

**REAL-TIME STEREO VISION USING LASER SCANNING  
AND POSITION SENSITIVE PHOTODETECTORS:  
ANALYTICAL AND EXPERIMENTAL RESULTS**

**A Thesis  
by  
HANSPETER SCHAUB**

**Submitted to the Office of Graduate Studies of  
Texas A&M University  
in partial fulfillment of the requirements for the degree of  
MASTER OF SCIENCE**

**August 1994**

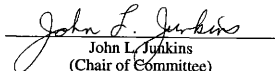
**Major Subject: Aerospace Engineering**

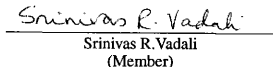
REAL-TIME STEREO VISION USING LASER SCANNING  
AND POSITION SENSITIVE PHOTODETECTORS:  
ANALYTICAL AND EXPERIMENTAL RESULTS

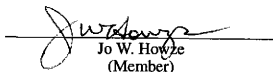
A Thesis  
by  
HANSPETER SCHAUB

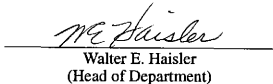
Submitted to the Office of Graduate Studies of  
Texas A&M University  
in partial fulfillment of the requirements for the degree of  
MASTER OF SCIENCE

Approved as to style and content by:

  
John L. Jenkins  
(Chair of Committee)

  
Srinivas R. Vadali  
(Member)

  
Jo W. Howze  
(Member)

  
Walter E. Haisler  
(Head of Department)

August 1994

Major Subject: Aerospace Engineering

## ABSTRACT

Real Time Stereo Vision Using Laser Scanning and Position Sensitive

Photodetectors: Analytical and Experimental Results. (August 1994)

Hanspeter Schaub, B.S., Texas A&M University;

Chair of Advisory Committee: Dr. John L. Jenkins

A three-dimensional scanning system is studied using two one-dimensional position sensitive detectors (PSDs) and a laser light source illuminating the object. This electro-optical concept enables real-time stereo triangulation measurements with a system having only one moving part, the laser scan mirror. The PSD is a silicon photodiode which produces an analog voltage proportional to the tangent of the incidence angle of the incoming light. While performing a scan, the laser light spot traces a profile of the target in the scan plane of the two PSD sensors. The diffuse reflection is focused onto the photosensitive strip of each PSD. The two resulting angles define the two-dimensional coordinates of the laser light spot within the scan plane. The sensors will perform a well-defined relative motion about the object. The profile scan is then transformed into a complete set of three-dimensional coordinates by knowing the inertial position and orientation of the PSD camera system at all times.

Each PSD was calibrated with an accuracy of  $0.005^\circ$  and a field of view of about  $22^\circ$ . The reflective light could be detected up to 9 inches away. Beyond this distance the signal to noise ratio became too small. Because the diffuse light signals are so weak, electronic noise became an issue. Six inches away from the sensors the maximum triangulation error due to electronic noise was less than 0.5mm.

The sensor position and orientation was determined by measuring the incidence angles for several targets whose location is known. Due to the very sensitive nature of this process an iterative technique was used to further fine tune the sensor configuration. The final configuration caused less than 3.5% distortion in the three-dimensional scans.

## ACKNOWLEDGEMENT

*This work was accomplished due to the vision and guidance of Dr. John L. Junkins, my committee chair. His support allowed me to move away from the purely theoretical domain and wet my feet in the practical domain. I thank you for the challenge, since I have learned and grown greatly from it.*

I must also thank Laurie Wittig for her help with the airbearing; Johnny Hurtado for his assistance with the signal acquisition; John Grillo and Conrad Wilson for their help in constructing the sensors; Barry Sims for his help building a target structure; Steve White for doing the photographic documentation; Ivan Brigman for his offered assistance in fixing the project; Mormon Hughes for supplying me with materials; and Texas A&M for the financial support of my graduate studies.

# TABLE OF CONTENTS

	Page
ABSTRACT .....	iii
ACKNOWLEDGEMENT .....	iv
LIST OF FIGURES .....	vii
LIST OF TABLES .....	ix
INTRODUCTION .....	1
HARDWARE DESCRIPTION .....	3
POSITION SENSITIVE DEVICE .....	3
SIGNAL PROCESSING BOARD .....	4
ONE DIMENSIONAL SENSOR UNIT .....	4
THEORY .....	6
CALIBRATION .....	6
CONFIGURATION .....	8
Sensor Unit Position .....	8
Sensor Unit Orientation .....	11
TRIANGULATION .....	12
TRIANGULATION SENSITIVITY .....	16
EXPERIMENTAL RESULTS .....	23
CALIBRATION .....	23
PSD Signal .....	23
Electronic Noise .....	27
Sensor Unit Pointing Accuracy .....	31
Sensor Unit Tracking Accuracy .....	32
CONFIGURATION .....	34
SCANNING RESULTS .....	37
Two-Dimensional Profile Scans .....	37
Three-Dimensional Scans .....	40

**TABLE OF CONTENTS (cont.)**

	Page
CONCLUSIONS AND RECOMMENDATIONS .....	44
REFERENCES .....	45
APPENDICES .....	46
APPENDIX A .....	47
APPENDIX B .....	48
VITA .....	86

## LIST OF FIGURES

	Page
Figure 1: Three-Dimensional Scanning System Layout. ....	1
Figure 2: PSD chip 1L5-SP. ....	3
Figure 3: PSD Electrical Equivalent Circuit. ....	3
Figure 4: Basic Operating Circuit of OT300LS Signal Processing Board. ....	4
Figure 5: One-Dimensional Sensor Unit. ....	5
Figure 6: Illustration of the Incident Light Projection onto the Photo-Sensitive Strip. ..	6
Figure 7: Illustration of SU Origin Determination. ....	8
Figure 8: Illustration of the SU Position Solution Space. ....	9
Figure 9: Illustration of SU Orientation Determination. ....	11
Figure 10: Triangulation of an Arbitrary Point. ....	12
Figure 11: Illustration of Vectors in the Camera and Inertial Reference Frames. ....	14
Figure 12: Camera Flight Path in the x-y Plane. ....	15
Figure 13: x-Coordinate Sensitivities versus $X_1$ , $Z_1$ , $X_2$ , and $Z_2$ . ....	18
Figure 14: Configuration Parameter Variations Due to Errors in Configuration Angle Measurements. ....	19
Figure 15: Illustration of the Configuration Solution Spaces. ....	19
Figure 16: Range of (x,z) Values Due to Configuration Errors. ....	20
Figure 17: PSD Voltage versus Tangent of Airbearing Angle. ....	23
Figure 18: Non-Linear Term of the PSD Voltage versus Airbearing Angle Relationship. ....	24
Figure 19: Comparison of the Calibration Modelling error. ....	24
Figure 20: Variation of Calibration Model at Different Target Distances. ....	25

# LIST OF FIGURES (cont.)

	Page
Figure 21: Steady State Zero PSD Signal with a Target Distance of Five Inches. ....	28
Figure 22: Steady State Non-Zero PSD Signal with a Target Distance of Five Inches. ...	29
Figure 23: Steady State Zero PSD Signal with a Target Distance of Eight Inches. ....	30
Figure 24: Steady State Non-Zero PSD Signal with a Target Distance of Eight Inches. ..	31
Figure 25: Sensor Unit Pointing Accuracy with a Target Distance of Five Inches. ....	32
Figure 26: Sensor Unit Tracking Error with a Target Distance of Five Inches. ....	33
Figure 27: Sensor Unit Tracking Error with a Target Distance of Eight Inches. ....	34
Figure 28: Photograph of the Three-Dimensional Scanning System. ....	35
Figure 29: Schematic of a Configuration Setup. ....	35
Figure 30: SU Angle Variations Due to Electrical Noise. ....	38
Figure 31: Triangulation Variations Due to Electrical Noise. ....	38
Figure 32: Profile Scan of a Straight Ruler. ....	39
Figure 33: Three-Dimensional Scan of Cylindrical Tube. ....	40
Figure 34: Three-Dimensional Scan of a Rectangular Piece of Wood. ....	41
Figure 35: Three-Dimensional Scan of a Small Bottle. ....	42
Figure 36: Profile Scan of a Large Plastic Thread. ....	42
Figure 37: Profile Scan of White and Black Areas of a Ruler. ....	43



**LIST OF TABLES**

	<b>Page</b>
Table 1: Chebyshev Polynomial Coefficients .....	26
Table 2: Comparison of Reconfiguration Targets .....	36
Table 3: Comparison of Sensor Unit Positions .....	37
Table 4: Chebyshev Coefficients for Ruler Approximation .....	39

## INTRODUCTION

Non-intrusive remote sensing capabilities are becoming increasingly important in robotics and experimental analysis. While it is possible to measure such parameters as distance, velocity or temperature, it remains difficult to measure the exact three-dimensional shape of an object in near-real-time and without touching it. This paper presents a three-dimensional scanning system using two one-dimensional position sensitive detectors (PSDs) and a laser light source illuminating the target object. Each PSD measures the angle of the incident laser light. Together they track the diffuse reflection of the laser light as it sweeps the object's surface and performs a line scan.

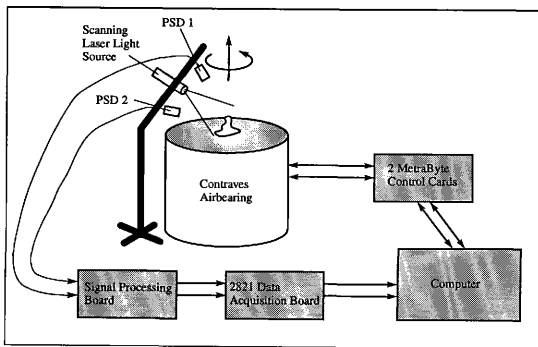


Figure 1: Three-Dimensional Scanning System Layout.

The sensors themselves can move relative to the target to observe the entire object either due to target motion, sensor motion, or both. By knowing the sensor position and orientation at all times, two-dimensional profile scans can be converted into a three-dimensional model. Figure 1 shows the experimental system layout. A circular PSD motion is achieved by rotating the object on

the airbearing while keeping the PSDs steady. This electro-optical concept enables real time stereo triangulation measurements with a system consisting of only one moving part, the laser scanning mirror. Such a non-contact sensing device has many applications in research and manufacturing. A robot equipped with this type of vision system would be able to perceive the shape, position and orientation of objects.<sup>1</sup> In medicine, such systems would be useful to help construct more exact prosthesis or to help plan procedures in reconstructive or cosmetic surgery. Researchers, engineers and artists would be able to obtain exact computer representations of their physical models.

Systems are available today to do the last named task. Cyberware Laboratory Inc. has a system which performs three-dimensional scans using a video system to track a laser light target.<sup>2</sup> A system developed by Intelligent Automation Systems Inc. has a similar setup, but it uses a fan of multiple laser light planes which are observed by a video system. The system proposed here will use analog PSDs to track a single laser light target. The advantages include simpler construction and design, higher resolution and less post-processing.

The purpose of this study is to demonstrate the feasibility and limitations of the proposed three-dimensional scanning system. A prototype system was built and tested to compare to the theoretical predictions. The theoretical and experimental results will stress three categories: (1) calibration of the PSDs and their accuracy; (2) finding the position and orientation (configuration) of the PSDs relative to a coordinate system and the sensitivity thereof; and (3) calculating and analyzing the three-dimensional scans.

## HARDWARE DESCRIPTION

The function of several key hardware components are explained in this section. A complete list of all the hardware components used and their model numbers is found in Appendix A.

