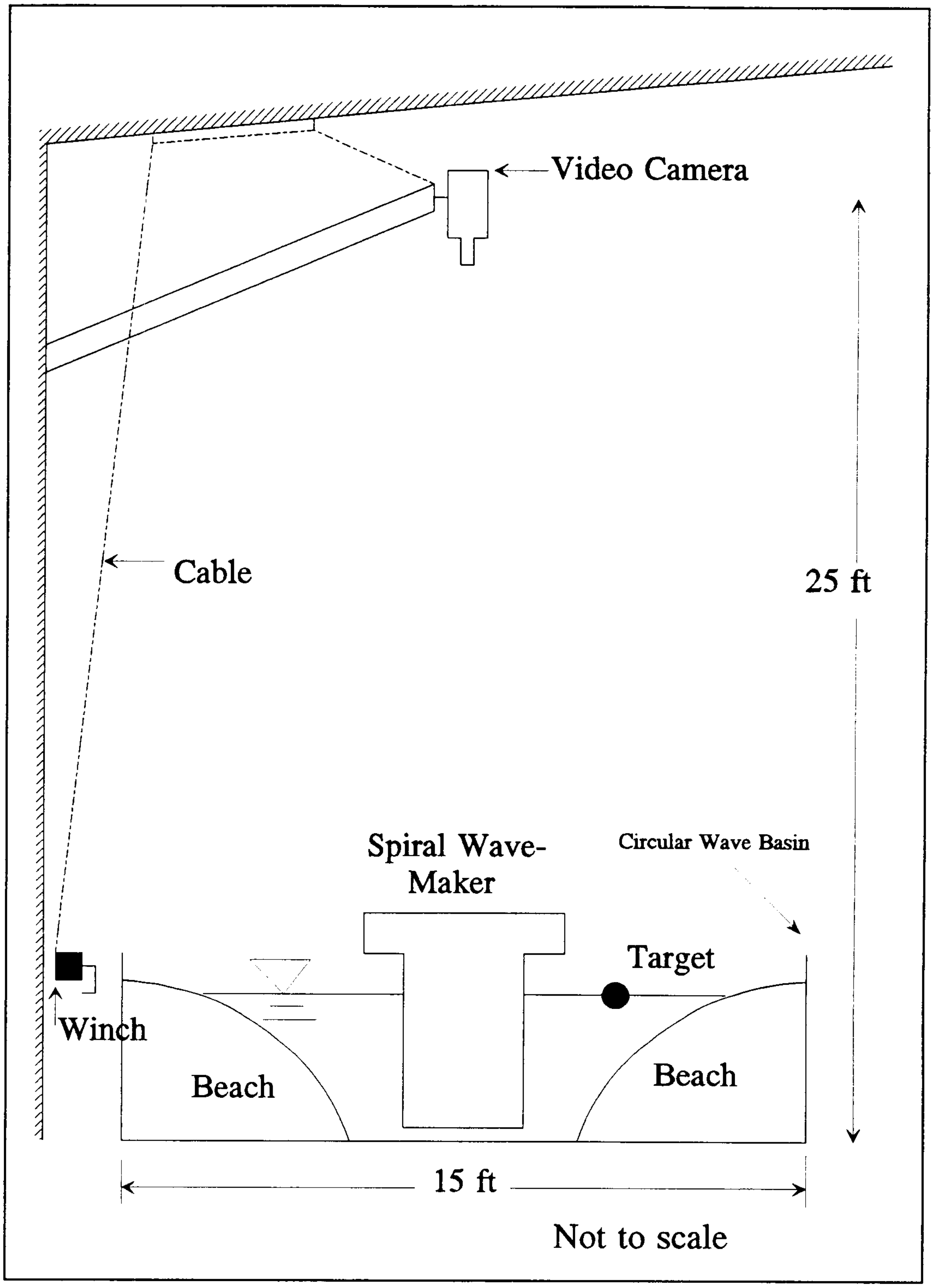


Figure 5.5 Experimental setup for longshore currents in a circular wave basin

Table 5.3 Time series from spiral wave maker, unscaled pixels and feet

TIME CODE	PIXELS		FEET	
HH:MM:SS:FF	X	Y	x	y
03:09:32:04	232.00	62.00	5.95	1.59
03:09:32:03	232.00	62.00	5.95	1.59
03:09:32:02	230.50	62.00	5.91	1.59
03:09:32:01	230.00	60.00	5.90	1.54
03:09:32:00	229.00	60.50	5.87	1.55
03:09:31:29	228.50	58.00	5.86	1.49
03:09:31:28	227.00	56.00	5.82	1.44
03:09:31:27	227.00	56.00	5.82	1.44
03:09:31:26	227.00	56.00	5.82	1.44
03:09:31:25	226.00	54.00	5.79	1.38
03:09:31:24	225.71	52.86	5.79	1.36
03:09:31:23	225.00	52.50	5.77	1.35
03:09:31:22	224.50	52.00	5.76	1.33
03:09:31:21	224.00	52.00	5.74	1.33
03:09:31:20	223.80	51.20	5.74	1.31
03:09:31:19	223.00	51.00	5.72	1.31
03:09:31:18	222.50	51.00	5.71	1.31
03:09:31:17	222.00	51.00	5.69	1.31
03:09:31:16	221.60	51.60	5.68	1.32
03:09:31:15	221.50	53.00	5.68	1.36
03:09:31:14	221.00	53.00	5.67	1.36
03:09:31:13	221.00	55.00	5.67	1.41
03:09:31:12	220.50	57.00	5.65	1.46
03:09:31:11	220.50	57.00	5.65	1.46
03:09:31:10	219.50	60.00	5.63	1.54
03:09:31:09	219.00	60.00	5.62	1.54
03:09:31:08	218.80	60.80	5.61	1.56
03:09:31:07	218.29	61.14	5.60	1.57
03:09:31:06	217.29	61.14	5.57	1.57
03:09:31:05	217.00	62.00	5.56	1.59
03:09:31:04	215.83	59.33	5.53	1.52
03:09:31:03	215.71	59.14	5.53	1.52
03:09:31:02	215.00	57.50	5.51	1.47
03:09:31:01	214.29	56.86	5.49	1.46
03:09:31:00	213.50	55.33	5.47	1.42
03:09:30:29	213.00	56.00	5.46	1.44
03:09:30:28	212.20	54.80	5.44	1.41

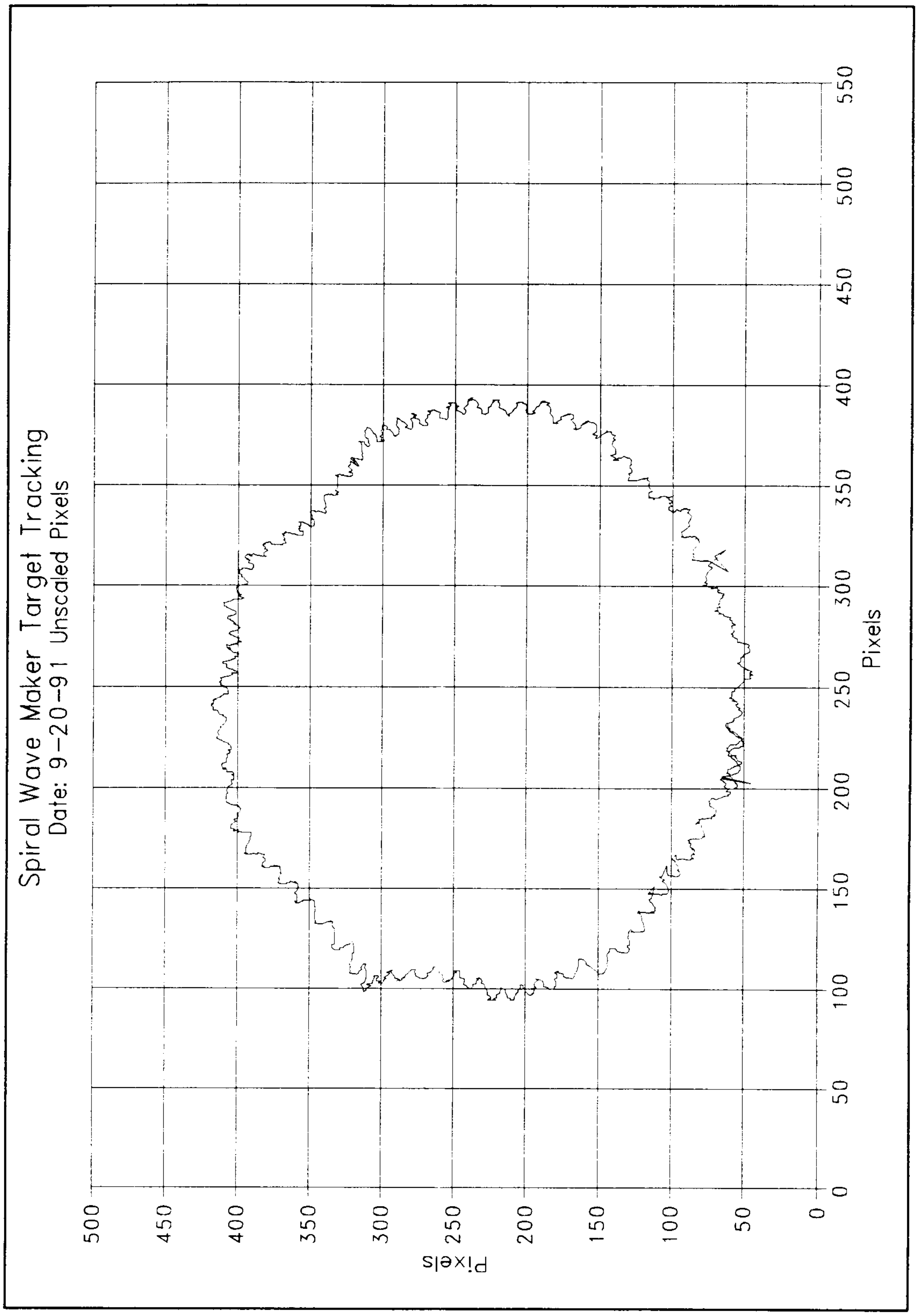


Figure 5.6 Longshore current model, video target tracking, raw unscaled pixels

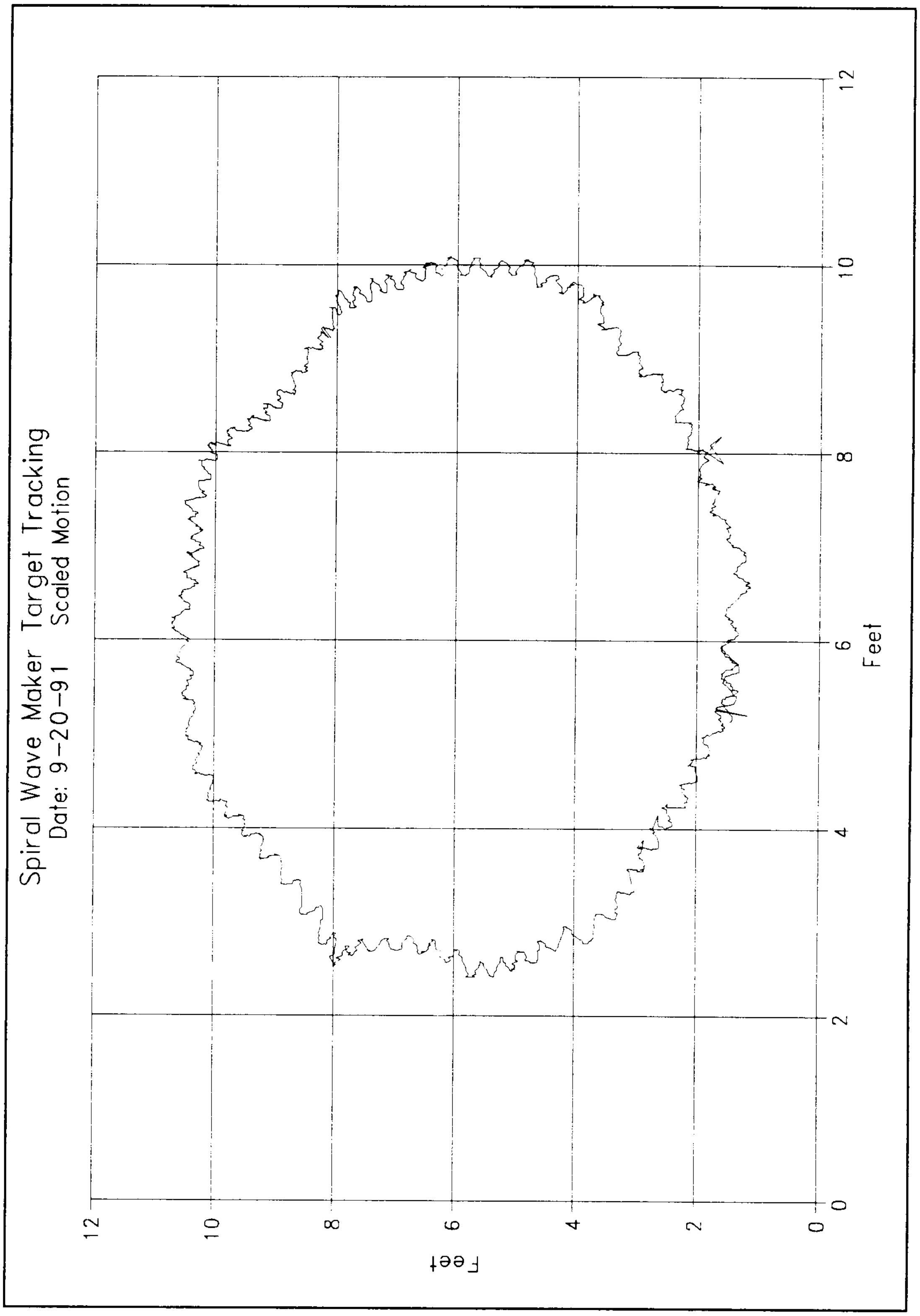


Figure 5.7 Longshore current model, video target tracking, feet

This was the first test using the 2-D model, since then several changes have been made. The original data was sampled at 4-5 Hz. and the non-steady sampling rate posed problems for data conversion. Light reflections were also a problem, thus a white background and black target were used. Precise 30 Hz. sampling is now possible and reflective painted targets are used to overpower water surface reflections. Multiple targets can be tracked now to create a better representation of the wave field in fewer cycles.

The required field of view dictates the target size. A slightly buoyant ball of 2.0 inch diameter is used to approximate a water particle. The buoyant ball is a source of error in the model, since the target does not represent true water particle motion due to its greater inertia. If a smaller field were required, a correspondingly smaller target could be used.

The single camera model limitations are apparent in this example. Only x,y locations are recoverable, but z elevations could be useful to determine water particle excursion and wave heights at various locations within the field. The 3-D model compensates for distortion and uses a more comprehensive calibration procedure. It has the potential for greater accuracy. An experimental verification of the 3-D model is presented in the next chapter.

5.4 Two Dimensional Model Procedure

A detailed description of the two dimensional procedure is presented in the following outline. Software names and required files are listed as well as instructions for anticipated problems.

1. Estimate anticipated motion. If motion is non-planar use 3-D model.
2. Orient camera image plane parallel to target motion plane. Maximize distance to target while retaining image field of view required for target tracking.
3. Place calibration points in target motion plane.
4. Place lights behind camera to illuminate calibration points.
5. Adjust field of view to encompass expected motion and lock camera controls.
6. Determine location of points in 2-D space by manual survey measurement.
7. Determine location of points in image plane.
8. Create ASCII text file, adhering to specified format, that contains calibration points 2-D world location and point identification numbers This is completed for each set of calibration points.
9. Execute program "subcapt.exe"
    a. Files required: "balls.dat" calibration world location file.
    b. Files created:
        i. "cam.dat" point screen location file.
        ii. "world.dat" point world location file.
        iii. "calibrat.prn" calibration point detail file.
10. Adjust size of box by following instruction on screen, if necessary.
11. Select gain adjust by holding down both mouse buttons simultaneously.
12. Move mouse to increase or decrease target contrast, then hit mouse button to exit.
13. Move box over calibration point.
14. Activate centroiding by hitting right mouse button.
15. Enter the number of the point selected on the screen.
16. Screen location and world location of point are written to files.
17. Process more calibration points? If yes return to step 13.

18. The 2-D scale factor is found using "balls.dat" and "world.dat. The world distance between points is divided by the screen distance between the same points.
19. Remove calibration points from view.
20. Place target in field of view.
21. Place lights behind camera to illuminate target.
22. Choose sampling rate desired, 5-6 Hz., or pseudo-real-time 30 Hz.
    a. If Pseudo-real-time, video record the event of interest, then set the video tape at the end of the event.
    b. If 5-6 Hz sampling, prepare event to be viewed.
23. Execute program "tsbackm.exe"
    a. Files required: None
    b. Files Created: user defined coordinate file.
24. Select adjust gain from menu.
25. Follow on screen instructions to obtain best contrast.
26. Select "multi-track" menu item.
27. Enter number of targets to be tracked and follow on screen instructions.
28. Either set video tape to pause and set the switch on the VCR to remote, or start the live event.
29. Place box(s) over target(s), adjusting box size where necessary, and follow on screen instructions.
30. Monitor processing in case of problem.
31. Stop tape or event and post process data file.
32. Multiply screen location differences by scale factor to determine real world motions. This is done using a spread sheet and the scale factor calculated earlier.

6.0 THREE DIMENSIONAL EXPERIMENTAL VERIFICATION

6.1 Overview

The 3-D, two camera procedure is more versatile and accurate than the single camera 2-D procedure. This chapter checks the validity and explains the implementation of the 3-D model. Video results are compared to results from electronic displacement measurement techniques.

The experiment was designed to check the algorithm, test the procedure, and examine the feasibility of the 3-D, two camera measurement process. The experiment used much of the hardware from the random mooring study described in Section 5.2.

The 3-D verification experiment setup is described, including lighting, timing and Cable Extension Transducers (CET) placement. The testing procedure is detailed, data processing technique is explained and the analyzed data are presented. Implementation suggestions are also presented.

6.2 Three Dimensional Experimental Setup

The experimental configuration is similar to the random mooring study. A 12 inch diameter sphere is mounted on a one inch square rod. The rod is secured to a support table at each end. The tables are secured with weights to prevent unwanted motion. Two overhead towers support a calibration grid above the rod. One camera is placed on each side of the rod with a tilt and pan angle of approximately 45 degrees relative to the rod center line. This provides out-of-plane motion for each camera. The analysis of the two images, however, should indicate a linear motion. A high power halogen light is placed behind and near each camera to provide illumination of the calibration points multiple spherical targets. The physical layout is illustrated in Figures 6.1 and 6.2.

A two inch spherical target is placed on a threaded rod seven inches above the large sphere. The spherical target has a common planar representation from any viewing angle. If three targets are used, six degrees of freedom can be resolved from the video data.

Since the motion is constrained to one dimension, only one target is required. The target is coated with 3M, 7210 Scotchlite reflective paint. The paint has the ability to channel incoming light rays and refract the light back to the source in a cone of light that encompasses the source. The reflective paint is used to minimize ambient light problems and to increase the contrast between target and background.

A calibration grid is outlined on the platform above the rod. This consists of a set of 16 calibration-wire support holes, spaced at precisely known distances in a sheet of plywood. This forms the basis of the world coordinate system. The longitudinal axis of the rod was aligned with the longitudinal axis of the grid of holes; this is the world x axis. The spacing of the holes, rod alignment and axis configuration are illustrated in Figure 6.3.

The CET (a string potentiometer) is placed at the camera end of the rod and a wire attached to the sphere. Longitudinal (x axis) position along the rod is obtained from the CET, accurate to 0.0625 inches and sampled at 30 Hz. The

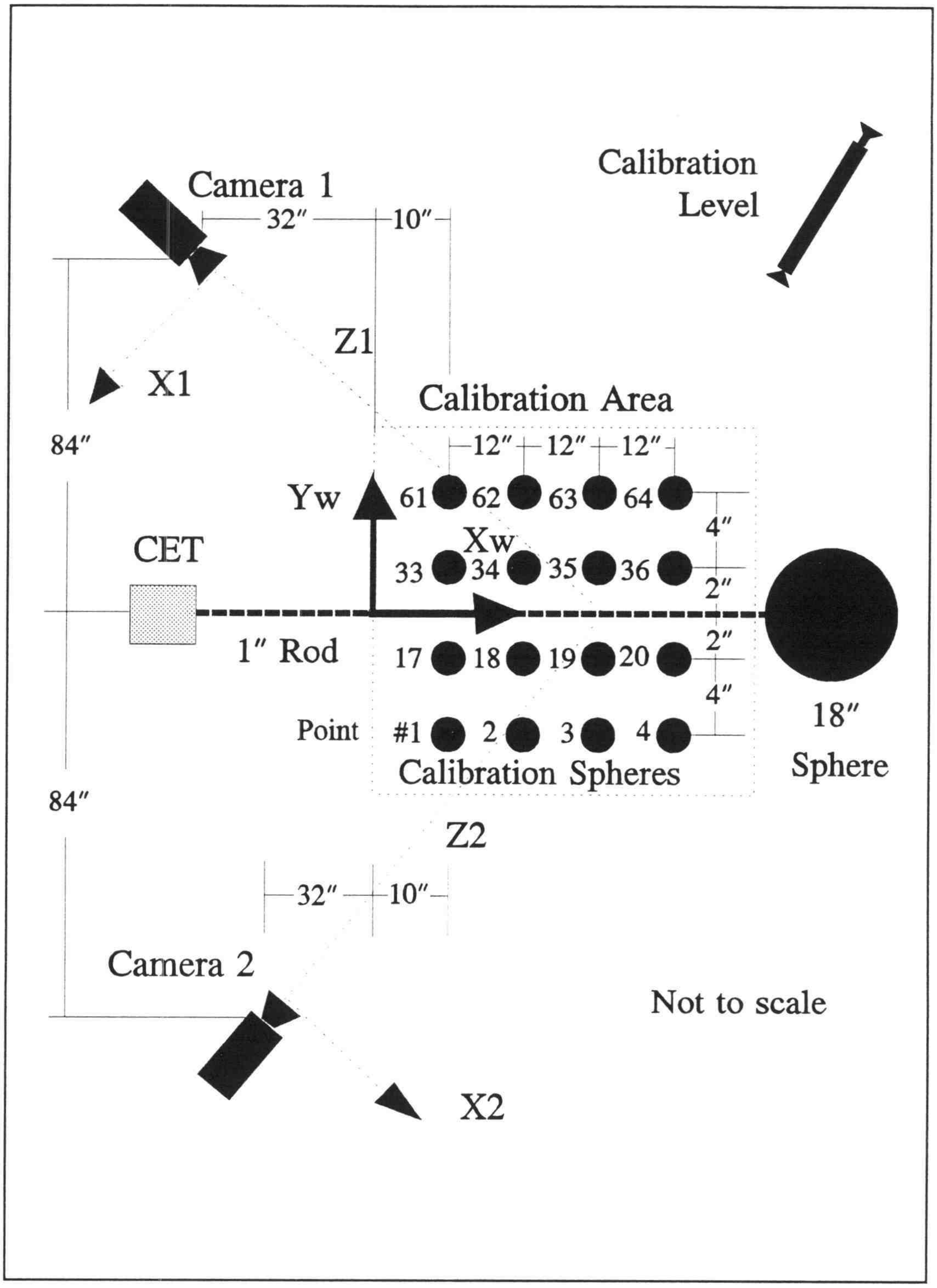


Figure 6.1 Experimental setup of 3-D translation experiment, plan view

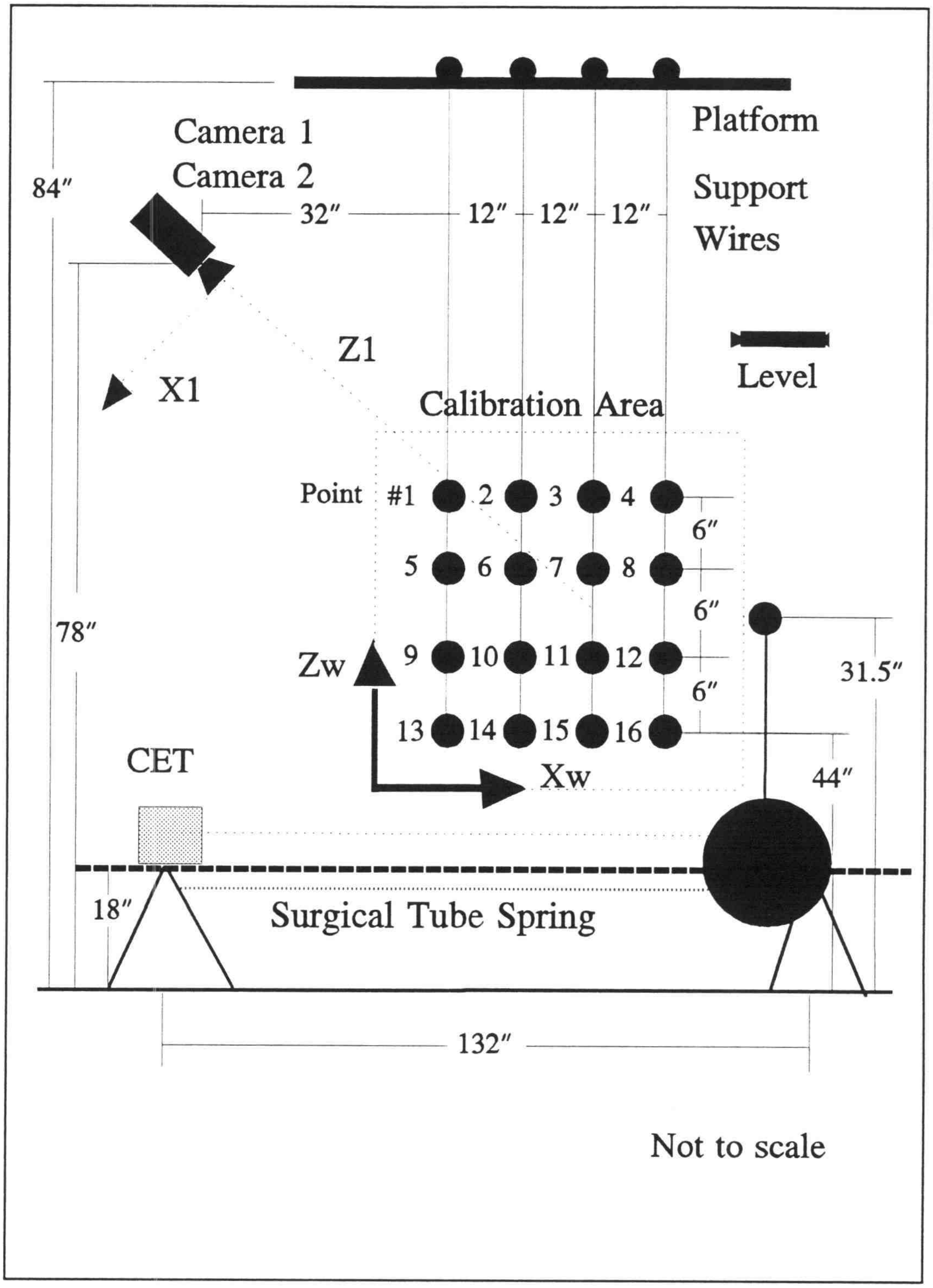


Figure 6.2 Experimental setup of 3-D translation, elevation view

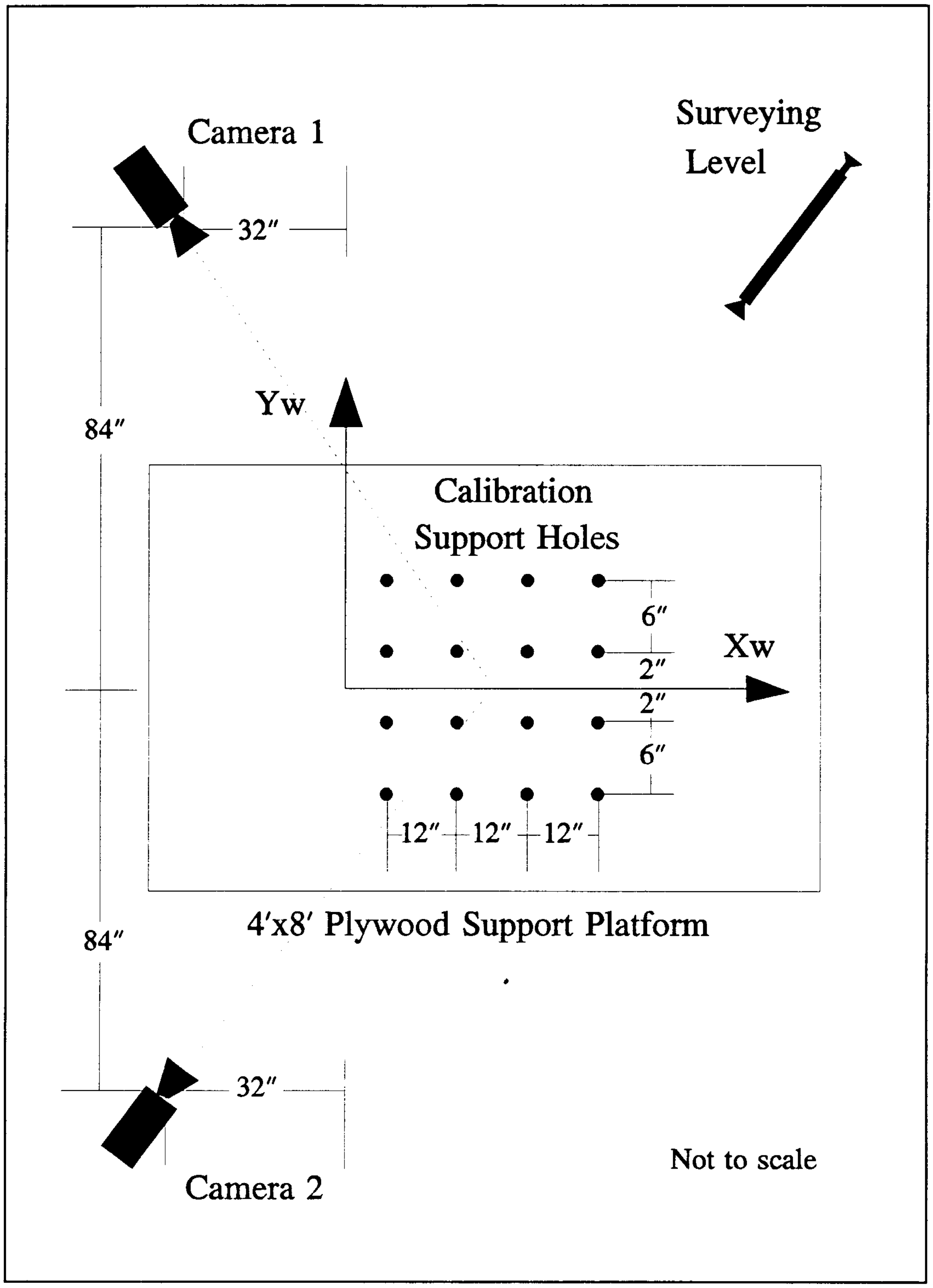


Figure 6.3 Experimental location of calibration support holes, plan view

CET data is collected using the following equipment: PC-AT, LabTech Notebook data acquisition software, WaveTek Rockland anti-aliasing filter, Unimeasure CET and a custom CET signal conditioner.

Since the target ball is rigidly attached above the sphere and the guide rod is aligned with the world x axis, the CET data is directly comparable to the world x coordinate data. Motion is constrained to the x direction. Some perturbations are observed in the world x and y direction, however, and measured with the video system.

The sphere is spring supported with surgical tubing in the +/- x directions. The rod is secured to a table at both ends. This provides a restoring force in the longitudinal direction along the rod, and allows forced simple harmonic motion and damped periodic behavior to be observed.

Both video cameras and the CET data are synchronized using a WWV universal standard, atomic clock time signal. This is done to ensure individual position information from each camera and the CET's can be correlated with a common time base. The video image from each camera is recorded on S-VHS tape for processing and archival purposes.

6.3 Calibration Procedure

The calibration of the video cameras is performed before any data is acquired. The calibration is recorded on video tape to verify that the cameras were not moved during the test.

The calibration is performed using 16, two inch balls that are coated with 3M reflective paint. Four balls are threaded onto thin wires at a center to center spacing of 6.0 +/-0.0625 inches. This is accomplished by placing small metal collars with allen screw locks at 6.0 inch intervals. Then four wires with four balls each are hung from the calibration support platform holes. A small weight is attached to the bottom of each wire to insure that the wire is plumb. One row, aligned with the longitudinal axis of the rod, is set at a time. A surveying level is used to adjust the top ball of each column to the same height, +/- 0.0625 inches. Small nylon cord lock devices are used to simplify this adjustment. A small sign denoting the row number is placed in the field of view of both cameras. This is shown in Figure 6.4.

The lights and cameras are turned on and the lights adjusted for maximum reflection to the camera. The auto focus and auto iris are turned off and the remote controls locked out to eliminate the possibility of changing any setting after calibration. Several minutes of video image from each camera are recorded. The four columns of balls are then moved to the next row of holes, leveled to the same elevation as the initial row. The sign is changed to reflect the new row number and additional video images are recorded from each camera. This is repeated for the two remaining rows. This creates 64 calibration points whose precise location are known in the world coordinate system. Some of these calibration procedures are similar to those used by Jim Walton of 4-D Video, Sebastapol, California.