### POSITION SENSITIVE DEVICE

A PSD averages the light intensity along a photo-sensitive strip and returns a voltage proportional to the centroid of the energy distribution. These position sensors have an advertised analog resolution of better than one part in a million and have a very small position non-linearity. Figure 2 shows the shape and size of the PSD. The photo-sensitive strip is only 5mm long, making it a very compact chip. Due to its small size, the sensor is very responsive with a rise time of  $0.05\mu\text{s}$ .<sup>3</sup>

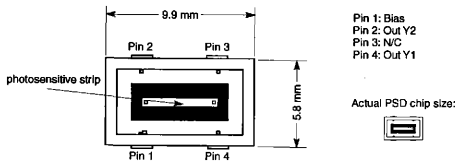


Figure 2: PSD chip 1L5-SP.

Figure 3 shows the equivalent circuit of a PSD. The photo-sensitive strip is divided into two sections, each acting like a solar panel converting light into electrical currents  $Y1$  and  $Y2$ .

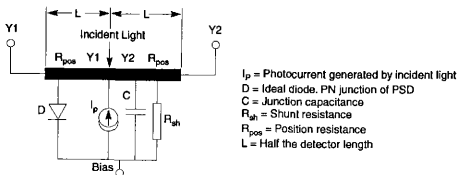


Figure 3: PSD Electrical Equivalent Circuit.

If the incident light is not centered on the strip, a current imbalance occurs. The magnitude and sign of the imbalance determines the location of the mean of the incoming light intensity.

## SIGNAL PROCESSING BOARD

An analog processing board takes the two currents and transforms them into a voltage  $v$  which is proportional to the non-dimensional position  $x/L$  as is shown in (1).

$$v = \frac{x}{L} = \frac{Y2 - Y1}{Y2 + Y1} \quad (1)$$

The basic operating circuit of the processing board is shown in Figure 4. The currents are added, subtracted and divided in an analog manner yielding an analog output proportional to the mean incident light position. This quantity is scaled to a value between -10V and +10V. The processing board has a second output which returns the sum of  $Y1$  and  $Y2$ . This sum is effectively a measure of the signal strength of the incident light.

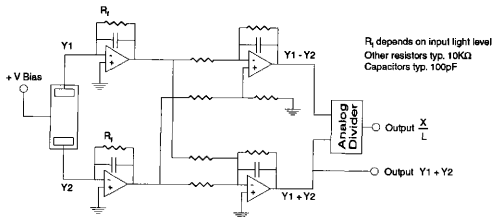


Figure 4: Basic Operating Circuit of OT300LS Signal Processing Board.

## ONE DIMENSIONAL SENSOR UNIT

The PSD along with the focusing lens and the casing form a one-dimensional sensor unit (SU). A cross-section of such a unit is shown to scale in Figure 5. The lens mount can be unscrewed to access and replace the PSD chip. The sensor itself is rigidly mounted inside a 30 mm barrel, 9mm

away from the lens. A lens with a diameter and a focal length of 9mm was chosen. It balances well the field of view (FOV) and the light through-put problem. The lens itself is coated to reduce chromatic aberration. The FOV is about  $22^\circ$  with the photo-sensitive strip length of 5 mm. The SU has three cables going to it. Two are for the currents  $I_1$  and  $I_2$ , the third is for a bias voltage applied to the chip.

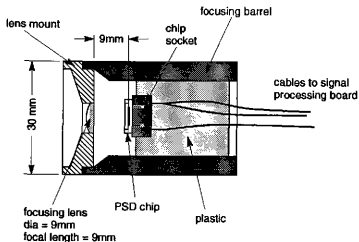


Figure 5: One-Dimensional Sensor Unit.

This sensor was designed in this study and assembled from a combination of off-the-shelf and custom-made components. In particular the lens, lens mount, focusing barrel and PSD chip were purchased off-the-shelf to suit our design. The other components were custom made.

## T H E O R Y

### CALIBRATION

The calibration process establishes the exact relationship between the processing board voltage  $v$  and the incident angle  $\theta$ . Figure 6 illustrates how the incident light gets projected onto the photo-sensitive strip.

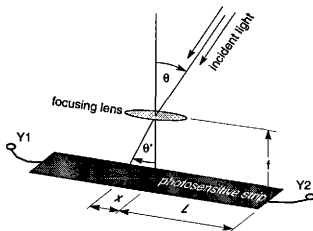


Figure 6: Illustration of the Incident Light Projection onto the Photo-Sensitive Strip.

Studying Figure 6 the relationship between  $x$  and  $\theta'$  is found as:

$$\frac{x}{f} = \tan \theta' \quad (2)$$

After dividing (2) by  $L$  and making use of equation (1) the following relationship between  $v$  and  $\theta$  can be found.

$$v = \frac{f}{L} \tan \theta' \quad (3)$$

For small angles,  $\tan \theta$  and  $\tan \theta'$  can be considered directly proportional by a proportionality factor  $k$ . This factor  $k$  depends on the geometry of the lens.

$$\tan \theta' \approx k \tan \theta \quad (4)$$

After substituting (4) into (3) the final relationship is found.

$$v = \frac{f}{L} k \tan \theta \quad (5)$$

Equation (5) assumes that the PSD is perfectly linear, namely that (1) is exact. Discrete Chebyshev polynomials were chosen to model the exact nonlinear relationship between  $v$  and  $\tan \theta$ . A Chebyshev polynomial of degree  $n$  is defined as:<sup>4</sup>

$$T_n(x) = \cos(n \arccos x) \quad (-1 \leq x \leq 1) \quad (6)$$

A  $(N-1)$ -th order Chebyshev polynomial approximation of the transcendental function  $\tan \theta(v)$  is defined in equation (7).

$$\tan \theta(v) \approx \sum_{n=0}^{N-1} c_n T_n(v) - \frac{1}{2} c_0 \quad (7)$$

The variables  $c_n$  are the Chebyshev polynomial coefficients. For an arbitrary function  $f(x)$  defined in  $[-1,1]$  these coefficients are defined as:

$$c_n = \frac{2}{N} \sum_{k=1}^N f \left( \cos \left( \frac{\pi(k - \frac{1}{2})}{N} \right) \right) \cdot \cos \left( \frac{\pi n(k - \frac{1}{2})}{N} \right) \quad (8)$$

The use of Chebyshev polynomials has several advantages over a regular polynomial curve fit. All polynomials are bounded by  $\pm 1$ . The Chebyshev coefficients  $c_n$  directly show the importance of that particular order. The first order approximation of  $\tan \theta(v)$  should stand out by several orders of magnitude because of the nearly linear behavior predicted in (5).

A Chebyshev approximation is more accurate to calculate than a least squares polynomial curve fit since a stable recursive formula (9) exists to calculate the Chebyshev polynomials. Avoiding having to calculate values to large powers saves a lot of computational time. More importantly, the orthogonality properties of the Chebyshev polynomials permit the high degree coefficients to be accurately computed without a matrix inverse.

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \quad (n \geq 1) \quad (9)$$

Please note that the tangent of the incidence angle, not the actual incidence angle is approximated in equation (7). All further formulas will be written in terms of the tangent of the angles.



Trigonometric calculations are very computationally intensive operations. This method eliminates calculating these transcendental functions by measuring the tangent of the angle directly!<sup>5</sup> This faster algorithm makes real-time calculations possible.

## CONFIGURATION

The SU position and orientation together are called the system configuration. This section presents a method to establish the configuration. A two-dimensional coordinate frame called the camera frame is first selected. The origin and orientation of this frame is arbitrary. All the profile scans are performed in this camera coordinate frame.

### Sensor Unit Position

The SU positions are determined by measuring the angles of three locations whose coordinates are known within the camera frame. Each location is placed on the z-axis an equal distance apart from the origin as shown in Figure 7.

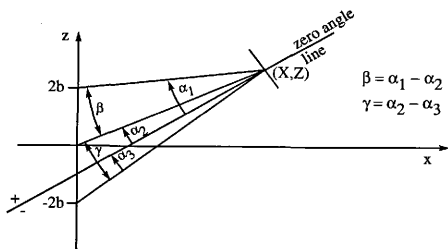


Figure 7: Illustration of SU Origin Determination.

The laser illuminates the targets successively and the three angle  $\alpha_i$  are measured. The angles  $\beta$  and  $\gamma$  are defined as illustrated in Figure 7. Given three known points and the two angles  $\beta$  and  $\gamma$  only two solutions for the SU position are possible. One has a positive, the other a negative x-coordinate. Assuming that the x-coordinate is positive in the camera frame, a unique solution can be determined!

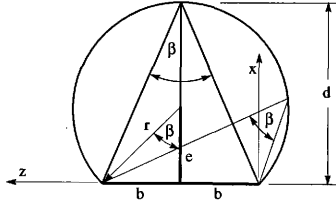


Figure 8: Illustration of the SU Position Solution Space.

For each angle  $\beta$  and  $\gamma$  the solution space is a circle going through the camera frame origin and the corresponding outer point as shown in Figure 8. By finding the intersection of the two circles, the SU position is determined. The distance  $e$  measured from the  $z$ -axis to the center of the solution space circles is very useful in the following development.

$$d = \frac{b}{\tan \frac{\beta}{2}} = b \cot \frac{\beta}{2} \quad (10)$$

$$b^2 + e^2 = (d - e)^2 = r^2 \quad (11)$$

$$e = \frac{b (\cot \frac{\beta}{2} - 1)}{2 \cot^2 \frac{\beta}{2}} = \frac{b}{2} (\cot \frac{\beta}{2} - \tan \frac{\beta}{2}) = \frac{b}{\tan \beta} \quad (12)$$

Since the PSDs will return the tangent of the angles  $\alpha_i$  and not the angles themselves, (12) can be rewritten making use of a trigonometric identity for  $\tan(\alpha \pm \beta)$  given in (13).

$$\tan(\alpha \pm \beta) = \frac{\tan \alpha \pm \tan \beta}{1 \mp \tan \alpha \tan \beta} \quad (13)$$

The distance  $e$  can be developed for the angle  $\gamma$  in a similar manner. Denoting  $e_1$  as the distance for the angle  $\beta$  and  $e_2$  for  $\gamma$ , they are defined as:

$$e_1 = b \frac{1 + \tan \alpha_1 \tan \alpha_2}{\tan \alpha_1 - \tan \alpha_2} \quad e_2 = b \frac{1 + \tan \alpha_2 \tan \alpha_3}{\tan \alpha_2 - \tan \alpha_3} \quad (14a,b)$$

The equations for the two solution space circles are defined as:

$$(x - e_1)^2 + (z - b)^2 = r_1^2 \quad (15)$$

$$(x - e_2)^2 + (z + b)^2 = r_2^2 \quad (16)$$

Here it was assumed that the first circle goes through  $(0, 2b)$  and the second circle goes through  $(0, -2b)$ . Both circles join at the origin. Consequently the solution at  $(0, 0)$  can be sampled out later. After expanding (15) and (16) and making use of

$$r_i^2 = b^2 + e_i^2 \quad (17)$$

the following two equations must be simultaneously solved.

$$x^2 - 2xe_1 + z^2 - 2zb = 0 \quad (18)$$

$$x^2 - 2xe_2 + z^2 + 2zb = 0 \quad (19)$$

To solve for  $x$  in terms of  $z$ , set (18) equal to (19) and solve for  $x$ . Here the fact that  $(x, z)$  is not  $(0, 0)$  was utilized.

$$x = \frac{-2bz}{e_1 - e_2} \quad (20)$$

Equation (20) can now be substituted into equation (19) to solve for  $x$  and  $z$ . The following two equations are the SU position  $(X_i, Z_i)$ . The subscript  $i$  denotes whether it was the first or the second SU.

$$X_i = \frac{4b^2(e_1 + e_2)}{4b^2 + (e_1 - e_2)^2} \quad (21)$$

$$Z_i = \frac{2b(e_2^2 - e_1^2)}{4b^2 + (e_1 - e_2)^2} \quad (22)$$

### Sensor Unit Orientation

The SU orientation can be defined by the slope of the zero angle line (ZAL) shown in Figure 7. Knowing  $\alpha_2$  and the SU position  $(X_i, Z_i)$  the slope of the ZAL can be found as shown in Figure 9.