The use of the platform supported calibration wires as targets allow precise x,y location of the calibration points. By pre-measuring the wires, using a level

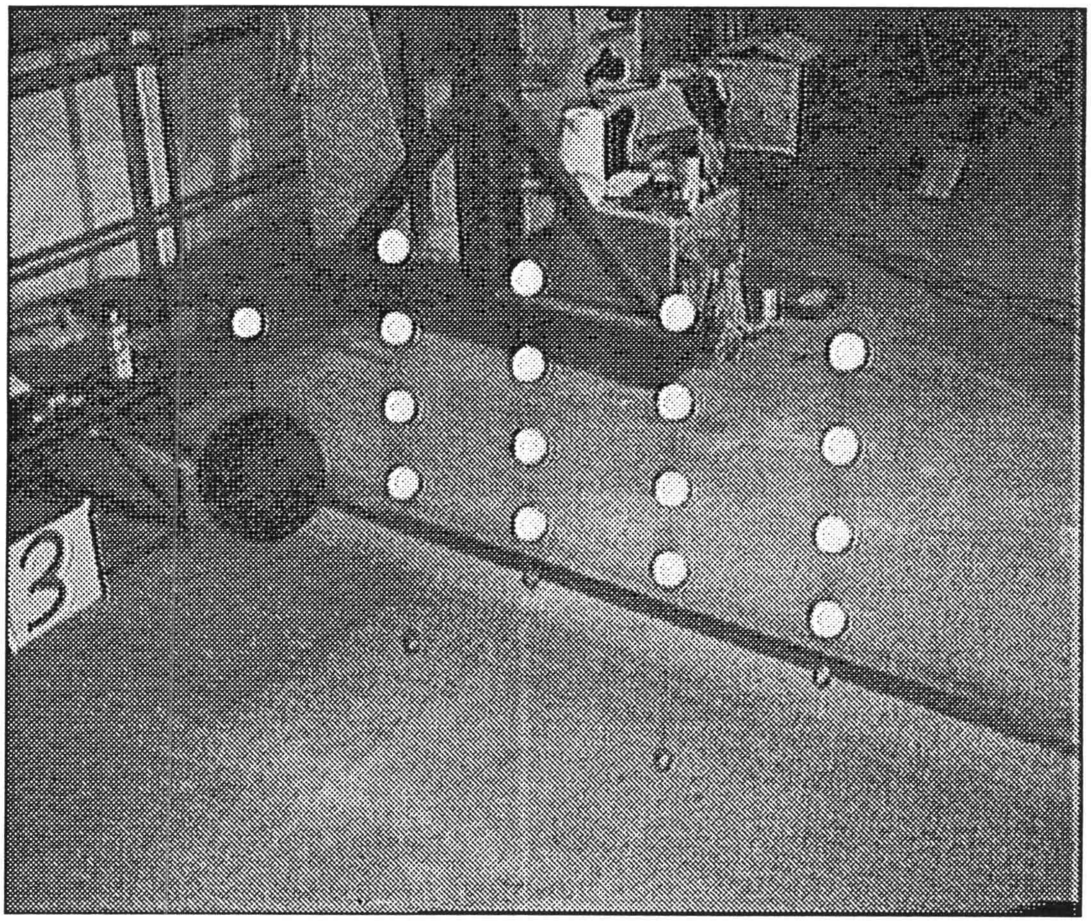


Figure 6.4 Digitized image of a single row of calibration points and the target ball

and placing the collars at the pre-determined distances, the inter-point spacing is set, establishing the z location of each ball. The use of the plumb line insures that all targets in a column of a specific row are at the same x,y location.

The use of the reflective paint creates excellent target-background contrast from any viewing angle. It also ensures that the target appears solid in the digitized video image. By having a solid spherical target, with a normal reflective surface, the video centroid becomes the center of the target ball, for each viewing angle.

Confusion between points is minimized by setting one row at a time. If 64 balls are hung at the same time it would be impossible to get a clear view of every ball from each camera. This would also require 64 painted balls, which would consume additional effort and expense.

6.4 Experimental Procedure

Once the calibration is completed, the sphere with the target ball attached is centered in the calibrated space. The surgical tubing and the CET cable are attached to the ball. Two tests are performed in air, similar to the random mooring test.

In the first test, the ball is plucked. After both video tape recorders are started and the CET data acquisition initiated, the ball is pulled from the equilibrium position approximately twenty four inches. It is then released. The resulting motion is similar to the damped oscillation of a spring-mass-dashpot system.

The second test consists of supplying a manual sinusoidal forcing function to the ball in air, approximating wave forces. An approximate one second period is induced and sampled at 30 Hz. In both tests, the cameras are set with a shutter speed of 1/1000 of second. If the shutter option is not used, the motion is blurred significantly, due to the large velocity of the target,

6.5 Video Post Processing

The calibration and tests are recorded onto video tape with a common time source, allowing each tape to be processed separately. This allows the calibration to be performed and repeated. The video tape also provides a permanent record for archival purposes.

The calibration section of the tapes is processed first, before any tests are performed. This is done to verify that the camera positions are resolvable. The tape from camera one is played and custom sub-pixel point location software, "subcapt.exe", is used to locate each calibration ball and superimpose the centroid on the video tape image.

The tracking software, "tsbackm.exe", facilitates gain adjustment to maximize contrast of the ball against the background. An adjustable size box mask is placed around the calibration ball with the mouse, and then established by a button press. The software then calculates the centroid of the target inside the box, using the centroid formulae, Equations 4.2.1 and 4.2.2. If the target is composed of more than ten pixels, the centroid is reported to decimal accuracy. It is not possible to display sub-pixel locations on the screen, however this accuracy is critical to the calibration program itself.

The centroid location is entered into a file. The point identification number (Tag) is input and the world x,y and z are then entered into a world file. The sequential point numbers are entered into a file "calocate.prn" along with the exact x,y and z location previously measured. This file is read at start up by the sub-pixel location software, "subcapt.exe". This speeds calibration, and reduces the possibility of error. It is transparent to the user.

All 64 calibration balls are located in the first camera tape. The screen and world location are written into files using the "subcapt.exe" software. The second camera tape is processed and the world and screen location recorded. Four files, camera one, camera two, world one, world two, are processed by the calibration software "calb.exe" and also back projected on the screen by verification software, "back.exe", to visually corroborate the calibration.

After verifying the calibration, the test runs are performed. The video recordings from these runs are processed, one camera at a time. The 2-D verification software, "tsbackm.exe" is used for the processing. The screen location of the target ball is written to a user selectable, intermediate file with the corresponding time, frame by frame. The second camera video is then processed. The two screen location files are then input to the world location software, "locate.exe".

The world location software reads each of the camera calibration files, the screen location files, and calculates the 3-D world location of the targets, frame by frame. To the software, the moving target is a series of still images. The 3-D world locations of the target are then compared to the CET data.

6.6 Data Analysis

Once the individual camera video tapes are processed and the raw screen locations resolved, the calibration is performed. The calibration software, "calb.exe" reads the screen location file, and the world location point file, then performs the calibration. The resulting intrinsic and extrinsic camera parameters, defined in Section 4.4, are written to a file for use by the verification and location software, "back.exe" and "locate.exe" respectively. The second camera is processed in the same manner.

The 3-D back projection verification algorithm, "back.exe", takes the camera parameter files and raw screen location files from each camera, and then calculates the world location for each calibration point. These values are compared with the actual measured world location. The maximum x, y and z errors as well as the average x, y and z errors are determined.

The calculated world location for each calibration point is then entered into the fundamental coordinate transformation Equation 4.4.1. The undistorted location is then determined using the pinhole camera Equations 4.4.4, and 4.4.5. These values are input into Equations 4.4.6 and 4.4.7. The distorted two dimensional values are then input into Equations 4.4.8 and 4.4.9 and a frame buffer location is determined. These locations are then plotted along with the original point locations on the overlay screen using "showcalb.exe". The original video is also viewed simultaneously.

The accuracy of the calibration is determined at this time. The initial analysis showed the back projected location of some points visibly different from the original locations. The average errors and the maximum errors were an order of magnitude larger than the world point measurement accuracy. The fixed camera parameters, sensor pixel distances (dx, dy) and the image center offset (Cx ,Cy) were then optimized to give the best results. This is justifiable since the values for dx and dy are reported by the camera manufacturer, and slight variability can exist in the CCD manufacturing process. The image offset (Cx, Cy) are not directly measurable and change with varying zoom factors.

In Tsai (1986), several variables are discussed that can be optimized. They primarily are the image center locations. Tsai reports that the image center can vary by over 25 pixels from the numerical image center, in both X and Y directions. The image sensor size reported by the camera manufacturer may also be inaccurate by 1%, and that would be +/- 0.05 mm, or up to 7.2 pixels.

The camera sensor size and image center values are changed by a small amount (Cx and Cy +/- 25 pixels, sensor size +/- 0.2 mm) and the calibration software executed. This is repeated until the average error is minimized. The amount the parameters are adjusted are within the values recommended by Tsai, this optimization process takes less than ten minutes and is accomplished by varying parameters in the "camparm.dat" file until the smallest error output is seen from the "back.exe" calibration software.

The final calculated point locations are then back projected onto the screen along with the original points using "showcalb.exe". Most points correspond closely to the original, some are slightly off by three to six pixels. Accepting round-off errors and the inability to display sub-pixel locations, this level of calibration closure is within tolerable limits.

Once the calibration is completed, the experimental runs are performed and the video processed. Point location software "locate.exe", similar to the verification software "back.exe", is used to determine the target location in each frame. These values are entered into a file to be compared with the CET data. As mentioned earlier, the motion is seen by the software as a series of still images, similar to the calibration point images.

6.7 Results

The video data provides x, y and z positions, however, only the x location can be numerically verified. When the video and the CET data were first viewed graphically an interesting anomaly was discovered. The coordinated universal time (WWV), was not the same. This is shown in Figure 6.5 and in Figure 6.10, where the CET data is offset a constant 0.2 seconds (six frames) from the video data. The video lags the CET Data. The 0.2 second offset appeared in both the plucking and the sinusoidal test. The time offset is caused by hardware delays in both data acquisition systems. The delay is small and is eliminated numerically by subtracting 0.2 seconds from the video position time record. This accounts for the

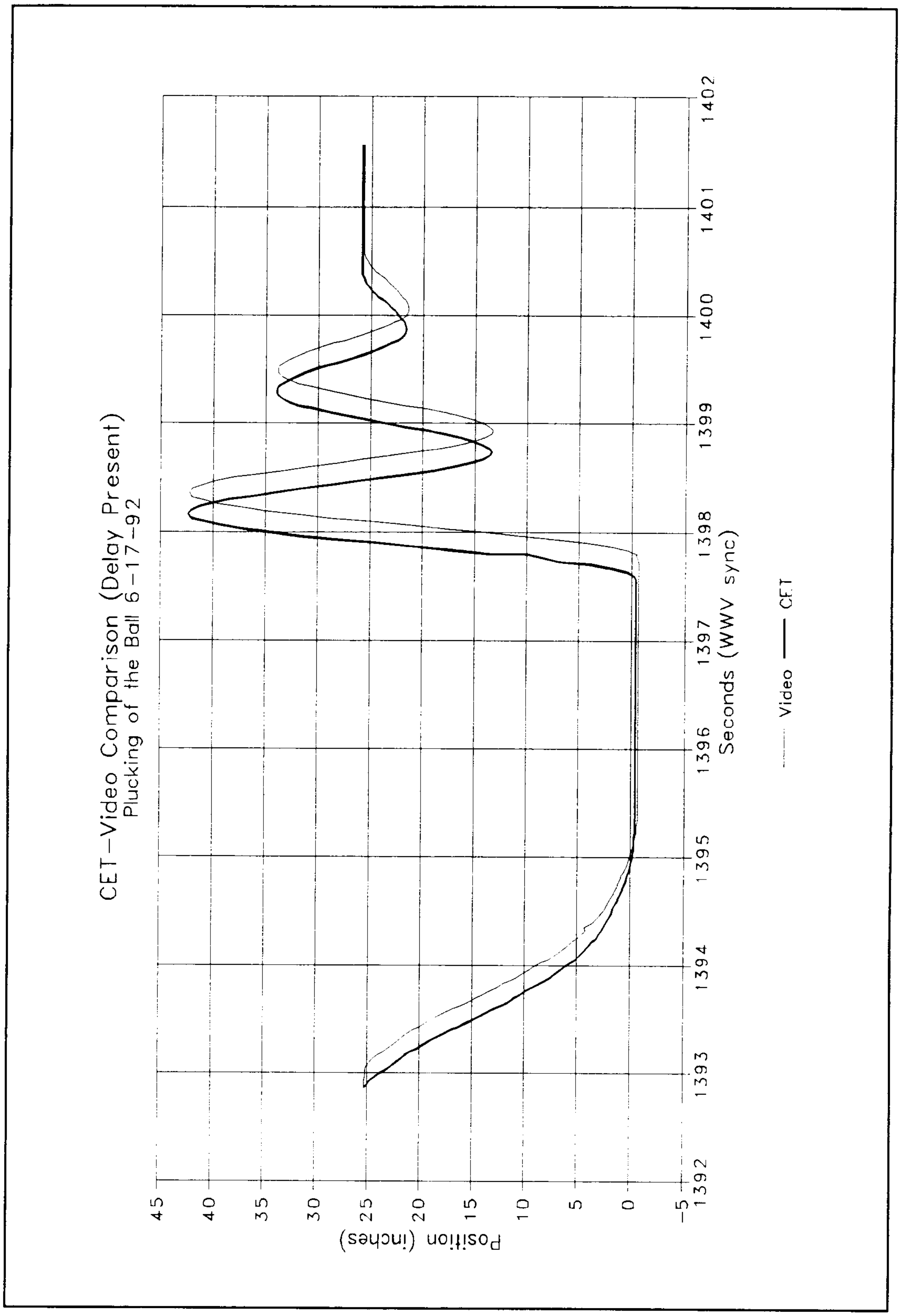


Figure 6.5 Comparison of video X position and measured CET response

offset between video and CET. The CET data could also have a delay, but since it was used as a control it was assumed to have no error.

When the offset is graphically eliminated, near perfect correlation is observed as shown in Figure 6.6. The video trajectory coincides with the CET data. One dimensional motion appears one dimensional in a three dimensional measurement space. Neither camera image plane was parallel to the plane of motion, which is a requirement in the 2-D case. Also, as the motion approaches the edge of the field of view the results do not degrade, as is the case with the 2-D test. The individual data points are illustrated in Figure 6.7.

The y and z trajectories are shown in Figure 6.8. The z location shows slight variance with time with maximum displacement of +/- 0.2 inches. The y location shows a more significant deviation, with a maximum displacement of +/- 0.725 inches, from what should ideally be zero. During the pluck test, noticeable y motion is visually observed. As the ball is pulled into the initial location a torque may be applied to the apparatus. When released, the target ball vibrates in the y direction approximately +/-0.75 inches. This vibration is enhanced by the elasticity of the target ball support rod. Motion in the z direction is also observable when the target ball is loaded vertically. Both of these motions are due to tolerances in the rod-ball linear bearing connection and a slight flexure of the target support mechanism.

The same observations are true in the manually forced periodic motion. The time offset is constant throughout the test and an interesting periodic motion is shown in the Y displacement of Figure 6.9. A smaller portion of the test is

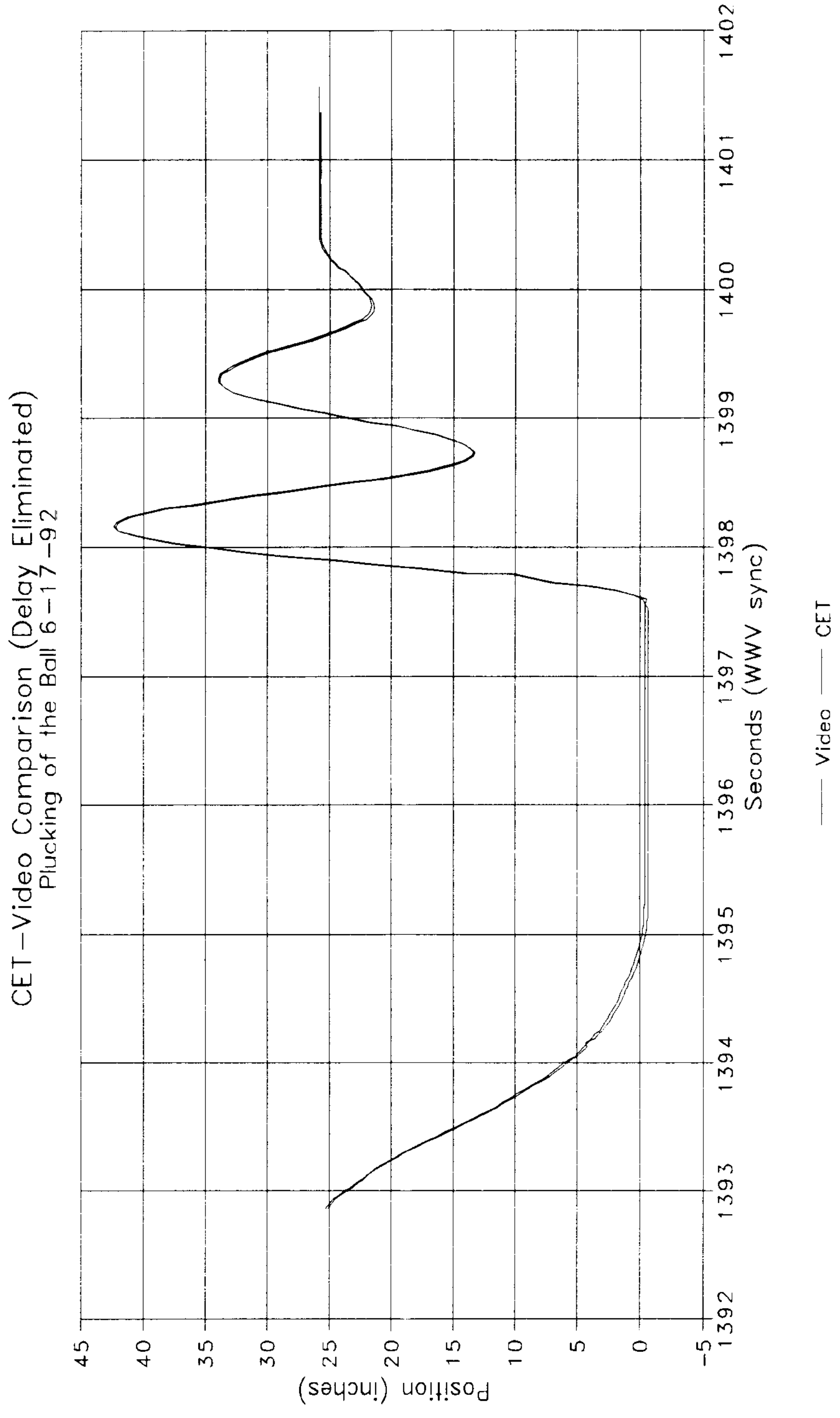


Figure 6.6 CET measurement projected forward six frames (0.2 seconds)

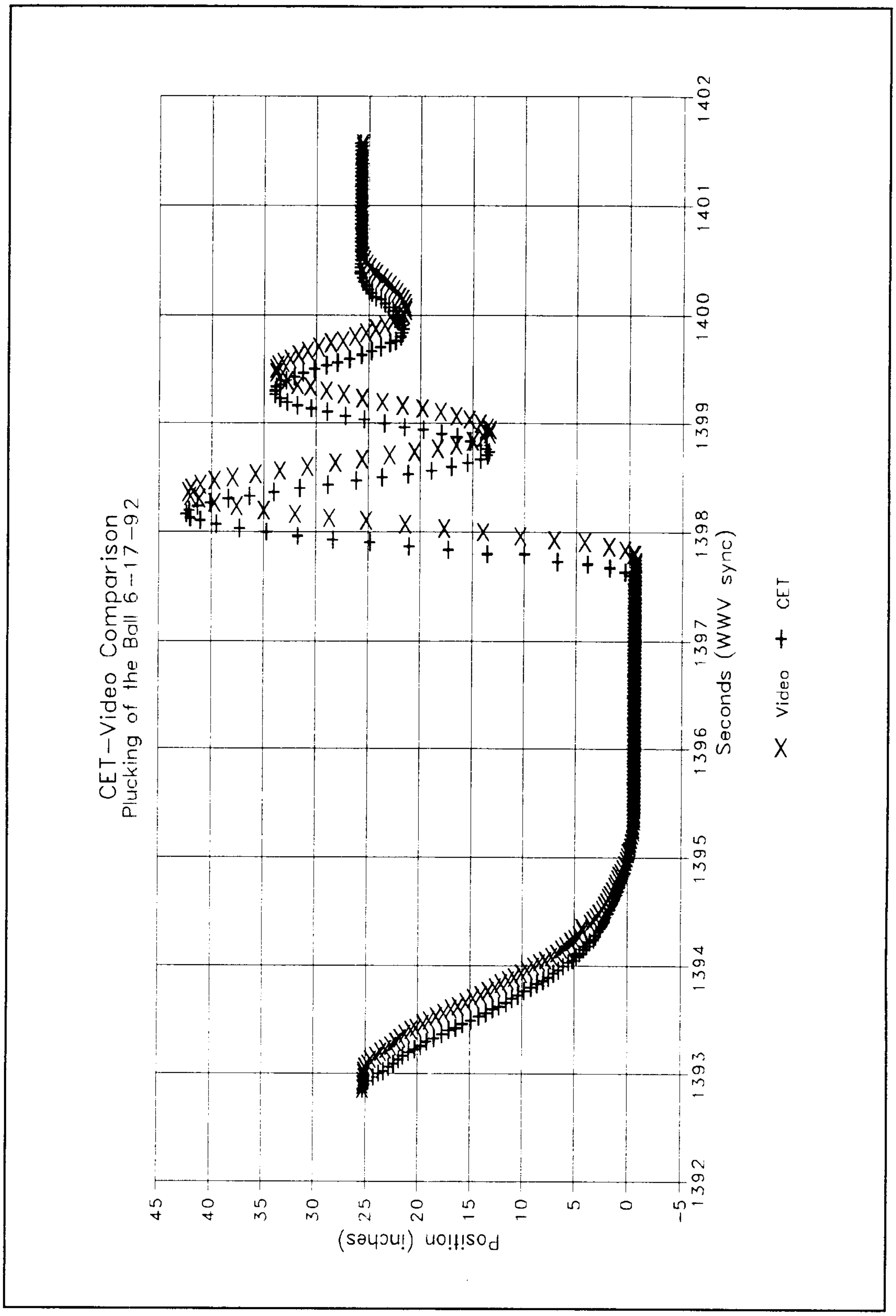


Figure 6.7 Digitized video and CET records without time corrections

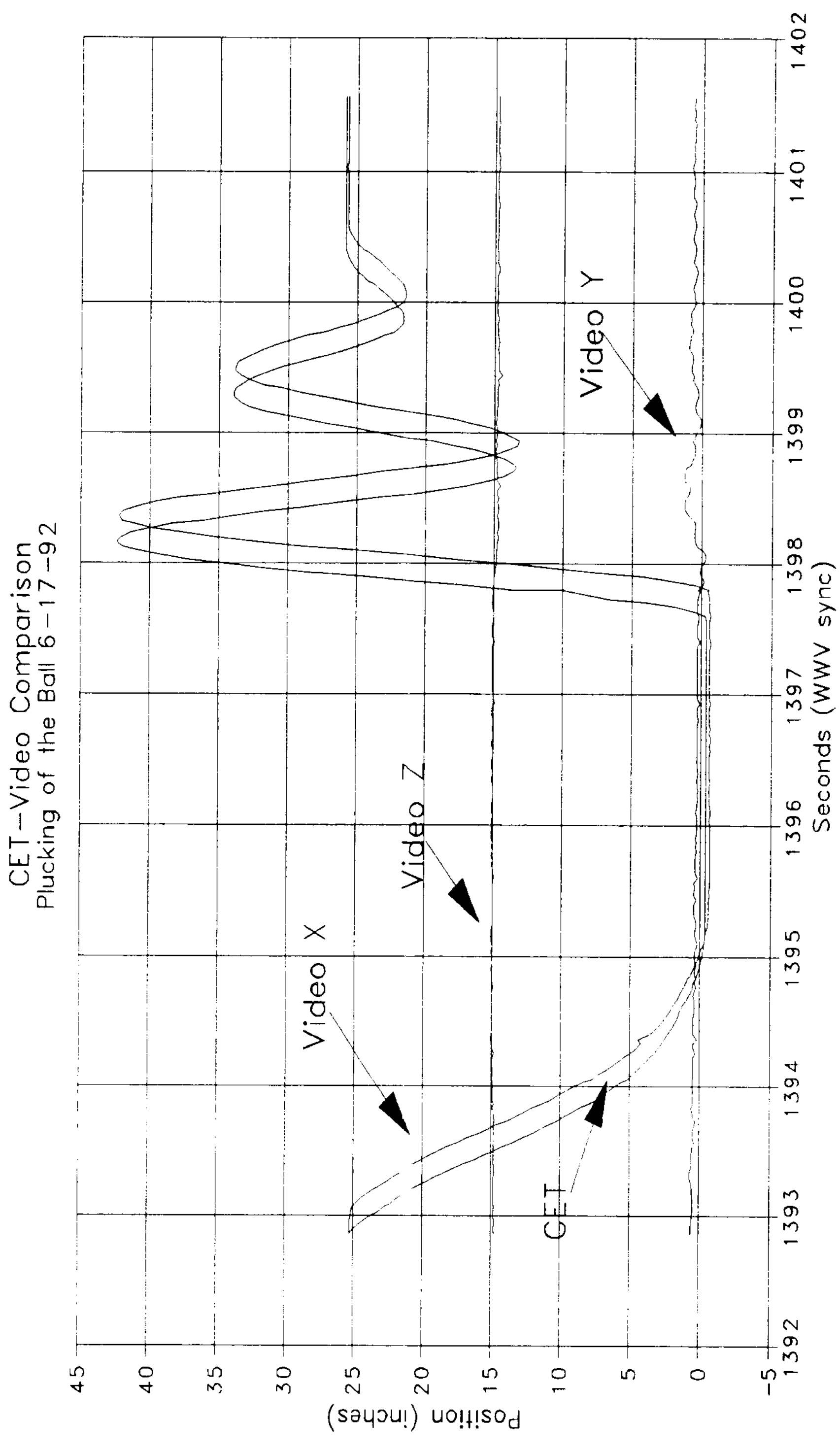


Figure 6.8 Video X,Y,Z trajectories

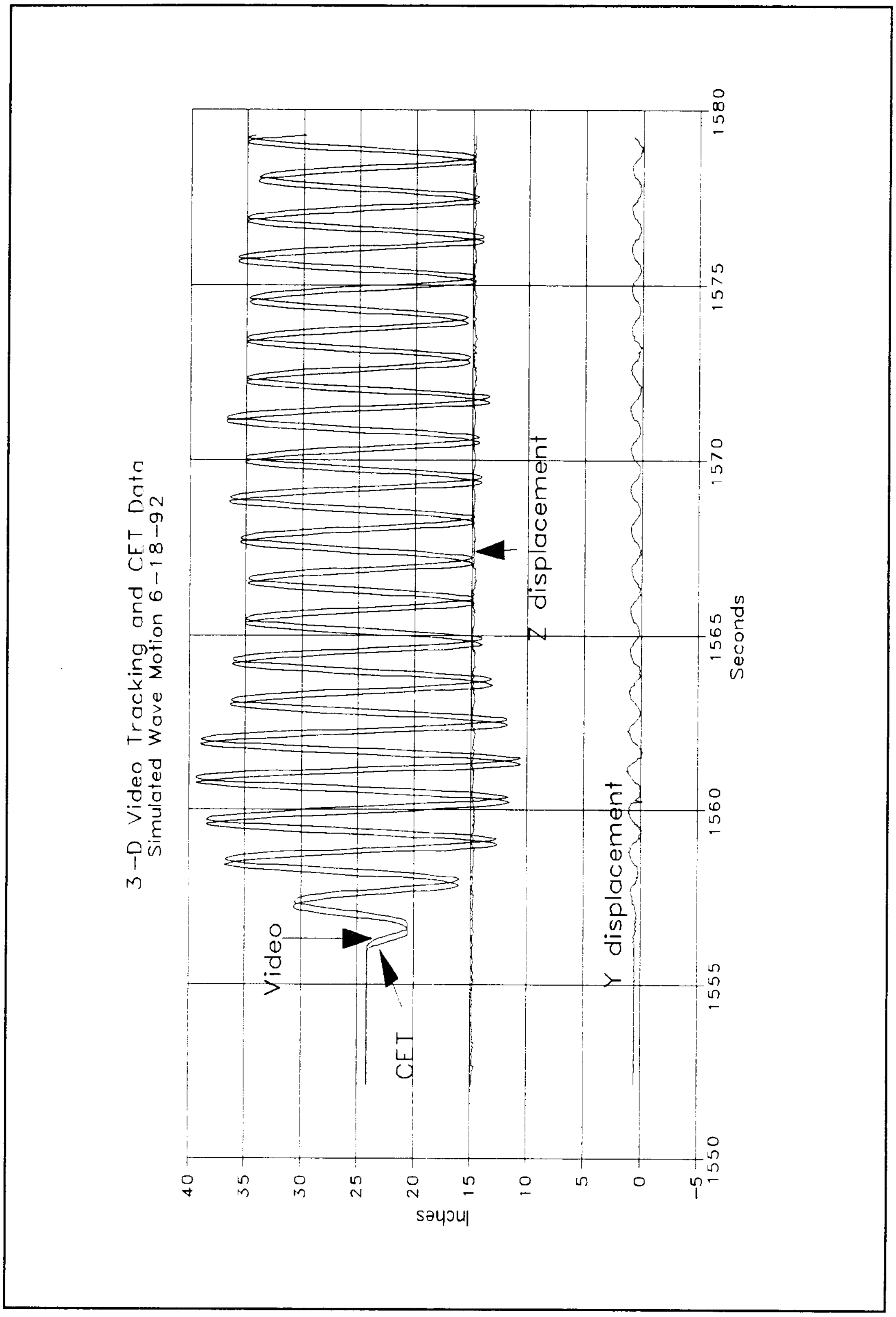


Figure 6.9 Periodic motion video measurement and comparison with CET position

enhanced in Figure 6.10, and the time offset eliminated in Figure 6.11. As with the pluck test, the video X position and CET data are nearly identical.

The sinusoidal motion in the y direction is caused by torsional vibration due to a loose bearing and an elastic rod supporting the target sphere. This was not intended but highlights the usefulness of the video system. The video system eliminates the need for multiple CET's and a complex decoding algorithm in 3-D location of a target.

The constraining rod was misaligned by approximately one inch over a 30 inch path length. This was determined by dropping plumb lines from either end of the world axis coordinate support frame. Figure 6.12 shows the amount of skew of the rod in relation to the world coordinate system. Slight y vibrations are also visible in Figures 6.9, 6.10, 6.11 and 6.13, as well as vibrations in the z direction. These are not directly verifiable, but were observed to be within the visually estimated values.