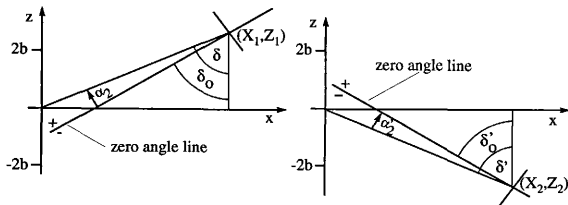


Figure 9: Illustration of SU Orientation Determination.

The angles with a ' mark the angles of the second SU shown in the right illustration of Figure 9. The angle  $\delta$  can be calculated using the SU position.

$$\delta = \text{atan} \frac{X_1}{Z_1} \quad \delta' = \text{atan} \frac{X_2}{Z_2} \quad (23a,b)$$

The angle  $\delta_0$  will define the slope of the ZAL of each SU. It is defined as:

$$\delta_0 = \delta - \alpha_2 \quad \delta'_0 = \delta' - \alpha'_2 \quad (24a,b)$$

Further calibration computations require the tangent of  $\delta$ . Taking the tangent of (24a,b) and making use of (13) and (23a,b) the following results are obtained.

$$\tan \delta_0 = \frac{\frac{X_1}{Z_1} - \tan \alpha_2}{1 + \frac{X_1}{Z_1} \tan \alpha_2} \quad \tan \delta'_0 = \frac{\frac{X_2}{Z_2} - \tan \alpha'_2}{1 + \frac{X_2}{Z_2} \tan \alpha'_2} \quad (25a,b)$$

Note the orientation of the angles. Any angle above the ZAL (in the +z direction) is considered positive. The angle  $\delta'_0$  in Figure 9 is negative, which agrees with the a negative slope of the ZAL.

## TRIANGULATION

When tracking an arbitrary laser spot, each SU returns a voltage; these two voltages define the two incidence angles. Let  $\alpha$  be the angle returned by the first SU, and  $\alpha'$  be the angle of the second SU.

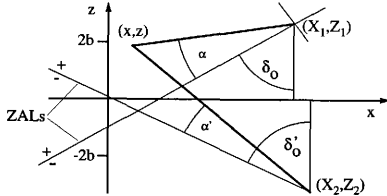


Figure 10: Triangulation of an Arbitrary Point.

The triangulation problem is now reduced to finding the intersection of two lines as seen in Figure 10. The two lines of interest are drawn in bold. The slope  $m_i$  of each line is defined as:

$$m_1 = \cot(\delta_0 + \alpha) \quad m_2 = \cot(\delta'_0 + \alpha') \quad (26a,b)$$

Again the trigonometric calculations can be avoided by making use of (13). The only singularities occur if either line is vertical, an impossible scenario because it is out of the field of view.

$$m_1 = \frac{1 - \tan \delta_0 \tan \alpha}{\tan \delta_0 + \tan \alpha} \quad m_2 = \frac{1 - \tan \delta'_0 \tan \alpha'}{\tan \delta'_0 + \tan \alpha'} \quad (27a,b)$$

The equations for the two lines are:

$$z = m_1(x - X_1) + Z_1 \quad z = m_2(x - X_2) + Z_2 \quad (28a,b)$$

After solving the two equations (28a,b), the  $x$  and  $z$  coordinates of the observed point can be found as given below in (29) and (30). The only unknown variables in these equations are the two slopes which are determined from the measured angles as shown in (27). Equations (27a,b) were not substituted to keep equations (29) and (30) more legible. All other variables are configuration

parameters which were previously established.

$$x = \frac{m_1 X_1 - m_2 X_2 + Z_2 - Z_1}{m_1 - m_2} \quad (29)$$

$$z = Z_1 + m_1 \frac{m_2 (X_1 - X_2) + Z_2 - Z_1}{m_1 - m_2} \quad (30)$$

To transform these two-dimensional coordinates into three-dimensional inertial coordinates the following two reference frames are used.

- N: Inertial reference frame with an origin located over the center of the airbearing. The orientation coincides with the initial camera frame orientation.
- S: Camera reference frame. This frame contains only  $x$  and  $z$  coordinates since it is the scanning plane. The  $y$  coordinate is always zero.

The transformation matrix  $C(t)$  will transform any vector with components in S into the same vector with components in N through (31).

$$r^N = C(t) r^S \quad (31)$$

The SU positions with S components are defined as:

$$r_{SU1}^S = \{X_1, 0, Z_1\} \quad r_{SU2}^S = \{X_2, 0, Z_2\} \quad (32a,b)$$

The camera position vector is defined as the mean of the two SU position vectors.

$$r_c^S = \frac{1}{2} (r_{SU1}^S + r_{SU2}^S) \quad (33)$$

Let  $r_p^S(t)$  be the position vector of a observed point with components in S as shown in (34). The variables  $x$  and  $z$  are defined in (29) and (30).

$$r_p^S(t) = \{x, 0, z\} \quad (34)$$

Figure 11 illustrates the relationship between the camera and position vector in the inertial and camera reference frames. Note that the camera flight path vector  $r^N(t)$  tracks the inertial camera position, not the origin of the camera frame S!

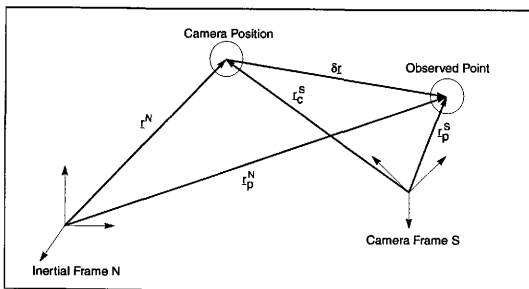


Figure 11: Illustration of Vectors in the Camera and Inertial Reference Frames.

The inertial coordinates of the observed point are found as:

$$r_p^N(t) = r_c^N(t) + \delta r^N(t) \quad (35)$$

The vector pointing from the camera position to the observed point in the camera reference frame is given in (36).

$$\delta r^S(t) = r_p^S(t) - r_c^S \quad (36)$$

Making use of (31) this vector is transformed into the inertial reference frame.

$$\delta r^N(t) = C(t) \delta r^S(t) \quad (37)$$

After substituting (37) and (36) into (35) the following formula is found. It calculates the inertial three-dimensional coordinates for any arbitrary camera flight path  $r_c^N(t)$  and camera orientation matrix  $C(t)$ .

$$r_p^N(t) = r_c^N(t) + C(t) (r_p^S(t) - r_c^S) \quad (38)$$

Note that since the initial camera orientation was chosen to coincide with the inertial frame the transformation matrix  $C$  initially is equal to the identity matrix.

$$C(0) = I \quad (39)$$

Since the setup used has a circular relative motion equation (38) can be further refined. All vectors are made dependent on the airbearing angle  $\theta$ . All profile scans are done while holding the airbearing steady. This angle will define the angular camera position of the circular flight path. Note that initially this angle is zero.

$$\theta(0) = 0 \quad (40)$$

The radius of the circular flight path is  $R_0$ , defined as:

$$R_0 = |r_c^S \cdot \{1, 0, 0\}| \quad (41)$$

The inertial camera position vector  $r^N(\theta)$  describes a circle parallel to the inertial x-y plane. The vertical offset between this horizontal circle and the coordinate origin of the inertial reference frame is denoted as  $z_c$ .

$$z_c = C(\theta) \cdot r_c^S \cdot \{0, 0, 1\} \quad (42)$$

Figure 12 illustrates the camera flight path. The relative camera motion is in the positive sense because the airbearing will have a negative rotation. The angle  $\theta$  is measured from the positive x-axis since the initial camera position is at  $x_c = +R_0$ . This was established in the configuration setup by assuring that all SUs will have positive x coordinates.

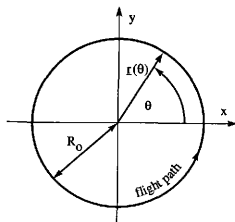


Figure 12: Camera Flight Path in the x-y Plane.



The inertial camera position vector is defined as:

$$\mathbf{r}^N(\theta) = \begin{pmatrix} \cos\theta \\ \sin\theta \\ 0 \end{pmatrix} R_0 + \begin{pmatrix} 0 \\ 0 \\ z_c \end{pmatrix} = \begin{pmatrix} R_0 \cos\theta \\ R_0 \sin\theta \\ z_c \end{pmatrix} \quad (43)$$

The orientation matrix  $C$  defines a rotation about the z-axis.<sup>(6)</sup> It is defined as:

$$C(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (44)$$

Using (43) and (44), the three-dimensional position vector  $\mathbf{r}_p^N$  can be calculated with the air-bearing angle  $\theta$ .

## TRIANGULATION SENSITIVITY

The transformation of coordinates from the camera frame to the inertial frame will have very little error since the only parameter is the airbearing angle  $\theta$ . This angle can be measured with an accuracy of greater than one arcsecond!<sup>(7)</sup> Yet the accuracy of the two-dimensional coordinates in the camera frame will heavily depend on the accuracy of the configuration parameters and, of course, the PSD measurement accuracy. This section will analyze the sensitivity of any arbitrary camera frame coordinates  $(x_i, z_i)$  versus the configuration parameters  $X_1, Z_1, \delta_0, X_2, Z_2$ , and  $\delta_0'$ . The slopes of the two intersecting lines during triangulation will depend on the measured incidence angles  $\alpha_i, \alpha'_i$  and the SU orientation angles as given in (45a,b).

$$m_{1i} = \cot(\delta_0 + \alpha_i) \quad m_{2i} = \cot(\delta'_0 + \alpha'_i) \quad (45a,b)$$

The formulas for the coordinates  $(x_i, z_i)$  are given in (46) and (47).

$$x_i = \frac{m_{1i}X_1 - m_{2i}X_2 + Z_2 - Z_1}{m_{1i} - m_{2i}} \quad (46)$$

$$z_i = Z_1 + m_{1i} \frac{m_{2i}(X_1 - X_2) + Z_2 - Z_1}{m_{1i} - m_{2i}} \quad (47)$$

The sensitivity parameters are found after taking the partial derivatives of  $x_i$  and  $z_i$  versus the

configuration parameters. Equation (48) through (59) define all sensitivity parameters.

$$\frac{\partial x_i}{\partial X_1} = \frac{m_{1i}}{m_{1i} - m_{2i}} \quad (48)$$

$$\frac{\partial x_i}{\partial Z_1} = \frac{-1}{m_{1i} - m_{2i}} \quad (49)$$

$$\frac{\partial x_i}{\partial \delta'_0} = \frac{x_i - X_1}{(m_{1i} - m_{2i}) \sin^2(\alpha_i + \delta'_0)} \quad (50)$$

$$\frac{\partial x_i}{\partial X_2} = \frac{-m_{2i}}{m_{1i} - m_{2i}} \quad (51)$$

$$\frac{\partial x_i}{\partial Z_2} = \frac{1}{m_{1i} - m_{2i}} \quad (52)$$

$$\frac{\partial x_i}{\partial \delta'_0} = \frac{X_2 - x_i}{(m_{1i} - m_{2i}) \sin^2(\alpha'_i + \delta'_0)} \quad (53)$$

$$\frac{\partial z_i}{\partial X_1} = \frac{m_{1i} m_{2i}}{m_{1i} - m_{2i}} \quad (54)$$

$$\frac{\partial z_i}{\partial Z_1} = \frac{-m_{2i}}{m_{1i} - m_{2i}} \quad (55)$$

$$\frac{\partial z_i}{\partial \delta'_0} = \left( \frac{z_i - Z_1}{m_{1i} - m_{2i}} - \frac{z_i - Z_1}{m_{1i}} \right) \frac{1}{\sin^2(\delta'_0 + \alpha_i)} \quad (56)$$

$$\frac{\partial z_i}{\partial X_2} = \frac{-m_{1i} m_{2i}}{m_{1i} - m_{2i}} \quad (57)$$

$$\frac{\partial z_i}{\partial Z_2} = \frac{m_{1i}}{m_{1i} - m_{2i}} \quad (58)$$

$$\frac{\partial z_i}{\partial \delta'_0} = \frac{Z_1 - z_i - m_{1i}(X_1 - X_2)}{(m_{1i} - m_{2i}) \sin^2(\delta'_0 + \alpha'_i)} \quad (59)$$

Note that the sensitivities not only depend on all the configuration parameters, but also on the incidence angles. This means that the sensitivity of a point depends on where in the field of view

the point lies.