A data anomaly is observed in the y displacement at 1560.25 seconds in Figure 6.13. When the raw screen data was reviewed the error was discovered in the X screen location from camera two. The X value was slightly larger than it should have been. When the video tape was reviewed no problems were apparent in that particular frame. The video tape was reprocessed and the location software executed. The anomaly was reduced. The problem is assumed to be caused by VCR to image processor transfer. Figures 6.9, 6.10 and 6.11 are the reprocessed results. Another possible explanation is a dimple or rough area on the rod that

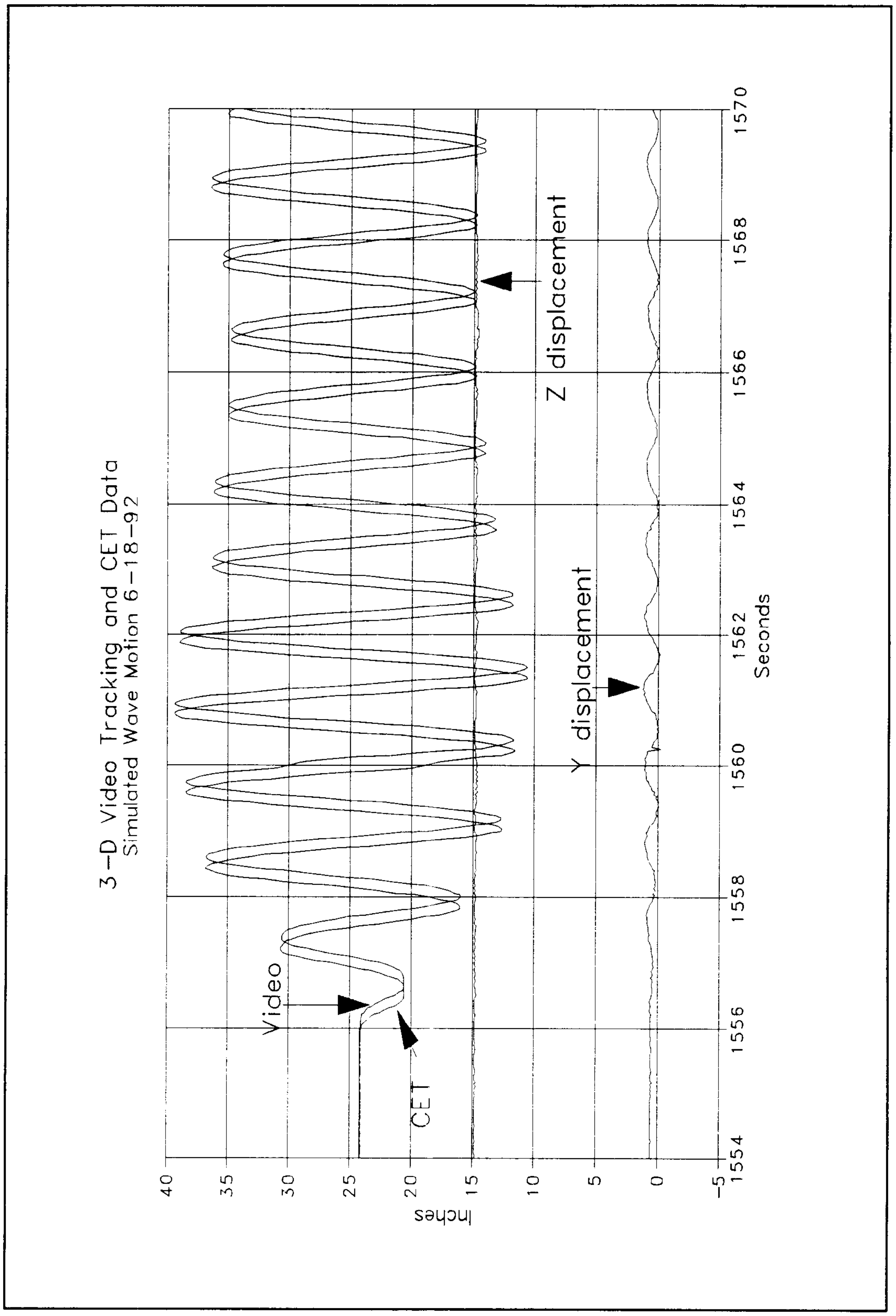


Figure 6.10 Enhanced video and CET measurement comparison

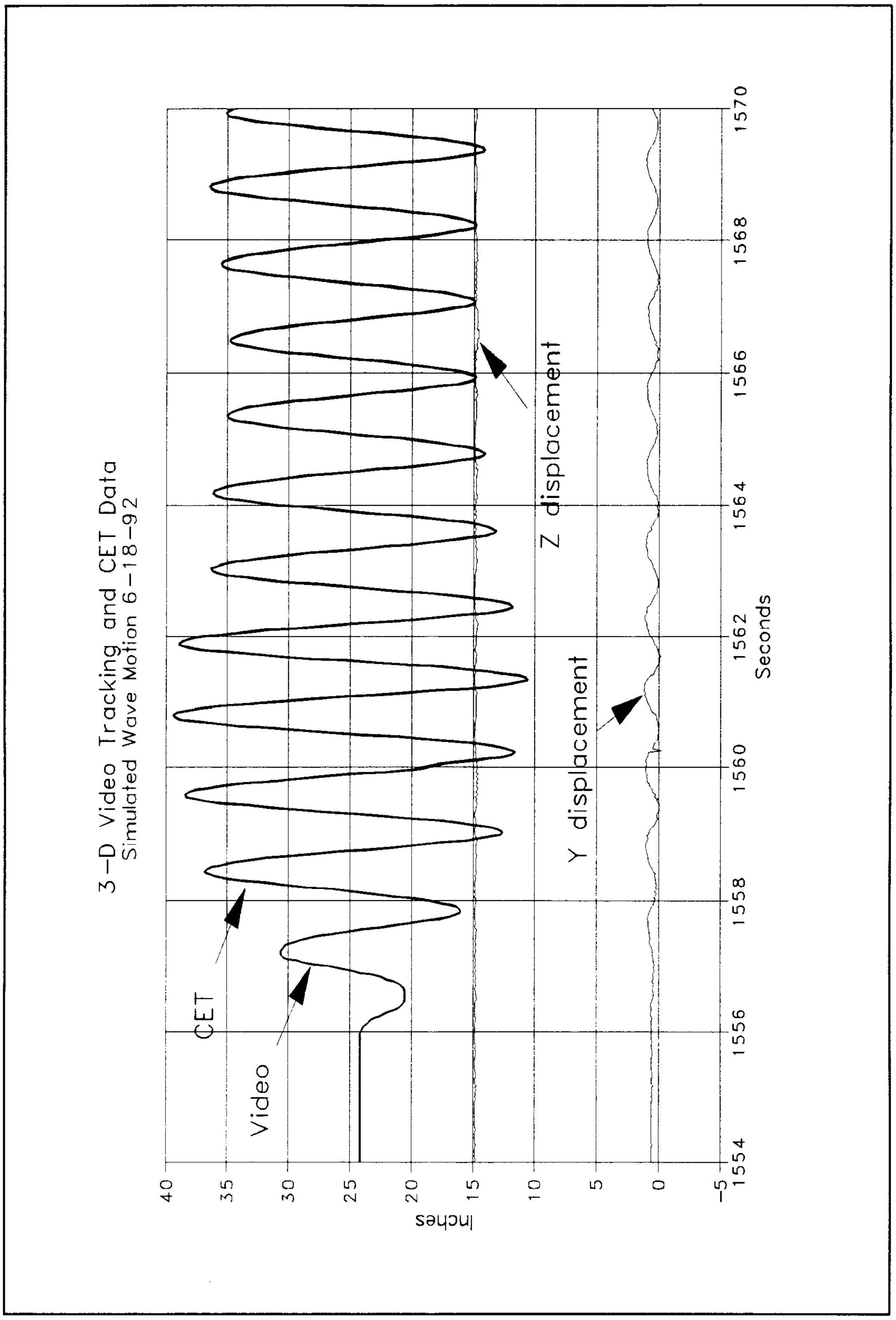


Figure 6.11 Comparison of video and CET x displacements with time lag removed

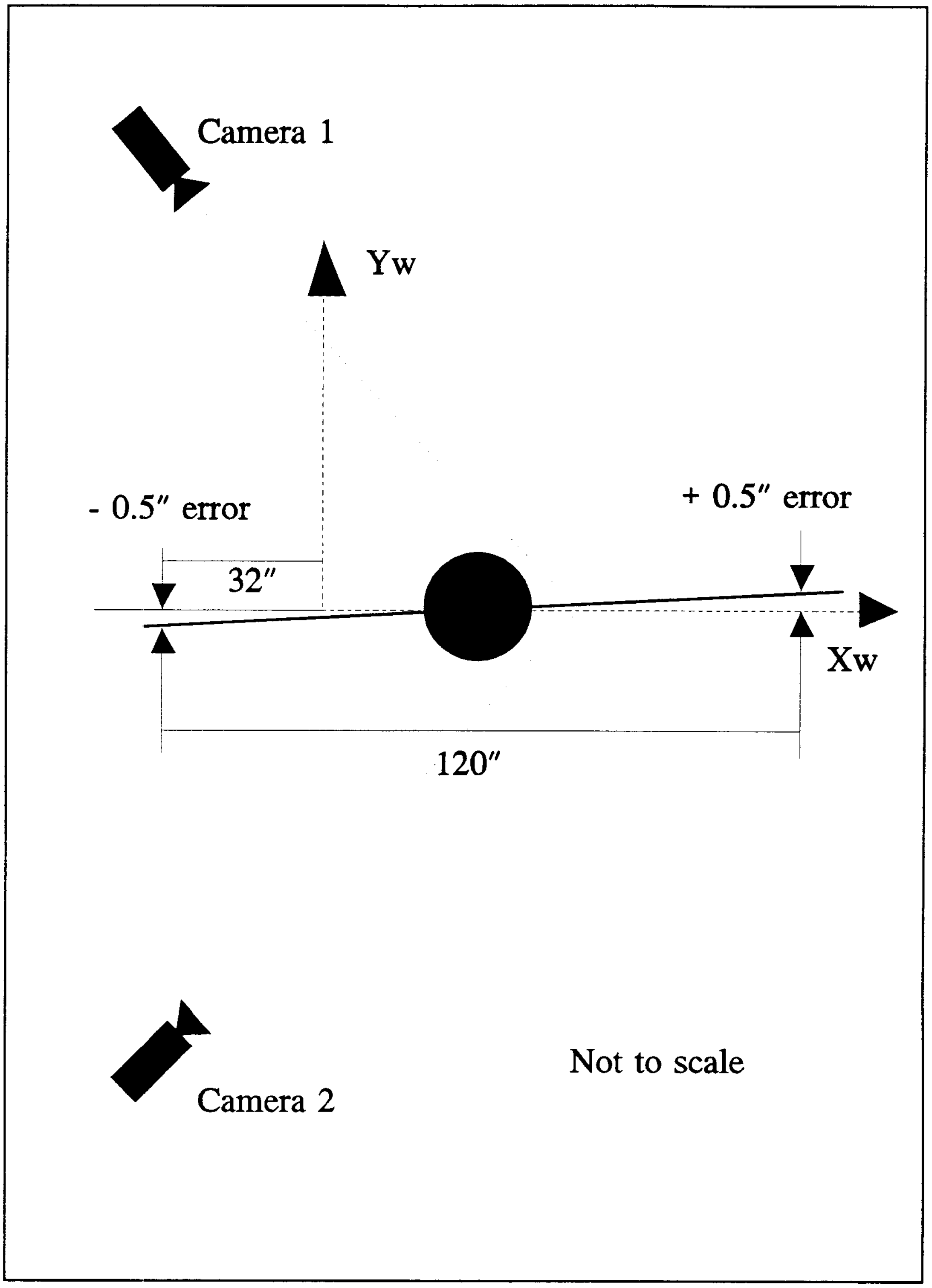


Figure 6.12 Support rod misalignment

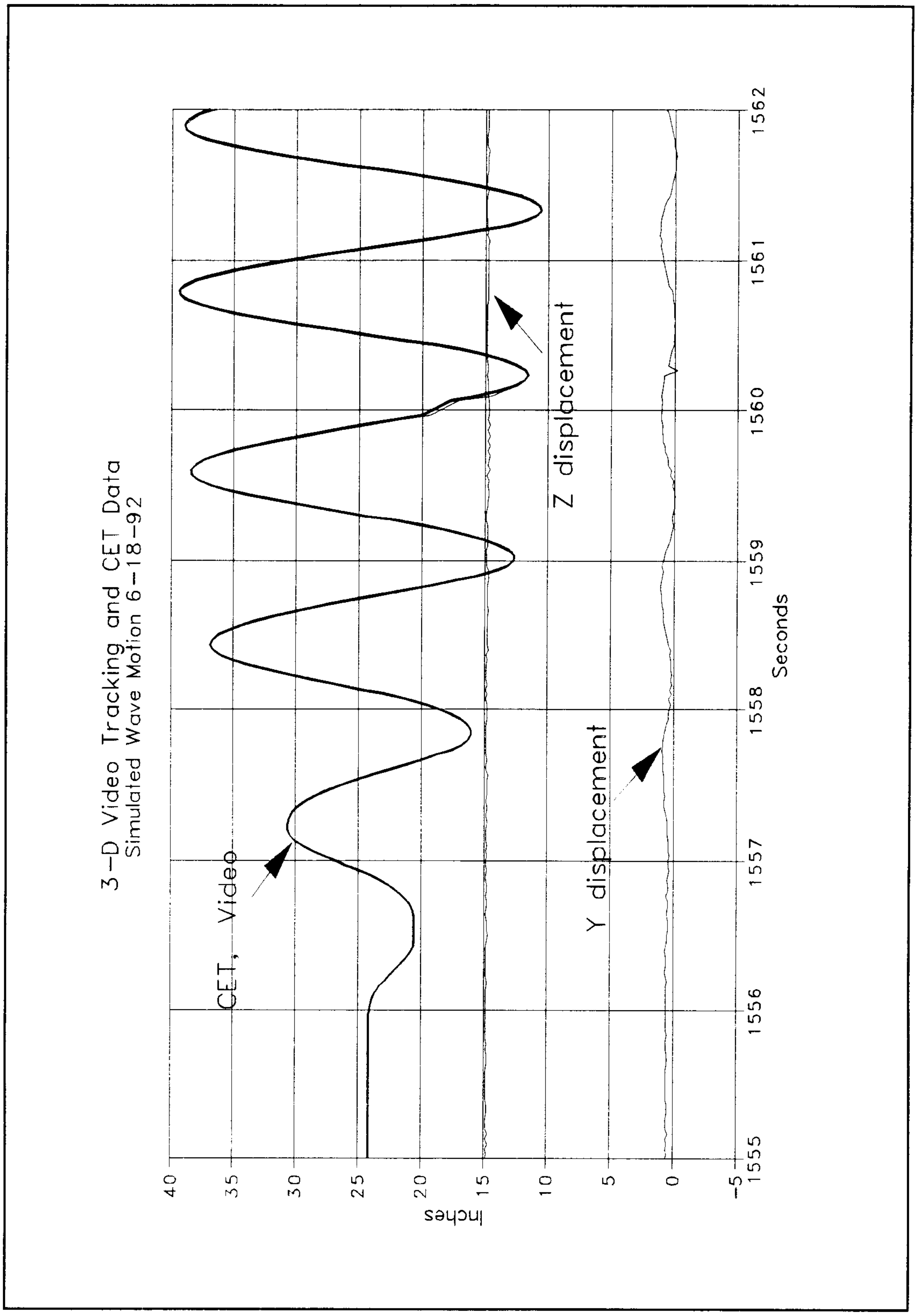


Figure 6.13 Video and CET position with image processing error at T=1560.25 seconds

caused momentary seizure. The CET data also shows a slight variance in 1560.10-1560.25 second time span, coinciding with the video problem.

The accuracy of the video location system is shown in Figure 6.14. The figure shows one second of video with 30 samples. The time offset was reduced but not totally eliminated because it is not an even frame multiple. If the fractional offset is removed the two data sets plot virtually on top of each other.

6.8 Discussion

The results obtained from the calibration are acceptable, after optimization of the image center position. When the calibration points are back projected onto the screen, the points are very near or coincident with the original point location.

One would not expect the results from the calibration to be better than the error in the initial measurement. The calibration balls are not perfectly round, with a nominal diameter of two inches. Depending on which axis is measured, the diameters vary between 2.0625 and 1.9375 inches. Also the grid on the support platform and the initial ball elevations are measured to accuracy resolution of +/- 0.03125 (1/32) inch.

The most critical value is the pixel-inch ratio at the target location. The field of view for each cameras is estimated to be 128 inches and the horizontal pixel field is 512. That results in one pixel spanning 0.25 inch. As discussed earlier, sub-pixel location is limited to 0.5 pixel, which corresponds to 0.125 inch video resolution at the target ball location. This is the limiting error. The average x, y and z error for all 64 calibration points is 0.11 inches with maximums of 0.231, 0.436 and 0.149 inches respectively.

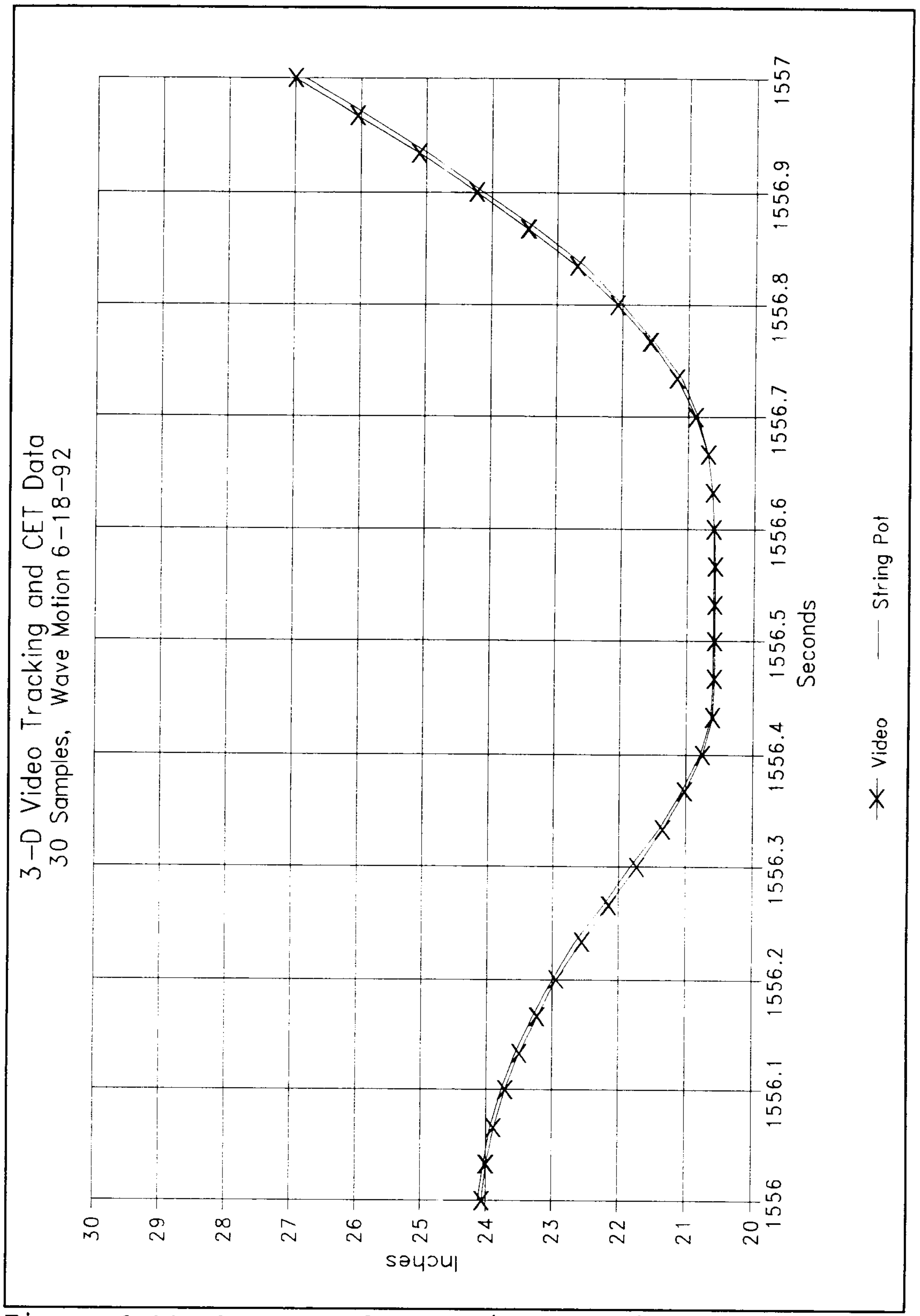


Figure 6.14 One second comparison of video and CET data

One interesting observation of the calibration is the fact that no calibration points are in the line of motion of the target. The target is three inches from the upper and lower rows of calibration spheres and two inches from the side rows of spheres.

The reason for the larger y maximum error is related to the calibration point location. The largest y errors are at the points that are at maximum distance from both cameras. This is at the far end of the support rod. The distance to the target is increased at those points, and the distance represented by one pixel is much larger, on the order of 0.5 inches per pixel.

The measurement accuracy in the y direction is also influenced by the camera location. Resolution of the y-axis is a combination of X,Y screen location from each camera. At the angles the cameras are oriented the screen mapping is significantly distorted. The z location, on the other hand, is more dependent upon the Y screen location of each camera. The Y screen axis is in the same plane as the z axis.

The moving target results agree well with the string potentiometer CET data, with a few explainable anomalies, the most notable being the 0.2 second time offset. The source is primarily a delay caused by the digitizing hardware. By performing several tests it has been determined that the time read by the data acquisition system is delayed a fraction of a second. Similarly the time sent to the VCR is also delayed a fraction of a second. These delays are not precisely the same, thus the offset of 0.2 seconds.

The misaligned coordinate system illustrates the advantages of the video system. By using only CET's it would be difficult to rapidly resolve the position of a object moving in three dimensions. The vibration of the target ball also illustrates the versatility and sensitivity of the video system.

The most important step in the 3-D algorithm application is the calibration. If many bright large calibration targets are placed very accurately, excellent results are obtained. It is recommended that the calibration be performed, image center optimized and the results back projected to visually verify the target locations. This should be done before any data runs are performed, eliminating the possibility of lost runs due to a bad calibration.

The calibration tape should also be viewed for each camera periodically to verify stationarity of the cameras. This is accomplished viewing a live image, then the video image. This was done in the pluck test and slight movements were noticed which were caused by a sagging tilt motor. The camera was manually brought back into alignment using the original calibration tape. It is also recommended that the calibration process be repeated at the end of a test sequence and the results compared to the initial calibration.

The calibration procedure works well for this medium scale setup. It would also work as well with a smaller setup and smaller targets. In large scale experiments, spanning tens of feet, it is difficult to build a large calibration support grid that could support a person to install and adjust the wire target assemblies. In these cases individually supported large target balls could be precisely located with optical or electronic surveying equipment.

6.9 Three Dimensional Model Procedure

A detailed description of the three dimensional procedure is presented in the following outline. Software names and required files are listed as well as recommended solutions for anticipated problems.

1. Determine if expected target motion is two or three dimensional.
   a. If two dimensional, use the 2-Dimensional procedure in Chap. 5.
   b. If 3-Dimensional, continue.
2. Establish world coordinate system. It should be selected such that comparisons with data from other measurement systems are straight forward.
3. Determine camera image sensor size and resolution.
4. Orient cameras so the target motion area is in the center of the field of view. The optic axis of the first two cameras should be approximately 90 degrees apart and tilted 45 degrees from horizontal for motions with a horizontal component.
5. Adjust the field of view to encompass expected motion, then lock the camera controls.
6. Set calibration points by hanging reflective spheres on wires or other support methods.
7. Determine the world location of calibration points which bracket the anticipated motions.
8. Place lights behind cameras to illuminate the calibration points.
9. Determine the location of points in image plane of each camera view.
10. Create an ASCII text file, adhering to the specified format, that contains calibration points 3-D world location and point identification numbers. This is done once for each set of calibration points.

11. Execute program "subcapt.exe" to locate calibration points with sub-pixel accuracy.
    a. File required: "balls.dat" calibration world location file.
    b. Files created:
        i. "cam.dat" point screen location file.
        ii. "world.dat" point world location file.
12. Adjust size of mask box by following instruction on screen.
13. Select gain adjust by holding down both mouse buttons simultaneously.
14. Move mouse to increase or decrease target contrast, then hit any mouse button to exit.
15. Move mask box over calibration point.
16. Activate centroiding by hitting right mouse button.
17. Enter the tag number of the point selected on the screen.
18. Screen location and world location of point are written to files.
19. Process more calibration points? If yes return to step 13 and repeat.
20. Process another camera view? If yes return to step 12 after renaming previous output files so they do not get erased. The calibration software requires that the second camera calibration points be processed in the same order with the same total number of points.
21. Make backup copies of required files and place originals in directory with program "calb.exe".
22. Execute "calb.exe" to perform calibration.
    a. Files required:
        i. "camparm.dat" fixed camera parameters
        ii. "world.prn" world calibration point locations
        iii. "cam1.prn" calibration point screen locations
        iv. "cam2.prn" calibration point screen locations
    b. Files created:
        i. "outcam1.prn" calibration results, camera 1
        ii. "outcam2.prn" calibration results, camera 2

23. Intermediate results will be displayed on screen. Errors will also be displayed. If the determinate at any step is not 1.0 or very close, a prompt will be given for a second calibration.
24. Follow instructions; a prompt for a file header will be given.
25. Were errors encountered? If yes, check world point location and number, screen location and number, and correlation of the same point between the world and screen, and return to step 23. If there is still a problem check the camera location and determine if the calibration points are distributed with in the field of view.
26. Execute "back.exe" to perform 3-D point location of calibration points, then calculate location of calibration points on the 2-D screen based on measured 3-D locations and calibration output.
    a. Files required:

i.	"camparm.dat"	fixed camera information
ii.	"outcam1.prn"	results from calibration
iii.	"outcam2.prn"	results from calibration
iv.	"cam1.prn"	calibration point screen location
v.	"cam2.prn"	calibration point screen location
vi.	"world.prn"	calibration point world location

    b. Files created:

i.	"calced.prn"	error summary and run description
ii.	"list.prn"	individual point location list
iii.	"fblocate.prn"	calculated screen locations
iv.	"leftiti.prn"	calculated screen locations
v.	"rightiti.prn"	decoupled calculation method

27. Execute "showcalb.exe", calibration point locations are back projected over original calibration point locations by projecting their computed locations. Files required:

i.	"cam1.prn"	calibration point screen location
ii.	"cam2.prn"	calibration point screen

iii. "fblocate.prn" calculated screen locations

iv. "leftiti.prn" calculated screen locations

v. "rightiti.prn" decoupled calculation method

28. Following instructions, are the calculated locations within a few pixels of the actual screen locations? If no, there is a problem with camera location, calibration point location, camera information, or screen location, or some other problem, start over by returning to step 4. If the calculated and actual points are close (10-20 pixels), the values for Cx, Cy, dx and dy can be modified slighly in the "camparm.dat" file. The numerical calibration should be performed again to get new coefficients.

29. If the calculated points are coincident or very near the original points, the calibration is complete.
30. Remove calibration points from view and set aside for calibration at end of test, then set target(s) to be tracked in field of view.
31. Arrange lights for best contrast between target and background.
32. Set video recorders and tape event with one time code as source for both, by splitting the time code from one timer and putting it on both VCR's.
33. Execute program "tsbackm.exe" for each tape.
    a. Files required: none
    b. Files created: user defined coordinate file ("rawpos.txt")
34. Select adjust gain from menu.
35. Follow on screen instructions to get best contrast.
36. Select "multi-track" menu item.
37. Enter number of targets to be tracked and follow on screen instructions.
38. Either set video tape to player "pause" and set the switch on the VCR to "remote" so that it is ready for processing, or start the live event.
39. Place box(s) over target(s), adjusting box size where necessary, and follow on screen instructions.
40. Monitor processing in case the box loses the target or the event ends.

41. Are there additional tapes to process? If yes, choose "file name" option from menu and change file name (to reflect which camera the tape is from) for next tape then go to step 34.
42. Edit output data files from "tsback.exe" so initial starting time is the same for each camera view.
43. Execute program "locate.exe"
    a. Files required:

i.	"camparm.dat"	error summary and run description
ii.	"outcam1.prn"	results from calibration
iii.	"outcam2.prn"	results from calibration
iv.	"cam1vid.prn"	individual frame by frame location
v.	"cam2vid.prn"	individual frame by frame location

    b. Files created: "location.prn" 3-D location of target
44. Output file location.prn has frame by frame 3-D location of target.

7.0 CONCLUSIONS

7.1 Summary

There is a specific need to sense position and orientation of discrete bodies used in experiments at the O.H Hinsdale Wave Research Laboratory, Oregon State University. After analysis of a range of problems, various possible solutions have been explored to satisfy the goal of three-dimensional, dynamic position sensing.

A video system has been selected, allowing access to a wide variety of computer based digital image processors. Recording the video on tape then permits viewing of an experiment at a future date to verify data or explain anomalies. Other possible solutions use proprietary image analysis hardware and do not allow for archiving the process being analyzed. This is a major limitation of non-video based systems. With video, the problem of time synchronization is also solved since an accepted standard of video timing is already in use (SMPTE).

After researching available imaging hardware, a compromise was made based on cost, speed, versatility and ease of use. The system was then acquired, including support hardware: image digitizer, video cameras, monitors and video cassette recorders. One important feature was a video recorder that could read and write SMPTE time code and be controlled by a host computer. A separate hardware box was acquired to perform the task.

After hardware acquisition was complete, software development was initiated. Simple routines were developed to control the video cassette recorder with the computer, and to perform image manipulations. After some effort, the ability to extract the location of a moving object in the video was developed. A user friendly interface was included in the development.

A single camera model was developed which combined many of the simple routines generated earlier. This mode allowed for dynamic object location, but the motion was limited to a plane that was parallel to the camera image plane. The model was tested and compared to traditional string potentiometer position sensing techniques. The video based method compared favorably with this method. The single camera model was simple and easy to use but had limitations to accuracy and could not be applied to situations with out of plane 3-D motions. For this reason, the multi-camera model was developed.

The multi-camera three dimensional model relies on the elementary routines developed earlier for 2-D video processing but the theoretical basis for the calibration and location algorithms were developed by Tsai (1986) and Lee, respectively. Several camera calibration and position location algorithms are reported in the literature, but the methods proposed by these authors have proven to be the most versatile and easiest to implement. The computer code for the calibration and the location algorithms was programmed as part of this study.

An experiment was devised to test the calibration and location codes. It consisted of a target that was constrained to a one dimensional motion trajectory. A traditional string potentiometer was installed to report the target location for verification of the video system. By constraining the target to one dimensional motion, the video could easily be compared with the string potentiometer record. The calibration was concluded to be successful when acceptable results were obtained from the calibration verification algorithm.

The results obtained from the three dimensional video algorithm were in close agreement with the string potentiometer measurements. The three dimensional algorithms provided for increased accuracy and more versatile motions when compared to the two dimensional algorithm. The two dimensional algorithm has specific uses, where simplicity and ease of use are paramount, but the three dimensional algorithm should be used when accuracy is required.

7.2 Recommendations

As with any data collection method, limitations exist with the video system. The most critical factor is lighting and target-background contrast. The two dimensional and three dimensional methods both require high contrast because the image processor only recognizes shades of gray and not shapes or textures.

Best contrast is achieved by using reflective targets with high power directed light and subdued secondary lighting. Reflections can be reduced by achieving a slight visually unfocused image. This also increases the apparent size of the target. If the target is moving rapidly, shuttered cameras must be used to freeze the motion.

The target itself must be large enough to be represented by several pixels on the screen. More pixels allow for more accurate sub-pixel centroid location. The screen representation of the target is a function of the camera position and zoom lens setting as well as the target physical size. It is difficult to develop an optimum relationship among all the variables, therefore use of the system involves art as well as science. However, the theoretical developments and algorithms are mathematically precise.

Given an accurate calibration, accurate target locations are possible. The calibration technique discussed in this report proved to be a very good method. It would work well on a smaller scale also, but an alternate method should be used for large scale experimental setups. Calculating the calibration parameters and locations is also strongly recommended before any data runs are performed. This could prevent unnecessary repetitions of the experiment.

7.3 Future Research

This research developed the initial three dimensional measurement and tracking system for use at the O.H. Hinsdale Wave Research Laboratory. Future study of calibration sensitivity and three dimensional sensitivity should be performed. Additional development and integration of software is also recommended, this would include a complete image processing and image tracking suite of software tools. The developed alogorithims for calibration, tracking and analysis could be combined in an easy to use enviroment, such as Microsoft Windows, or another graphical operation system. This work should be performed by someone trained in software design and development.

Sensitivity analysis of both the calibration and the three dimensional algorithms is needed. This should include various camera positions, number of cameras, number of calibration points, calibration point locations, camera lens settings, lighting variations, target sizing, and target spacing.

The calibration software should be improved by allowing for more that 32 calibration points, even though good results were obtained with 32. This limit was imposed by the memory limitations and individual programming techniques. The calibration, calibration verification and target location software should be incorporated into the windowed target tracking software environment.

The hardware may also be upgraded if increased resolution is desired. Systems with twice the resolution are becoming economically attractive. This would increase the accuracy and could be implemented easily. Hardware also exists that can perform real time tracking of multiple targets. The latter, combined with the target location software and two cameras, could provide for real time three dimensional dynamic target location.

One final area of research left unexplored is water wave run up on structures. This adds one more complexity. A non-ridged deformable body changes shape and color thus making tracking by traditional methods difficult. The position algorithms could still be used if the same point on the surface could be located in two camera views.

EPILOGUE

When near perfect results are obtained in any experiment, initial joy is squelched by a feeling of uneasiness. Were the equations optimized for the particularly camera locations? Would other operators obtain the same results? These are questions that are left unanswered.

It is hoped that the system developed will be put to practical use and will withstand the test of time, and become an integral part of the data acquisition package used by the staff at the O.H. Hinsdale Wave Research Laboratory and by others. LAJ

REFERENCES

Ayache, Nicholas and Lustman, Francis, "Trinocular Stereo Vision for Robotics", IEE Transactions on Pattern Analysis and Machine Intelligence, Vol 13, No. 1, January 1991, pp 73-85.

(EPST) Encyclopedia of Physical Science and Technology, Vol. 12, 1987, page 238.

Holman, Robert, Personal communications, 1990.

Imaging Technology Incorporated, ITI-151 Technical Reference Manual.

Katzev, David, "Simulation of Costal Processes in a Circular Wave Basin", Oregon State University, Masters Thesis, 1992.

Lee, Jen-Feng, S. Hirai, P. Liang, S Hackwood, "Robot Hand-Eye Coordination", Center for Robotic Systems in Microelectronics, UC Santa Barbara.

Lenz, Reimar K., "Techniques for Calibration of the Scale Factor and Image Center for High Accuracy 3D Machine Vision Metrology", IEEE International Conference On Robotics and Automation", VOL. 1, 1987, pp. 68-75.

Ljung, Lennard, System Identification: Theory for the User, Prentice-Hall, 1987, page 464.

Saraga, P. and Jones, B. M., "Simple Assembly Under Visual Control", Artificial Vision for Robots, 1977, pp 92-112.

Tsai, Roger Y., "A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses", IBM Research Report 1986.

Tsai, Roger Y., "A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses", IEEE Journal of Robotics and Automation, VOL. RA-3, No. 4, August 1987, pp. 323-344.

Tsai, Roger Y., "Review of RAC-Based Camera Calibration", Vision, SME's Quarterly on Vision Technology, VOL. 5, No. 3, November 1988A.

Tsai, Roger Y., "Overview of a Unified Calibration Trio for Robot Eye, Eye-to-Hand, and Hand Calibration using 3D Machine Vision", Proceedings of SPIE, International Society for Optical Engineering, VOL 1003, November 1988B, pp. 202-213.

Videomedia, V-lan Technical Reference Manual

Wolf, Paul R., Elements of Photogrammetry, McGraw-Hill, 1974.

APPENDICES

APPENDIX A

REAL-TIME TARGET REQUIREMENTS

NOTE: This section was written before several advances were made in software and tracking techniques. The ability to track an object frame by frame now exists. The VCR can be accurately controlled with single frame precision. Also the discovery of reflective paint reduces the problem with reflections. By creating a slightly out of focused image many reflections are also eliminated.

A.1 Overview

The selection of a trackable target is dependent on various interrelated factors. There are some basic guidelines that can be followed for target design. These guidelines have been established by performing an experiment using several target sizes. From this, an absolute minimum target size was found, then a realistic target size was selected. Even though only one camera view, light position and zoom factor were used, relations were developed that are independent of these factors.

The critical factors in target design are size, color and spacing of multiple targets. Other factors that indirectly influence target recognition are camera related, such as height, zoom setting and iris setting. A major problem is extraneous light sources and their reflections, that cannot always be controlled. Recognition is further complicated if the targets are on or in water due to the ability of water to reflect or transmit light depending on its surface profile. A major factor that can usually be controlled is lighting, but it is difficult to predict the best lighting configuration.

The selection of a camera position and zoom setting is related to the object motion sequence. If the object has no mean translational motion, the object can occupy more of the field of view. If the object exhibits a mean translation, it must occupy a small portion of the overall field which decreases the ability to resolve the target small scale motion.

A.2 Target Size Experiment

The experiment was conducted in the small circular wave basin at the O.H. Hinsdale Wave Research Laboratory at Oregon State University. Ten targets consisting of solid black circles on a white background were used. The circles were sequentially placed on a platform attached to the wave maker. The motion was started and the object was tracked at the real-time maximum rate of 3-4 Hz. The number of pixels representing the target was averaged over 50 non-zero samples. A time series of the centroid location and actual target size were recorded for each target. None of the camera parameters were changed and the lighting conditions remained the same throughout the experiment.

This experiment provided some guidelines in target design based on required field of views. The only variable was the target size while all other parameters were held constant. Using basic geometry and known camera parameters, the results were extrapolated to other possible configurations. Alternate lighting configurations were not tested, since it is difficult to formulate a simple theory related to light intensity in target recognition. More light does not always provide better results, since contrast between target and background is what the system detects, not overall brightness.

The camera was positioned 28.5 feet (8.7 m) above the wavemaker with the horizontal field of view parallel to the East-West building wall, Figure A.1. The camera lens was set at full wide angle. The horizontal field of view was 20 feet, 5 inches (6.22 m), the vertical field of view was 15 feet, 5 inches (4.69m).

Object motion was not necessary to establish minimum target size, but was included to better model an actual application. In these experiments the target motion was small relative to the field of view. The target was attached to the wave maker and a known wave maker trajectory program was initiated. The actual path of the wave maker is shown in Figure A.2.

Ten circular solid black targets were tracked under the same conditions. The sampling efficiency is the percent ratio of 50 sample images to the number of images required to get 50 acceptable images. If 100 samples are taken and the centroid is found in only 50 images, then the efficiency is only 50%. The total number of pixels found reflects the camera iris setting, and the image processors gain setting. The maximum pixel count is 10,000, highest accuracy and fastest sampling rate occur with total pixels found under 1100.