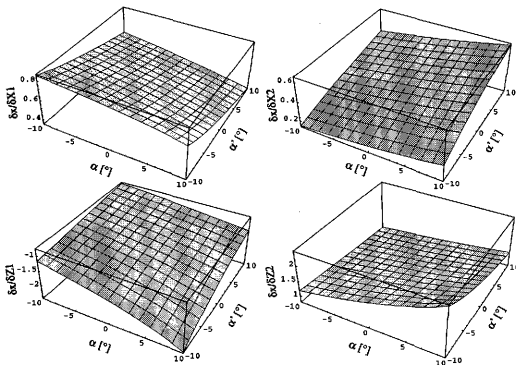


Figure 13: x-Coordinate Sensivities versus  $X_1$ ,  $Z_1$ ,  $X_2$ , and  $Z_2$ .

Figure 13 shows how the sensitivities of  $x_i$  can vary over the field of view. The SU orientation angle  $\delta_0$  was steady at  $64.03^\circ$ . Depending on the magnitude and sign of the configuration error,  $x_i$  could be enlarged or reduced. The sensitivities of  $z_i$  behave very similarly.

The accuracy of the configuration parameters will depend heavily on the accuracy of the three configuration angles  $\alpha_1$ ,  $\alpha_2$ , and  $\alpha_3$ . Since the laser spot will be placed on the three targets by hand, these angles will not have the same high accuracy the airbearing angle will have. Some sample placements found that the configuration angles will vary no more than  $0.01^\circ$ . While this is a small range, it is sufficient to cause a large range of possible configuration parameters.

The possible range of configuration parameters is illustrated in Figure 14 below. The true SU position and orientation used are given in the figure along with the configuration angle ranges used. The imaginary configuration targets were placed 20 mm apart. Clearly the configuration parameters can vary over a considerable range!

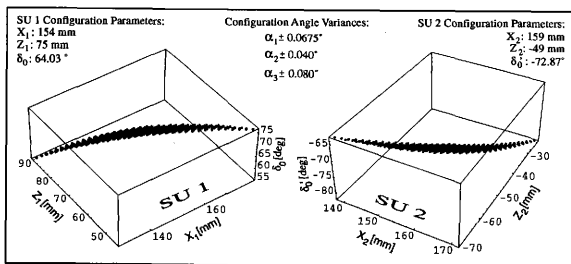


Figure 14: Configuration Parameter Variations Due to Errors in Configuration Angle Measurements.

One cause of this sensitivity is that the SUs will be as far as possible from the target to maximize the scanning area. Since the field of view is rather narrow, the configuration targets cannot be placed very far apart! The two circles that need to be intersected during the configuration calibration process are very close together, making the process very sensitive to errors in configuration angles. Figure 15 shows two sample circles and how the intersection is not very distinct. The configuration process could be made less sensitive by enlarging the field of view of the SUs allowing for greater distances between the configuration targets.

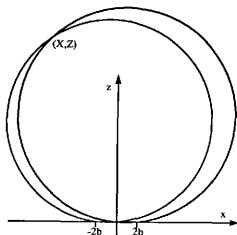


Figure 15: Illustration of the Configuration Solution Spaces.

To illustrate how much the  $(x,z)$  values could vary a sample point at  $(20,20)$  was used in Figure 16. The configuration parameters were varied over all possible values given in Figure 14.

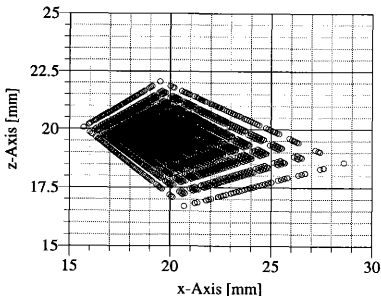


Figure 16: Range of  $(x,z)$  Values Due to Configuration Errors.

The  $x$  values vary from -20% to +45% of the true values. The  $z$  values vary from -17.5% to +10%. Note that these are the extremes, however. Most errors are roughly within  $\pm 10\%$ . This large range of  $(x,z)$  coordinates demonstrates a need to improve the accuracy of the configuration parameter calculation. The following method uses the initial configuration parameters as the initial values of a nonlinear iterative least squares method to fine tune the configuration parameters closer to their true values.

Let the vector  $r_i$  point to a known point  $(x_i, z_i)$  and the vector  $\sigma$  contain all the configuration parameters as given in (60).

$$\sigma = [X_1 \ Z_1 \ \delta_0 \ X_2 \ Z_2 \ \delta'_0]^T \quad (60)$$

The nonlinear function for  $r_i$  is given in equation (61).

$$r_i = f_i(\sigma, \alpha, \alpha') = \begin{bmatrix} x(\sigma, \alpha, \alpha') \\ z(\sigma, \alpha, \alpha') \end{bmatrix} \quad (61)$$

The nonlinear function  $f$  is linearized with a first order Taylor series expansion about a set of configuration parameters  $\sigma_0$ . The higher order terms are cut off.

$$r_i = f_i(\sigma_0, \alpha, \alpha') + \frac{\partial}{\partial \sigma} f_i(\sigma_0, \alpha, \alpha') \delta \sigma + HOT \quad (62)$$

Equation (62) can be solved for the triangulation error  $\delta r_i$ .

$$\delta r_i = r_i - f_i(\sigma_0, \alpha, \alpha') = \frac{\partial}{\partial \sigma} f_i(\sigma_0, \alpha, \alpha') \delta \sigma + HOT \quad (63)$$

Let the estimated configuration parameters be labelled as  $\hat{\sigma}$ . Equation (63) can be rewritten in terms of the estimated parameters. Since the higher order terms are dropped an error  $e_i$  is introduced.

$$\delta r_i = \frac{\partial}{\partial \sigma} f_i(\hat{\sigma}, \alpha, \alpha') \delta \sigma + e_i \quad (64)$$

By measuring the incidence angles of  $N$  sets of known locations  $r_i$  the following set of equations are found.

$$\delta r = \begin{bmatrix} \delta r_1 \\ \dots \\ \delta r_i \\ \dots \\ \delta r_N \end{bmatrix} = \begin{bmatrix} \frac{\partial x_i}{\partial X_1} \frac{\partial x_i}{\partial Z_1} \frac{\partial x_i}{\partial \delta_0} \frac{\partial x_i}{\partial X_2} \frac{\partial x_i}{\partial Z_2} \frac{\partial x_i}{\partial \delta'_0} \\ \frac{\partial z_i}{\partial X_1} \frac{\partial z_i}{\partial Z_1} \frac{\partial z_i}{\partial \delta_0} \frac{\partial z_i}{\partial X_2} \frac{\partial z_i}{\partial Z_2} \frac{\partial z_i}{\partial \delta'_0} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \frac{\partial x_N}{\partial X_1} \frac{\partial x_N}{\partial Z_1} \frac{\partial x_N}{\partial \delta_0} \frac{\partial x_N}{\partial X_2} \frac{\partial x_N}{\partial Z_2} \frac{\partial x_N}{\partial \delta'_0} \\ \frac{\partial z_N}{\partial X_1} \frac{\partial z_N}{\partial Z_1} \frac{\partial z_N}{\partial \delta_0} \frac{\partial z_N}{\partial X_2} \frac{\partial z_N}{\partial Z_2} \frac{\partial z_N}{\partial \delta'_0} \end{bmatrix} \delta \sigma + \begin{bmatrix} e_1 \\ \dots \\ e_i \\ \dots \\ e_N \end{bmatrix} \quad (65)$$

Let  $A$  be the  $(2N \times 6)$  sensitivity matrix evaluated at  $\hat{\sigma}$ . The error vector  $e$  is defined as:

$$e = \begin{bmatrix} e_1 & \dots & e_N \end{bmatrix}^T = \delta r - A \delta \sigma \quad (66)$$

The goal of the iteration method is to minimize the error vector  $e$ . Let the cost function  $J$  be defined as the square of the error vector.

$$J = e^T \cdot e \quad (67)$$

After substituting (66) into (67) and expanding the expressions the following second order cost function is found.

$$J = \delta \mathbf{r}^T \delta \mathbf{r} - 2 \delta \boldsymbol{\sigma}^T \mathbf{A}^T \delta \mathbf{r} + \delta \boldsymbol{\sigma}^T (\mathbf{A}^T \mathbf{A}) \delta \boldsymbol{\sigma} \quad (68)$$

Minimizing the cost function will also minimize the error vector. At an extremum the first derivative of  $J$  will be zero!

$$\frac{\partial J}{\partial \boldsymbol{\sigma}} = -2 \mathbf{A}^T \delta \mathbf{r} + 2 (\mathbf{A}^T \mathbf{A}) \delta \boldsymbol{\sigma} = 0 \quad (69)$$

Equation (69) can be simplified and solved for  $\delta \boldsymbol{\sigma}$ .

$$\delta \boldsymbol{\sigma} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \delta \mathbf{r} \quad (70)$$

$N$  has to be at least three to have a uniquely determined error in configuration parameters. If  $N$  is greater than three equation (70) finds the error in a least squares manner.<sup>(8)</sup> The relationship between the configuration parameter error, the estimated and the actual configuration parameters is given in (71).

$$\boldsymbol{\sigma}_{true} \approx \boldsymbol{\sigma}_{updated} = \hat{\boldsymbol{\sigma}} + \delta \boldsymbol{\sigma} \quad (71)$$

After the configuration parameters have been updated the  $(x, z)$  coordinates can be recalculated and compared to the true  $(x, z)$  values. If the error is too large, the iteration is repeated. The success of this gradient method will depend on many factors. The initial configuration measurements will have to be sufficiently close to the true values for the iteration to converge. The accuracy of the final configuration parameters depends on how many points were measured and how accurately the coordinates of these points are known.

## EXPERIMENTAL RESULTS

The source code for all the programs used during calibration, configuration and three-dimensional scans can be found in Appendix B.

### CALIBRATION

#### PSD Signal

To determine the actual relationship between the PSD voltage  $v$  and the tangent of the incident angle  $\theta$ , the SU was mounted on the airbearing with the lens over the center of the airbearing platform. A laser was aimed at a white target on the same level as the SU. The airbearing was then rotated to different positions to measure the corresponding PSD voltage. At each position, the signal was oversampled to reduce the effects of electronic noise. As predicted in equation (5), the relationship is very linear (see Figure 17).

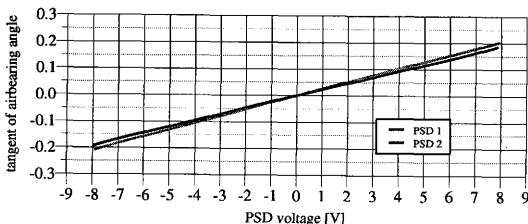


Figure 17: PSD Voltage versus Tangent of Airbearing Angle.

The difference in slopes is a result of having two slightly different gains on each processing board. The scale in Figure 17 is too large to see the non-linear term of the relationship, but shows that these are indeed near linear position detectors. Figure 18 was created by taking the PSD data shown in Figure 17 and subtracting the linear term. The magnitude of the residual non-linear term is two orders of magnitude smaller than the linear term. This corresponds to having a non-linearity of less than 1%, as predicted by the manufacturer.