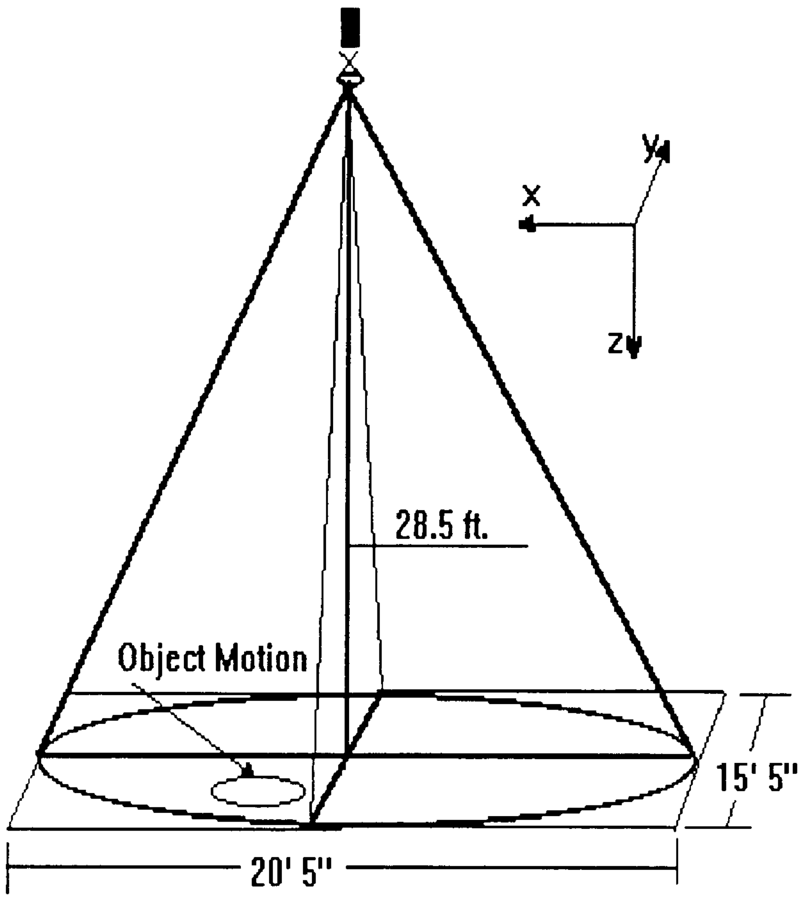


Figure A.1 Camera position and field of view

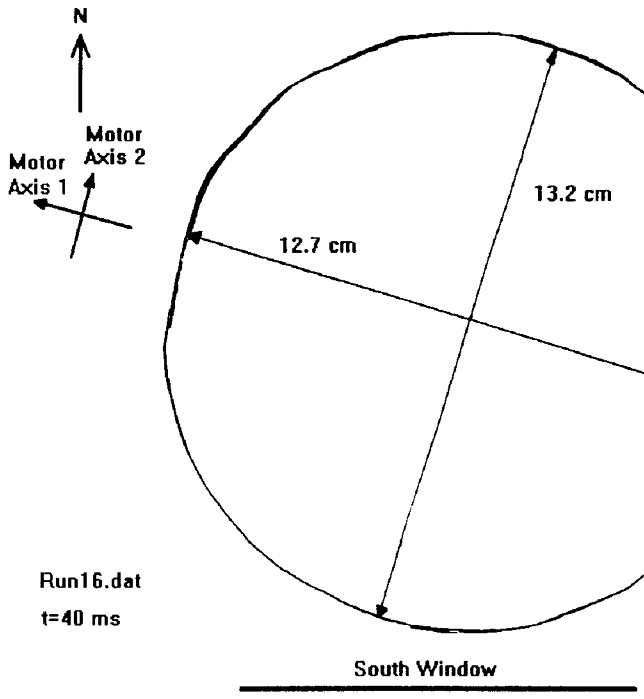


Figure A.2 Actual path of circular wavemaker.

The exact same experimental setup was repeated one day later. All lighting was in the same position, and outdoor lighting was also similar. The only variable was the software video input gain. The two runs are shown in Figure A.3. Run 2 reflects a lower input gain, and a smaller detectable target size. The data from Run 2 was used for all subsequent target size predictions since it resulted in a larger more realistic target size. The results from Run 2 are shown in Table A.1

A threshold sampling efficiency was set at 100 percent since missing data points can cause difficulty in secondary processing of the results. When the target size is selected 100 percent sampling efficiency is required.

Figure A.4 shows the target size in terms of area of the target. Target area rather than target diameter was shown since the target is represented on the video screen by a collection of pixels, which also have a proportional area. By simply requiring a fixed number of pixels for target design, the targets may be non-circular and still have the required surface area visible to the camera. The software would give the coordinates of the object video centroid, which would not necessarily be the target centroid.

The data was collected using the single pass algorithm which samples approximately 3 to 4 times per second, and the motion of the target had a 2 second period. The centroid location, time, centroid pixel count and total pixels found from Run 2 are quantified in Table A.2.

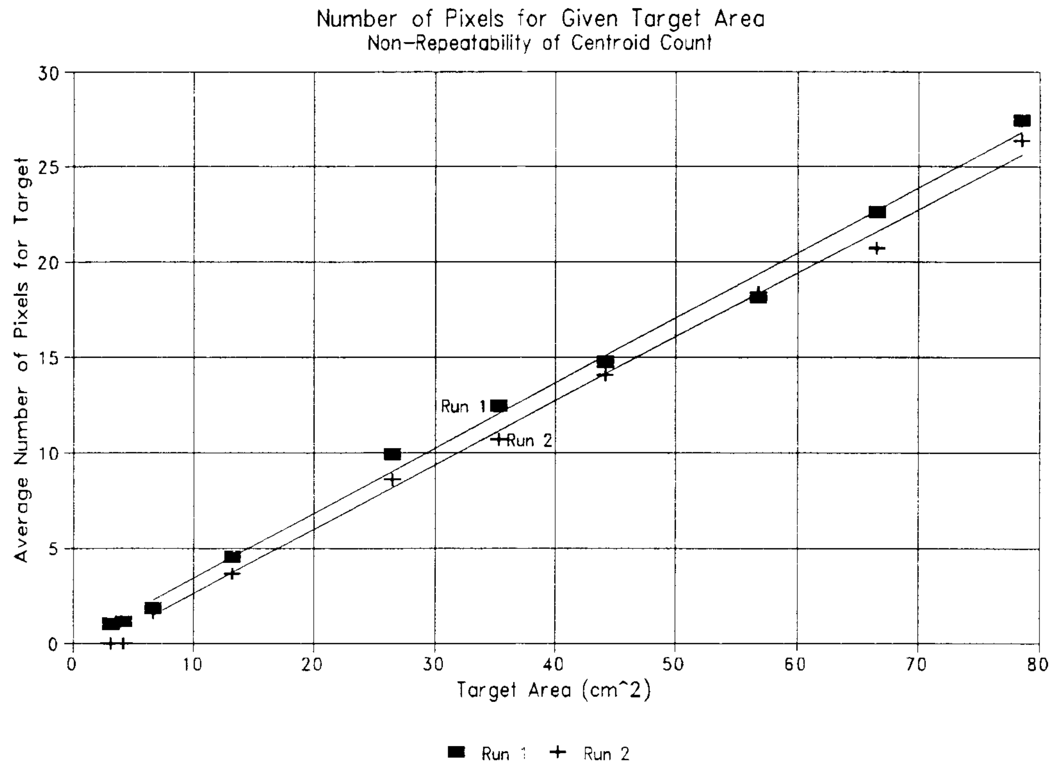


Figure A.3 Number of pixels representing a given target area

TABLE A.1 Target size effect on efficiency

Object Number	Centroid Pixel Average	Size (cm)	Efficiency (%)	Total Found (avg)
1	26.38	10	100	1100
2	20.78	9.2	100	1069
3	18.38	8.5	100	1124
4	14.05	7.5	100	1072
5	10.48	6.7	100	1069
6	8.62	5.8	100	1122
7	3.68	4.1	100	1055
8	1.6	2.9	94	1066
9	0	2.3	0	0
10	0	2.0	0	0

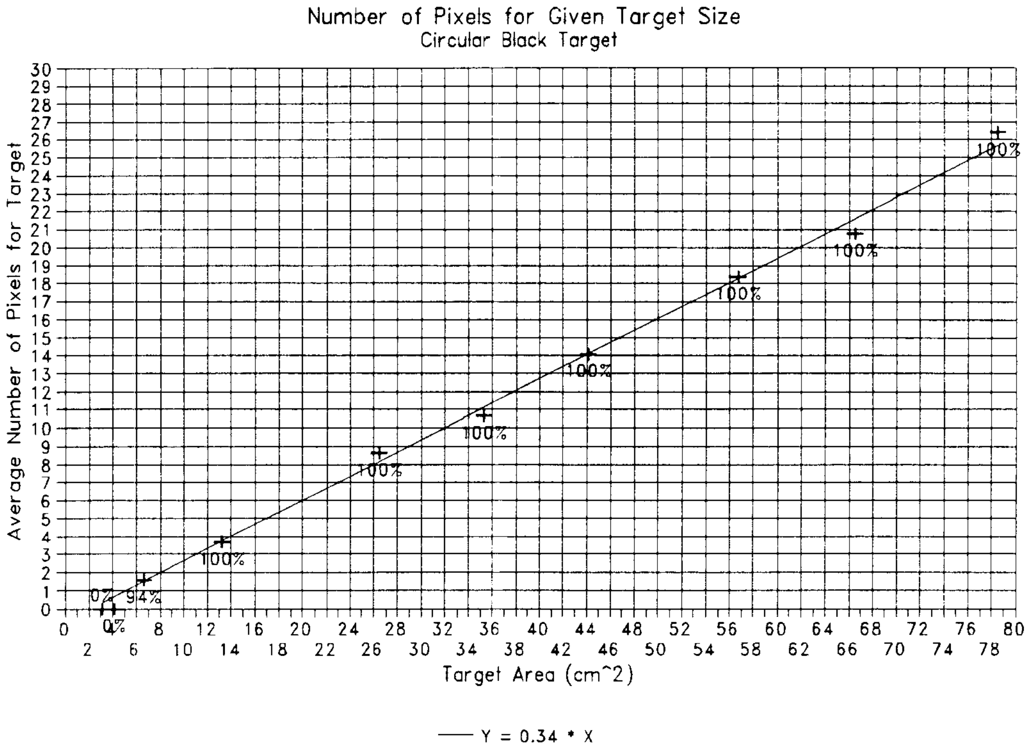


Figure A.4 Number of pixels found for given target size, and location efficiency.

TABLE A.2 Centroid, time and pixel count, run 2

X	Y	SMPTE Time	Centroid Pixels	Total Pixels
307	81	00:07:02:14	8	1062
309	74	00:07:02:26	7	1080
315	75	00:07:03:08	11	1100
317	81	00:07:03:18	9	1041
313	86	00:07:03:29	13	1095
308	84	00:07:04:11	12	1045
307	77	00:07:04:21	9	1087
311	73	00:07:05:02	12	1093
316	77	00:07:05:14	12	1067
316	83	00:07:05:25	11	1067
311	86	00:07:06:06	10	1051
307	82	00:07:06:17	12	1055
308	76	00:07:06:28	10	1069
313	74	00:07:07:08	12	1082
317	79	00:07:07:19	9	1077
315	85	00:07:08:00	9	1060
309	85	00:07:08:11	12	1085
307	80	00:07:08:23	9	1053
309	75	00:07:09:03	8	1114
315	75	00:07:09:14	9	1044
317	81	00:07:09:26	9	1085
313	86	00:07:10:06	10	1057
307	84	00:07:10:17	12	1073
307	77	00:07:10:29	9	1070
311	73	00:07:11:09	12	1061
316	77	00:07:11:20	12	1058
316	83	00:07:12:02	11	1073
311	87	00:07:12:12	12	1044
307	82	00:07:12:23	10	1081
308	75	00:07:13:05	14	1052

Shown in Table A.2 is a sample of the data from the 6.7 cm target, operating at 100% efficiency, the average centroid pixel count is 10.68, even though the centroid pixel count is quite variable. In the complete record the minimum value is 6, the maximum is 14, the average is 10.68 and the standard deviation is 1.86. The other targets sizes have similar statistical results as shown in Table A.3.

Processing the same sequence multiple times to fill in missing points would not have added significantly more information to this experiment. The minimum target size could have been calculated from one still frame, but a sequence of tape, 10 seconds, was analyzed to show the variability of the centroid pixel count.

This variability has two causes. The first is minor lighting and contrast changes by water surface reflections. The second source of variability is the hardware interpolation of pixel locations. The round target is represented by square pixels. As the target moves, there is a point where each pixel sees only a portion of the target. The hardware makes the choice to either report that it has seen the target or report only background. This is hardware controlled and not user controlled.

Edge effects are magnified when the target is moving. As the target gets smaller, its perimeter gets smaller. Less pixels are likely to be at the target edges, and therefore, the individual pixel counts should be closer to the

TABLE A.3 Target size statistical results

Object Number	Size (cm)	Minimum Pixels	Maximum Pixels	Average	Standard Deviation
1	10	17	32	26.38	3.36
2	9.2	7	26	20.78	4.25
3	8.5	7	22	18.38	3.38
4	7.5	5	18	14.04	2.63
5	6.7	6	14	10.68	1.86
6	5.8	4	12	8.62	1.83
7	4.1	2	5	3.68	0.88
8	2.9	0	3	1.51	0.72
9	2.3	0	0	0	0
10	2.0	0	0	0	0

average. This is verified in Table A.3 which shows the standard deviation decreasing with target size.

A.3 Target Size Prediction

From this experiment a simple method of meeting the requirements of the image processing system has been developed. It was based on the relationship between the camera field of view to the minimum recognizable target size. Only one piece of information about the event to be measured is required: the amount of net translational motion over a sampling period in the experiment.

If there is little translation, the field of view can be small and result in better motion resolution. If large translation is are experienced, the field of view must be larger, which decreases small scale resolution. The field of view is a function of the distance of the camera to the target and of the cameras zoom lens setting. The same field of view is possible from an infinite combination of camera distances and lens settings.

The first step to develop an acceptable target size is to select a minimum sampling efficiency. Since missing data poses a problem later when the data is processed to extract velocities and accelerations, 100 percent efficiency was selected. As can be seen in Figure A.5, for 100 percent efficiency a target size of at least 4 pixels are required.

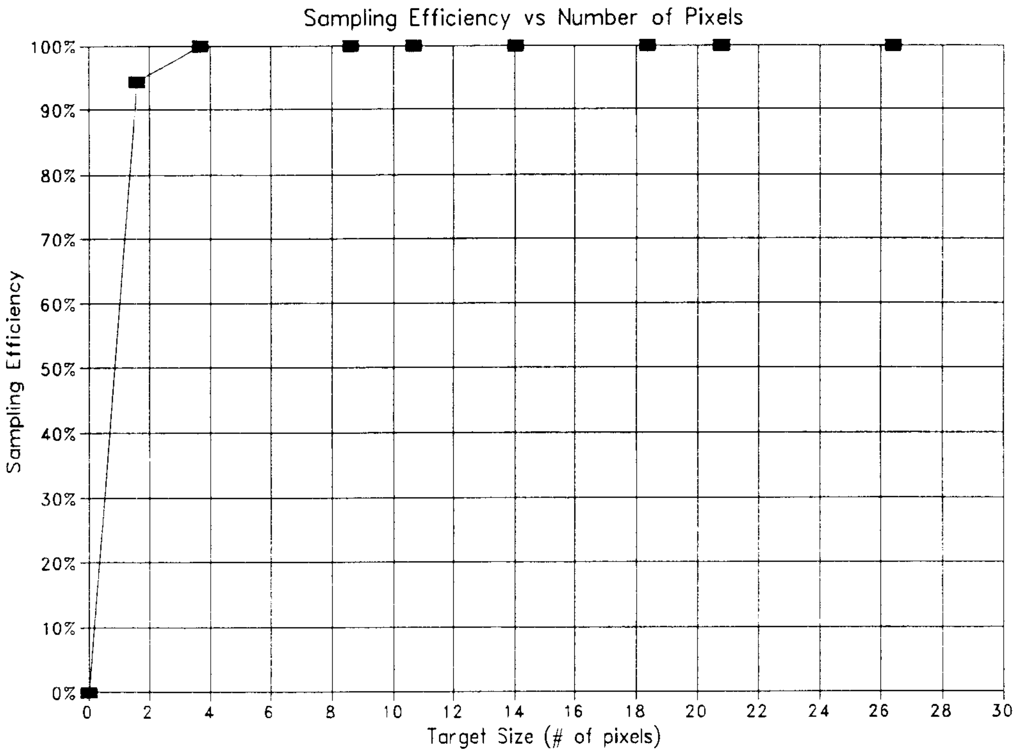


Figure A.5 Sampling efficiency as a function of target size

A more useful way of expressing the same data is in the non-dimensional form of target size divided by the horizontal field of view. This is shown in Figure A.6 and Figure A.7.

A problem introduced by the NTSC RS-170 video standard is the horizontal field of view is always larger than the vertical field of view. The limiting condition is the vertical field of view, seen in Figure A.7. To attain the same efficiency the target needs to be 30 percent larger than required by the horizontal field of view (Figure A.6). This can also be explained by the fact that the horizontal field is approximately 30 percent larger than the vertical field of view.

To attain 100 percent efficiency the target must be at least 0.9 percent of the vertical field of view. This will also result in a target that is represented by at least four pixels. The predicted size as shown in Figure A.8, is larger than the experimental data suggests. A target with a 4.1 cm diameter had 100 percent efficiency and a average pixel count of 3.68.

After a field of view and target size are selected, the camera height must be determined. The horizontal field is larger and remains approximately 30 percent larger throughout the full zoom range of the lens. The vertical and horizontal field of view for a full range of camera heights and lens zoom settings is show in Figures A.9 and A.10. An experimental configuration is feasible if the camera height and the field width desired plot between the zoom and wide angle lines in Figure A.9 and A.10. These figures will not

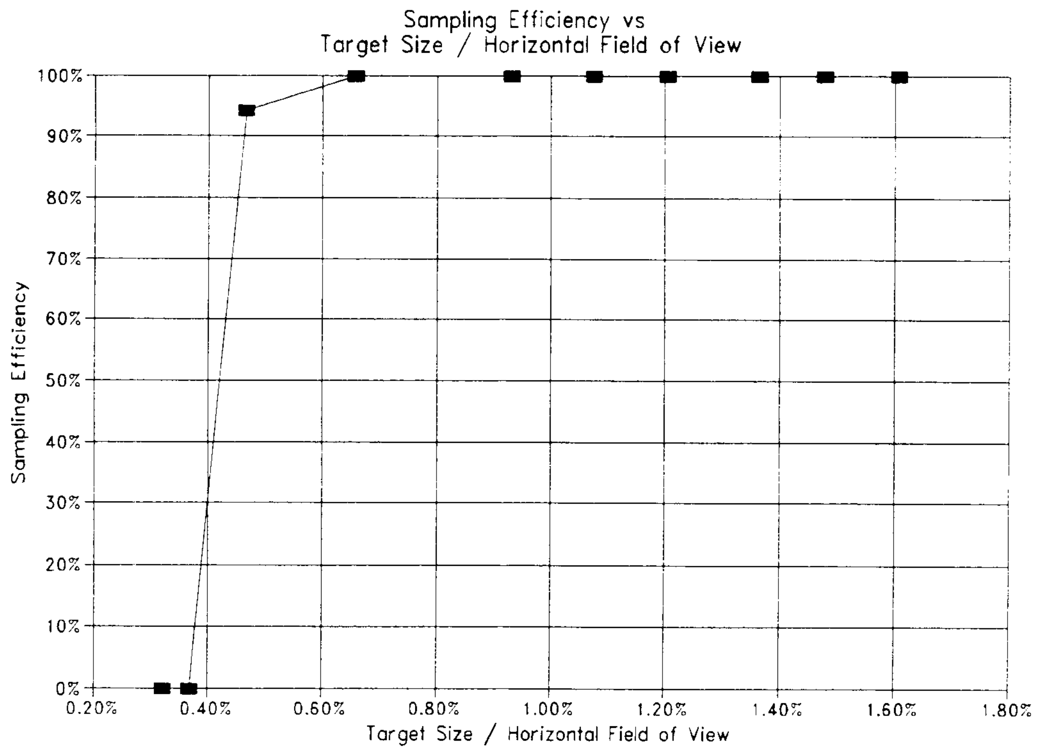


Figure A.6 Sampling efficiency as a function of non-dimensional target size/horizontal field of view

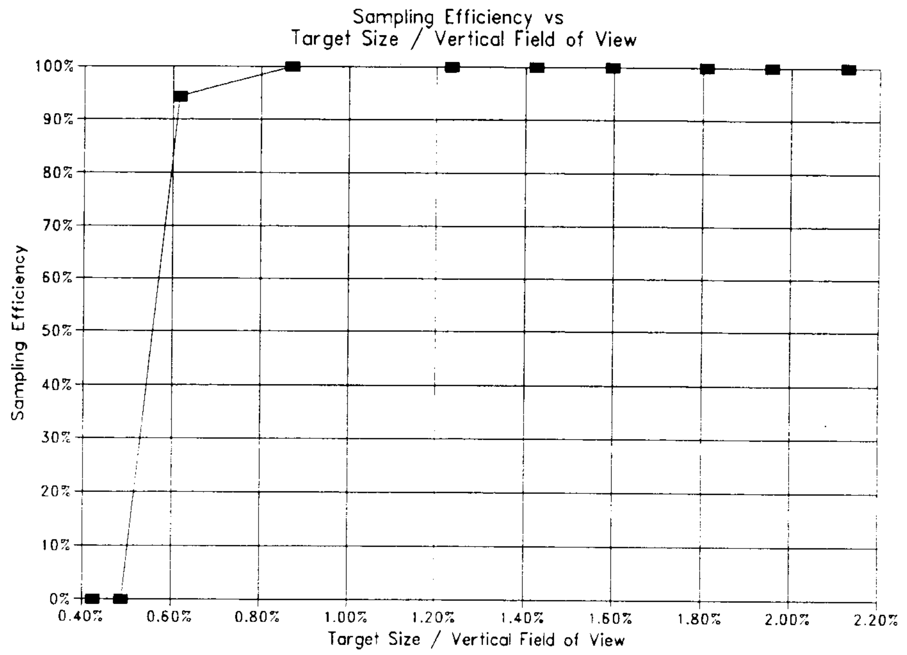


Figure A.7 Sampling efficiency as a function of non-dimensional target size/vertical field of view

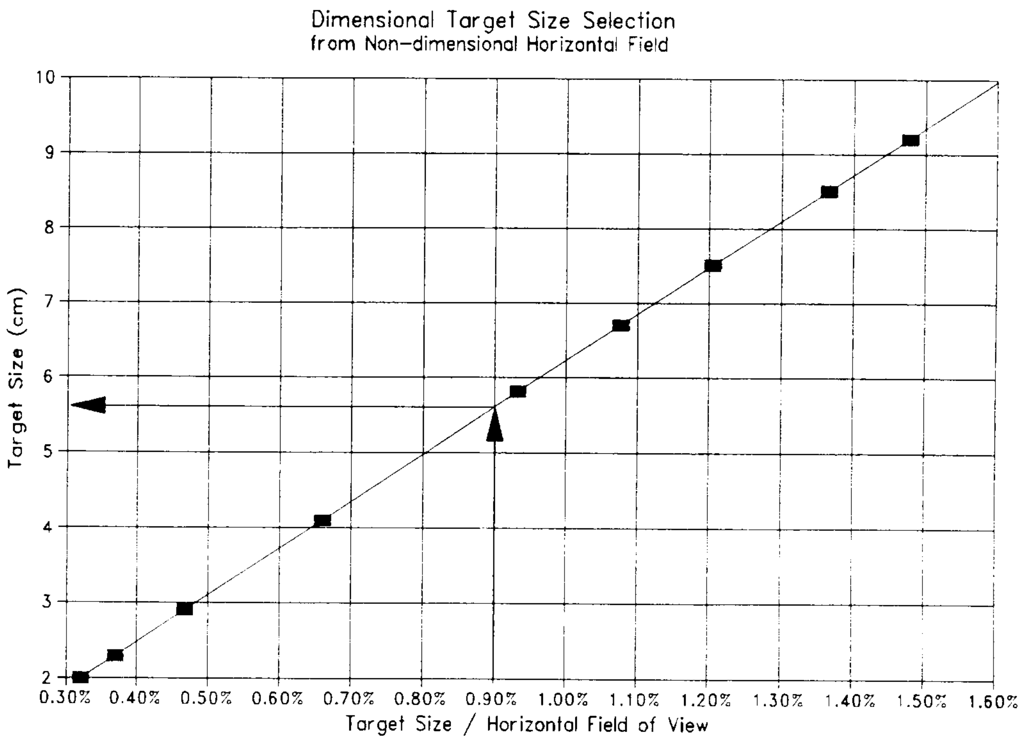


Figure A.8 Dimensional target size selection using non-dimensional horizontal field of view

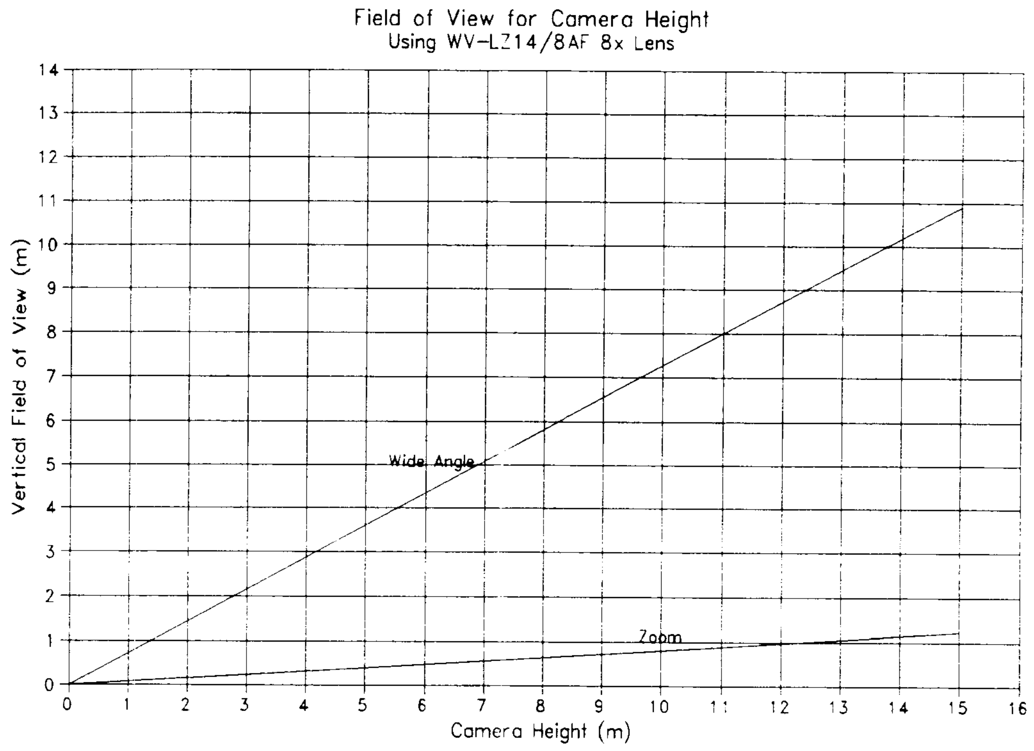


Figure A.9 Vertical field of view for WV-LZ14/8AF lens

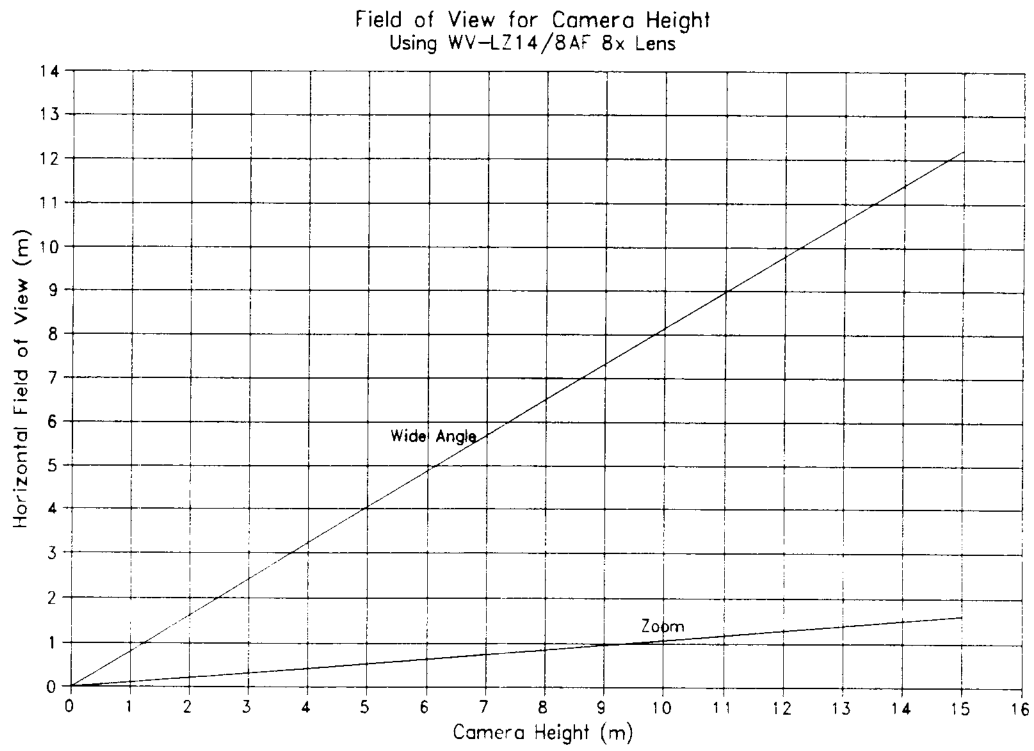


Figure A.10 Horizontal field of view for WV-LZ14/8AF lens

predict the exact field however, because several uncontrollable factors influence the field.

It is impossible to measure the exact field with the existing monitors, since they use an "over-scanning" picture display process. Since the video picture is larger than the physical screen, a small margin of the picture is not shown. This prevents black margins on the sides, top and bottom of the screen, but when the field is measured, only the visible field is measured. All data reported is viewable field. All predictions are theoretical fields. Since the image processor scans the exact video field, this causes the actual number of pixels seen on the screen to be less than 512 horizontal and 480 vertical, with the corners having the least total visible pixels.

A.4 Limitations and Sources of Errors

The limitations of the system and these predictions fall into three groups; 1) video hardware, 2) image processor, 3) other factors. The video hardware poses some significant limitations that must be understood before any experiment is designed to use the tracking system. The image processor also introduces several problems that must be understood. The other factors are user controllable and are not as significant.

The video hardware that was selected conforms to the industry standard NTSC RS-170 video format. It operates at 30 video frames per second. One frame is composed of two fields that are interlaced together, a field is either every odd numbered scan line or every even numbered scan line. By selecting RS-170 video, cost was minimized, functions were maximized and compatibility was preserved. Unfortunately RS-170 was not designed with image processing in mind.

The most significant limitation is the sampling rate of 30 Hz. This cannot be increased or decreased. Only specialized non-standard cameras have controllable sampling rates. By processing the same sequence several times and offsetting the sampling point, a composite file can be constructed which represents "real-time" 30 Hz sampling. The problem of measuring camera field of view is not resolvable unless a non-standard over scanning monitor is used. The most significant problem is the aspect ratio (horizontal lines of resolution / vertical lines of resolution) of the RS-170 video. If the target only travels in a circular path, then there will be sections of the horizontal field that will not be used, this reduces the effective resolution. The RS-170 video aspect ratio problem also causes problems with the image processor.

The image processor has a larger horizontal field than vertical field. This is to make it compatible with the RS-170 video. This results in pixels that are not square and the horizontal pixel count is not the same as a vertical pixel count. This causes calibration problems and measurement problems which are usually transparent to the end user.

Even though there are more horizontal pixels than vertical pixels, the horizontal field is larger. The result is that the resolving power in the vertical direction is 25 percent better than in the horizontal direction.

The image processor has a variable gain on the video input signal which allows the picture to be changed from all black to all white. It is analogous to a camera iris. If the gain is increased the picture becomes "overexposed" and too many pixels are found, the target gets lost in the back ground noise. If the gain is too low the target is not found at all. Adjusting the gain is one of the more critical steps in using the system since it affects the target recognition and centroiding accuracy.

If the target is underexposed only a portion of the outline of the target is seen and the true centroid is not reported. If the target is overexposed, more than the target is seen, and once again the true centroid is not located. Even if the gain is optimized and the target is stationary, the centroid is only accurate to within 2 pixels. This results in 1 part in 256 horizontal, and 1 part in 240 vertical resolution. If multiple passes are made on the same frames, the centroid can be located better in a statistical sense and this improves the resolution. The exposure is a function of the image processor gain, the camera iris setting and the lighting configuration.

Appropriate lighting of the test area improves the function of the video equipment and the image processor. Lighting problems have simple solutions, such as moving lights, using more or less lights, or changing the target color or background color. The results of these actions are difficult to predict and trial and error methods are required to develop an acceptable testing setup. The lighting creates the target to background contrast. The higher the contrast the more efficiently and accurately the system operates in discerning targets.

A.5 Recommendations

When an experiment is designed to use the image processing and tracking system several things must be considered: 1) The inherent problems introduced by the video hardware, aspect ratio and sampling rate; 2) The problems involved with the image processor itself, different aspect ratio, centroiding accuracy, and gain selection; and 3) maximizing target contrast.

Once these problems are understood a target size can be selected using the previous figures. These figures are, however, no substitute for actual testing of actual targets. They give approximate values for size, not exact values. The minimum value indicated for the target size is not necessarily the size that will give the best tracking results.

When a target is selected there is another step which requires user judgment, the gain adjustment. When the gain is increased more total pixels are found, but not necessarily more target pixels. Target pixel count must be maximized while minimizing total pixel count. This only works to a point where the contrast is the limiting factor. Physical lighting configurations must then be changed to maximize target to background contrast. This step is critical before any permanent record is made on video tape, since it cannot be changed later.

The last consideration in target design that has not been addressed is the choice of black or white. A black target on a white background was used since reflections from uncontrolled light sources blend with the white back ground and thus does not pose a problem.

If white targets are used on a black background light reflections off the water surface are confused with the target. All reflections cannot be removed since some light is required for the camera to operate and give an acceptable picture with good contrast. The other alternative is to make the targets much brighter than the background by using reflective paint or small lights for the target. The intensity of the light is what must be maximized for best results since the size is not the critical factor with this method.

When multiple targets are tracked they must be spaced an acceptable distance to allow proper operation. If the targets are too close, the software has difficulty distinguishing between targets. This is an experiment design parameter that must be tested with the actual setup to find the minimum acceptable distance. It is possible to modify the software to look for a specific pattern which can theoretically reduce the target spacing. It is also possible to extrapolate target motion to anticipate the location.

The rate at which the system samples was not varied in the test, but was set to the maximum real-time rate of 3-4 Hz. If full 30 Hz sampling were required the sequence would have been processed using a multiple pass algorithm. This processes the tape once with user supervision, then the tape automatically rewinds to the same starting point, the sample point is offset by one frame and processing is done again. This is done 10 times and a composite of the tracked objects motion is recorded in a file. This gives pseudo-real-time data in ten times the actual sample length. A one minute run would take about 12 minutes to get 30 Hz resolution. If only 3 Hz resolution is required this can be attained in one minute, or real-time.

APPENDIX B

SPECIFICATIONS

COMPONENT SPECIFICATIONS:

Image Processor resolution	512 Horizontal, 480 Vertical
Camera Speed	1/30--1/2000 sec
Camera Lens	Variable zoom (8X)
Video, SVHS	RS-170, 480 lines resolution
Computer	486-25, 200 Mg Hard Drive

TRACKING SYSTEM SPECIFICATIONS

Maximum real-time sampling	5-6 Hz
Maximum post processing sampling	30 Hz
Maximum trackable points	Unlimited
Field of View	Depends on camera distance to object plane and zoom factor
Minimum target size	0.2% of field of view (1 pixel)
Minimum recommended target size (target size is proportional to field of view)	1.0% of field of view (5 pixel group)
Accuracy of target location on screen	(+/-) 0.25 - 0.5 pixel
Tracking method	Intensity recognition, box masking

STEREO VISION SPECIFICATION

Theoretical accuracy limit (with +/- 0.1 pixel location)	1/4000 (R.Y.Tsai, 1986)
Minimum Number Cameras Required	2
Minimum know calibration points	7 (theoretical)
Suggested number of calibration points	31
Current accuracy of 3D point placements (average)	1/1350 0.07% of diagonal field

APPENDIX C

SOURCE CODE

```
                    Calb.c    10-20-92      Version 3.0

        10-16  Performs camera calibration
        30 is the maximum cal points due to the NP 60 value and memory
        this can be improved
        10-25 Does both cameras in one pass
        10-30 changed 1/S.x factor in both sections
        11-19-91 Tsai calibration is now a subroutine,
                it is called twice (camera 1, then camera 2
        10-20-92 gave files headers and changed file struture

                    By Andrew Jansky
```

```
/**** NOTE: THE UNITS MUST BE THE SAME ON SENSOR SIZE AND MEASUREMENTS ***/
#define VERSION    "Version 3.0  10-20-92\n"
#define CAMPARM    "camparm.dat"  /* camera fixed parameter file */
#define CAMERA_ONE "cam1.prn"     /* target location file      */
#define CAMERA_TWO "cam2.prn"     /* target location file      */
#define OUT_CAM1   "outcam1.prn"  /* calibration output results  */
#define OUT_CAM2   "outcam2.prn"  /* calibration output results  */

#include <time.h>
#include <string.h>


#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include "nr.h"
#include "nrutil.h"

#define DEBUG 0    /*  1 will give intermediate output  0 will not      */
#define NP 70      /*  70 Number of rows and colums in least square matrix */
#define MAX_PTS 33 /**25 this number is a factor of memory and least_square*/
#define WP 3
#define MAX_ARRAY 32
#define MAXSTR 80


void calb_tsai(void);
void least_square(double **, double **, int , int , int );
void gaussjf(double **, int, double **, int);

struct camera {   /******* Camera parameter structure      **************/
        double r1;  double r2;  double r3;
        double r4;  double r5;  double r6;     /** Rotation matrix         */
        double r7;  double r8;  double r9;

        double tx;  double ty;  double tz;     /** Translation matrix      */

        double Sx;                             /** Scan error              */
        double f;                              /** Effective focal length  */
        double k1;                             /** Radial lens distortion  */
        double Ncx;     double Ncy;            /** Image sensor lines      */
        double Cx;      double Cy;         /* Where camera center is on IP plane*/
        double sensorx; double sensory;        /** Sensor size (same units) */
        double field;  /*** diagonal field size  ***/
```

```
            };

struct  frame_buffer{                        /** Frame buffer coordinates **/
            double x;
            double y;
            int    total; /* total pixels */
            };
double IPNfx; IPNfy;        /**** Image processor scan resoulution        **/

struct world_coordinate{                     /** World point coordinates  **/
            double x;
            double y;
            double z;
            int    t;   /* point tag    */
            };

struct world_coordinate world[MAX_ARRAY];

struct camera cam;          /**** camera 1 parameters fixed each setup   **/


struct frame_buffer fb[MAX_ARRAY]; /**** point coordinate for cam1         **/
FILE *fp;
FILE *Cam;


int i;    /** Counter for number of ground control points           **/
double dx1,dy1,xu1,yu1,x,y; /**intermediate values**/
double **a,**b;
float **det,d;
int *indx;
int j,k,l;


main()
{
int w=7;   /** matrix cols        , same for all 2 camera solutions **/
int n=4;
int m=1;   /** number of solutions, same for all 2 camera solutions **/
char *axis[]={" ","X=","Y=","Z="};
char dummy [MAXSTR];
char tmpbuf[128];
a=dmatrix(1,NP,1,NP);
b=dmatrix(1,WP,1,NP);
det=matrix(1,3,1,3);


/***************** Read Fixed values for both cameras    ******************/
if  ((fp = fopen (CAMPARM,"r")) == NULL)
            nrerror("Camera fixed parameter file not found (camparm.dat)\n");

            fgets(dummy,MAXSTR,fp);
            fgets(dummy,MAXSTR,fp);
            fscanf(fp,"%lf %lf  %lf   %lf   %lf   %lf   %lf ",&cam.Ncx,&cam.Ncy,
                                                    &IPNfx,&cam.Cx,&cam.Cy,&cam.sensorx,&cam.sensory);
            fgets(dummy,MAXSTR,fp);
            fscanf(fp,"%lf   %lf ",&cam.field,&cam.field);

            fclose(fp);

if(DEBUG==1){
printf("Left\n");
printf("Ncx=%lf  Ncy=%lf  Nfx=%lf   Cx=%lf   Cy=%lf  SensX=%lf  SensY=%lf\n",
                                        cam.Ncx,cam.Ncy,IPNfx,cam.Cx,cam.Cy,cam.sensorx,cam.sensory);

}
        dx1=(cam.sensorx/cam.Ncx)*(cam.Ncx/IPNfx);/** calc intermediate values*/
        dy1=(cam.sensory/cam.Ncy);

/************* Open image processor coordinate files **********************/

if  ((Cam = fopen (CAMERA_ONE,"r")) == NULL)
        nrerror("Camera information file not found (cam1.prn)\n");
        for(i=0;i<12;i++)
        fgets(dummy,MAXSTR,Cam);                        /* strip header  */

if  ((fp = fopen (OUT_CAM1,"w")) == NULL)
        nrerror("Data file outcam1.prn not created\n");
        fprintf(fp,"%s \n",OUT_CAM1);
        _strtime ( tmpbuf );
        fprintf(fp,"%s \n",tmpbuf);
        _strdate ( tmpbuf );
        fprintf(fp,"%s \n",tmpbuf);
        fprintf(fp,"This is an intermedate calibration file for use with\n");
        fprintf(fp,"back.exe, and locate.exe\n");
        fprintf(fp," r1                  r2              r3\n");
        fprintf(fp," r4                  r5              r6\n");
        fprintf(fp," r7                  r8              r9\n\n");
        fprintf(fp," Tx                   Ty              Tz\n\n");
        fprintf(fp," Sx                  f               K1\n\n");
        fprintf(fp," Max dist between        PointA            PointB\n");

fprintf(fp,"=====================================================
=====\n");

calb_tsai();   /**** perform tsai calibration    *******/

fclose(Cam);
fclose(fp);

/*********************** Second Pass Here   ********************/
/*********could exit here for one camera calibration       ******/

if  ((Cam = fopen (CAMERA_TWO,"r")) == NULL)
        nrerror("Camera information file not found (cam2.prn)\n");
        for(i=0;i<12;i++)
        fgets(dummy,MAXSTR,Cam);                        /* strip header  */


if  ((fp = fopen (OUT_CAM2,"w")) == NULL)
        nrerror("Data file outcam2.prn not created\n");
        fprintf(fp,"%s \n",OUT_CAM2);
        _strtime ( tmpbuf );
        fprintf(fp,"%s \n",tmpbuf);
        _strdate ( tmpbuf );
        fprintf(fp,"%s \n",tmpbuf);
        fprintf(fp,"This is an intermedate calibration file for use with\n");
        fprintf(fp,"back.exe, and locate.exe\n");
        fprintf(fp," r1                  r2              r3\n");
        fprintf(fp," r4                  r5              r6\n");
        fprintf(fp," r7                  r8              r9\n\n");
        fprintf(fp," Tx                   Ty              Tz\n\n");
        fprintf(fp," Sx                  f               K1\n\n");
        fprintf(fp," Max dist between        PointA            PointB\n");
fprintf(fp,"=========================================================
=====\n");

calb_tsai();    /**** perform tsai calibration    *******/

fclose(Cam);
fclose(fp);

free_matrix(det,1,3,1,3);
free_dmatrix(b,1,WP,1,NP);
free_dmatrix(a,1,NP,1,NP);
printf("\n\n\nCalibration performed by Calb.exe, ");
printf(VERSION);
}
/****** END MAIN *******/

/*************** calb_tsai() ****** Tsai calibration *********************/
void calb_tsai(void)
{
int qq,rr;
int track_dpt[2]={0,0};
double dx,dy,dz,dpt,dpt_old;

i = 1; /*** Zero offset for control point data          ***/
cam.Sx = 1.0; /** Assumption in Tsai for multi-plane case     ***/

/******************* Calibration begins here           ******************/
while ((!feof(Cam))&&(i<MAX_PTS)) {
  if(i>=(MAX_PTS-1)) break;

  fscanf(Cam,"%i ",&world[i].t);
  fscanf(Cam,"%lf ",&fb[i].x);
  fscanf(Cam,"%lf ",&fb[i].y);
  fscanf(Cam,"%i ",&fb[i].total);
  fscanf(Cam,"%lf ",&world[i].x);
  fscanf(Cam,"%lf ",&world[i].y);
  fscanf(Cam,"%lf ",&world[i].z);

  if(DEBUG==1)
          printf("%d\t%8.2lf   %8.2lf   %8.2lf    %8.2lf    %8.2lf\n"
          ,i,world[i].x,world[i].y,world[i].z,fb[i].x,fb[i].y);
  /** Convert frame buffer coordinates to Image plane coordinates  ***/
  fb[i].x = dx1 * cam.Sx * ((double)fb[i].x-cam.Cx);
  fb[i].y = dy1 * ((double)fb[i].y-cam.Cy);

  /** build A matrix     ***/
  a[i][1] = fb[i].y * world[i].x;
  a[i][2] = fb[i].y * world[i].y;
  a[i][3] = fb[i].y * world[i].z;

  a[i][4] = fb[i].y;

  a[i][5] = -fb[i].x * world[i].x;
  a[i][6] = -fb[i].x * world[i].y;
  a[i][7] = -fb[i].x * world[i].z;

  b[1][i] = fb[i].x;
  i++;
}

  i--; /*** slight glitch counter is one too high at this point  **/
```

```
printf("\n");
printf("Total points used in camera calibration = %d\n",i+1);

if(DEBUG==1) printf("Input b matrix\n");
for (l=1;l<=i;l++){ if(DEBUG==1)printf("number=%i   %3.5lf\n",l,b[1][l]);}

least_square(a,b,i,7,1); /**** could be (a,b,n,w,m)   ****/

if(DEBUG==1) printf("Answer matrix  A\n");
for (k=1;k<=7;k++) {
          for (l=1;l<=1;l++){ if(DEBUG==1)printf("%3.5lf\n",a[k][l]);}
          }


cam.ty = fabs(1/sqrt(a[5][1]*a[5][1]+a[6][1]*a[6][1]+a[7][1]*a[7][1]));

/***** Selection of cam.ty sign       *******/
cam.r1=a[1][1]*cam.ty;
cam.r2=a[2][1]*cam.ty;
cam.r3=a[3][1]*cam.ty;
cam.tx=a[4][1]*cam.ty;
cam.r4=a[5][1]*cam.ty;
cam.r5=a[6][1]*cam.ty;
cam.r6=a[7][1]*cam.ty;

/***** Using the first point to check the sign, another possible bug***/
x = cam.r1*world[1].x + cam.r2*world[1].y + cam.r3*world[1].z + cam.tx;
y = cam.r4*world[1].x + cam.r5*world[1].y + cam.r6*world[1].z + cam.ty;
/*****  y equation differs from Tsai, he says y->cam.tx not cam.ty  ***/

{
if(DEBUG==1){
                    printf(" The sign Factor\n");
                          printf("x/fb[1].x = %f   y/fb[1].y=%f\n",x/fb[1].x ,y/fb[1].y);
                          printf(" %f    %f\n",y/fb[1].x ,x/fb[1].y);
                          }

if((x/fb[1].x > 0.0)&&(y/fb[1].y>0.0)){
          cam.ty=cam.ty;
          if(DEBUG==1) printf("The Siqn is the same on Ty\n");
          }

else {cam.ty = -cam.ty;
                    if(DEBUG==1) printf("The Siqn is the opposite on Ty\n");
                    }
}
KLUDGE1:

cam.Sx=sqrt(a[1][1]*a[1][1]+a[2][1]*a[2][1]+a[3][1]*a[3][1])*fabs(cam.ty);

/* cam.r1 = a[1][1]*cam.ty*cam.Sx;
cam.r2 = a[2][1]*cam.ty*cam.Sx;
cam.r3 = a[3][1]*cam.ty*cam.Sx;*/     /*sometimes better positions*/


cam.r1 = a[1][1]*cam.ty/cam.Sx;    /* from the paper*/
cam.r2 = a[2][1]*cam.ty/cam.Sx;
cam.r3 = a[3][1]*cam.ty/cam.Sx;


cam.tx = a[4][1]*cam.ty;
cam.r4 = a[5][1]*cam.ty;
```

```
     cam.r5 = a[6][1]*cam.ty;
     cam.r6 = a[7][1]*cam.ty;

     if(DEBUG==1) printf("Tx= %lf       Sx=%lf\n",cam.tx,cam.Sx);

     /*** Find last row by the cross product of the first two  ***/
     cam.r7 = cam.r2*cam.r6-cam.r3*cam.r5;
     cam.r8 =   cam.r3*cam.r4-cam.r1*cam.r6;
     cam.r9 =   cam.r1*cam.r5-cam.r2*cam.r4;


   indx=ivector(1,3);
   det[1][1]=(float)cam.r1; det[1][2]=(float)cam.r2; det[1][3]=(float)cam.r3;
   det[2][1]=(float)cam.r4; det[2][2]=(float)cam.r5; det[2][3]=(float)cam.r6;
   det[3][1]=(float)cam.r7; det[3][2]=(float)cam.r8; det[3][3]=(float)cam.r9;

     ludcmp(det,3,indx,&d); /***** get value for determinant    *****/

     for(j=1;j<=3;j++) d *= det[j][j];
     printf("Determinant of rotation matrix = %f\n\n",d);

     if(fabs(d-1.0)>0.01){
               printf("\nThere could be a problem with the input points\n");
               printf("since the determinant is not near 1.000\n");
               printf("hit any key to continue, garbage may result, however\n\n\a");
               getch();
               }
     /*** load up matrix for finding f, k1 and tz        ***/



     if(DEBUG==1)  printf("Loading up to find f,K1 and Tz\n");

     for(k=1;k<=i;k++){
     if(DEBUG==1) printf("a[%i][1] a[%i][2] a[%i][3] b[1][%i]\n ",k,k,k,k);
     a[k][1] = world[k].x*cam.r1+world[k].y*cam.r2+world[k].z*cam.r3+cam.tx;
     a[k][2] = (world[k].x*cam.r1+world[k].y*cam.r2+world[k].z*cam.r3+cam.tx)*
                                 (fb[k].x*fb[k].x+fb[k].y*fb[k].y);
     a[k][3] = (-fb[k].x);
     b[1][k] =fb[k].x*(world[k].x*cam.r7+world[k].y*cam.r8+world[k].z*cam.r9);
     }
     /*** loading up second part of matrix  ***/
     for(k=(i+1);k<=(int)(i*2);k++){
     if(DEBUG==1) printf("loading second part of a and b\n");
     if(DEBUG==1) printf("a[%i][1] a[%i][2] a[%i][3] b[1][%i]\n ",k,k,k,k);
                    a[k][1] = world[k-i].x*cam.r4+world[k-i].y*cam.r5+world[k-i].z
                                             *cam.r6+cam.ty;
                    a[k][2] = (world[k-i].x*cam.r4+world[k-i].y*cam.r5+world[k-i].z
                                             *cam.r6+cam.ty)*(fb[k-i].x*fb[k-i].x+fb[k-i].y*fb[k-i].y);
                    a[k][3] = (-fb[k-i].y);
                    b[1][k]= fb[k-i].y*(world[k-i].x*cam.r7+world[k-i].y
                                             *cam.r8+world[k-i].z*cam.r9);
     }
     least_square(a,b,(int)(i*2),3,1);

     cam.k1=a[2][1]/a[1][1];
     cam.f=a[1][1];
     cam.tz=a[3][1];

    /***** Big GOTO KLUDGE  ******/
    if( cam.f <= 0.0){
             printf("You Just Kludged\n");
```

```
            cam.ty = (-cam.ty);
            goto KLUDGE1;
            }
/***        Find largest distance between points          ***/
/***        dpt  distance    track_dpt=the point tag      ***/
dpt=dpt_old=0.0;
for(qq=1;qq<=i;qq++){
        for(rr=(qq+1);rr<=i;rr++){
                dx=world[qq].x-world[rr].x;
                dy=world[qq].y-world[rr].y;
                dz=world[qq].z-world[rr].z;
                dpt = sqrt(dx*dx+dy*dy+dz*dz);

                if( dpt>=dpt_old ){
                        dpt_old=dpt;
                        track_dpt[0]=qq;  /*** keep track of which points are max ***/
                        track_dpt[1]=rr;  /*** distance from each other           ***/
                        }
                }
        }
/********** this could be a subroutine here   *********/
   /*** print values for R, T, f, Sx and k1  ***/
   printf("\t\tCam\n");
   printf("\tr1=%lf\tr2=%lf\tr3=%lf\n",cam.r1,cam.r2,cam.r3);
   printf("\tr4=%lf\tr5=%lf\tr6=%lf\n",cam.r4,cam.r5,cam.r6);
   printf("\tr7=%lf\tr8=%lf\tr9=%lf\n\n",cam.r7,cam.r8,cam.r9);
   printf("\ttx=%lf\tty=%lf\ttz=%lf\n\n",cam.tx,cam.ty,cam.tz);
   printf("\tSx=%lf\tf=%lf\tk1=%lf\n\n",cam.Sx,cam.f,cam.k1);

 printf("\n\tdst=%lf\tpt1=%d\t\tpt2=%d\n",dpt_old,track_dpt[0],track_dpt[1]);

   fprintf(fp,"%5.15lf\t%5.15lf\t%5.15lf\n",cam.r1,cam.r2,cam.r3);
   fprintf(fp,"%5.15lf\t%5.15lf\t%5.15lf\n",cam.r4,cam.r5,cam.r6);
   fprintf(fp,"%5.15lf\t%5.15lf\t%5.15lf\n\n\n",cam.r7,cam.r8,cam.r9);
   fprintf(fp,"%5.15lf\t%5.15lf\t%5.15lf\n\n",cam.tx,cam.ty,cam.tz);
   fprintf(fp,"%5.15lf\t%5.15lf\t%5.15lf\n",cam.Sx,cam.f,cam.k1);

   fprintf(fp,"\n\t%lf\t\t%d\t\t%d\n",dpt_old,track_dpt[0],track_dpt[1]);

}
/*************** End calb_tsai **************************/

void least_square(double **a,double **b,int n,int w,int m)
/**** Returns answer in a   ***/
/*int m,n,w;
double **a,**b; */
{
int j,k,l;
double **ai,**at,**u,**x,**t;

ai=dmatrix(1,NP,1,NP);
u=dmatrix(1,NP,1,NP);

x=dmatrix(1,NP,1,WP);
t=dmatrix(1,NP,1,NP);
at=dmatrix(1,NP,1,NP);

        /*** Zero out transpose matrix to prevent overflows ***/
        for (l=1;l<=n;l++){
                for (k=1;k<=n;k++) at[k][l]=0.0;
        }
```

```
/*** create A transpose ***/
for (k=1;k<=n;k++){
            for (l=1;l<=w;l++) at[l][k]=a[k][l];
            }

/*** Print A transpose ***/
        if(DEBUG==1)printf("\nmatrix  AT\n");
        for (k=1;k<=w;k++) {
                for (l=1;l<=n;l++) if(DEBUG==1)printf("%12.6f",at[k][l]);
                if(DEBUG==1)printf("\nROW=%d\n",k);
        }



        /*** create ATA,  NxM  Matrix Multiplicaton      ***/
        for (l=1;l<=w;l++) {
                for (k=1;k<=n;k++){
                        t[k][l]=0.0;
                        for (j=1;j<=n;j++)
                                t[k][l] += (at[k][j]*a[j][l]);
                }
        }



/*** Print ATA ***/
        if(DEBUG==1)printf("\nmatrix  ATA\n");
        for (k=1;k<=w;k++) {
                for (l=1;l<=w;l++) if(DEBUG==1)printf("%12.6f",t[k][l]);
                if(DEBUG==1)printf("\n");
        }

        for (l=1;l<=n;l++){
                for (k=1;k<=w;k++) ai[k][l]=a[k][l];
                for (k=1;k<=m;k++) x[l][k]=b[l][k];
        }

gaussjf(t,w,x,m); /**** NOT THE BEST WAY TO INVERT, maybe ludcmp() *****/

/*** Print ATA inverse using gaussjordan ***/
        if(DEBUG==1)printf("\nmatrix  ATA inverse\n");
        for (k=1;k<=w;k++) {
                for (l=1;l<=w;l++) if(DEBUG==1)printf("%12.6f",t[k][l]);
                if(DEBUG==1)printf("\n");
        }

        /*** create ATA-1 * AT,  NxM  Matrix Multiplicaton      ***/
        for (l=1;l<=n;l++) {
                for (k=1;k<=w;k++){
                        u[k][l]=0.0;
                        for (j=1;j<=w;j++)
                                u[k][l] += (t[k][j]*at[j][l]);
                }
        }

        if(DEBUG==1)printf("\nmatrix  ATA-1*AT \n");
        for (k=1;k<=w;k++) {
                for (l=1;l<=n;l++) if(DEBUG==1)printf("%12.6f",u[k][l]);
                if(DEBUG==1)printf("\n");
        }

        /*** create ATA-1 * AT * B,  NxM  Matrix Multiplicaton      ***/
```

```
          for (l=1;l<=m;l++)  {
                    for (k=1;k<=w;k++) {
                              a[k][l]=0.0;
                              for (j=1;j<=n;j++)
                                        a[k][l] += (u[k][j]*b[l][j]);
                    }
          }

          if(DEBUG==1)printf("\nAnswer From Gaussfj \n");
          for (k=1;k<=n;k++)  {
                    for (l=1;l<=w;l++) if(DEBUG==1)printf("%12.6f",a[k][l]);
                    if(DEBUG==1)printf("\n");
          }

free_dmatrix(t,1,NP,1,NP);
free_dmatrix(x,1,NP,1,WP);
free_dmatrix(u,1,NP,1,NP);
free_dmatrix(ai,1,NP,1,NP);
free_dmatrix(at,1,NP,1,NP);
}
/*****  End least_square()  ********/
```

```
SHOWCALB.C    4-10-92
    Shows calibration points in various phases of calculation
    Used to verify calibration accuracy
  4-10-92 basic concept developed
 10-20-92 modified to accept new file headers and formats
                 By Andrew Jansky
```

```
#define CAM_PARM "camparm.dat"    /*** camera fixed parameter file  **/
#define OUT_LEFT "outcam1.prn"    /*** output from calibration      **/
#define OUT_RIGHT "outcam2.prn"   /*** output from calibration      **/
#define CAMERA1    "cam1.prn"     /*** original point locations     **/
#define CAMERA2    "cam2.prn"     /*** original point locations     **/
#define CALCED   "calced.prn"     /*** summary of calibration results**/
#define LIST     "list.prn"       /*** point by point list of results**/
#define FB_LOCATE "fblocate.prn"
#define VID1_ITI  "vid1iti.prn"
#define VID2_ITI  "vid2iti.prn"
#define MAXSTR 127

#include <conio.h>
#include <ctype.h>
#include <dos.h>
#include <io.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <itex150.h>
#include "regop.h"

static char *str[]={ "0","1","2","3","4","5","6","7","8","9","10","11","12",
              "13","14","15","16","17","18","19","20","21","22","23","24","25","26",
              "27","28","29","30","31","32","33","34","35","36","37","38","39","40"};


int bstat,row,col,rc,cc,bcount;
FILE *fil1;
FILE *Cam1;
FILE *Cam2;
FILE *lefti;
FILE *righti;
main()
{
int key=0;
int frame_flag=0;
int *ct,*rt;
int num1=0;
int num2=0;
int num;
int cam1[100][2];
int cam2[100][2];
int cam1nr[100][2];
int cam2nr[100][2];
int lft[100][2];
int rgt[100][2];
int left[100][2];
int leftnr[100][2];
int right[100][2];
int rightnr[100][2];
int c,r,i,end;
float x,y;
int temp;
float tempf;

char dummy [127];

          if( !(fil1 = fopen(FB_LOCATE,"r"))){
                    printf("Cannot open ");
                    printf( FB_LOCATE );
                    exit(0);
                    }
          for(i=0;i<12;i++)
          fgets(dummy,MAXSTR,fil1);                    /* strip header */

          if( !(Cam1 = fopen(CAMERA1,"r"))){
                    printf("Cannot open ");
                    printf( CAMERA1 );
                    exit(0);
                    }
          for(i=0;i<12;i++)
          fgets(dummy,MAXSTR,Cam1);                     /* strip header */

          if( !(Cam2 = fopen(CAMERA2,"r"))){
                    printf("Cannot open ");
                    printf( CAMERA2 );
                    exit(0);
                    }
          for(i=0;i<12;i++)
          fgets(dummy,MAXSTR,Cam2);                     /* strip header */

          if( !(lefti = fopen(VID1_ITI,"r"))){
                    printf("Cannot open ");
                    printf( VID1_ITI);
                    exit(0);
                    }
          for(i=0;i<12;i++)
          fgets(dummy,MAXSTR,lefti);                   /* strip header */

          if( !(righti = fopen(VID2_ITI,"r"))){
                    printf("Cannot open ");
                    printf( VID2_ITI );
                    exit(0);
                    }
          for(i=0;i<12;i++)
          fgets(dummy,MAXSTR,righti);                   /* strip header */


load_cfg("150.cfg");
initsys();


adi_initluts();                /**** Enable overlay on B2****/
adi_hbanksel(1);                 /**** Lut bank 1 is red   ****/
adi_hgroupsel(RED);
adi_clearlut(255);
adi_hgroupsel(GREEN|BLUE);
adi_clearlut(0);

adi_hbanksel(2);                 /**** Lut bank 2 is green ****/
adi_hgroupsel(GREEN);
```

```
adi_clearlut(255);
adi_hgroupsel(RED|BLUE);
adi_clearlut(0);

adi_hbanksel(3);
adi_hgroupsel(BLUE);
adi_clearlut(255);
adi_hgroupsel(RED|GREEN);
adi_clearlut(0);

adi_hbanksel(4);                /**** lut 4 white    ****/
adi_hgroupsel(BLUE|GREEN);
adi_clearlut(255);
adi_hgroupsel(RED);
adi_clearlut(250);

adi_hbanksel(5);                /*** pink            ****/
adi_hgroupsel(BLUE|GREEN|RED);
adi_clearlut(255);
adi_hgroupsel(GREEN|BLUE);
adi_clearlut(128);


adi_lutmode(DYNAMIC);

adi_camera(0);
select_path(LOOP_BACK);




 i=0;
 while( !feof(fil1))       /** feof stops input so it doesnt lock **/
                        {

                        fscanf(fil1,"%f",&x);
                        fscanf(fil1,"%f",&y);

                        cam1[i][0]=(int)x;
                        cam1[i][1]=(int)y;

                        fscanf(fil1,"%f",&x);
                        fscanf(fil1,"%f",&y);

                        cam2[i][0]=(int)x;
                        cam2[i][1]=(int)y;

                        fscanf(fil1,"%f",&x);
                        fscanf(fil1,"%f",&y);

                        cam1nr[i][0]=(int)x;
                        cam1nr[i][1]=(int)y;

                        fscanf(fil1,"%f",&x);
                        fscanf(fil1,"%f",&y);

                        cam2nr[i][0]=(int)x;
                        cam2nr[i][1]=(int)y;

                        fscanf(Cam1,"%i",&temp);
                        fscanf(Cam1,"%f",&x);
```

```
                    fscanf(Cam1,"%f",&y);
                    fscanf(Cam1,"%i",&temp);
                    fscanf(Cam1,"%f",&tempf);
                    fscanf(Cam1,"%f",&tempf);
                    fscanf(Cam1,"%f",&tempf);
                    lft[i][0]=(int)x;
                    lft[i][1]=(int)y;

                    fscanf(Cam2,"%i",&temp);
                    fscanf(Cam2,"%f",&x);
                    fscanf(Cam2,"%f",&y);
                    fscanf(Cam2,"%i",&temp);
                    fscanf(Cam2,"%f",&tempf);
                    fscanf(Cam2,"%f",&tempf);
                    fscanf(Cam2,"%f",&tempf);

                    rgt[i][0]=(int)x;
                    rgt[i][1]=(int)y;

                    fscanf(lefti,"%f",&x);
                    fscanf(lefti,"%f",&y);

                    left[i][0]=(int)x;
                    left[i][1]=(int)y;

                    fscanf(lefti,"%f",&x);
                    fscanf(lefti,"%f",&y);

                    leftnr[i][0]=(int)x;
                    leftnr[i][1]=(int)y;

                    fscanf(righti,"%f",&x);
                    fscanf(righti,"%f",&y);

                    right[i][0]=(int)x;
                    right[i][1]=(int)y;

                    fscanf(righti,"%f",&x);
                    fscanf(righti,"%f",&y);

                    rightnr[i][0]=(int)x;
                    rightnr[i][1]=(int)y;

                    i++;
                    }
end=i-1;
printf("Enter selection\n1=corrected and back project\n2=and delay\n3=all\n");
scanf( "%d",&key);

for(i=0;i<end;i++){
/*blu*/
                    line(B2,0,lft[i][0]-5,lft[i][1],lft[i][0]+5,lft[i][1],3);
                    line(B2,0,lft[i][0],lft[i][1]-5,lft[i][0],lft[i][1]+5,3);
                    if(i<=39)
                    text(B2,0,lft[i][0]-5,lft[i][1]-18,HORIZONTAL,1,3,str[i+1]);

/*rd*/ line(B2,0,cam1[i][0]-5,cam1[i][1],cam1[i][0]+5,cam1[i][1],1);
                    line(B2,0,cam1[i][0],cam1[i][1]-5,cam1[i][0],cam1[i][1]+5,1);
                    circle(B2,0,cam1[i][0],cam1[i][1],4,1,1,1);

/*wht*/ line(B2,0,left[i][0]-5,left[i][1],left[i][0]+5,left[i][1],4);
                    line(B2,0,left[i][0],left[i][1]-5,left[i][0],left[i][1]+5,4);
                    if(key= =2) getch();
                    if(key= =3){
/*grn*/ line(B2,0,cam1nr[i][0]-5,cam1nr[i][1],cam1nr[i][0]+5,cam1nr[i][1],2);
                    line(B2,0,cam1nr[i][0],cam1nr[i][1]-5,cam1nr[i][0],cam1nr[i][1]+5,2);


/*pinkt*/ line(B2,0,leftnr[i][0]-5,leftnr[i][1],leftnr[i][0]+5,leftnr[i][1],5);
                    line(B2,0,leftnr[i][0],leftnr[i][1]-5,leftnr[i][0],leftnr[i][1]+5,5);
                    }

          /*  printf("pt=%d Xlf=%d  Ylf=%d   CX=%d  CY=%d  CXnorad=%d  CYnorad=%d\n",
                    i+1,lft[i][0],lft[i][1],cam1[i][0],cam1[i][1],cam1nr[i][0],cam1nr[i][1]);
                    */
                    }

printf("Blue=original points     red=calculated     green=calced,no radial\n");
printf("hit a key to show second camera\n");
getch();
fb_clf(B2,0);


fb_clf(B2,0);
for(i=0;i<end;i++){
/*blu*/ line(B2,0,rgt[i][0]-5,rgt[i][1],rgt[i][0]+5,rgt[i][1],3);
                    line(B2,0,rgt[i][0],rgt[i][1]-5,rgt[i][0],rgt[i][1]+5,3);
                    if(i<=39)
                    text(B2,0,rgt[i][0],rgt[i][1]-18,HORIZONTAL,1,3,str[i+1]);

/*rd*/ line(B2,0,cam2[i][0]-5,cam2[i][1],cam2[i][0]+5,cam2[i][1],1);
                    line(B2,0,cam2[i][0],cam2[i][1]-5,cam2[i][0],cam2[i][1]+5,1);
                    circle(B2,0,cam2[i][0],cam2[i][1],4,1,1,1);


/*wht*/ line(B2,0,right[i][0]-5,right[i][1],right[i][0]+5,right[i][1],4);
                    line(B2,0,right[i][0],right[i][1]-5,right[i][0],right[i][1]+5,4);

                    if(key= =2) getch();
                    if(key= =3){
/*grn*/  line(B2,0,cam2nr[i][0]-5,cam2nr[i][1],cam2nr[i][0]+5,cam2nr[i][1],2);
                    line(B2,0,cam2nr[i][0],cam2nr[i][1]-5,cam2nr[i][0],cam2nr[i][1]+5,2);

/*pink*/line(B2,0,rightnr[i][0]-5,rightnr[i][1],rightnr[i][0]+5,rightnr[i][1],5);
                    line(B2,0,rightnr[i][0],rightnr[i][1]-5,rightnr[i][0],rightnr[i][1]+5,5);
                    }

          /*  printf("pt=%d Xlf=%d  Ylf=%d   CX=%d  CY=%d  CXnorad=%d  CYnorad=%d\n",
                    i+1,rgt[i][0],rgt[i][1],cam2[i][0],cam2[i][1],cam2nr[i][0],cam2nr[i][1]);
            */
                    }
/*
printf("Blue=original points     red=calculated     green=calced,no radial\n");
printf("White=single decoupled R and T     Pink=no radial decoupled R T  ");
*/

fclose(fil1);
fclose(Cam2);
fclose(Cam1);
fclose(lefti);
}
/**************          end Main  ***************************************/
```

```
              BACK.C    10-20-92  VERSION 3.0

    10-15  Reads values from various files including camera parameter
           file and from two seperate cameras, then calculates the
           world coordinates for a list of points, and compares this
           to the actual value.  The input calibration points are used

    11-7   Added back projection calculation for checking values
    11-19  changed all calculations to double's using doubles from calb
    6-15-92  tested with calbv2 and got excellent results, no numerical
             changes should be required.
    10-20-92  changed file headers and removed requirement for calb
             points to be in same order and number for each camera

    *******  Could eliminate vid1 and vid2 files, seems decoupled
             calculation results tend to confuse showcalb display
             Must track the calculations for it that are not necessary
             if removed.
                    By Andrew Jansky
```

```
#define VERSION "Version 3.0  10-20-92\n"
/**** NOTE: THE UNITS MUST BE THE SAME ON SENSOR SIZE AND MEASUREMENTS ***/

#define CAM_PARM  "camparm.dat"    /*** camera fixed parameter file   **/
#define OUT_LEFT  "outcam1.prn"    /*** output from calibration       **/
#define OUT_RIGHT "outcam2.prn"    /*** output from calibration       **/
#define CAM1      "cam1.prn"       /*** original point locations      **/
#define CAM2      "cam2.prn"       /*** original point locations      **/
#define CALCED    "calced.prn"     /*** summary of calibration results**/
#define LIST      "list.prn"       /*** point by point list of results**/
#define FB_LOCATE "fblocate.prn"
#define VID1_ITI  "vid1iti.prn"
#define VID2_ITI  "vid2iti.prn"

#include <time.h>
#include <string.h>

#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include "nr.h"
#include "nrutil.h"

#define DEBUG 0    /*  1 will give intermediate output  0 will not     */
#define NP 26      /*  Number of rows and colums in least square matrix */
#define MP 26
#define MAXSTR 127
#define MAX_ARRAY 131