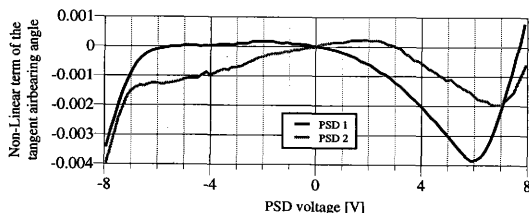


Figure 18: Non-Linear Term of the PSD Voltage versus Airbearing Angle Relationship.

The relationship between PSD voltage and the tangent of the incidence angle was modeled with a set of Chebyshev polynomials. The accuracy of the model depends on the order of the Chebyshev approximation and the airbearing step size. A range of step sizes and Chebyshev orders were tested to find an optimal combination for minimum modelling error.

The modelling error in Figure 19 was calculated as the sum of the absolute difference between the model and the actual measurements. After an order of 21 the model errors flatten out quickly. They decrease only very slightly for higher orders.

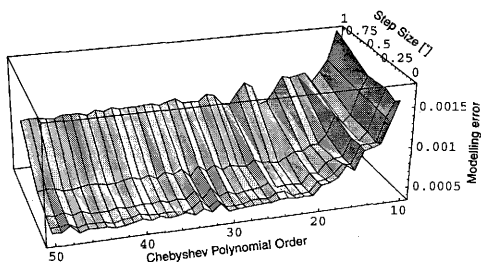


Figure 19: Comparison of the Calibration Modelling error.

The step size variable dictates the number and separation of the measurements points. It was found to be less important than the Chebyshev orders. Going to step size smaller than  $0.25^\circ$  provided very little gain, but increased the calibration time greatly. The slowest factor of the calibration process was moving the airbearing to a new position. Therefore all calibrations were done with an airbearing step size of  $0.25^\circ$  using Chebyshev polynomials up to an order of 20.

All above data was taken with the laser spot six inches from the SU. At other distances the PSD voltage versus the tangent of the incidence angle relationship turned out to be slightly different. Due to depth-of-field effects, as the laser spot moved closer or further away the focus changed, causing a slight change in the voltage-displacement relationship.

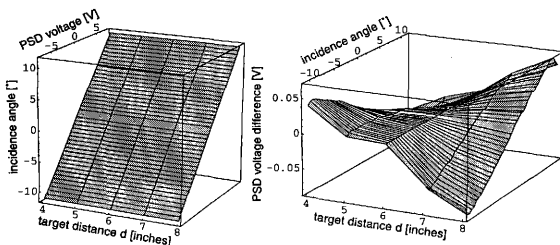


Figure 20: Variation of Calibration Model at Different Target Distances.

Figure 20 shows how the calibration of a PSD changes at different target distances. The left picture shows the exact calibration, yet shown on a full scale, no difference can be seen. The difference is illustrated in the right picture. The calibration at a distance of six inches was subtracted from all the other calibrations. The PSD voltage difference compared to the six inch calibration are within 0.08V. The 12-bit A/D board accuracy is 0.00488V.<sup>(9)</sup> This focus problem can cause an error 16 times the A/D board accuracy!

To account approximately for this focus problem, a two-step process was used. After measuring two PSD voltages, the incidence angles are calculated assuming the target was six inches away.

The found  $(x, z)$  coordinates are then used to find a better approximation of the target distance  $d$ . The calibration process was only done at target distances of four, five, six, seven, and eight inches away. The actual incidence angles are found by using the new target distance to linearly interpolate between two sets of calibration constants. Since the calibration variations are small and smooth, the depth-of-field errors were reduced to nearly the same magnitude as the A/D board errors.

Table 1: Chebyshev Polynomial Coefficients

Polynomial Order	PSD 1	PSD 2
0	-2.898711435807941e-003	-4.255639488248909e-003
1	1.865738889927913e-001	2.063986745698595e-001
2	-9.846744572974053e-004	-1.861139324875055e-003
3	2.106006352562894e-003	4.423704187360154e-004
4	5.252012010075907e-004	2.242101351328072e-004
5	9.108117735722156e-004	5.419014729523152e-004
6	8.597118071953371e-006	1.090098371598563e-004
7	4.161009380323950e-005	1.926561369360239e-004
8	-1.165380879027579e-004	3.221203249737141e-005
9	-1.343764758365768e-004	5.387506617794756e-005
10	-2.031394474557149e-005	-2.264253354230158e-006
11	-2.924012407003152e-005	-1.760042462490284e-005
12	4.453145084760879e-005	-2.387506428718035e-005
13	1.485919646587206e-005	-4.927841901712399e-005
14	1.728001587333897e-005	-5.981918851286550e-006
15	1.487004939920165e-005	-1.096258768214211e-005
16	-9.524834239409149e-007	1.577971919347374e-005
17	-1.590003411420417e-005	-9.829060372190252e-006
18	-1.615288137675631e-005	5.634727640673226e-006
19	-6.552275211586792e-006	1.378458940959435e-005
20	9.459421974195356e-006	1.991662323445494e-005

The Chebyshev polynomial coefficients for the two PSDs are given in Table 1. Note that the first order coefficients are two orders of magnitude larger than the other coefficients, showing again the near linear behavior of the PSD.

### Electronic Noise

Since the PSDs are detecting only the diffuse reflection of the laser light, not the laser light itself, the signal strength is very weak. To operate optimally, the signal processing board requires a signal strength of 1 volt. The diffuse reflection of a white matte surface yields a signal strength of only 0.03 volts at maximum gain setting. A shiny surface gave a weaker signal, since most of the light was concentrated along the total reflection ray direction.

Having such weak currents going from the PSD chip to the processing board means that electronic noise becomes an issue. An attempt to reduce the noise was made by shielding and grounding all cables and grounding any metal equipment in the vicinity. After all remedial actions were taken, some small amount of electronic noise remained.

The electronic noise can be broken down into two parts. There is a 60 Hz component and a higher frequency noise superimposed on it. Several types of analog and digital filters were examined to remove the noise from the main signal. The goal of the filter was to reduce the noise level to less than the A/D board accuracy of 0.00488 volts while keeping the overall system very responsive. A very clean signal was achieved using an analog RC filter. This filter was second order and had a cut-off frequency of around 5 Hz. It got rid of both the 60Hz and the high frequency noise, but it was not used because it had considerable lag and therefore could not be used to track rapid image motion. Being second order and having such a low cut-off frequency made the system far too sluggish. When tracking a 10°/s motion it had a 0.1° lag. This is 20 times the A/D board accuracy! No satisfactory analog filter was found that reduced the noise to an acceptable level and kept the system responsive.

A digital low-pass filter was tested. Trying to cancel both types of noise yielded the same result as the analog filters. The system became too sluggish. Another digital filter was tested where only the higher frequency noise was reduced leaving the 60Hz noise. The 60Hz noise was then cancelled by taking a fast Fourier transform (FFT) of the signal and smoothing out the 60Hz peak

and then performing an inverse FFT. This worked fine for steady signals where the true signal was a constant line and the 60Hz component was clearly visible. Yet when examining the signal of an actual profile scan, the 60Hz noise component was not distinctive in the FFT. Trying to smooth the 60 Hz noise caused more error in the true signal than was there to begin with.

The digital filter finally adopted was a localized averaging scheme. Each data point was averaged with a set number of data points before and after the current data point. This process was relatively fast and reduced the noise levels to an acceptable level without making the system performance sluggish. Since this filter used a small amount of “future” data, it would not be acceptable for most real-time applications. However, for near-real-time stereo imaging this process was the most attractive filter tested.

To show the level of noise present and the effect of the moving average filter, several steady state test runs are presented. Here the laser spot and the SU are held steady while the computer samples the signal. The true signal should always be a constant voltage.

Both cables that carry the two PSD currents to the processing board are very close to each other and will therefore pick up almost the same noise. The PSD voltage formula is given in equation (1). In the numerator, one current is subtracted from the other current. If each cable picked up identically the same noise, it would cancel itself in the processing board.

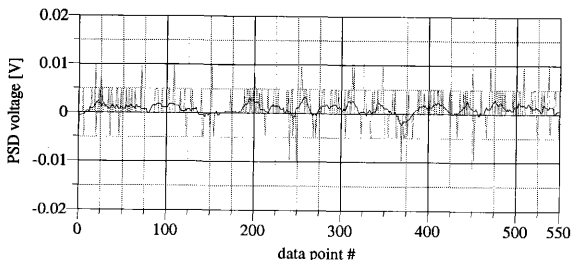


Figure 21: Steady State Zero PSD Signal with a Target Distance of Five Inches.

Figure 21 shows a steady state signal with a PSD voltage of about zero. The black line is the filtered signal and the grey line is the unfiltered signal. Keep in mind that the A/D board accuracy is 0.00488 volts, which corresponds with the horizontal grids of the graph for easier viewing. The filtered signal variations remain less than the A/D board accuracy as desired. The smoothing factor used on all these steady state tests is six. Since the cables did not pick up the exact same electronic noise, a small residual noise remains even after the cancellation. The high frequency noise component is caused by the processing board. Very weak signals are processed in the operational amplifiers and the analog dividers.

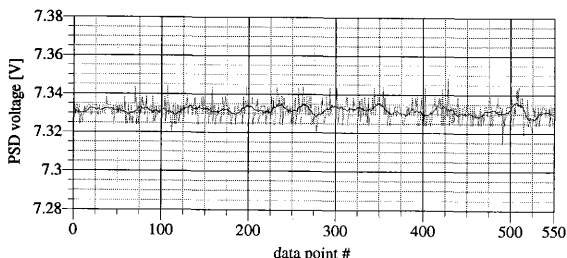


Figure 22: Steady State Non-Zero PSD Signal with a Target Distance of Five Inches.

Figure 22 shows the same steady state test at a non-zero PSD voltage. The 60Hz noise level has grown larger. A distinct sinusoidal wave can be seen. The filtered signal variations remain around the A/D board accuracy. Notice, however, that the 60Hz noise is causing more error to the filtered signal than the higher frequency noise. The larger the magnitude of the PSD voltage, the larger the 60Hz noise component grew. This enlargement could be an effect of the PSD chip, but this is unlikely. It is also unlikely that electronic noise caused the phenomena, since there is no reason for the 60Hz component enlargement to depend on the incidence angle of the light. The only remaining explanation is that it is an optical effect. Slight power fluctuation causes the laser signal to be unsteady. These shifts in the energy distribution cause small shifts in the perceived centroid

of the laser spot. The laser used utilizes a five volt DC power supply transformed from 60Hz AC power. No AC to DC transformations are perfect and there probably remain some small fluctuations in the laser power supply. This hypothesis is strengthened by a separate observation. The neon light tube in the ceiling and the desk light cause the same effect, though at a larger magnitude. Both types of light sources are unsteady and fluctuate in strength with a 60Hz frequency. The neon lights are sufficient to yield a strong diffuse reflection with a signal strength of more than 0.1 volts. At non-zero PSD voltages a large sinusoidal wave of about two volts can be observed on the position signal. After orienting the SU to let the ambient light be evenly distributed over the PSD chip, the average PSD voltage returned is zero. The 60Hz sine wave is virtually canceled! There is still the same signal strength, but the oscillations have been greatly reduced to about 0.1 volts. This is the same effect as observed with the laser spot. The 60Hz noise component is minimized at a zero PSD voltage and grows larger with larger PSD voltage magnitudes!

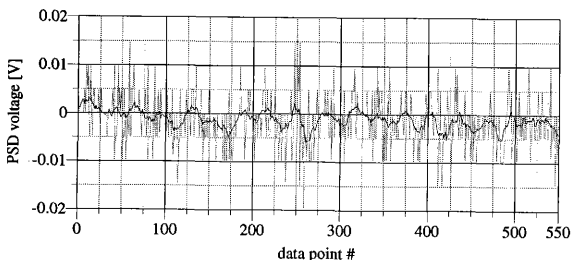


Figure 23: Steady State Zero PSD Signal with a Target Distance of Eight Inches.