void least_square(double **, double **, int , int , int );
void gaussjf(double **, int, double **, int);

struct camera {   /******* Camera parameter structure     **************/
        double r1;  double r2;  double r3;
        double r4;  double r5;  double r6;      /** Rotation matrix       **/
        double r7;  double r8;  double r9;

        double tx;  double ty;  double tz;      /** Translation matrix    **/
```

```
            double Sx;                        /** Scan error              **/
            double f;                         /** Effective focal length   **/
            double k1;                        /** Radial lens distortion   **/
            double Ncx;     double Ncy;            /** Image sensor lines        **/
            double Cx;      double Cy;       /** Where camera center is on IP plane*/
            double sensorx; double sensory;         /** Sensor size (same units) **/
            double field;  /*** diagonal field size  ***/
            };

struct  frame_buffer{                      /** Frame buffer coordinates **/
            double x;
            double y;
            int    total; /* total pixels */
            };
double IPNfx;  IPNfy;        /**** Image processor scan resoulution         ***/

struct world_coordinate{                   /** World point coordinates  ***/
            double x;
            double y;
            double z;
            int    t;    /* point tag    */
            };

struct world_coordinate World1[MAX_ARRAY]; /** world coordinate array 1   **/
struct world_coordinate World2[MAX_ARRAY]; /** world coordinate array 2   **/
struct world_coordinate world;

struct camera c1;          /**** camera 1 parameters fixed each setup   ***/
struct camera c2;          /**** camera 2 parameters fixed each setup   ***/

struct frame_buffer Fb1[MAX_ARRAY]; /**** point coordinate array for cam1***/
struct frame_buffer Fb2[MAX_ARRAY]; /**** point coordinate array for cam2***/
struct frame_buffer fb1;            /**** point coordinate for cam1      ***/
struct frame_buffer fb2;            /**** point coordinate for cam2      ***/



main()
{
int n=4;   /** matrix rows         , same for all 2 camera solutions **/
int w=3;   /** matrix cols         , same for all 2 camera solutions **/
int m=1;   /** number of solutions, same for all 2 camera solutions **/
int count=0;
int count2=0;
int tag_count=0; /** count of points processed ***/
int i=0;
int ii=0;
int j,k,l;
char tmpbuf[128];
double **a,**b;
double **R1;
double **R2;
double **temp;
double **temp2;

double Xu,Yu;                 double Xu2,Yu2;
double Xd,Yd;                 double Xd2,Yd2;
double Xf,Yf;                 double Xf2,Yf2;
double Xf_norad,Yf_norad;   double Xf2_norad,Yf2_norad;

double diffx=0.0   ,diffy=0.0   ,diffz=0.0;
double maxx=0.0    ,maxy=0.0    ,maxz=0.0;
double avgerrx=0.0 ,avgerry=0.0 ,avgerrz=0.0;
double max_dist;

double dx1,dx2,dy1,dy2,xu1,xu2,yu1,yu2,x1,y1,x2,y2;/**intermediate values**/

char dummy [MAXSTR];

FILE *fp;
FILE *fpp;
FILE *cam1;
FILE *cam2;
FILE *iti;
FILE *vid1;
FILE *vid2;



a=dmatrix(1,NP,1,NP);
b=dmatrix(1,NP,1,MP);
R1=dmatrix(1,3,1,3);
R2=dmatrix(1,3,1,3);

temp=dmatrix(1,3,1,1);
temp2=dmatrix(1,3,1,1);

/**************** Read Fixed values for both cameras    *******************/

if ((fp = fopen (CAM_PARM,"r")) == NULL)
        nrerror("Data file camparm.dat not found\n");

        fgets(dummy,MAXSTR,fp);
        printf("Reading---->");
        printf("%s",dummy);
        fgets(dummy,MAXSTR,fp);
        fscanf(fp,"%lf %lf  %lf   %lf   %lf  %lf   %lf ",&c1.Ncx,&c1.Ncy,
                                        &IPNfx,&c1.Cx,&c1.Cy,&c1.sensorx,&c1.sensory);
        fgets(dummy,MAXSTR,fp);
        fscanf(fp,"%lf %lf  %lf   %lf   %lf  %lf   %lf ",&c2.Ncx,&c2.Ncy,
                                        &IPNfx,&c2.Cx,&c2.Cy,&c2.sensorx,&c2.sensory);
        fgets(dummy,MAXSTR,fp);
        fscanf(fp,"%lf  %lf ",&c1.field,&c2.field);
        fclose(fp);

if(DEBUG==1){
printf("Left\n");
printf("Ncx=%lf Ncy=%lf Nfx=%lf  Cx=%lf  Cy=%lf SensX=%lf SensY=%lf\n",
                                c1.Ncx,c1.Ncy,IPNfx,c1.Cx,c1.Cy,c1.sensorx,c1.sensory);
printf("Right\n");
printf("Ncx=%lf Ncy=%lf Nfx=%lf  Cx=%lf  Cy=%lf SensX=%lf SensY=%lf\n",
                                c2.Ncx,c2.Ncy,IPNfx,c2.Cx,c2.Cy,c2.sensorx,c2.sensory);
}

/**************** Read Calibration Data for cam1 ************************/
printf("Reading---->Camera 1 Calibration File\n");
if ((fp = fopen (OUT_LEFT,"r")) == NULL)
        nrerror("Camera one calibration file not found (outcam1.prn)\n");
        for(i=0;i<15;i++)
        fgets(dummy,MAXSTR,fp);                    /* strip header */


        fscanf(fp,"%lf          %lf          %lf ",&c1.r1,&c1.r2,&c1.r3);
        fscanf(fp,"%lf          %lf          %lf ",&c1.r4,&c1.r5,&c1.r6);
        fscanf(fp,"%lf          %lf          %lf ",&c1.r7,&c1.r8,&c1.r9);

        fscanf(fp,"%lf    %lf    %lf ",&c1.tx,&c1.ty,&c1.tz);
        fscanf(fp," ");
        fscanf(fp,"%lf    %lf    %lf ",&c1.Sx,&c1.f,&c1.k1);
        fscanf(fp," ");
        fscanf(fp,"%lf ",&max_dist);
        fclose(fp);

/**************** Read Calibration Data for cam2 ************************/
printf("Reading---->Camera 2 Calibration File\n");
if ((fp = fopen (OUT_RIGHT,"r")) == NULL)
        nrerror("Camera two calibration file not found (outcam2.prn)\n");
        for(i=0;i<15;i++)
        fgets(dummy,MAXSTR,fp);                    /* strip header */


        fscanf(fp,"%lf    %lf    %lf ",&c2.r1,&c2.r2,&c2.r3);
        fscanf(fp,"%lf    %lf    %lf ",&c2.r4,&c2.r5,&c2.r6);
        fscanf(fp,"%lf    %lf    %lf ",&c2.r7,&c2.r8,&c2.r9);

        fscanf(fp,"%lf    %lf    %lf ",&c2.tx,&c2.ty,&c2.tz);
        fscanf(fp," ");
        fscanf(fp,"%lf    %lf    %lf ",&c2.Sx,&c2.f,&c2.k1);
        fscanf(fp," ");
                fscanf(fp," ");
        fscanf(fp,"%lf ",&max_dist);

        fclose(fp);


 if(DEBUG==1){           /*** print values for R, T, f, Sx and k1 ***/
        printf("\t\tCam1\n");
        printf("\tr1=%lf\tr2=%lf\tr3=%lf\n",c1.r1,c1.r2,c1.r3);
        printf("\tr4=%lf\tr5=%lf\tr6=%lf\n",c1.r4,c1.r5,c1.r6);
        printf("\tr7=%lf\tr8=%lf\tr9=%lf\n\n",c1.r7,c1.r8,c1.r9);
        printf("\ttx=%lf\tty=%lf\ttz=%lf\n\n",c1.tx,c1.ty,c1.tz);
        printf("\tSx=%lf\tf=%lf\tk1=%lf\n\n\n",c1.Sx,c1.f,c1.k1);
        printf("\t\tCam2\n");
        printf("\tr1=%lf\tr2=%lf\tr3=%lf\n" ,c2.r1,c2.r2,c2.r3);
        printf("\tr4=%lf\tr5=%lf\tr6=%lf\n",c2.r4,c2.r5,c2.r6);
        printf("\tr7=%lf\tr8=%lf\tr9=%lf\n\n",c2.r7,c2.r8,c2.r9);
        printf("\ttx=%lf\tty=%lf\ttz=%lf\n\n",c2.tx,c2.ty,c2.tz);
        printf("\tSx=%lf\tf=%lf\tk1=%lf\n",c2.Sx,c2.f,c2.k1);
        }
/*********** Open image processor coordinate files **********************/
if ((cam1 = fopen (CAM1,"r")) == NULL)
        nrerror("Data file for raw camera 1 not found\n");
        for(i=0;i<12;i++)
        fgets(dummy,MAXSTR,cam1);                    /* strip header */

if ((cam2 = fopen (CAM2,"r")) == NULL)
        nrerror("Data file for raw camera 2 not found\n");
        for(i=0;i<12;i++)
        fgets(dummy,MAXSTR,cam2);                    /* strip header */

/****************** Multiple point Loop begins here   ******************/
if ((fp = fopen (CALCED,"w")) == NULL)
        nrerror("Data file calced.prn not found\n");
        fprintf(fp,"%s \n",CALCED);
        _strtime ( tmpbuf );
        fprintf(fp,"%s \n",tmpbuf);
```

```
            _strdate ( tmpbuf );
            fprintf(fp,"%s \n\n",tmpbuf);
            fprintf(fp,"This file contains the results of the back.exe          \n");
            fprintf(fp,"The location of known calibration points are calculated \n");
            fprintf(fp,"and the results compared to the true values             \n");
            fprintf(fp,"This is the summary file                          \n\n\n\n");

fprintf(fp,"=================================================================
====\n");

if ((fpp = fopen (LIST,"w")) == NULL)
            nrerror("Data file list.prn not found\n");
            fprintf(fpp,"%s \n",LIST);
            _strtime ( tmpbuf );
            fprintf(fpp,"%s \n",tmpbuf);
            _strdate ( tmpbuf );
            fprintf(fpp,"%s \n\n",tmpbuf);
            fprintf(fpp,"This file contains the results of the back.exe          \n");
            fprintf(fpp,"The location of known calibration points are calculated \n");
            fprintf(fpp,"and the results compared to the true values             \n");
            fprintf(fpp,"This is the point by point comparison files     \n\n\n\n");

fprintf(fp,"=================================================================
====\n");

if ((iti = fopen (FB_LOCATE,"w")) == NULL)
            nrerror("Data file fblocat.prn not found\n");
            fprintf(iti,"%s \n",FB_LOCATE);
            _strtime ( tmpbuf );
            fprintf(iti,"%s \n",tmpbuf);
            _strdate ( tmpbuf );
            fprintf(iti,"%s \n\n",tmpbuf);
            fprintf(iti,"This file contains the results of the back.exe          \n");
            fprintf(iti,"The file is used by showcalb.exe to display calculated  \n");
            fprintf(iti,"screen locations of calibration points.  Varyious forms \n");
            fprintf(iti,"of back projection are used                      \n\n\n\n");

fprintf(iti,"=================================================================
====\n");

if ((vid1 = fopen (VID1_ITI,"w")) == NULL)
            nrerror("Data file vid1iti.prn not found\n");
  fprintf(vid1,"%s \n",VID1_ITI);
  _strtime ( tmpbuf );
  fprintf(vid1,"%s \n",tmpbuf);
  _strdate ( tmpbuf );
  fprintf(vid1,"%s \n\n",tmpbuf);
  fprintf(vid1,"This file contains the results of the back.exe          \n");
  fprintf(vid1,"The file is used by showcalb.exe to display calculated  \n");
  fprintf(vid1,"screen locations of calibration points.  Decoupled      \n");
  fprintf(vid1,"values with and with out radial distortion shown  \n\n\n\n");

fprintf(vid1,"=================================================================
=====\n");

if ((vid2 = fopen (VID2_ITI,"w")) == NULL)
            nrerror("Data file vid2iti.prn not found\n");
  fprintf(vid2,"%s \n",VID2_ITI);
  _strtime ( tmpbuf );
  fprintf(vid2,"%s \n",tmpbuf);
  _strdate ( tmpbuf );
  fprintf(vid2,"%s \n\n",tmpbuf);
```

```
   fprintf(vid2,"This file contains the results of the back.exe          \n");
   fprintf(vid2,"The file is used by showcalb.exe to display calculated  \n");
   fprintf(vid2,"screen locations of calibration points.  Decoupled       \n");
   fprintf(vid2,"values with and with out radial distortion shown  \n\n\n\n");

fprintf(vid2,"=====================================================
=====\n");

/*    c1.Sx=1.0/c1.Sx;
           c2.Sx=1.0/c2.Sx;      */

printf("Calculating");

while ((!feof(cam1))&&(i<MAX_ARRAY)) {
   if(i>=(MAX_ARRAY-1)) break;
                        if(count<68) printf(".");
                        count++;
   fscanf(cam1,"%i ",&World1[i].t);
   fscanf(cam1,"%lf ",&Fb1[i].x);
   fscanf(cam1,"%lf ",&Fb1[i].y);
   fscanf(cam1,"%i ",&Fb1[i].total);
   fscanf(cam1,"%lf ",&World1[i].x);
   fscanf(cam1,"%lf ",&World1[i].y);
   fscanf(cam1,"%lf ",&World1[i].z);
   i++;
   }

count=0;
ii=0;
while ((!feof(cam2))&&(i<MAX_ARRAY)) {
   if(ii>=(MAX_ARRAY-1)) break;
                        if(count<68) printf(".");
                        count++;

   fscanf(cam2,"%i ",&World2[ii].t);
   fscanf(cam2,"%lf ",&Fb2[ii].x);
   fscanf(cam2,"%lf ",&Fb2[ii].y);
   fscanf(cam2,"%i ",&Fb2[ii].total);
   fscanf(cam2,"%lf ",&World2[ii].x);
   fscanf(cam2,"%lf ",&World2[ii].y);
   fscanf(cam2,"%lf ",&World2[ii].z);
   ii++;
   }
printf("\n");

/*** step through cam1 and look for common points then calculate the ***/
/*** back projected value.                                      ***/

for(count=0;count<i;count++){
             for(count2=0;count2<ii;count2++){
               if(World1[count].t==World2[count2].t) {
               tag_count++;
               fb1.x=Fb1[count].x;
               fb1.y=Fb1[count].y;
               fb2.x=Fb2[count2].x;
               fb2.y=Fb2[count2].y;
               world.x=World1[count].x; /*** both views of point should have **/
               world.y=World1[count].y; /*** common world location            **/
               world.z=World1[count].z;
               world.t=World1[count].t;
               printf("Calculating tag number = %i\n" , World1[count].t);
```

```
/***find the ending ************/
        a[1][1]=world.x;
        a[2][1]=world.y;
        a[3][1]=world.z;
        /**** calculate projected location back on image plane ****/
        R1[1][1]=c1.r1;    R1[1][2]=c1.r2;    R1[1][3]=c1.r3;
        R1[2][1]=c1.r4;    R1[2][2]=c1.r5;    R1[2][3]=c1.r6;
        R1[3][1]=c1.r7;    R1[3][2]=c1.r8;    R1[3][3]=c1.r9;

        R2[1][1]=c2.r1;    R2[1][2]=c2.r2;    R2[1][3]=c2.r3;
        R2[2][1]=c2.r4;    R2[2][2]=c2.r5;    R2[2][3]=c2.r6;
        R2[3][1]=c2.r7;    R2[3][2]=c2.r8;    R2[3][3]=c2.r9;



        /***  NxM  Matrix Multiplicaton  cam1    ***/
        for (l=1;l<=1;l++) {
                for (k=1;k<=3;k++) {
                        temp[k][l]=0.0;
                        for (j=1;j<=3;j++)
                                temp[k][l] += (R1[k][j]*a[j][l]);
                }
        }
        /***  NxM  Matrix Multiplicaton  cam2    ***/
        for (l=1;l<=1;l++) {
                for (k=1;k<=3;k++) {
                        temp2[k][l]=0.0;
                        for (j=1;j<=3;j++)
                                temp2[k][l] += (R2[k][j]*a[j][l]);
                }
        }

        /****recalc begins*********/
        temp[1][1]=temp[1][1]+c1.tx;
        temp[2][1]=temp[2][1]+c1.ty;
        temp[3][1]=temp[3][1]+c1.tz;

        Xu=c1.f*(temp[1][1]/temp[3][1]);
        Yu=c1.f*(temp[2][1]/temp[3][1]);

        Xd=2.0*Xu/(1+sqrt(1.0-4.0*c1.k1*(Xu*Xu+Yu*Yu)));
        Yd=2.0*Yu/(1+sqrt(1.0-4.0*c1.k1*(Xu*Xu+Yu*Yu)));

        Xf=c1.Sx*Xd*IPNfx/c1.sensorx+c1.Cx;
        Yf=Yd*c1.Ncy/c1.sensory+c1.Cy;

        Xf_norad=c1.Sx*Xu*IPNfx/c1.sensorx+c1.Cx;
        Yf_norad=Yu*c1.Ncy/c1.sensory+c1.Cy;
        /**************** Second camera          **************/
        temp2[1][1]=temp2[1][1]+c2.tx;
        temp2[2][1]=temp2[2][1]+c2.ty;
        temp2[3][1]=temp2[3][1]+c2.tz;

        Xu2=c2.f*(temp2[1][1]/temp2[3][1]);
        Yu2=c2.f*(temp2[2][1]/temp2[3][1]);

        Xd2=2.0*Xu2/(1+sqrt(1.0-4.0*c2.k1*(Xu2*Xu2+Yu2*Yu2)));
        Yd2=2.0*Yu2/(1+sqrt(1.0-4.0*c2.k1*(Xu2*Xu2+Yu2*Yu2)));

        Xf2=c2.Sx*Xd2*IPNfx/c2.sensorx+c2.Cx;
        Yf2=Yd2*c2.Ncy/c2.sensory+c2.Cy;
```

```
            Xf2_norad=c2.Sx*Xu2*IPNfx/c2.sensorx+c2.Cx;
            Yf2_norad=Yu2*c2.Ncy/c2.sensory+c2.Cy;

            fprintf(vid1,"%5.1f %5.1f   %5.1f %5.1f \n"
            ,Xf,Yf,Xf_norad,Yf_norad);
            fprintf(vid2,"%5.1f %5.1f   %5.1f %5.1f \n"
            ,Xf2,Yf2,Xf2_norad,Yf2_norad);
            /*****END RECALC**********/
            dx1=(c1.sensorx/c1.Ncx)*(c1.Ncx/IPNfx);  /** calc intermediate values**/
            dy1=(c1.sensory/c1.Ncy);

            dx2=(c2.sensorx/c2.Ncx)*(c2.Ncx/IPNfx);
            dy2=(c2.sensory/c2.Ncy);

            x1=dx1 * c1.Sx * ((double)fb1.x-c1.Cx);  /**** POTENTIAL BUG  1/S.x ***/
            y1=dy1 * ((double)fb1.y-c1.Cy);

            x2=dx2 * c2.Sx * ((double)fb2.x-c2.Cx);  /**** POTENTIAL BUG  1/S.x ***/
            y2=dy2 * ((double)fb2.y-c2.Cy);

/**** radial lens distortion correction    ****/

/*      xu1=x1*(1+sqrt(1.0-4.0*c1.k1*(x1*x1+y1*y1)))/2.0;
             yu1=y1*(1+sqrt(1.0-4.0*c1.k1*(x1*x1+y1*y1)))/2.0;

             xu2=x1*(1+sqrt(1.0-4.0*c2.k1*(x2*x2+y2*y2)))/2.0;
             yu2=y1*(1+sqrt(1.0-4.0*c2.k1*(x2*x2+y2*y2)))/2.0;   */

            xu1=x1/(1+c1.k1*(x1*x1+y1*y1));
              yu1=y1/(1+c1.k1*(x1*x1+y1*y1));

            xu2=x2/(1+c2.k1*(x2*x2+y2*y2));
              yu2=y2/(1+c2.k1*(x2*x2+y2*y2));

            a[1][1]=c2.r1-(xu2/c2.f)*c2.r7;
            a[1][2]=c2.r2-(xu2/c2.f)*c2.r8;
            a[1][3]=c2.r3-(xu2/c2.f)*c2.r9;

            a[2][1]=c2.r4-(yu2/c2.f)*c2.r7;
            a[2][2]=c2.r5-(yu2/c2.f)*c2.r8;
            a[2][3]=c2.r6-(yu2/c2.f)*c2.r9;

            a[3][1]=c1.r1-(xu1/c1.f)*c1.r7;
            a[3][2]=c1.r2-(xu1/c1.f)*c1.r8;
            a[3][3]=c1.r3-(xu1/c1.f)*c1.r9;

            a[4][1]=c1.r4-(yu1/c1.f)*c1.r7;
            a[4][2]=c1.r5-(yu1/c1.f)*c1.r8;
            a[4][3]=c1.r6-(yu1/c1.f)*c1.r9;

            b[1][1]=((xu2/c2.f)*c2.tz)-c2.tx;
            b[1][2]=((yu2/c2.f)*c2.tz)-c2.ty;

            b[1][3]=((xu1/c1.f)*c1.tz)-c1.tx;
            b[1][4]=((yu1/c1.f)*c1.tz)-c1.ty;

            if(DEBUG==1) printf("matrix  A\n");
            for (k=1;k<=n;k++) {
                        for (l=1;l<=w;l++){ if(DEBUG==1)printf("%12.6f",a[k][l]);}
                        if(DEBUG==1)printf("\n");
            }
```

```
 if(DEBUG==1)printf("\nmatrix  B\n");
 for (k=1;k<=m;k++){
           for (l=1;l<=n;l++) if(DEBUG==1)printf("%12.6f",b[k][l]);
                              if(DEBUG==1)printf("\n");
           }

         /* perform least square calculation on a and b  */

         least_square(a,b,n,w,m); /**** could be (a,b,n,w,m)  ****/
         /***  calculated values for xw,yw,zw returned in a[1..3][1]         ****/

         /**** calculate projected location back on image plane ****/
         R1[1][1]=c1.r1;    R1[1][2]=c1.r2;   R1[1][3]=c1.r3;
         R1[2][1]=c1.r4;    R1[2][2]=c1.r5;   R1[2][3]=c1.r6;
         R1[3][1]=c1.r7;    R1[3][2]=c1.r8;   R1[3][3]=c1.r9;


         R2[1][1]=c2.r1;    R2[1][2]=c2.r2;   R2[1][3]=c2.r3;
         R2[2][1]=c2.r4;    R2[2][2]=c2.r5;   R2[2][3]=c2.r6;
         R2[3][1]=c2.r7;    R2[3][2]=c2.r8;   R2[3][3]=c2.r9;


         /***  NxM  Matrix Multiplicaton  cam1    ***/
         for (l=1;l<=1;l++) {
                   for (k=1;k<=3;k++){
                             temp[k][l]=0.0;
                             for (j=1;j<=3;j++)
                                       temp[k][l] += (R1[k][j]*a[j][l]);
                   }
         }
         /***  NxM  Matrix Multiplicaton  cam2    ***/
         for (l=1;l<=1;l++) {
                   for (k=1;k<=3;k++){
                             temp2[k][l]=0.0;
                             for (j=1;j<=3;j++)
                                       temp2[k][l] += (R2[k][j]*a[j][l]);
                   }
         }

if(i>=0){
         temp[1][1]=temp[1][1]+c1.tx;
         temp[2][1]=temp[2][1]+c1.ty;
         temp[3][1]=temp[3][1]+c1.tz;

         Xu=c1.f*(temp[1][1]/temp[3][1]);
         Yu=c1.f*(temp[2][1]/temp[3][1]);

         Xd=2.0*Xu/(1+sqrt(1.0-4.0*c1.k1*(Xu*Xu+Yu*Yu)));
         Yd=2.0*Yu/(1+sqrt(1.0-4.0*c1.k1*(Xu*Xu+Yu*Yu)));

         Xf=c1.Sx*Xd*IPNfx/c1.sensorx+c1.Cx;
         Yf=Yd*c1.Ncy/c1.sensory+c1.Cy;

         Xf_norad=c1.Sx*Xu*IPNfx/c1.sensorx+c1.Cx;
         Yf_norad=Yu*c1.Ncy/c1.sensory+c1.Cy;

         /*************** Second camera          **************/
         temp2[1][1]=temp2[1][1]+c2.tx;
         temp2[2][1]=temp2[2][1]+c2.ty;
         temp2[3][1]=temp2[3][1]+c2.tz;

         Xu2=c2.f*(temp2[1][1]/temp2[3][1]);
```

```
          Yu2=c2.f*(temp2[2][1]/temp2[3][1]);

          Xd2=2.0*Xu2/(1+sqrt(1.0-4.0*c2.k1*(Xu2*Xu2+Yu2*Yu2)));
          Yd2=2.0*Yu2/(1+sqrt(1.0-4.0*c2.k1*(Xu2*Xu2+Yu2*Yu2)));

          Xf2=c2.Sx*Xd2*IPNfx/c2.sensorx+c2.Cx;
          Yf2=Yd2*c2.Ncy/c2.sensory+c2.Cy;

          Xf2_norad=c2.Sx*Xu2*IPNfx/c2.sensorx+c2.Cx;
          Yf2_norad=Yu2*c2.Ncy/c2.sensory+c2.Cy;
 if(DEBUG==1)
  printf("%5.1f %5.1lf  %5.1lf %5.1lf   %5.1lf  %5.1lf   %5.1lf  %5.1lf\n"
          ,Xf,Yf,Xf2,Yf2,Xf_norad,Yf_norad,Xf2_norad,Yf2_norad);

  fprintf(iti,"%5.1lf %5.1lf  %5.1lf %5.1lf   %5.1lf  %5.1lf   %5.1lf  %5.1lf\n"
          ,Xf,Yf,Xf2,Yf2,Xf_norad,Yf_norad,Xf2_norad,Yf2_norad);
          }


          i++;
          diffx +=  fabs(a[1][1]-world.x);
          diffy +=  fabs(a[2][1]-world.y);
          diffz +=  fabs(a[3][1]-world.z);

          if (fabs(a[1][1]-world.x) > fabs(maxx))  maxx = a[1][1]-world.x;
          if (fabs(a[2][1]-world.y) > fabs(maxy))  maxy = a[2][1]-world.y;
          if (fabs(a[3][1]-world.z) > fabs(maxz))  maxz = a[3][1]-world.z;

          avgerrx +=  fabs(a[1][1]-world.x)/((c1.field+c2.field)/2.0);
          avgerry +=  fabs(a[2][1]-world.y)/((c1.field+c2.field)/2.0);
          avgerrz +=  fabs(a[3][1]-world.z)/((c1.field+c2.field)/2.0);

          fprintf(fpp,"Tag calculated  %d      AvgError   %lf\n",world.t,
                                                (diffx+diffy+diffz)/(3.0*(double)i));

          fprintf(fpp,"Calculated %8.2lf    %8.2lf    %8.2lf\n",a[1][1],a[2][1],
                                                                                        a[3][1],
          fprintf(fpp,"World      %8.2lf    %8.2lf    %8.2lf\n",world.x,world.y,

world.z);
          fprintf(fpp,"Difference %8.2lf    %8.2lf    %8.2lf\n\n",a[1][1]-world.x,

a[2][1]-world.y,a[3][1]-world.z);
} /**** end of count2 search loop  **/
} /**** end of count  search loop  **/
} /**** End of points        **/

printf("\n\nPoints Calculated  %d            AvgError  %3.10lf\n",tag_count,
           (diffx+diffy+diffz)/(3.0*(double)tag_count));


printf("AvgError inches\tX = %3.3lf\tY = %3.3lf\tZ = %3.3lf\n",
           (diffx/(double)i),(diffy/(double)i),(diffz/(double)tag_count));

printf("AvgErr/max dist\tX = %3.3lf%%\tY = %3.3lf%%\tZ = %3.3lf%%\n",
          (100.0*((diffx/(double)i)/max_dist)),(100.0*((diffy/(double)i)/max_dist)),
          (100.0*((diffz/(double)i)/max_dist)));

printf("AvgErr %% field\tX = %3.3lf%%\tY = %3.3lf%%\tZ = %3.3lf%% \n",
           (100.0*avgerrx/(double)tag_count),(100.0*avgerry/(double)tag_count),
           (100.0*avgerrz/(double)tag_count));
```

```
printf("MaxError      \tX = %3.3lf\tY = %3.3lf\tZ = %3.3lf\n",
            maxx,maxy,maxz);

        printf("Enter Description:  ");
        gets(dummy);
        fprintf(fp,"%s\n",dummy);

fprintf(fp,"\n\nPoints Calculated  %d      AvgError  %lf\n",tag_count,
            (diffx+diffy+diffz)/(3.0*(double)tag_count));

fprintf(fp,"Diagonal field of view\tcam1 = %3.1lf\tcam2 = %3.1f\n\n",
            c1.field,c2.field);

fprintf(fp,"AvgError inches\tX = %3.3lf\tY = %3.3lf\tZ = %3.3lf\n",
            (diffx/(double)tag_count),(diffy/(double)tag_count),
            (diffz/(double)tag_count));

fprintf(fp,"AvgErr/max dist\tX = %3.3lf%%\tY = %3.3lf%%\tZ = %3.3lf%%\n",
           (100.0*((diffx/(double)tag_count)/max_dist)),
           (100.0*((diffy/(double)tag_count)/max_dist)),
           (100.0*((diffz/(double)tag_count)/max_dist)));


fprintf(fp,"AvgErr %% field\tX = %3.3lf%%\tY = %3.3lf%%\tZ = %3.3lf%%\n",
            (100.0*avgerrx/(double)tag_count),
            (100.0*avgerry/(double)tag_count),
            (100.0*avgerrz/(double)tag_count));

fprintf(fp,"MaxError      \tX = %3.3lf\tY = %3.3lf\tZ = %3.3lf\n",
            maxx,maxy,maxz);


fclose(cam1);
fclose(cam2);

fclose(fp);
fclose(fpp);
fclose(iti);
fclose(vid1);
fclose(vid2);

free_dmatrix(b,1,NP,1,NP);
free_dmatrix(a,1,NP,1,NP);
free_dmatrix(R1,1,3,1,3);
free_dmatrix(R2,1,3,1,3);
free_dmatrix(temp,1,3,1,1);
free_dmatrix(temp2,1,3,1,1);
printf("\n\nVerification Program Back.exe, ");
printf(VERSION);
}
/****** END MAIN *******/

void least_square(double **a,double **b,int n,int w,int m)
/**** Returns answer in a  ***/
/*int m,n,w;
double **a,**b; */
{
int j,k,l;
double **ai,**at,**u,**x,**t;

ai=dmatrix(1,NP,1,NP);
u=dmatrix(1,NP,1,NP);
```

```
x=dmatrix(1,NP,1,NP);
t=dmatrix(1,NP,1,NP);
at=dmatrix(1,NP,1,NP);

        /*** Zero out transpose matrix to prevent overflows ***/
        for (l=1;l<=n;l++){
                for (k=1;k<=n;k++) at[k][l]=0.0;
        }

  /*** create A transpose ***/
  for (k=1;k<=n;k++){
          for (l=1;l<=w;l++) at[l][k]=a[k][l];
          }

  /*** Print A transpose ***/
        if(DEBUG==1)printf("\nmatrix  AT\n");
        for (k=1;k<=w;k++) {
                for (l=1;l<=n;l++) if(DEBUG==1)printf("%12.6f",at[k][l]);
                if(DEBUG==1)printf("\nROW=%d\n",k);
        }



        /*** create ATA,   NxM  Matrix Multiplicaton       ***/
        for (l=1;l<=w;l++) {
                for (k=1;k<=n;k++) {
                        t[k][l]=0.0;
                        for (j=1;j<=n;j++)
                                t[k][l] += (at[k][j]*a[j][l]);
                }
        }



  /*** Print ATA ***/
        if(DEBUG==1)printf("\nmatrix  ATA\n");
        for (k=1;k<=w;k++) {
                for (l=1;l<=w;l++) if(DEBUG==1)printf("%12.6f",t[k][l]);
                if(DEBUG==1)printf("\n");
        }

        for (l=1;l<=n;l++){
                for (k=1;k<=w;k++) ai[k][l]=a[k][l];
                for (k=1;k<=m;k++) x[l][k]=b[l][k];
        }

  gaussjf(t,w,x,m); /**** NOT THE BEST WAY TO INVERT, maybe ludcmp() *****/

  /*** Print ATA inverse using gaussjordan ***/
        if(DEBUG==1)printf("\nmatrix  ATA inverse\n");
        for (k=1;k<=w;k++) {
                for (l=1;l<=w;l++) if(DEBUG==1)printf("%12.6f",t[k][l]);
                if(DEBUG==1)printf("\n");
        }

        /*** create ATA-1 * AT,   NxM  Matrix Multiplicaton       ***/
        for (l=1;l<=n;l++) {
                for (k=1;k<=w;k++) {
                        u[k][l]=0.0;
                        for (j=1;j<=w;j++)
                                u[k][l] += (t[k][j]*at[j][l]);
                }
          }

          if(DEBUG==1)printf("\nmatrix  ATA-1*AT \n");
          for (k=1;k<=w;k++)  {
                    for (l=1;l<=n;l++) if(DEBUG==1)printf("%12.6f",u[k][l]);
                    if(DEBUG==1)printf("\n");
          }

          /*** create ATA-1 * AT * B,   NxM  Matrix Multiplicaton       ***/
          for (l=1;l<=m;l++)  {
                    for (k=1;k<=w;k++) {
                              a[k][l]=0.0;
                              for (j=1;j<=n;j++)
                                        a[k][l] += (u[k][j]*b[l][j]);
                    }
          }

          if(DEBUG==1)printf("\nAnswer From Gaussfj \n");
          for (k=1;k<=n;k++)  {
                    for (l=1;l<=w;l++) if(DEBUG==1)printf("%12.6f",a[k][l]);
                    if(DEBUG==1)printf("\n");
          }

free_dmatrix(t,1,NP,1,NP);
free_dmatrix(x,1,NP.1,NP);
free_dmatrix(u,1,NP,1,NP);
free_dmatrix(ai,1,NP,1,NP);
free_dmatrix(at,1,NP,1,NP);
}
/*****  End least_square()  ********/
```

```
LOCATE.C    10-20-92  VERSION 3.0

    10-15 Reads values from various files including camera parameter
          file and from two seperate cameras, then calculates the
          world coordinates for a list of points, and compares this
          to the actual value.  The input calibration points are used

    11-7   Added back projection calculation for checking values
    11-19  changed all calculations to double's using doubles from calb
    6-15-92 Tested with calbv2 and got excellent results, not numerical
           changes should be required.
    6-15   Rename 'Locate.c',just reads camparm file and each camera
          calibration file, then opens files and does two camera
          stereo correlation and writes the 3-D results to a file (loc)
    10-20-92 modified reading of files to accept new headers

    ******** Need to incorporate syncronizing of times from both files
            so that it is automaticaly done

                        By Andrew Jansky
```

```
#d fine  VERSION  "Version 3.0  10-20-92\n"
/****  NOTE: THE UNITS MUST BE THE SAME ON SENSOR SIZE AND MEASUREMENTS  ***/
#define  CAM_PARM  "camparm.dat"
#define  OUT_CAM1  "outcam1.prn"
#define  OUT_CAM2  "outcam2.prn"
#define  LOCATE    "location.prn"
#define  CAM1      "cam1vid.prn"
#define  CAM2      "cam2vid.prn"

#include <time.h>
#include <string.h>

#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include "nr.h"
#include "nrutil.h"

#define DEBUG 0    /*  1 will give intermediate output  0 will not      */
#define NP 26      /*  Number of rows and colums in least square matrix */
#define MP 26
#define MAXSTR 80

void least_square(double **, double **, int , int , int );
void gaussjf(double **, int, double **, int);

struct camera {   /******* Camera parameter structure     **************/
          double r1;  double r2;  double r3;
          double r4;  double r5;  double r6;     /** Rotation matrix        **/
          double r7;  double r8;  double r9;

          double tx;  double ty;  double tz;     /** Translation matrix     **/

          double Sx;                       /** Scan error            **/
          double f;                        /** Effective focal length  **/
          double k1;                       /** Radial lens distortion  **/
          double Ncx;     double Ncy;            /** Image sensor lines     **/
          double Cx;      double Cy;       /** Where camera center is on IP plane*/
          double sensorx; double sensory;         /** Sensor size (same units) **/
          double field;  /*** diagonal field size  ***/
          };

struct  frame_buffer{                    /** Frame buffer coordinates **/
          double x;
          double y;
          };
double IPNfx;  IPNfy;       /**** Image processor scan resoulution        ***/


struct camera c1;          /**** camera 1 parameters fixed each setup   ***/
struct camera c2;          /**** camera 2 parameters fixed each setup   ***/

struct frame_buffer fb1;   /**** point coordinate for cam1              ***/
struct frame_buffer fb2;   /**** point coordinate for cam2              ***/



main()
{
int n=4;  /** matrix rows        , same for all 2 camera solutions **/
int w=3;  /** matrix cols        , same for all 2 camera solutions **/
int m=1;  /** number of solutions, same for all 2 camera solutions **/

int i=0;
int j,k,l;
double **a,**b;

double Xu,Yu;               double Xu2,Yu2;
double Xd,Yd;               double Xd2,Yd2;
double Xf,Yf;               double Xf2,Yf2;
double Xf_norad,Yf_norad;   double Xf2_norad,Yf2_norad;

double max_dist;

double dx1,dx2,dy1,dy2,xu1,xu2,yu1,yu2,x1,y1,x2,y2;/**intermediate values**/

char dummy [MAXSTR];
char tmpbuf[128];
char cam1_time[30];
char cam2_time[30];

FILE *fp;
FILE *cam1;
FILE *cam2;
FILE *loc;

a=dmatrix(1,NP,1,NP);
b=dmatrix(1,NP,1,MP);


/**************** Read Fixed values for both cameras    ******************/

if  ((fp = fopen (CAM_PARM,"r")) == NULL)
          nrerror("Data file camparm.dat not found\n");

          fgets(dummy,MAXSTR,fp);
          printf("Reading---->");
          printf("%s",dummy);
          fgets(dummy,MAXSTR,fp);
```

```
        fscanf(fp,"%lf %lf  %lf    %lf    %lf   %lf   %lf ",&c1.Ncx,&c1.Ncy,
                                                &IPNfx,&c1.Cx,&c1.Cy,&c1.sensorx,&c1.sensory);
        fgets(dummy,MAXSTR,fp);
        fscanf(fp,"%lf %lf  %lf    %lf    %lf   %lf   %lf ",&c2.Ncx,&c2.Ncy,
                                                &IPNfx,&c2.Cx,&c2.Cy,&c2.sensorx,&c2.sensory);
        fgets(dummy,MAXSTR,fp);
        fscanf(fp,"%lf   %lf ",&c1.field,&c2.field);
        fclose(fp);

if(DEBUG==1){
printf("Left\n");
printf("Ncx=%lf Ncy=%lf Nfx=%lf  Cx=%lf  Cy=%lf SensX=%lf SensY=%lf\n",
                                        c1.Ncx,c1.Ncy,IPNfx,c1.Cx,c1.Cy,c1.sensorx,c1.sensory);
printf("Right\n");
printf("Ncx=%lf Ncy=%lf Nfx=%lf  Cx=%lf  Cy=%lf SensX=%lf SensY=%lf\n",
                                        c2.Ncx,c2.Ncy,IPNfx,c2.Cx,c2.Cy,c2.sensorx,c2.sensory);
}

/***************** Read Calibration Data for cam1 **************************/
printf("Reading---->Camera 1 Calibration File\n");
if ((fp = fopen (OUT_CAM1,"r")) == NULL)
        nrerror("Data file for camera 1 not found\n");
        for(i=0;i<15;i++)
        fgets(dummy,MAXSTR,fp);                          /* strip header */

        fscanf(fp,"%lf          %lf          %lf ",&c1.r1,&c1.r2,&c1.r3);
        fscanf(fp,"%lf          %lf          %lf ",&c1.r4,&c1.r5,&c1.r6);
        fscanf(fp,"%lf          %lf          %lf ",&c1.r7,&c1.r8,&c1.r9);

        fscanf(fp,"%lf     %lf     %lf ",&c1.tx,&c1.ty,&c1.tz);
        fscanf(fp," ");
        fscanf(fp,"%lf     %lf     %lf ",&c1.Sx,&c1.f,&c1.k1);
        fscanf(fp," ");
        fscanf(fp,"%lf ",&max_dist);
        fclose(fp);

/***************** Read Calibration Data for cam2 **************************/
printf("Reading---->Camera 2 Calibration File\n");
if ((fp = fopen (OUT_CAM2,"r")) == NULL)
        nrerror("Data file for camera 2 not found\n");
        for(i=0;i<15;i++)
        fgets(dummy,MAXSTR,fp);                          /* strip header */

        fscanf(fp,"%lf     %lf     %lf ",&c2.r1,&c2.r2,&c2.r3);
        fscanf(fp,"%lf     %lf     %lf ",&c2.r4,&c2.r5,&c2.r6);
        fscanf(fp,"%lf     %lf     %lf ",&c2.r7,&c2.r8,&c2.r9);

        fscanf(fp,"%lf     %lf     %lf ",&c2.tx,&c2.ty,&c2.tz);
        fscanf(fp," ");
        fscanf(fp,"%lf     %lf     %lf ",&c2.Sx,&c2.f,&c2.k1);
        fscanf(fp," ");
                fscanf(fp," ");
        fscanf(fp,"%lf ",&max_dist);

        fclose(fp);


 if(DEBUG==1){            /*** print values for R, T, f, Sx and k1  ***/
        printf("\t\tCam1\n");
        printf("\tr1=%lf\tr2=%lf\tr3=%lf\n",c1.r1,c1.r2,c1.r3);
        printf("\tr4=%lf\tr5=%lf\tr6=%lf\n",c1.r4,c1.r5,c1.r6);
        printf("\tr7=%lf\tr8=%lf\tr9=%lf\n\n",c1.r7,c1.r8,c1.r9);
        printf("\ttx=%lf\tty=%lf\ttz=%lf\n\n",c1.tx,c1.ty,c1.tz);
        printf("\tSx=%lf\tf=%lf\tk1=%lf\n\n\n",c1.Sx,c1.f,c1.k1);
        printf("\t\tCam2\n");
        printf("\tr1=%lf\tr2=%lf\tr3=%lf\n" ,c2.r1,c2.r2,c2.r3);
        printf("\tr4=%lf\tr5=%lf\tr6=%lf\n",c2.r4,c2.r5,c2.r6);
        printf("\tr7=%lf\tr8=%lf\tr9=%lf\n\n",c2.r7,c2.r8,c2.r9);
        printf("\ttx=%lf\tty=%lf\ttz=%lf\n\n",c2.tx,c2.ty,c2.tz);
        printf("\tSx=%lf\tf=%lf\tk1=%lf\n",c2.Sx,c2.f,c2.k1);
        }
/************ Open image processor coordinate files ***********************/
if ((cam1 = fopen (CAM1,"r")) == NULL)
        nrerror("Data file for raw camera 1 not found\n");

if ((cam2 = fopen (CAM2,"r")) == NULL)
        nrerror("Data file for raw camera 2 not found\n");

if ((loc = fopen (LOCATE,"w")) == NULL)
        nrerror("Data file location.prn not created\n");

        printf("Enter file header description :  ");
        gets(dummy);
        fprintf(loc,"%s\n",dummy);

        fprintf(loc,"%s \n",LOCATE);
        _strtime ( tmpbuf );
        fprintf(loc,"%s \n",tmpbuf);
        _strdate ( tmpbuf );
        fprintf(loc,"%s \n",tmpbuf);
        fprintf(loc,"This file contains the final 3-D location of the\n");
        fprintf(loc,"targets location from the two 2-D location files\n\n\n\n");
        fprintf(loc," Time1            Time2              X       Y       Z \n");

fprintf(loc,"=====================================================
====\n");

/****************** Multiple point Loop begins here   ******************/


        for(i=1;i<=4;i++)               /*** clear file headers  ***/
        fgets(dummy,MAXSTR,cam1);
        for(i=1;i<=4;i++)
        fgets(dummy,MAXSTR,cam2);


while ((!feof(cam1))||(!feof(cam2))) {

        fscanf(cam1,"%s",&cam1_time);       /** reads time   **/
        fscanf(cam1,"%lf  %lf ",&fb1.x,&fb1.y);
        fscanf(cam2,"%s",&cam2_time);       /** reads time   **/
        fscanf(cam2,"%lf  %lf ",&fb2.x,&fb2.y);



        dx1=(c1.sensorx/c1.Ncx)*(c1.Ncx/IPNfx);  /** calc intermediate values**/
        dy1=(c1.sensory/c1.Ncy);

        dx2=(c2.sensorx/c2.Ncx)*(c2.Ncx/IPNfx);
        dy2=(c2.sensory/c2.Ncy);

        x1=dx1 * c1.Sx * ((double)fb1.x-c1.Cx);  /**** POTENTIAL BUG  1/S.x ***/
        y1=dy1 * ((double)fb1.y-c1.Cy);
        x2=dx2 * c2.Sx * ((double)fb2.x-c2.Cx);  /**** POTENTIAL BUG  1/S.x ***/
        y2=dy2 * ((double)fb2.y-c2.Cy);

/**** radial lens distortion correction    ****/

/*    xu1=x1*(1+sqrt(1.0-4.0*c1.k1*(x1*x1+y1*y1)))/2.0;
          yu1=y1*(1+sqrt(1.0-4.0*c1.k1*(x1*x1+y1*y1)))/2.0;

          xu2=x1*(1+sqrt(1.0-4.0*c2.k1*(x2*x2+y2*y2)))/2.0;
          yu2=y1*(1+sqrt(1.0-4.0*c2.k1*(x2*x2+y2*y2)))/2.0;   */

        xu1=x1/(1+c1.k1*(x1*x1+y1*y1));
          yu1=y1/(1+c1.k1*(x1*x1+y1*y1));

        xu2=x2/(1+c2.k1*(x2*x2+y2*y2));
          yu2=y2/(1+c2.k1*(x2*x2+y2*y2));

        a[1][1]=c2.r1-(xu2/c2.f)*c2.r7;
        a[1][2]=c2.r2-(xu2/c2.f)*c2.r8;
        a[1][3]=c2.r3-(xu2/c2.f)*c2.r9;

        a[2][1]=c2.r4-(yu2/c2.f)*c2.r7;
        a[2][2]=c2.r5-(yu2/c2.f)*c2.r8;
        a[2][3]=c2.r6-(yu2/c2.f)*c2.r9;

        a[3][1]=c1.r1-(xu1/c1.f)*c1.r7;
        a[3][2]=c1.r2-(xu1/c1.f)*c1.r8;
        a[3][3]=c1.r3-(xu1/c1.f)*c1.r9;

        a[4][1]=c1.r4-(yu1/c1.f)*c1.r7;
        a[4][2]=c1.r5-(yu1/c1.f)*c1.r8;
        a[4][3]=c1.r6-(yu1/c1.f)*c1.r9;

        b[1][1]=((xu2/c2.f)*c2.tz)-c2.tx;
        b[1][2]=((yu2/c2.f)*c2.tz)-c2.ty;

        b[1][3]=((xu1/c1.f)*c1.tz)-c1.tx;
        b[1][4]=((yu1/c1.f)*c1.tz)-c1.ty;

        if(DEBUG==1) printf("matrix  A\n");
        for (k=1;k<=n;k++)  {
                for (l=1;l<=w;l++){ if(DEBUG==1)printf("%12.6f",a[k][l]);}
                if(DEBUG==1)printf("\n");
        }


  if(DEBUG==1)printf("\nmatrix  B\n");
  for (k=1;k<=m;k++){
          for (l=1;l<=n;l++) if(DEBUG==1)printf("%12.6f",b[k][l]);
                          if(DEBUG==1)printf("\n");
          }

        /* perform least square calculation on a and b  */

        least_square(a,b,n,w,m); /**** could be (a,b,n,w,m)   ****/
        /***  calculated values for xw,yw,zw returned in a[1..3][1]        ****/

        i++;
/*** print results to screen (slow)  ***/
/*  printf("Point %i  X = %8.2lf  Y = %8.2lf  Z = %8.2lf\n"
                     ,i,a[1][1],a[2][1],a[3][1]); */
```

```
            fprintf(loc,"%s\t%s\t %8.2lf    %8.2lf    %8.2lf\n",cam1_time,
                                        cam2_time,a[1][1],a[2][1],a[3][1]);

} /**** End of points   ***/

printf("\n\nPoints processed: %d\n\n\n",i);
printf("Location performed by Locate.exe, ");
printf(VERSION);

fclose(cam1);
fclose(cam2);
fclose(loc);


free_dmatrix(b,1,NP,1,NP);
free_dmatrix(a,1,NP,1,NP);


}
/****** END MAIN *******/

void least_square(double **a,double **b,int n,int w,int m)
/**** Returns answer in a  ***/
/*int m,n,w;
double **a,**b; */
{
int j,k,l;
double **ai,**at,**u,**x,**t;

ai=dmatrix(1,NP,1,NP);
u=dmatrix(1,NP,1,NP);

x=dmatrix(1,NP,1,NP);
t=dmatrix(1,NP,1,NP);
at=dmatrix(1,NP,1,NP);

            /*** Zero out transpose matrix to prevent overflows ***/
            for (l=1;l<=n;l++){
                        for (k=1;k<=n;k++) at[k][l]=0.0;
            }

   /*** create A transpose  ***/
   for (k=1;k<=n;k++){
              for (l=1;l<=w;l++) at[l][k]=a[k][l];
              }

   /*** Print A transpose  ***/
            if(DEBUG==1)printf("\nmatrix  AT\n");
            for (k=1;k<=w;k++)  {
                        for (l=1;l<=n;l++) if(DEBUG==1)printf("%12.6f",at[k][l]);
                        if(DEBUG==1)printf("\nROW=%d\n",k);
            }



            /*** create ATA,  NxM  Matrix Multiplicaton       ***/
            for (l=1;l<=w;l++)  {
                        for (k=1;k<=n;k++) {
                                    t[k][l]=0.0;
                                    for (j=1;j<=n;j++)
                                                t[k][l] += (at[k][j]*a[j][l]);
```

```
                }
        }

    /*** Print ATA  ***/
        if(DEBUG==1)printf("\nmatrix  ATA\n");
        for (k=1;k<=w;k++)  {
                for (l=1;l<=w;l++) if(DEBUG==1)printf("%12.6f",t[k][l]);
                if(DEBUG==1)printf("\n");
        }

        for (l=1;l<=n;l++){
                for (k=1;k<=w;k++) ai[k][l]=a[k][l];
                for (k=1;k<=m;k++) x[l][k]=b[l][k];
        }

    gaussjf(t,w,x,m); /****  NOT THE BEST WAY TO INVERT, maybe ludcmp() *****/

    /*** Print ATA inverse using gaussjordan ***/
        if(DEBUG==1)printf("\nmatrix  ATA inverse\n");
        for (k=1;k<=w;k++)  {
                for (l=1;l<=w;l++) if(DEBUG==1)printf("%12.6f",t[k][l]);
                if(DEBUG==1)printf("\n");
        }

        /*** create ATA-1 * AT,   NxM  Matrix Multiplicaton       ***/
        for (l=1;l<=n;l++)  {
                for (k=1;k<=w;k++) {
                        u[k][l]=0.0;
                        for (j=1;j<=w;j++)
                                u[k][l] += (t[k][j]*at[j][l]);
                }
        }

        if(DEBUG==1)printf("\nmatrix  ATA-1*AT \n");
        for (k=1;k<=w;k++)  {
                for (l=1;l<=n;l++) if(DEBUG==1)printf("%12.6f",u[k][l]);
                if(DEBUG==1)printf("\n");
        }

        /*** create ATA-1 * AT * B,   NxM  Matrix Multiplicaton       ***/
        for (l=1;l<=m;l++)  {
                for (k=1;k<=w;k++) {
                        a[k][l]=0.0;
                        for (j=1;j<=n;j++)
                                a[k][l] += (u[k][j]*b[l][j]);
                }
        }

        if(DEBUG==1)printf("\nAnswer From Gaussfj \n");
        for (k=1;k<=n;k++)  {
                for (l=1;l<=w;l++) if(DEBUG==1)printf("%12.6f",a[k][l]);
                if(DEBUG==1)printf("\n");
        }

free_dmatrix(t,1,NP,1,NP);
free_dmatrix(x,1,NP,1,NP);
free_dmatrix(u,1,NP,1,NP);
free_dmatrix(ai,1,NP,1,NP);
free_dmatrix(at,1,NP,1,NP);
}
/*****  End least_square()  ********/
```

```
                    TSBACKM.C    4-21-92
MUST BE LINKED WITH TC.C AND TMB.C FOR ENTIRE PROGRAM TO WORK


4-21 backtrack guts, multiple boxes, subpixel centroid calb
4-21 changed box tmpx,y parameters so box does not jump when resized
    MOVING BOX ENABLED

                    By Andrew Jansky
```

```
#define MOVEX 10    /** distance to move box                **/
#define MOVEY 10    /** distance to move box                **/
#define MARGY 20    /** margin              **/
#define MARGX 20    /** margin              **/

#define  MAX_BOXES 50
#include "ts.h"
#include "regop.h"
void multi_track(void);
void set_multi_box(int box);
void multi_centroid(fentry *iarray,int bstat,int n);
void sub_pixel_location(void);
void adi_modify(void);
struct box_format{
            int x;      /*** beginning x pixel location    ***/
            int y;      /***          y                   ***/
            int dx;     /*** x size of box                 ***/
            int dy;     /*** y                             ***/
            };

struct centroid{
            int x;
            int y;
            };
struct d_centroid{
            double x;
            double y;
            };

struct centroid   roid[MAX_BOXES];
struct d_centroid   d_roid[MAX_BOXES];
struct box_format mbox[MAX_BOXES];

static char *str[]={ "0","1","2","3","4","5","6","7","8","9","10","11","12",
                  "13","14","15","16","17","18","19","20","21","22","23","24","25","26",
                  "27","28","29","30","31","32","33","34","35","36","37","38","39","40"};


void multi_track(void)
{
int fn;                         /*** fn feature number            ***/
int c,n,flag;
int bstat,row,col;
int number;   /**** number of object to track   **/
int bn;       /**** box number     **/
int ii=2;
```

```
struct T {
            int h;
            int m;
            int s;
            int f;
            }T;

char vlan_resp[15];

DWORD fcount;
DWORD delta;
fentry *farray;

                                                                                        /*** Allocate 10,000point
array  ***/
            if((farray=(fentry *)malloc(10001*sizeof(fentry)))==NULL){
                        wprintf("Cant't allocate space for feature data\n");
                        exit(0);
            }
            file_name();
            if(!wopen(3,4,8,70,0,LRED|_LGREY,REDX|_BLACK))error_exit(1);
            add_shadow();
            do{
            wprintf("Enter number of points to track (less than 50):\n ");
            scanf("%d",&number);
            if(number<=50) break;
            }while(1);

            /**** set number of boxes ****/
            for ( c=0; c<number ;c++){
            set_multi_box(c);
            }
            wclose();

            if(!wopen(3,4,20,70,0,LRED|_MAGENTA,BLACK|_MAGENTA))error_exit(1);
            add_shadow();

            if(*tflag==1)                /*** Set fn based on tflag          ***/
                      fn=1;
            if(*tflag==2)
                      fn=2;

vlan_com_open(com2_base);

wprintf("Using Camera Port 0");

select_path(LOOP_BACK);
grab(B1);

   vlan_com_open(com2_base);
   wpgotoxy(13,2);

            vlan_command("SR",vlan_resp);
            while(strcmp(vlan_resp,"LOCAL       ")==0){
                        wprintf ("Please switch VCR to 'remote' mode\n");
                        wprintf ("with control selector, then hit any key\n");
                        getch();
                        vlan_command("SR",vlan_resp);
                        }


   if(*tflag==1)
```

```
            adi_linlut(ADI,INPUT|RED|GREEN|BLUE,0);
   if(*tflag==2)
            invlut(ADI,RED|GREEN|BLUE,0);

   wpgotoxy(13,2);
   wprintf(" press Left mouse button to toggle thresholded vs normal\n");
   wprintf(" press Right mouse button to display pixels found (in green)\n");
   wprintf(" press Both mouse buttons together to reposition box\n");
   wprintf("              PRESS ANY KEY TO QUIT");

/************************ MAIN AQUISITION LOOP    ************************/

bstat=0;                /**** Initalizing various variables     ****/
flag=0;
delta_ms();
         hf_initfet();                /*** init F.E.T              ***/
         hf_mode(FEATURE);              /*** Feature mode             ***/
         hf_period(FRAME); /**** needs to be in the code **/

                 if(*tflag==1)
                          hf_spanfet(1,SPAN1,256);
                 if(*tflag==2)
                          hf_spanfet(2,0,SPAN2);

         fb_clf(B2,0);
alu_dbanksel(0,0);
alu_initluts();

   /* if( !(FPTR = fopen("temp.dat","w")))
            error_exit(4); */
         /*** NO HEADERS FOR MY EXPERIMENT ***/
 /* fprintf(FPTR,"Date:  %s\n",sysdate(5));
         fprintf(FPTR,"Time:  %s\n",systime(0));
         fprintf(FPTR,"BM1 x    BM1 y\n");
         fprintf(FPTR,"%d       %d\n",bm[0],bm[1]);
         fprintf(FPTR,"BM2 x    BM2 y\n");
         fprintf(FPTR,"%d       %d\n",bm2[0],bm2[1]);
         fprintf(FPTR," x        y         Frame  \n");
         fprintf(FPTR,"=====================================\n");
                          roid[0].x=0.0;
                          roid[0].y=0.0;
                          roid[1].x=0.0;
                          roid[1].y=0.0; */


         grab(ALOW);
         while(kbhit()==0){
                          msstatus(&bstat,&row,&col);
                          if(bstat==3){
                            if(!wopen(3,4,8,50,0,LGREEN|_LGREY,LGREEN|_LGREY))error_exit(1);
                                  add_shadow();
                                   vlan_command("SH0",vlan_resp);
                                  do{
                                           wprintf("Enter number of box to move:\n ");
                                           scanf("%d",&bn);
                                           bn--;
                                           if(bn<=(number-1)) break;
                                           wprintf("A number LESS than the total stupid!!\n");
                                           }while(1);

                                  set_multi_box(bn);
                                  wclose();
```

```
                                }

                        if(bstat==1){
                                if(flag==0){
                                                                                        /*** threshold luts, either case
***/
                                        if(*tflag==1)
                                                threshold(ADI,RED|GREEN|BLUE,0,256,SPAN1);
                                        if(*tflag==2)
                                                threshold(ADI,RED|GREEN|BLUE,0,SPAN2,0);
                                        flag++;
                                        }

                                else{           /*** resets luts                ***/

                                        if(*tflag==1)
                                                adi_linlut(ADI,INPUT|RED|GREEN|BLUE,0);
                                        if(*tflag==2)
                                                invlut(ADI,RED|GREEN|BLUE,0);
                                        flag=0;
                                        }
                                }

                /*  adi_waitvb();   akes it a constant time interval    ****/
                /*  adi_passvb();   ut does slow it down slightly       ****/

                fb_access(NONE,SCAN,SCAN);    /*** enables FRAMEA to be seen ***/
                vlan_command("JR",vlan_resp);
                        delay_(20);
                        vlan_command("LR",vlan_resp);
                        select_path(ALOW);
                        fb_clf(FRAMEA,0);

                        average(VDI,2,4);

                        hf_aq(SINGLE);

                        hf_ssel(VDA);

                        hf_waitaq();

        /*   fcount = h_fcount(0,VDI,FRAME,fn);  gets number of features*/

                         fcount=1;
                        if (fcount > 0L){

                                n=hf_getfarray (fn, farray, 0);

                                multi_centroid(farray, bstat,number);

                                hf_clear(1024);          /** Clear next HF buffer        ***/
                                                                                                /** 1024 does not
give shadow  ***/
                                                                                                /**    VERY
IMPORTANT            ***/
                                delta=delta_ms();         /*   gets time since last call ***/

                                wpgotoxy(ii++,2);
                        strchg(vlan_resp,'.',';');  /*changes . to ; so qpro imports ok*/
                        wprintf("pixels %d  \tdt = %ld   \tFrame = %s   ",n,delta,vlan_resp);
                        if(ii==8) ii=2;
                        wpgotoxy(10,2);
```

```
                                        wprintf("X1 = %d \tY1 =  %d  X2 = %d \tY2 =  %d  ",
                                        roid[0].x,roid[0].y,roid[1].x,roid[1].y);

fprintf(FPTR,"%13s\t\t%7.2f%7.2f\n",vlan_resp,d_roid[0].x,d_roid[0].y);

/*  gives the location of 5 points to the file
fprintf(FPTR,"%13s\t\t%7.2f%7.2f\t%7.2f%7.2f\t%7.2f%7.2f\t%7.2f%7.2f\t%7.2f%7.2f\n",
vlan_resp,d_roid[0].x,d_roid[0].y,d_roid[1].x,d_roid[1].y,d_roid[2].x,
d_roid[2].y,d_roid[3].x,d_roid[3].y,d_roid[4].x,d_roid[4].y);*/
                                                }
        }



vlan_command("SH0",vlan_resp);
free(farray);                           /***  free up malloc memory ***/
msgotoxy(9,24);
wclose();                               /***  close open window     ***/
fclose(FPTR);
}/***************************  END TRACK  *******************************/

void set_multi_box(int n)
{
int bstat,row,col;
int tmpx, tmpy;
int ct,rt,cc,rc;
int i;

select_path(LOOP_BACK);
wpgotoxy(2,2);
wprintf("Setting box number= %d  ",n+1);

ct=256;
rt=240;

        if(*tflag==1)
                threshold(ADI,RED|GREEN|BLUE,0,256,SPAN1);
        if(*tflag==2)
                threshold(ADI,RED|GREEN|BLUE,0,SPAN2,0);


        do{
        msstatus(&bstat,&row,&col);
        }while((bstat==2)||(bstat==1));   /*** traps against held button  **/

        mbox[n].dx=50;
        mbox[n].dy=50;

        bstat=0;
        while(bstat!=2){
                msstatus(&bstat,&row,&col);
                mscount(&rc,&cc);
                ct=ct+cc;
                rt=rt+rc;
                if((ct)>=512) ct=512;
                if((ct)<=0)   ct=0;
                if((rt)>=480) rt=480;
                if((rt)<=0)   rt=0;

                fb_clf(B2,0);

                mbox[n].x=ct;
                mbox[n].y=rt;
                rectangle(B2,0,mbox[n].x,mbox[n].y,mbox[n].dx,mbox[n].dy,1);
        text(B2,0,mbox[n].x-15,mbox[n].y-25,HORIZONTAL,1,1,"Leftchanges size");
        text(B2,0,mbox[n].x-15,mbox[n].y-12,HORIZONTAL,1,1,"Rightto track");
                tmpx=ct;
                tmpy=rt;
                for(i=0;i<n;i++){
                rectangle(B2,0,mbox[i].x,mbox[i].y,mbox[i].dx,mbox[i].dy,2);
                }
                if(bstat==1){
                                msgotoxy(tmpx,tmpy);
                                bstat=1;
                                while(bstat==1){
                                        msstatus(&bstat,&row,&col);
                                        mscount(&rc,&cc);
                                        ct=ct+cc;
                                        rt=rt+rc;
                                        if((ct)>=512) ct=512;
                                        if((ct)<=0)   ct=0;
                                        if((rt)>=480) rt=480;
                                        if((rt)<=0)   rt=0;

                                        mbox[n].dx=ct;
                                        mbox[n].dy=rt;
                                        fb_clf(B2,0);

                                        for(i=0;i<n;i++){
                                                rectangle(B2,0,mbox[i].x,mbox[i].y,

mbox[i].dx,mbox[i].dy,2);
                                                }

                                        rectangle(B2,0,mbox[n].x,mbox[n].y,mbox[n].dx,

mbox[n].dy,1);
                                        }
                                msgotoxy(tmpx,tmpy);
                                ct=tmpx;
                                rt=tmpy;

                                }
        }
fb_clf(B2,0);
msgotoxy(9,24);
}/****************** End set_multi_box()**********************************/

void multi_centroid(fentry *iarray,int bstat,int n)
{
int i;
struct total{           /**** centroid structure   ****/
        long int xt;
        long int yt;
                        int c;          /***** total pixels making up centroid  ***/
        };
struct  total t[MAX_BOXES];

fentry *fp;

for(i=0;i<n;i++){
t[i].c=0;
t[i].xt=0;
t[i].yt=0;
```

```
}


for (fp = iarray;fp->fn != 0xFF;fp++){



        for(i=0;i<n;i++){
                    if( (fp->x > mbox[i].x) && (fp->x < mbox[i].x+mbox[i].dx) &&
                          ((fp->y) > mbox[i].y) && ((fp->y) <mbox[i].y+mbox[i].dy)){
                           t[i].c++;
                           t[i].xt=t[i].xt+(fp->x);
                           t[i].yt=t[i].yt+(fp->y);
                           /* circle(B2,0,fp->x,fp->y,1,1,1,3);*/
                           }
                          }

        }

fb_clf(B2,0);


for(i=0;i<n;i++){
        if(t[i].c>=1){

        roid[i].x = (t[i].xt / t[i].c);
        roid[i].y = (t[i].yt / t[i].c);
  /* create sub-pixel location array  **/
        d_roid[i].x = ((double)t[i].xt / (double)t[i].c);
        d_roid[i].y = ((double)t[i].yt / (double)t[i].c);


                circle(B2,0,roid[i].x,roid[i].y,4,1,1,1); /** Draw cross-hair   ***/
                line(B2,0,roid[i].x-6,roid[i].y,roid[i].x+6,roid[i].y,1);
                line(B2,0,roid[i].x,roid[i].y-6,roid[i].x,roid[i].y+6,1);


          if(bstat==2)
                  rectangle(B2,0,mbox[i].x,mbox[i].y,mbox[i].dx,mbox[i].dy,1);

          if(i<=39){
                  if(bstat==2)
                    text(B2,0,mbox[i].x,mbox[i].y-13,HORIZONTAL,1,1,str[i+1]);
                  }
        }

        else{  text(B2,0,mbox[i].x,mbox[i].y-23,HORIZONTAL,1,2,"NONE");
                  rectangle(B2,0,mbox[i].x,mbox[i].y,mbox[i].dx,mbox[i].dy,2);

                  text(B2,0,mbox[i].x,mbox[i].y-13,HORIZONTAL,1,2,str[i+1]);
                  /*wprintf("\a");*/
                  }


        }
  /*** move box with old method  ***/
 /* for(i=0;i<n;i++){
        if(roid[i].x<=(mbox[i].x+MARGX))          mbox[i].x-=MOVEX;
        if(roid[i].x>=(mbox[i].x+mbox[i].dx-MARGX)) mbox[i].x+=MOVEX;

        if(roid[i].y<=(mbox[i].y+MARGY))          mbox[i].y-=MOVEY;
        if(roid[i].y>=(mbox[i].y+mbox[i].dy-MARGY)) mbox[i].y+=MOVEY;
```

```
            }  */
/* box mover new method still dont work */
for(i=0;i<n;i++){
  if(roid[i].x<=(mbox[i].x+mbox[i].dx/3))          mbox[i].x-=mbox[i].dx/3;
  if(roid[i].x>=(mbox[i].x+mbox[i].dx-mbox[i].dx/3))mbox[i].x+=mbox[i].dx/3;
  if(roid[i].y<=(mbox[i].y+mbox[i].dy/3))          mbox[i].y-=mbox[i].dy/3;
  if(roid[i].y>=(mbox[i].y+mbox[i].dy-mbox[i].dy/3))mbox[i].y+=mbox[i].dy/3;
  }




}/******* End multi_centroid   ******************/

/*******Cant remember what this is here for but it looks to good to kill***/
/******* Think it is a sub pixle calibration routine     ****/
void sub_pixel_location(void)
{
FILE *CALBFILE;
int bstat,row,col;
int tmpx, tmpy;
int ct,rt,cc,rc;
int i,n;
int dx,dy,x,y;
int fn;
double xbarf,ybarf;
int xbar,ybar;
int point=0;

struct total{            /**** centroid structure   ****/
            long int xt;
            long int yt;
                            int c;         /***** total pixels making up centroid  ***/
            };
struct  total tot;

fentry *fp;
fentry *array;

  if( !(CALBFILE = fopen("calibrat.prn","w")))
            error_exit(4);

         /***  Allocate 10,000 point array  ***/
         if((array=(fentry *)malloc(10001*sizeof(fentry)))==NULL){
                  wprintf("Cant't allocate space for feature data\n");
                  exit(0);
         }

         if(*tflag==1)                /*** Set fn based on tflag         ***/
                  fn=1;
         if(*tflag==2)
                  fn=2;

KLUDGE2:
         if(!wopen(3,4,8,70,0,LRED|_LGREY,REDX|_BLACK))error_exit(1);
         add_shadow();

         select_path(LOOP_BACK);

         ct=256;
         rt=240;
```

```
do{
msstatus(&bstat,&row,&col);
}while((bstat==2)||(bstat==1));    /*** traps against held button   **/

if(*tflag==1)
          threshold(ADI,RED|GREEN|BLUE,0,256,SPAN1);
if(*tflag==2)
          threshold(ADI,RED|GREEN|BLUE,0,SPAN2,0);


dx=50;
dy=50;

bstat=0;
while(bstat!=2){
          msstatus(&bstat,&row,&col);
          mscount(&rc,&cc);
          ct=ct+cc;
          rt=rt+rc;
          if((ct)>=512) ct=512;
          if((ct)<=0)   ct=0;
          if((rt)>=480) rt=480;
          if((rt)<=0)   rt=0;

          fb_clf(B2,0);

          x=ct;
          y=rt;
          rectangle(B2,0,x,y,dx,dy,1);
          text(B2,0,x-15,y-38,HORIZONTAL,1,1,"Bothchanges gain");
          text(B2,0,x-15,y-25,HORIZONTAL,1,1,"Leftchanges size");
          text(B2,0,x-15,y-12,HORIZONTAL,1,1,"Rightto Process");
          tmpx=ct;
          tmpy=rt;
          if(bstat==1){
                              msgotoxy(tmpx,tmpy);
                              bstat=1;
                              while(bstat==1){
                                        msstatus(&bstat,&row,&col);
                                        mscount(&rc,&cc);
                                        ct=ct+cc;
                                        rt=rt+rc;
                                        if((ct)>=512) ct=512;
                                        if((ct)<=0)   ct=0;
                                        if((rt)>=480) rt=480;
                                        if((rt)<=0)   rt=0;

                                        dx=ct;
                                        dy=rt;
                                        fb_clf(B2,0);


                                        rectangle(B2,0,x,y,dx,dy,1);
                                        }
                              msgotoxy(tmpx,tmpy);
                              ct=tmpx;
                              rt=tmpy;
                              }
           if(bstat==3){
           adi_modify();
                    do{
                    msstatus(&bstat,&row,&col);
```

```
                        }while((bstat==2)||(bstat==1));/** traps against held button**/
                }
        }
/***** centroiding part start *******/
        hf_initfet();                   /*** init F.E.T           ***/
        hf_mode(FEATURE);               /*** Feature mode         ***/
        if(*tflag==1)
                hf_spanfet(1,SPAN1,256);
        if(*tflag==2)
                hf_spanfet(2,0,SPAN2);

        grab(ALOW);
        fb_access(NONE,SCAN,SCAN);   /*** enables FRAMEA to be seen ***/
        select_path(ALOW);
        fb_clf(FRAMEA,0);
        average(VDI,2,4);
        hf_aq(SINGLE);
        hf_ssel(VDA);
        hf_clear(1024);
        hf_waitaq();
        n=hf_getfarray (fn, array, 0);


        tot.c=0;
        tot.xt=0;
        tot.yt=0;


for (fp = array; fp->fn != 0xFF; fp++){
                        if( (fp->x > x) && (fp->x < x+dx) &&
                                (fp->y > y) && (fp->y < y+dy)){
                                tot.c++;
                                tot.xt=tot.xt+(fp->x);
                                tot.yt=tot.yt+(fp->y);
                                }
        }

if(tot.c>=1){
        xbar = (tot.xt / tot.c);
        ybar = (tot.yt / tot.c);
        xbarf = ((double)tot.xt / (double)tot.c);
        ybarf = ((double)tot.yt / (double)tot.c);
        fb_clf(B2,0);
        circle(B2,0,xbar+18,ybar,4,1,1,1); /** Draw cross-hair   ***/
        text(B2,0,x-15,y-25,HORIZONTAL,1,1,"Leftto end");
        text(B2,0,x-15,y-12,HORIZONTAL,1,1,"Rightto continue");