This 60Hz noise became excessive after 8 volts. Therefore all PSD voltages were limited to within  $\pm 8$  volts. In Figures 21 and 22 the target distance was five inches, which is close enough to provide a strong signal. Figure 23 shows the same test run as Figure 21, but at a larger target distance of eight inches. Even though the PSD voltage is centered about zero, there is a larger 60Hz noise component than in Figure 21. The laser power influence is the same, but the electronic 60Hz

component is larger here because of the greater signal. The signal strength drops off with the square of the distance. The overall noise level has roughly doubled. The filtered signal is slightly larger than the A/D board accuracy.

Figure 24 shows the same test run as Figure 22, except for being at a larger target distance of eight inches. The filtered signal is now bounded by approximately three times the A/D board accuracy. The 60Hz noise is very distinct here. The maximum allowable target distance was found to be nine inches. Beyond that, the signal to noise ratio became too small.

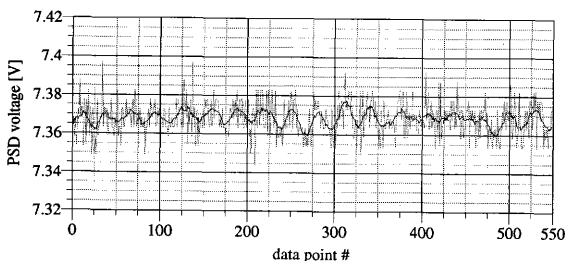


Figure 24: Steady State Non-Zero PSD Signal with a Target Distance of Eight Inches.

While the noise levels shown are not severe, they could be greatly reduced in a future design by building a printed circuit board containing the signal processing electronics and a chip socket for the PSD. This would reduce the noise in two ways: (1) the distance from the chip to the signal processing would be drastically reduced; (2) the printed circuit boards are inherently more noise resistant since all circuits are close together and well grounded. A laser light source with a very stable power output would help reduce some of the 60Hz noise components.

### Sensor Unit Pointing Accuracy

The accuracy and repeatability of the calibration process was tested by having the airbearing rotate the SU through the field of view. At certain increments the airbearing would stop and the



PSD voltage would be sampled. Oversampling a steady signal removed all the electronic noise. The resulting voltage was then transformed into an angle using the Chebyshev coefficients. Figure 25 illustrates the difference between the calculated position angle and the actual airbearing position. The test run was performed with a target distance of five inches.

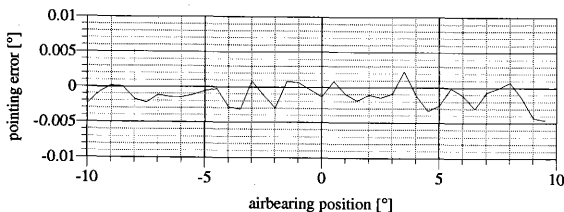


Figure 25: Sensor Unit Pointing Accuracy with a Target Distance of Five Inches.

The pointing accuracy lies just within the A/D board accuracy. This result was also very repeatable, demonstrating that the PSD chip was mounted sufficiently rigid in the sensor unit.

The remaining pointing errors seen in Figure 25 are due to calibration inaccuracies and a minute flexibility within the sensor unit. To simulate the actual scanning process as closely as possible, the SU should remain steady with no position varying forces acting on it. This was accomplished by rigidly attaching the SU to the airbearing!

### Sensor Unit Tracking Accuracy

To test how well a SU could track a moving target, the SU was left attached on the airbearing and the airbearing performed a finite rotation. During dynamic motion, the actual airbearing angle was compared to the SU angle. Ideally, the tracking error should always be zero. Yet because each position measurement takes a finite amount of time, there will be some lag if the target moves too fast! The A/D board access time to read a voltage was measured at around  $220\mu\text{s}$ . For this access time to cause a perceivable tracking error, the target would have to move  $22^\circ/\text{s}$  or  $0.06\text{Hz}$ . This access time therefore limits the profile scans to be done with a slow moving laser only!

The PSD chip rise time is advertised at  $0.05\mu\text{s}$ . For this rise time to cause a tracking error perceivable by the A/D board, the target would have to move at 271Hz. This is several orders of magnitudes larger than what the A/D access time dictates. Therefore the PSD chip is responsive enough not to cause any tracking error.

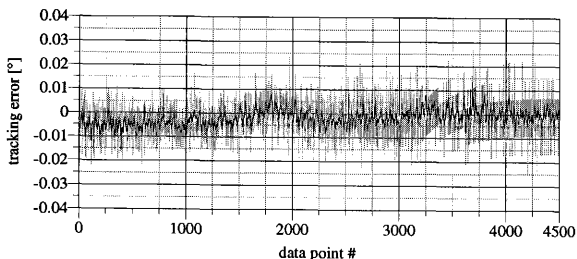


Figure 26: Sensor Unit Tracking Error with a Target Distance of Five Inches.

The airbearing motion used for the tracking test started at rest at  $-10^\circ$ , accelerated to a coasting motion of  $+20^\circ/\text{s}$  and came to rest at a position of  $+10^\circ$ . Figure 26 shows the tracking error for a test run at a target distance of five inches. As with the steady state plots, the black line is the filtered signal and the grey is the unfiltered signal. The filter used a smoothing factor of six. The tracking error starts out at zero and then quickly drops to  $-0.005^\circ$ . This lag was due to the A/D board access time. The airbearing was rotating at the critical speed mentioned earlier! After about the 2000th data point measurement, the airbearing is very close to the  $+10^\circ$  position and is moving very slowly to avoid overshooting the target. At the end of the maneuver, the tracking error approaches zero, indicating there was no significant residual pointing error.

Figure 27 shows the same tracking test, but at a target distance of eight inches. As expected the noise levels are higher. The tracking error during the coasting motion is again  $-0.005^\circ$ , as predicted. Note that the noise in the filtered signal gets smaller towards the 1100th data point and then begins to increase again. This happens because the PSD voltage goes through the zero voltage point where the noise cancels itself.

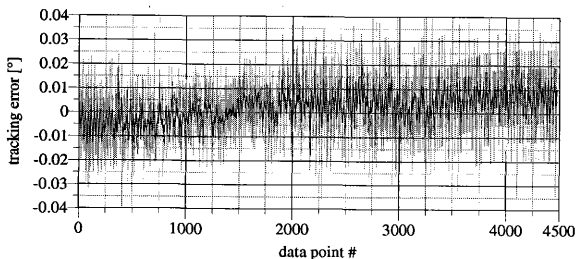


Figure 27: Sensor Unit Tracking Error with a Target Distance of Eight Inches.

To improve the tracking performance of the SU, a faster A/D board and computer would be needed. The A/D sampling and access time are the primary features limiting the present system to tracking relatively slow targets.

## CONFIGURATION

The configuration calibration process determines the position and orientation of each SU. Both SUs are placed on a support platform, but independent from the airbearing. This platform also carries the laser light source and the scanning mirror mounted on a motor as is seen in Figure 28. The SUs and the scanning laser motion are aligned in the same plane. The approximate distance of the SUs to the configuration targets are measured by hand and stored in a file. The configuration program will use these approximate distances when adjusting for the focus.

A vertical target was placed over the center of the airbearing. The target should be a matte black color to absorb all the laser light. Three thin white targets are placed exactly one inch apart on the vertical target. The center mark will become the coordinate origin. An elevated origin setup shown in Figure 29 was chosen over having the origin on the airbearing platform for visibility reasons.

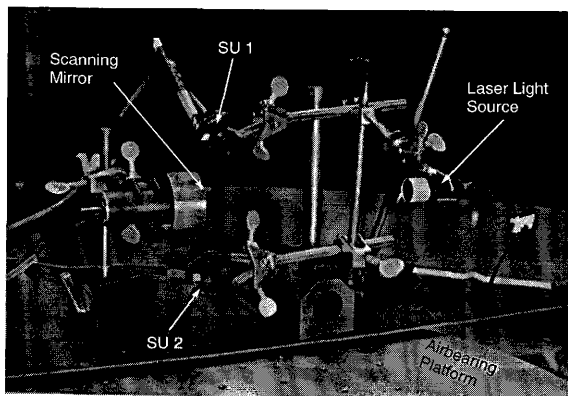


Figure 28: Photograph of the Three-Dimensional Scanning System.

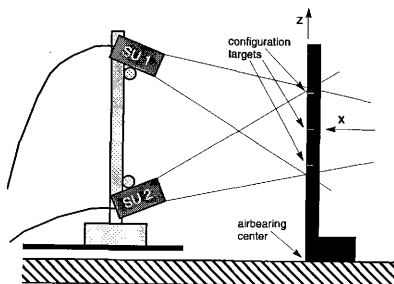


Figure 29: Schematic of a Configuration Setup.

As explained earlier, the configuration process is very sensitive to errors in the configuration

angles. To minimize configuration angle variations the white targets should be as thin as possible, while allowing enough return light to yield sufficient signal strength. During the laser placement, the PSD signal strength can be observed on an oscilloscope. As the laser is centered on the mark the signal strength should be at a maximum. With these precautions the configuration angle variations shown in Figure 14 were achieved. While these angles were very small, they were enough to cause significant configuration errors. Scanned objects were found to be 5-10% wider in width and shorter in height than physically correct.

To fine tune the configuration parameters six target points were used. Three were the original configuration targets over the center of the airbearing and the other three were on a vertical line at  $x=+20\text{mm}$ . This was sufficient for the nonlinear least-squares method to fine tune the parameters. Table 2 shows a comparison of the hand-measured coordinates of the six targets and the coordinates of the nonlinear least-squares converged estimates. The first point stands out with the largest difference. All other points have difference of less than 0.4mm. The reason the iteration method did not converge exactly to the measured coordinates is that the hand measurements inherently have same placement error associated with them, as did the configuration placement. Again to reduce the placement error all measurements were taken three times and averaged. After fine tuning, the configuration parameters, the distortions of scanned objects were reduced almost an order of magnitude to less than 1-3%!

Table 2: Comparison of Reconfiguration Targets

Target Number	Hand Measured Coordinates [mm]	Converged Coordinates [mm]	Difference [mm]
1	(19,12.7)	(19.7,12.77)	0.703
2	(19,0)	(18.96,0.02)	0.044
3	(19,-12.7)	(19.42,-12.77)	0.426
4	(0,25.4)	(0.14,25.41)	0.140
5	(0,0)	(-0.34,-0.18)	0.385
6	(0,-25.4)	(0.14,-25.26)	0.198

The final SU positions are compared to the hand measured positions in Table 3. The differences vary from -4% to +10%. Keep in mind that it is very difficult to measure the SU positions accurately by hand. Some of the differences are simply due to inaccurate hand measurements.

Table 3: Comparison of Sensor Unit Positions

SU Number	$X_i$ [mm]	$Z_i$ [mm]	measured $X_i$ [mm]	measured $Z_i$ [mm]
1	148.3	81	154	75
2	167	-54	159	-54

Another cause for the differences in SU positions is the laser diode. After focusing the laser beam to a point there is a half-moon shape glow about this point. This glow acts as optical noise causing a slightly different angle to be perceived. A pin-hole mask was ineffective in blocking the glow since the optical noise originates from the center of the laser beam. The configuration parameter fine tuning process tries to account for this error by shifting the parameters slightly from their true values. This "aliasing" is common in any estimation process where the model does not capture all of the physics.

During the configuration calibration process it became evident that the laser spot size must be as small as possible. The first test runs were performed without the focusing lens. Having the light spread out over a finite region meant that the center of the light intensity was off from the geometric center. In the worst case situations this error was enough to cause distortions of 15-20%!

To increase the configuration accuracy it would be beneficial to have a larger field of view, ideally using a fish-eye lens. The further apart the configuration targets can be placed, the more accurate the result will be. The intersection of the two configuration circles would be more distinct.

## SCANNING RESULTS

### Two-Dimensional Profile Scans

The electronic noise will cause some errors in the SU angles. The range of these errors depends on where in the field of view the observed spot is.