        line(B2,0,xbar-6+18,ybar,xbar+6+18,ybar,1);
        line(B2,0,xbar+18,ybar-6,xbar+18,ybar+6,1);/*18 fudge factor for disp*/
        }
wprintf("Hit left button to end\n");
wprintf("Hit right button to continue\n");
wprintf("Calibration point Sub-pixel location\n");
wprintf(" xbar = %f   ybar= %f  count = %i",xbarf,ybarf,tot.c);
fprintf(CALBFILE,"%i   %f   %f   %i\n",point++,xbarf,ybarf,tot.c);


/***** centroiding part end   *******/
select_path(ALOW);
fb_access(NONE,SCAN,SCAN);
fb_setpan(FRAMEB,18);
```

```
bstat=0;
 while(bstat!=1){
             msstatus(&bstat,&row,&col);
             if(bstat==2){
               wclose();
               goto KLUDGE2;
               }
             }


fb_clf(B2,0);
   if(*tflag==1)               /** cleaning up back to normal  **/
                adi_linlut(ADI,INPUT|RED|GREEN|BLUE,0);
   if(*tflag==2)
                invlut(ADI,RED|GREEN|BLUE,0);

select_path(ALOW);
grab(ALOW);

fclose(CALBFILE);
free(array);
wclose();


msgotoxy(9,24);

}/*********************endsub-pixel location  ****************************/
```

```
#include "ts.h"
#include "regop.h"
void track_automatic(void);

void track_automatic(void)
{
int fn;                          /*** fn feature number          ***/
int n,flag;
int bstat,row,col;
int ii=2;
int endflag=0;
static int thr;
static int passes;
struct T {
            int h;
            int m;
            int s;
            int f;
            }T;

char vlan_resp[15];
char vlan_resp2[15];
char vlan_resp3[15];
char vlan_resp4[15];
DWORD fcount;
DWORD delta;
fentry *farray;


                                                                         /*** Allocate 10,000point
array  ***/
            if((farray=(fentry *)malloc(10001*sizeof(fentry)))==NULL){
                        wprintf("Cant't allocate space for feature data\n");
                        exit(0);
            }

            if(!wopen(3,4,20,70,0,LRED|_MAGENTA,BLACK|_MAGENTA))error_exit(1);
            add_shadow();


            if(*tflag==1)               /*** Set fn based on tflag        ***/
                        fn=1;
            if(*tflag==2)
                        fn=2;


select_path(LOOP_BACK);
grab(B1);



   if(*tflag==1)
               adi_linlut(ADI,INPUT|RED|GREEN|BLUE,0);
   if(*tflag==2)
               invlut(ADI,RED|GREEN|BLUE,0);


/* if(rstop==0){
            fprintf(one,"\nCalibration coefficient\n");
            fprintf(one,"%f\n",fact);
            fprintf(one,"BM1 x     BM1 y\n");
            fprintf(one,"%d        %d\n",bm[0],bm[1]);
```

```
        fprintf(one,"BM2 x     BM2 y\n");
        fprintf(one,"%d        %d\n",bm2[0],bm2[1]);*/
        fprintf(one," Pass %i \n",passes);
        fprintf(one," x         y         Frame   \n");
        fprintf(one,"=====================================\n");
  /* } */
/************************ MAIN AQUISITION LOOP   ************************/

bstat=0;                /**** Initalizing various variables    ****/
flag=0;
delta_ms();
        hf_initfet();               /*** init F.E.T              ***/
        hf_mode(FEATURE);              /*** Feature mode           ***/
        hf_aq(CONTINUOUS);             /*** 2 buffers alow continuous  ***/

                if(*tflag==1)
                        hf_spanfet(1,SPAN1,256);
                if(*tflag==2)
                        hf_spanfet(2,0,SPAN2);

        if(passes==0){
                vlan_com_open(com2_base);
                delay_(30);
                bstat=3;
                boxtype=3;
                set_ann();
                vlan_command("LR",startframe);
                thrstart=thr;
                passes++;
                }
        else{
                thr=thrstart;
                do{
                  vlan_command("LR",vlan_resp);
                  wprintf ("At frame  %s, starting at %s\n",vlan_resp,startframe);
                  if(kbhit()!=0) break;          /*** outlet for saftey  **/
                  }while(strcmp(vlan_resp,startframe)!=0);
                }
        hf_mode(FEATURE);
        hf_ssel(VDI);
        hf_period(FRAME);
        hf_clear(ALL);
        hf_aq(CONTINUOUS);
        strcpy(vlan_resp2,"00:00:00:00");

        do{
                  vlan_command("LR",vlan_resp);
                 do{
                        vlan_command("LR",vlan_resp);
                        if(strcmp(vlan_resp,endframe)==0)endflag=1;
                           /* T.h=atoi(strleft(vlan_resp,2));
                                T.m=atoi(strmid(vlan_resp,4,2));
                                T.s=atoi(strmid(vlan_resp,7,2)); */
                        T.f=atoi(strmid(vlan_resp,10,2));
                                if( (T.f==0)||(T.f==14)) break;
                        }while(1);



                                adi_waitvb();
                                adi_passvb();
                                /* write_reg(hf[CAQEN],0);
                                        write_reg(hf[ACQ],1);
                                        while(read_reg(hf[ACQ]));*/

                               n=hf_getfarray (fn, farray, 0);
                               vlan_command("LR",vlan_resp2);

                               /*  write_reg(hf[CMODE],3);
                                        write_reg(hf[CLR],1);
                                        while(read_reg(hf[CLR]));*/

                        /*  vlan_command("LR",vlan_resp3);*/

                               showfeat(farray, bstat);  /**** call feature display   ***/

                        /*  vlan_command("LR",vlan_resp4);*/

                               hf_clear(1024);         /** Clear next HF buffer       ***/
                                                                                 /** 1024 does not
give shadow  ***/
                                                                                 /**    VERY
IMPORTANT            ***/
          /*  wprintf("Frame=%s    %s      %s\n",vlan_resp,vlan_resp2,vlan_resp3);*/
                   wprintf("Frame=%s    %s      \n",vlan_resp,vlan_resp2);
                 if(endflag==1) break;
                 if(kbhit()!=0) break;

          /*  fprintf(one,"%3d%5d%13s%13s%13s%13s\n",cen[0],cen[1],vlan_resp,vlan_resp2,vlan_resp3,vlan_resp4);*/
                   fprintf(one,"%3d%5d%13s%13s\n",cen[0],cen[1],vlan_resp,vlan_resp2);

         }while(1);



fb_clf(B2,0);
if(rstop==0) fprintf(one,"-1          -1                -1");

else       fprintf(one,"%4d%10%d20%d\n",rstop,rstop,rstop);

vlan_command("LR",endframe);
free(farray);                      /***  free up malloc memory ***/
msgotoxy(9,24);
wclose();                          /***  close open window     ***/
}
/***************************  END TRACK  ******************************/
/*****          TC.C      ******/
#include "Ts.h"
int bcolor[]={32,0,48,112};
/* error message table */
char *error_text[]= {
         NULL,  /* errnum =  0, no error        */
         "windowing error",              /* errnum == 1, windowing error */
         "No mouse or driver",           /* errnum==2, no mouse          */
         "Memory allocation error",
         "Problems opening file, probably no Ram disk or Track.cfg is messed up"

};


/*-------------------------------------------------------------------------*/
/* this function initializes  video, mouse, keyboard systems */
 void initialize(void)
```

```
{
        /* initialize the  video system and save current screen info */
        videoinit();
        readcur(&crow,&ccol);
        if((savescrn=ssave())==NULL) error_exit(3);

        /* if mouse exists, turn on full mouse support */
        if(msinit()) {
                mssupport(MS_FULL);
                msgotoxy(12,49);
                }
        else{
                if(!wopen(9,26,13,55,0,WHITE|_BLUE,BLINK|WHITE|_LGREY))error_exit(1);
                add_shadow();
                wputs("Sorry mouse must be present, check driver");
                delay_(80);
                srestore(savescrn);
                gotoxy_(crow,ccol);
                wclose();
                error_exit(2);
                }

        /* attach [Alt-X] to the confirm_quit() function */
        setonkey(0x2d00,confirm_quit,0);

        /* initialize help system, help key = [F1] */
        whelpdef("TRACK.HLP",0x3b00,YELLOW|_RED,LRED|_RED,WHITE|_RED,REDX|_LGREY
                ,pre_help);

}

/*---------------------------------------------------------------------------*/
/* this function pops open a window and confirms that the user really */
/* wants to quit the demo.  If so, it terminates the demo program.    */

 void confirm_quit(void)
{
        struct _onkey_t *kblist;

        kblist=chgonkey(NULL); /* hide any existing hot keys */
        if(_mouse&MS_CURS) mshidecur();
        if(!wopen(9,24,13,55,0,WHITE|_BLUE,BLINK|WHITE|_LGREY))error_exit(1);
        add_shadow();
        wputs("\n Quit, are you sure?\a \033N\b");
        clearkeys();
        showcur();
        if(wgetchf("YN",'N')=='N'){
                wclose();
                hidecur();
                if(_mouse&MS_CURS) msshowcur();
                chgonkey(kblist);       /* restore any hidden hot keys */
                }
                else{
                normal_exit();
                }
}

/*---------------------------------------------------------------------------*/

 void normal_exit(void)
{
        close_default();
```

```
            srestore(savescrn);
            gotoxy_(crow,ccol);
            if(_mouse) mshidecur();
            showcur();
            exit(0);
}
/*-------------------------------------------------------------------------*/
/* this function handles abnormal termination.  If it is passed an  */
/* error code of 1, then it is a windowing system error.  Otherwise */
/* the error message is looked up in the error message table.       */

 void error_exit(int errnum)
{
            if(errnum) {
                        printf("\n%s\n",(errnum==1)?werrmsg():error_text[errnum]);
            exit(errnum);
            }
}

/*-------------------------------------------------------------------------*/
/* this function will add a shadow to the active window */

 void add_shadow(void)
{
            wshadow(LGREY|_BLACK);
}

/*-------------------------------------------------------------------------*/

 void open_back_wind(void)
{
            register int i;
            int j;
            int hunds;
            union REGS regs;
            regs.h.ah=0x2c;                         /* DOS Get System Time */
            int86(0x21,&regs,&regs);
            j=regs.h.dl;
            j=(int)(j/25);
            if(!wopen(0,0,23,79,5,0,(int)bcolor[j]))error_exit(1);

/*for(i=1;i<59;i++) wprintf("\033F%c  Wave Research                    ",i);*/

}

/*-------------------------------------------------------------------------*/

 void open_stat_wind(void)
{
            wfillch('\260');
            wopen(24,0,24,79,5,0,LGREY|_RED);
            wprints( 0, 1,LGREY|_RED,"[F1]=Help");
            wcenters(0   ,LGREY|_RED,"Select operation.");
            wrjusts( 0,78,LGREY|_RED,"[Alt-X]=Quit any time");
            wfillch(' ');
}

/*-------------------------------------------------------------------------*/

 void open_titl_wind(void)
 {
if(!wopen(1,12,6,67,0,LRED|_MAGENTA,LRED|_MAGENTA))error_exit(1);
```

```
add_shadow();
wcenters(0,WHITE|_MAGENTA,"O. H. Hinsdale Wave Resarch Lab");
wcenters(1,WHITE|_MAGENTA,"Object Tracking and Image Processing Workstation");
wcenters(2,WHITE|_MAGENTA,"by Andrew Jansky   1990-91    ");
wcenters(3,WHITE|_MAGENTA,VERSION);

}

/*--------------------------------------------------------------------------*/

 void pre_menu1(void)
{
add_shadow();
hidecur();
}

/*--------------------------------------------------------------------------*/

 void do_nothing(void)
{
if(!wopen(15,15,20,60,4,LRED|_MAGENTA,LMAGENTA|_MAGENTA))error_exit(1);
add_shadow();
wcenters(1,WHITE|_MAGENTA,"This routine is under development");
wcenters(2,BLINK|WHITE|_MAGENTA,"SORRY");
delay_(44);
wclose();
}

/*--------------------------------------------------------------------------*/

 void pre_help(void)
{
        add_shadow();
        setonkey(0x2d00,confirm_quit,0);
}

/*--------------------------------------------------------------------------*/
/* this function updates the on-screen clock in the pull-down demo. */

 void update_clock(void)
{
  prints(0,72,LGREEN,systime(2));
}
/*--------------------------------------------------------------------------*/

/* this function is used by the pull-down demo. */
/* It is used to execute a DOS command.         */

 void execute(void)
{
        static int busy=NO;
        char command[61];
        int *scrn;

        if(busy) return; else busy=YES;     /* avoid recursion */
        showcur();
        if(_mouse) mshidecur();
        if(!wopen(8,7,10,70,0,LMAGENTA|_RED,LMAGENTA|_RED))error_exit(1);
        wtitle("[ Execute DOS Command ]",TCENTER,LMAGENTA|_RED);
        winpbeg(BLUEX|_LGREY,BLUEX|_LGREY);
        winpdef(0,1,command,"****************************************"
                "********************",0,0,NULL,0);
```

```
          if(!winpread()) {
                    if((scrn=ssave())==NULL) error_exit(3);
                    cclrscrn(LGREY|_BLACK);
                    system(command);
                    printf("\nPress any key to continue....");
                    waitkey();
                    srestore(scrn);
          }
          hidecur();
          if(_mouse) msshowcur();
          wclose();
          busy=NO;
}

/*---------------------------------------------------------------------------*/
```

```
subcapt.c    4-25-92
        1-7  captures calibration points from screen to file using
             point tag map and other goodies
        1-8  got rid of * usage in get_position, problems with pointer
        4-16 added subpixel centroid calculation for points
        4-25 captures frame points using subpixel method and prompts for
             tag number..
      **5-8  hf_period(FRAME) acquires full frame instead of field!!!!!
        OPENS   TARGETS.DAT    read
                CALOCATE.PRN   write
        10-20changed file headers and cleaned up the text output

                        By Andrew Jansky
```

```
#define VERSION        "Subcapt Version 3.0"
#define TARGET_INFO    "targets.dat"    /** target ball location file      */
#define WORLD_CAMERA   "calocate.prn"   /** calibration location and world */


#define MAX_ARRAY 133
#define SPAN1 254   /** Low cut of feature 1  high cut is 256    **/
#define SPAN2 2     /** High cut of feature 2 low cut is 0       **/

#include "cxldef.h"
#include "cxlkey.h"
#include "cxlmou.h"
#include "cxlstr.h"
#include "cxlvida.h"
#include "cxlwin.h"
#include <conio.h>
#include <ctype.h>
#include <dos.h>
#include <io.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <itex150.h>
#include "regop.h"


static void adi_modify(void);
static void add_shadow(void);
static void error_exit    (int errnum);
static void initialize(void);
static int *savescrn,crow,ccol;              /*** screen variables    ****/
static char *error_text[]= {
        NULL,  /* errnum =  0, no error          */
        "windowing error",              /* errnum == 1, windowing error */
        "No mouse or driver",           /* errnum==2, no mouse          */
        "Memory allocation error",
        "Problems opening file, probably no Ram disk"    };

static char *str[]={ "0","1","2","3","4","5","6","7","8","9","10",
                     "11","12","13","14","15","16","17","18","19","20",
                     "21","22","23","24","25","26","27","28","29","30",
                     "31","32","33","34","35","36","37","38","39","40",
                    "41","42","43","44","45","46","47","48","49","50",
                    "51","52","53","54","55","56","57","58","59","60",
                    "61","62","63","64","65","66","67","68","69","70",
                    "71","72","73","74","75","76","77","78","79","80",
                    "81","82","83","84","85","86","87","88","89","90",
                    "91","92","93","94","95","96","97","98","99","100",
                    "101","102","103","104","105","106","107","108","109","110",
                    "111","112","113","114","115","116","117","118","119","120",
                    "121","122","123","124","125","126","127","128","129","130",
                    "131","132"
                    };

struct point{
        double n;     /* point number  */
        double d;  /* point description, 0 not used, 1 used  */
        double x;  /*world x*/
        double y;  /*world y*/
        double z;  /*world z*/
        double sxf; /*screen x float */
        double syf; /*screen y float */
        int sx;     /*screen x*/
        int sy;     /*screen y*/
        };

struct point pt[MAX_ARRAY];
int choice;
int choices[MAX_ARRAY];
char tmp[130];
int bstat,row,col,rc,cc,bcount,ct,rt;
int tmpx, tmpy;
FILE *fil0;
FILE *fil1;
char tmpbuf[128];

int *tflag;
struct total{          /**** centroid structure   ****/
          long int xt;
          long int yt;
                         int c;        /***** total pixels making up centroid  ***/
          };
struct  total tot;


main()
{
int key;
int ans;
int count=0;
int t=0;

int num1=0;
int cnt=0;
int num;

int c,r,i;
int n;
int fn;
fentry *fp;
fentry *array;
int dx,dy,x,y;
double xbarf,ybarf;
int xbar,ybar;
int point_num=0;
*tflag=1;       /* sets tracking flag to normal threshold, 0 for inverse */

        if( !(fil0 = fopen(TARGET_INFO,"r"))){
                printf( "Could not open %s \n",TARGET_INFO);
                exit(0);
                }

        if( !(fil1 = fopen(WORLD_CAMERA,"w"))){
                printf( "Could not open %s \n",WORLD_CAMERA);
                exit(0);
                }
        fprintf(fil1," %s \n",WORLD_CAMERA);  /*  print file name at header  */
        _strtime( tmpbuf );                   /*  get dos time         */
        fprintf(fil1," %s \n", tmpbuf);       /*  print dos time       */
        _strdate( tmpbuf );
        fprintf(fil1," %s \n\n\n", tmpbuf);   /*  print dos date       */
        fprintf(fil1,
        "This file is written by subcapt.exe and used by calb.exe, back.exe \n");
        fprintf(fil1,
        "and showcalb.exe.  It contains point tag, screen centroid, pixel\n");
        fprintf(fil1,
        "count and world point location\n\n");

        fprintf(fil1,
        "Point  X      Y      Pixel  X      Y      Z\n");
        fprintf(fil1,
        "Tag    cent   cent   Total  w      w      w\n");
        fprintf(fil1,
"==========================================================
==\n");

initialize();

load_cfg("150.cfg");
initsys();


adi_initluts();                 /**** Enable overlay on B2****/
adi_hbanksel(1);                  /**** Lut bank 1 is red   ****/
adi_hgroupsel(RED);
adi_clearlut(255);
adi_hgroupsel(GREEN|BLUE);
adi_clearlut(0);

adi_hbanksel(2);                  /**** Lut bank 2 is green ****/
adi_hgroupsel(GREEN);
adi_clearlut(255);
adi_hgroupsel(RED|BLUE);
adi_clearlut(0);

adi_hbanksel(3);                  /**** Lut bank 2 is green ****/
adi_hgroupsel(BLUE);
adi_clearlut(255);
adi_hgroupsel(RED|GREEN);
adi_clearlut(0);

adi_hbanksel(4);                  /**** Lut bank 2 is green ****/
adi_hgroupsel(BLUE|GREEN|BLUE);
adi_clearlut(255);
adi_hgroupsel(BLUE);
```

```
adi_clearlut(250);


adi_lutmode(DYNAMIC);

            if(!wopen(3,4,21,75,0,LRED|_LGREY,REDX|_BLACK))error_exit(1);
            add_shadow();
            wtitle(VERSION,TCENTER,LRED|_LGREY);
adi_camera(0);

/*
wpgotoxy(1,2);
wprintf("Hit y to adjust gain, any other key to proceed");
select_path(LOOP_BACK);
if(getch()=='y')
adi_modify();
*/

select_path(LOOP_BACK);


delay_(20);

bstat=0;

/*** read in world coordinates from file ****/
for(i=0;i<5;i++){               /** clear file header  **/
   fgets( tmp, 125, fil0 );
   }
while( !feof(fil0))
                        {
                        fscanf(fil0,"%lf ",&pt[cnt].n);
                        fscanf(fil0,"%lf ",&pt[cnt].d);
                        if(pt[cnt].d==1.0){
                                    fscanf(fil0,"%lf " ,&pt[cnt].x);
                                    fscanf(fil0,"%lf " ,&pt[cnt].y);
                                    fscanf(fil0,"%lf ",&pt[cnt].z);
                                    }
                        wpgotoxy(4,2);
                        wprintf("Reading point: %lf %i ",pt[cnt].n,cnt);
                        wpgotoxy(5,2);
                        wprintf("x= %.2lf\ty=%.2lf\tz= %.2lf",pt[cnt].x,pt[cnt].y,pt[cnt].z);

                        if((int)pt[cnt].n!=cnt) printf("\a");
                        cnt++;
                        }
wclear();
wpgotoxy(1,2);
wprintf("Total calibration points entered: %i ",cnt);

            /*** Allocate 10,000 point array ***/
            if((array=(fentry *)malloc(10001*sizeof(fentry)))==NULL){
                        wprintf("Cant't allocate space for feature data\n");
                        exit(0);
            }

            if(*tflag==1)                   /*** Set fn based on tflag         ***/
                        fn=1;
            if(*tflag==2)
                        fn=2;

            dx=50;    /* default box values */
```

```
          dy=50;
          ct=256;
          rt=240;

printf("\a");
cnt=0;

          do{
          msstatus(&bstat,&row,&col);
          }while((bstat==2)||(bstat==1));    /*** traps against held button   **/

          if(*tflag==1)
                    threshold(ADI,RED|GREEN|BLUE,0,256,SPAN1);
          if(*tflag==2)
                    threshold(ADI,RED|GREEN|BLUE,0,SPAN2,0);

wpgotoxy(2,2);
wprintf("ESC to end");
wpgotoxy(3,2);
wprintf("Left button to change box size");
wpgotoxy(4,2);
wprintf("Right button to calculate point centroid and enter into file");
wpgotoxy(5,2);
wprintf("Both buttons to adjust contrast");
wpgotoxy(6,2);
wprintf("Using 0 as a target number aborts file entry");
do{
          BAD_GOTO:  /*** lazy way out bad, bad, bad thing to do    **/

          if(kbhit()!=0)
          key=getch();


                    msstatus(&bstat,&row,&col);
                    mscount(&rc,&cc);
                    ct=ct+cc;
                    rt=rt+rc;
                    if((ct)>=512) ct=512;
                    if((ct)<=0)   ct=0;
                    if((rt)>=480) rt=480;
                    if((rt)<=0)   rt=0;

                    x=ct;
                    y=rt;
                    rectangle(B2,0,x,y,dx,dy,1);
                    text(B2,0,x-15,y-38,HORIZONTAL,1,1,"Bothchanges gain");
                    text(B2,0,x-15,y-25,HORIZONTAL,1,1,"Leftchanges size");
                    text(B2,0,x-15,y-12,HORIZONTAL,1,1,"Rightto Process");
                    tmpx=ct;
                    tmpy=rt;
                    if(bstat==1){
                                        msgotoxy(tmpx,tmpy);
                                        bstat=1;
                                        while(bstat==1){
                                                  msstatus(&bstat,&row,&col);
                                                  mscount(&rc,&cc);
                                                  ct=ct+cc;
                                                  rt=rt+rc;
                                                  if((ct)>=512) ct=512;
                                                  if((ct)<=0)   ct=0;
                                                  if((rt)>=480) rt=480;
                                                  if((rt)<=0)   rt=0;
                                        dx=ct;
                                        dy=rt;
                                        fb_clf(B2,0);
                                        fb_access(FRAMEA,SCAN,NOSCAN);
                                        rectangle(B2,0,x,y,dx,dy,1);
                                        }
                              ct=tmpx;
                              rt=tmpy;
                              msgotoxy(tmpx,tmpy);
                              }
                if(bstat==3){
                adi_modify();
                        do{
                        msstatus(&bstat,&row,&col);
                        }while((bstat==2)||(bstat==1)); /** traps against held button**/
                }


  if(bstat==2) {
/***** centroiding part start *******/
         hf_initfet();                  /*** init F.E.T             ***/
         hf_mode(FEATURE);                  /*** Feature mode            ***/
         if(*tflag==1)
                 hf_spanfet(1,SPAN1,256);
         if(*tflag==2)
                 hf_spanfet(2,0,SPAN2);


         fb_access(FRAMEA,SCAN,NOSCAN);   /*** enables FRAMEA to be seen ***/
         fb_clf(FRAMEA,0);
         average(VDI,2,4);
         hf_aq(SINGLE);
         hf_period(FRAME);  /*** need to be here for full frame **/
         hf_ssel(VDA);
         hf_clear(1024);
         hf_waitaq();

         n=hf_getfarray (fn, array, 0);


         tot.c=0;
         tot.xt=0;
         tot.yt=0;

fb_access(FRAMEB,SCAN,NOSCAN);
for (fp = array; fp->fn != 0xFF; fp++){
                    if( (fp->x > x) && (fp->x < x+dx) &&
                         (fp->y > y) && (fp->y < y+dy)){
                         tot.c++;
                         tot.xt=tot.xt+(fp->x);
                         tot.yt=tot.yt+(fp->y);
                         fb_wpixel(B2,fp->x,fp->y,2);
                         }
         }

if(tot.c>=1){
         xbar = (tot.xt / tot.c);
         ybar = (tot.yt / tot.c);
         xbarf = ((double)tot.xt / (double)tot.c);
         ybarf = ((double)tot.yt / (double)tot.c);

         circle(B2,0,xbar,ybar,4,1,1,1); /** Draw cross-hair   ***/
```

```
            text(B2,0,x-15,y-25,HORIZONTAL,1,1,"Leftto end");
            text(B2,0,x-15,y-12,HORIZONTAL,1,1,"Rightto continue");
            line(B2,0,xbar-6,ybar,xbar+6,ybar,1);
            line(B2,0,xbar,ybar-6,xbar,ybar+6,1);
            }

/***** centroiding part end   *******/

                              ans=0;
                              fb_access(FRAMEA,SCAN,NOSCAN);
                              do{
                                   wpgotoxy(8,2);
                                   wprintf("     ");
                                   wpgotoxy(8,2);
                                   wprintf("Enter valid target number:");
                                   showcur();
                                   ans=scanf("%i",&choice);
                                   if(pt[choice].d==0.0){
                                        ans=0;
                                        printf("\a\a");
                                        }
                                   if(choice==0) goto BAD_GOTO;
                                   hidecur();
                                   }while(ans==0);
                              choices[count]=choice;
                              pt[choice].sxf=xbarf;
                              pt[choice].syf=ybarf;
                              pt[choice].sx=xbar;
                              pt[choice].sy=ybar;
fprintf(fil1,"%i\t%.2lf\t%.2lf\t%i\t%.2lf\t%.2lf\t%.2lf\n",
            choice, xbarf, ybarf, tot.c, pt[choice].x, pt[choice].y, pt[choice].z);

wpgotoxy(8,2);
wprintf("Last target value %i                    ",choice);
wpgotoxy(12,2);
wprintf("Target %i located at:",choice);
wpgotoxy(13,2);
wprintf("xbar = %5.2f\tybar= %5.2f\tPixel count = %i   ",

xbarf,ybarf,tot.c);

wpgotoxy(14,2);
wprintf("Xw   = %5.2lf\tYw = %5.2lf\tZw = %5.2lf "
                        ,pt[choice].x,pt[choice].y,pt[choice].z);

                              printf("\a");
                              delay_(1);
                              cnt++;
                              count++;

                        /* bstat=0;
                             while(bstat!=2){
                             msstatus(&bstat,&row,&col);
                             } */

                        }


            for(i=0;i<count;i++){
            t=choices[i];
            line(B2,0,pt[t].sx-5,pt[t].sy,pt[t].sx+5,pt[t].sy,1);
            line(B2,0,pt[t].sx,pt[t].sy-5,pt[t].sx,pt[t].sy+5,1);
            if(t<131)
            text(B2,0,pt[t].sx,pt[t].sy-18,HORIZONTAL,1,2,str[t]);
            }

fb_clf(B2,0);
fb_access(FRAMEA,SCAN,NOSCAN);


            }while(key!=27);

wclose();
if(*tflag==1)               /** cleaning up back to normal  **/
              adi_linlut(ADI,INPUT|RED|GREEN|BLUE,0);
if(*tflag==2)
              invlut(ADI,RED|GREEN|BLUE,0);

fclose(fil0);
fclose(fil1);


free(array);
}
/**************         end Main *************************************/

/*****************************************************************************/
static void add_shadow(void)
{
            wshadow(LGREY|_BLACK);
}
/*****************************************************************************/
static void error_exit(int errnum)
{
            if(errnum) {
                        printf("\n%s\n",(errnum==1)?werrmsg():error_text[errnum]);
            exit(errnum);
            }
}
/*****************************************************************************/
static void initialize(void)
{
            /* initialize the  video system and save current screen info */

            videoinit();
            readcur(&crow,&ccol);
            if((savescrn=ssave())==NULL) error_exit(3);

            /* if mouse exists, turn on full mouse support */
            if(msinit()) {
                        mssupport(MS_FULL);
                        msgotoxy(12,49);
                        }
            clrscrn();
}


/*****************************adi_modify()*********************************/
/***** Adjusts the gain and offset of adi board using the mouse   *********/
void adi_modify(void)
{

int bstat,row,col,rc,cc,bcount;
int rt,ct;
rt=0;
ct=0;

fb_clf(B2,0);
grab(B1);


            if(!wopen(4,4,12,70,0,LRED|_MAGENTA,LRED|_MAGENTA))error_exit(1);
                        add_shadow();

wprintf("\n");
wprintf("Move mouse to adjust gain (and offset)\n");
wprintf("Press left button when done adjusting\n\n");


bstat=0;
adi_level(100,64);
while(bstat==0) {
            msstatus(&bstat,&row,&col);
            mscount(&rc,&cc);

                        ct=ct+cc;
                        rt=rt+rc;

                        if( (ct) >= 255)  ct=255;
                        if( (ct) <= 0)    ct=0;

                        if( (rt) >= 255)  rt=255;
                        if( (rt) <= 0)    rt=0;

            adi_level(ct,rt);
            wpgotoxy(6,12);
            wprintf("Gain:   %d   Offset:   %d        ",ct,rt);
            }
wclose();
msgotoxy(9,24);
}
/**************************End adi_modify() *******************************/
```