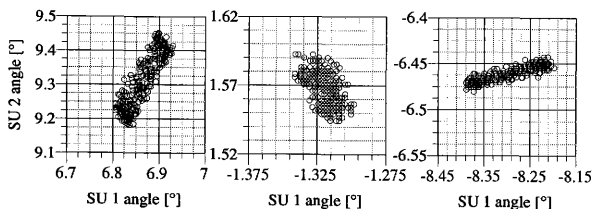


Figure 30: SU Angle Variations Due to Electrical Noise.

Figure 30 shows SU angle variations for three different locations. For each measurement, the laser is held steady at a certain location. The left, center and right pictures show the variations for the upper, center and lower half of the field of view, respectively. As expected, there is less error in the center of the field of view. The left and right pictures have elongated angle distributions.

The axis with the larger angle has a larger variation. This is because the further away from the PSD center the light spot is, the more the noise is perceivable. Figure 31 shows how these angle variations cause triangulation variations. The points observed are roughly in the center of the air-bearing, six inches from the SUs. Again the least variation is found in the center of the field of view. Note that the worst variation is only about 0.5 mm! Even though the processing board has to work with a very weak signal the noise levels are still tolerable.

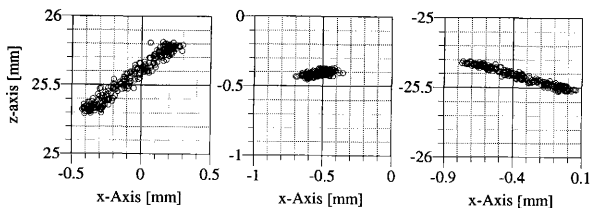


Figure 31: Triangulation Variations Due to Electrical Noise.

To measure how well triangulation is working for the first experiment, the side of a straight ruler was scanned. The result should be a straight line with no curvature. Figure 32 shows the actual scan. The scanner was able to recreate the flat surface very well. The first order curve fit was drawn in as a visual reference. The deviations from this line are minimal and only due to electronic noise. Their magnitude is less than 0.5mm. This confirms the maximum error due to noise shown in Figure 31.

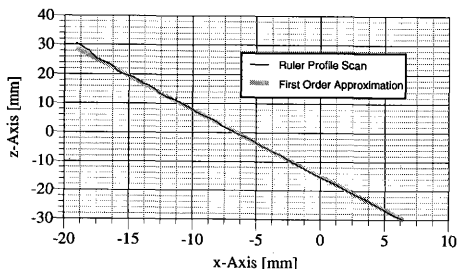


Figure 32: Profile Scan of a Straight Ruler.

Table 4: Chebyshev Coefficients for Ruler Approximation

Polynomial Order	Chebyshev Coefficients
0	0.238021
1	-31.562688
2	0.525353
3	-0.241915
4	0.280774
5	0.029339
6	-0.178053



The scan shown above in Figure 32 was analyzed by fitting a set of Chebyshev polynomials to it. Table 4 shows the resulting Chebyshev coefficients up to an order of six. The second and higher order terms are all at least two orders of magnitude smaller than the first order term. All these higher orders terms are trying to model the noise on top of the straight line. This means the only significant error is due to electronic noise, not calibration or configuration error.

### Three-Dimensional Scans

All three-dimensional scans were done with the scanning motor set at approximately 0.5Hz. This is about eight times faster than the critical speed of 0.06Hz. Having the laser spot move at this speed causes a triangulation error of only about 0.1mm over the airbearing center, and less error at a closer target distance! This error is far less than the electronic noise shown in Figure 31. Therefore it has only a minimal influence on the scans.

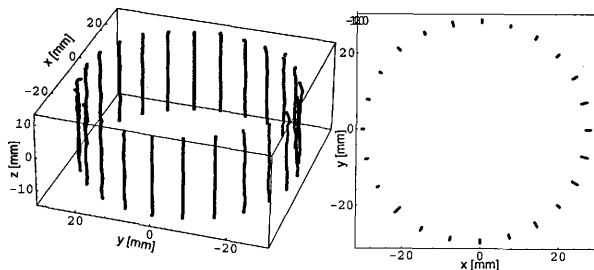


Figure 33: Three-Dimensional Scan of Cylindrical Tube.

The object used in Figure 33 was a cylindrical plastic tube. It was covered with white masking tape to make the surface less translucent. The actual outer diameter is 57mm. The tube was positioned over the center of the airbearing with its walls close to vertical. The scanned profiles shown in the left picture are very smooth and vertical. The slight unevenness is due to electronic noise. A top view is shown in the right picture. The circular shape of the tube is very well reproduced due to the high angular accuracy of the airbearing. The profile scans do not collapse to a point in this view

because of the electronic noise. The top view indicates the tube outer diameter to be 57.8mm, only a 1.4% enlargement.

Another object scanned was a white rectangular piece of wood. Its walls were close to vertical, but the wood grain contained grooves in it 1-2mm deep. Figure 34 shows the scan.

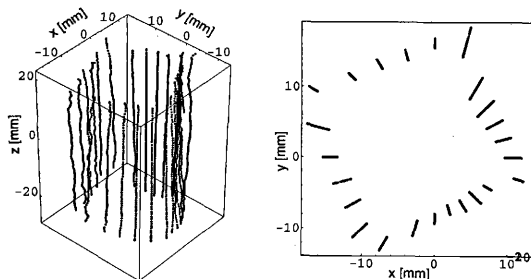


Figure 34: Three-Dimensional Scan of a Rectangular Piece of Wood.

The profile scans are far less smooth. It is possible to see where the surface has a distinct grain texture. The true side length of the rectangular block was 23mm. The top view shows a side length of about 23.7mm, an enlargement of about 3%.

Figure 35 shows a three-dimensional scan of a small bottle. It was covered with white masking tape to increase the signal strength. The shape of the bottle can be clearly seen. Note that some erroneous points appear where the main bottle transitions to the cap. As the laser swept over the edge both the cap and the bottle are illuminated for a split second causing this error.

Another problem was encountered when trying to scan a surface with sharp cavities. Figure 36 shows the profile scan of a large plastic thread. The actual thread shape has six teeth threads each with a triangular shape. The scanned shape shows the sharp outer edge of the teeth, yet the cavities were smoothed out. As the laser light hits the cavity it reflects off the walls illuminating the entire cavity! Depending upon the viewing angle and the laser incidence angle it is not possible to reli-

ably see the bottom of sharp crevice features. When the laser energy enters into the translucent plastic material, the light is reflected from a volume of the material rather than from a surface. The PSDs cannot clearly distinguish where the laser spot is, resulting in the smoothing effect.

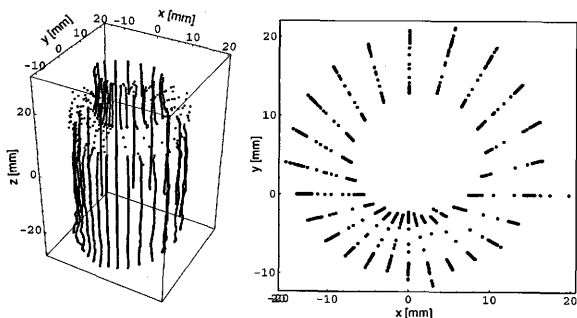


Figure 35: Three-Dimensional Scan of a Small Bottle.

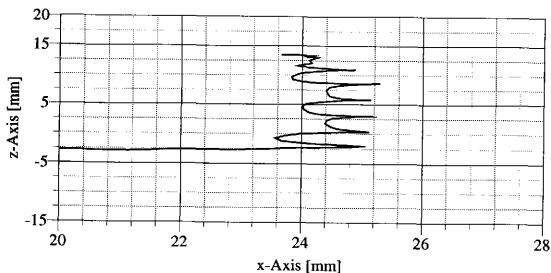


Figure 36: Profile Scan of a Large Plastic Thread.

If an objects' surface changes to a color other than white, the signal strength is reduced. This

by itself would not cause a problem with the triangulation if it were not for the electronic noise. With the reduced signal strength the noise becomes more dominant causing more triangulation errors. Figure 37 shows a ruler profile scan again. This time the laser light spot was allowed to sweep over the black letters of the ruler. These sections are clearly visible. The otherwise smooth graph makes some erratic jumps and then becomes smooth again. The center irregularity does not stand out like the other two because the noise happened to go through a low cycle there.

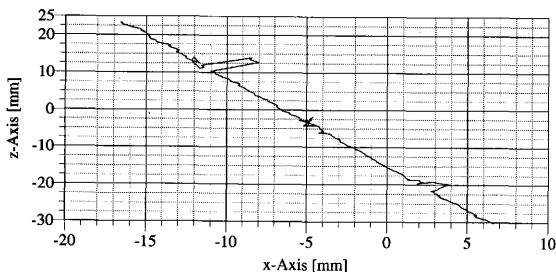


Figure 37: Profile Scan of White and Black Areas of a Ruler.

## CONCLUSIONS AND RECOMMENDATIONS

*This study addressed the real-time stereo triangulation of diffusely reflected laser energy. The PSDs tracked the moving diffuse laser light reflection very well. Having an analog position signal allows for very high resolutions as well as near-real-time processing. Their only drawback was the weak signal they returned to the processing boards. This weak signal was found very susceptible to electronic noise. The problem could be resolved in two ways: (1) use of a stronger laser to achieve a brighter reflection; (2) mount the PSD chip with the signal processing circuits on a printed circuit board; (3) and improved optics with a larger field of view and greater depth of field.*

The SU calibration was able to keep the pointing error to less than the A/D board accuracy of  $0.005^\circ$ . This accuracy was maintained for targets from four to eight inches away from the sensor. The defocusing problem was solved by calibrating the sensors at different distances and then linearly interpolating between the sets. The field of view of a SU was about  $22^\circ$ . The Chebyshev polynomials lent themselves well to model the PSD voltage versus tangent of the incidence angle relationship. Using these polynomials and measuring the tangent of the angles directly sped up the algorithms greatly. The accuracies were not realized reliably in the stereo triangulation measurement of bodies, mainly due to: (1) low signal strength of diffuse reflection; (2) faster scanning rates; (3) and various aliasing effects due to asymmetric diffuse reflections off particular surfaces.

Finding the configuration of the system is a very sensitive process. Minute errors in placing the laser on the three targets caused the estimated configuration parameters to vary greatly. This error causes distortions in the triangulation process. Fine tuning the parameters by measuring several targets and using a least squares approach to fit the answer helped reduce the distortion. The biggest difficulty here lies again in the accuracy of the target placement. Further it was found that the laser beam diameter plays a large role in the accuracy of the angle measurements. Only after focusing the beam to a point were reasonable answers achieved.

The three-dimensional scans performed very well. The distortion were less than 3%. A limitation of the system is the limited useful scanning area. A fish-eye lens would greatly increase the field of view of each SU and thus increase the scanning area.

## REFERENCES

- <sup>1</sup> Gerbig, Volker, "Laser Scanning in Industrial Robots," in *Recording Systems: High-Resolution Cameras and Recording Devices and Laser Scanning and Recording Systems*, Leo Beiser, Reimar K. Lenz, Editors, Proc. SPIE 1987, 287-295 (1993).
- <sup>2</sup> Data sheet of *Cyberware*; 1990; Cyberware Laboratory Inc., 8 Harris Court 3D, Monterey, California 93940.
- <sup>3</sup> *The PSD User Manual*. Version 1; 1989; SiTek Electro Optics, Ogardesvagen 13A, S-433 30 Partille, Sweden.
- <sup>4</sup> Press, William H., et al., *Numerical Recipes in C*. Cambridge: Cambridge University Press, 1992.
- <sup>5</sup> Junkins, John L., et al., "A Novel G&C Technology Transfer: Automated Notetaking in the Classroom of the Future," *17th Annual AAS Guidance and Control Conference*, Keystone, Colorado, February 2-6, 1994.
- <sup>6</sup> Junkins, John L., and Turner, James D., *Optimal Spacecraft Rotational Maneuvers*. New York: Elsevier Science Publishing Company Inc., 1986.
- <sup>7</sup> Technical manual of *Contraves Airbearing 50C*; 1986; Contraves Goerz Corporation, 610 Epsilon Drive, Pittsburgh, PA 15238.
- <sup>8</sup> Junkins, John L., *An introduction to optimal estimation of dynamical systems*. Alphen aan den Rijn, Netherlands: Sijthoff & Noordhoff International Publishers, 1978.
- <sup>9</sup> Technical Manual of *DT2821 Series*; 1988; Data Translation Inc., Marlboro, MO 01752-1192.

## APPENDICES

## **APPENDIX A**

### **HARDWARE LIST**

The following list contains all the hardware components used to construct the three-dimensional scanner prototype.

- two PSDs model 1L5SP from ONTRACK PHOTONICS
- two signal processing boards model OT300LS from ONTRACK PHOTOICS
- one Hewlett Packard twin low voltage power supply model 800A-2 to power the signal processing boards
- a Contraves 50C one-dimensional airbearing
- a Intel 386 computer to control the system
- two Metrabyte 24 bit high output current parallel digital interface cards model PIO-24 to allow the computer to communicate with the Contraves 50C airbearing
- one Data Translation DT2821 A/D board
- one Data Translation screw terminal panel DT707
- two scanning unit focusing lenses with a diameter of 9mm and a focal length of 9mm
- one 4mW class IIIb laser diode with a beam diameter of 6mm x 2 mm operating at a wavelength of 675nm
- one laser focusing lens with a diameter of 25mm and a focal length of 400mm
- one Maxon DC motor #RE035-071-33EAB200A to rotate the mirror
- one gold plated flat mirror size 16mm x 23mm
- one KEPCO bipolar operational power supply/amplifier to drive the DC motor



## APPENDIX B

### SOURCE CODE

The following source codes are appended alphabetically:

- airbear.h*: Header file containing all the necessary subroutines to control the airbearing. The function of each subroutines is outlined at the top of the file. Before any other subroutines are called the *configure()* subroutine must be called. *offmode()* disconnects the airbearing controls.
- calib.c*: This program performs the SU calibration for a certain target distance. After finding the zero PSD voltage position the airbearing is moved in certain increments. At each increment the airbearing position and the PSD voltage are measured and later approximated with a set of Chebyshev polynomials.
- cheby.h*: Header file containing the Chebyshev curve fitting subroutine and a subroutine to calculate a point on the curve fit.
- config.c*: This program determines the initial configuration parameters by measuring the incidence angles for three different positions. To reduce the error in positioning the laser spot on the targets, each target is measured three times.
- datbrd2.h*: Header file containing subroutines to communicate with the DT2821 A/D board. The board is initialized with *setup2821()* and terminated with *terminate2821()*. Other subroutines are explained at the top of the file.
- euler.h*: Header file containing subroutines to generate a direction cosine matrix for a rotation about the x, y, or z axis.
- recon.c*: This program reconfigures the system and fine tunes the configuration parameters. The sensitivity matrix is used to minimize the triangulation error. The iteration step size is set at 1.
- scan.c*: Main program to perform the three-dimensional scans. After measuring a profile scan the data is filtered and then stored in a dummy file. After all profile scans are taken the post-processor triangulates the data into three-dimensional coordinates.
- vector.h*: Header file containing structures and subroutines to perform three-dimensional vector and matrix algebra.

/\*

## AIRBEAR.H

This header file contains all the necessary declarations and subroutines to control the Contraves 50-C airbearing through the two MetraByte cards, Model PIO-24. Much of the code is based on work done by Laurie Wittig in her University Undergraduate Fellow in 1990/91. All subroutines declared are listed below.

written by: Hanspeter Schaub  
Date: August 17th, 1993  
Organization: Aerospace Department, Texas A&M University

subroutines defined are:

```
configure(): configures the air bearing and the interface cards
offmode(): resets all ports and goes to offmode. Must be used
            before terminating the program.
posmode(): sets the air bearing to position mode. Must be set
            before using set_pos()
ratemode(): sets the air bearing to rate mode. Must be set
            before using set_rate()
set_pos(): sets the air bearing at a certain position
set_rate(): sets the air bearing turn at a precision rate
set_mode(): sets the airbearing in a certain mode
set_port(): writes the coarse and fine words to the coarse
            and fine addresses.
read_pos(): returns the current position as a float
read_rate(): returns the current rate as a float in degrees/second
read_port(): reads in the coarse and fine words at the addresses
            given and transforms them into a float value
check_pos(): waits until the given position is reached
handshake(): checks for a handshake between the air bearing and the
            interface card
convert(): converts a float value into the four hexadecimal words
*****/
```

```
/******
   include the necessary header files
******/
```

```
#include <conio.h>
#include <bios.h>
```

```
/******
   define the macros used
******/
```

```
#define CARD1 0x303 /* hexadecimal address of card 1 */
#define CARD2 0x353 /* hexadecimal address of card 2 */
#define PA1 0x300
#define PB1 0x301
#define PC1 0x302
#define PA2 0x350
#define PB2 0x351
#define PC2 0x352
#define CONTWORD1 0x92 /* control word to configure card 1 */
#define CONTWORD2 0x88 /* control word to configure card 2 */
#define STROBE ~0x1
#define RESET ~0x0
```

```
/******
   declare return types
******/
```

```
float read_pos();
float read_rate();
float read_port();
int flag, flagi;
```

```

/*****
    configure(): This subroutine will configure the two MetraByte cards
                and initialize all the ports.
*****/
configure()
{
    /*    configure the cards */
    outp(CARD1,CONTWORD1);
    outp(CARD2,CONTWORD2);

    /*    initialize all ports to logic low and set flags */
    outp(PC1,RESET);
    outp(PC2,RESET);
    outp(PA2,RESET);
    outp(PB2,RESET);
    flagi = inp(PC2);
    flag = flagi;
}

/*****
    offmode(): This subroutine will stop the airbearing and reset all
              the ports. It must be called before terminating the
              program.
*****/
offmode()
{
    int dath,          /* contains the high data word */
        datl;         /* contains the low data word */

    /* stop the airbearing */
    ratemode();
    set_rate(0.0);

    dath=0x0;          /* set the word for "offmode" */
    datl=0x0;

    set_mode(dath,datl);
}

/*****
    ratemode(): This subroutine sets the airbearing in the
              rate mode.
*****/
ratemode()
{
    int dath,datl;

    dath = 0x0;
    datl = 0x2;        /* set the command words to "rate mode" */

    set_mode(dath,datl);
}

/*****
    set_rate(): This subroutine will set a certain precision
              turn rate. The rate is given in degrees/second.
*****/
set_rate(float rate)
{
    int addc, /* contains the coarse command address */
        addf, /* contains the fina command address */
        fdatl, /* contains the low byte of the fine precision */
        fdath, /* contains the high byte of the fine precision */
        cdatl, /* contains the low byte of the coarse precision */
        cdath; /* contains the high byte of the coarse precision */

```

```

/*****
calculate fine and coarse word bytes from rate
*****/
convert(rate,&cdath,&cdatl,&fdath,&fdatl);

addc = 0121;      /* coarse word rate address */
addf = 0125;      /* fine word rate address */

set_port(addc,addf,cdatl,cdath,fdatl,fdath);

}

/*****
read_rate(): This subroutine reads the current rate of
the airbearing and returns it as a float.
*****/
float read_rate()
{
    int addc,addf;
    float rate;

    addc = 021;      /* address for coarse rate input */
    addf = 025;      /* address for fine rate input */

    rate = read_port(addc,addf);

    return (rate);
}

/*****
posmode(): This subroutine sets the airbearing in the
position mode.
*****/
posmode()
{
    int dath,datl;

    dath = 0x0;      /* set the command words to "position mode" */
    datl = 0x1;

    set_mode(dath,datl);
}

/*****
set_pos(): This subroutine will set the airbearing at a
certain position in degrees.
*****/
set_pos(float pos)
{
    int addc, /* contains the coarse command address */
    addf, /* contains the fine command address */
    fdatl, /* contains the low byte of the fine precision */
    fdath, /* contains the high byte of the fine precision */
    cdatl, /* contains the low byte of the coarse precision */
    cdath; /* contains the high byte of the coarse precision */

    /*****
calculate fine and coarse word bytes from pos
*****/
    convert(pos,&cdath,&cdatl,&fdath,&fdatl);

    addc = 0111;      /* coarse word position address */
    addf = 0115;      /* fine word position address */

    set_port(addc,addf,cdatl,cdath,fdatl,fdath);
}

```

```

/*****
    read_pos(): This subroutine reads the current position of
                the airbearing and returns it as a float.
*****/
float read_pos()
{
    int addc, addf;
    float pos;

    addc = 011;          /* address for coarse position input */
    addf = 015;          /* address for fine position input */

    pos = read_port(addc, addf);

    return (pos);
}

/*****
    check_pos(): This subroutine waits until the given position is
                reached by the airbearing.
*****/
check_pos(pos)
float pos;
{
    int test=1;
    float x, inc = 0.0001;
    while(test) {
        x = read_pos();
        if ((x>pos-inc) && (x<pos+inc)) test = 0;
    }
}

/*****
    handshake(): This subroutine checks for a handshake between the
                air bearing and the interface cards.
*****/
handshake()
{
    while (flag==flagi)
        flag = inp(PC2);
    flag = flagi;
}

/*****
    convert(): This subroutine transforms a float number into the
                four hexadecimal integers needed to communicate
                with the data interface cards.
*****/
convert(number, cdath, cdatl, fdath, fdatl)
float number;
int *cdatl, *cdath, *fdath, *fdatl;
{
    int dl, sign = 0;
    float r;

    /* check sign of number */
    if (number<0.0) {
        sign = 1;
        number = -number;
    }

    dl = ((int)(number/100));
    r = number - dl*100;
    *cdath = dl*16;
    dl = ((int)(r/10));
    r -= dl*10;

```

```

        *cdath += d1;
        d1 = ((int)(x));
        r -= d1;
        *cdatl = d1*16;
        d1 = ((int)(x*10+.00005));
        r -= (float)(d1/10.);
        *cdatl += d1;
        d1 = ((int)(x*100+.0005));
        r -= (float)(d1/100.);
        *fdath = d1 + sign*16;
        d1 = ((int)(x*1000+.005));
        r -= (float)(d1/1000.);
        *fdatl = d1*16;
        d1 = ((int)(x*10000+.05));
        *fdatl += d1;
    }

    /*****
    read_port(): This subroutine reads the coarse and the fine words
                  from the the port and converts the words into a float.
    *****/
    float read_port(addrc, addf)
    int    addrc, addf;
    {
        int dath, datl, bits, sign, num;
        float number;

        num = 0xff00;

        /* set coarse address */
        outp(PC1, ~addrc);
        handshake();

        /* read in the coarse positon */
        dath = (~inp(PB1)) - num;
        datl = (~inp(PA1)) - num;
        number = (dath/16)*100. + (dath-(dath/16)*16)*10.;
        number += (datl/16) + (datl-(datl/16)*16)/10.;

        /* set fine address */
        outp(PC1, ~addf);
        handshake();

        /* read in fine address */
        dath = (~inp(PB1)) - num;
        datl = (~inp(PA1)) - num;

        /* check for the sign */
        bits = dath/16;
        sign = (0x1 & bits);
        dath -= bits*16;
        number += (dath-(dath/16)*16)/100.;
        number += (datl/16)/1000. + (datl - (datl/16)*16)/10000.;

        /* adjust for the sign */
        if (sign == 0)          /* number is negative */
            number = -number;

        /* reset ports */
        outp(PC1, RESET);
        outp(PA2, RESET);
        outp(PB2, RESET);

        return (number);
    }

    /*****
    set_mode(): This subroutine commands a certain mode as given
                by dath and datl.
    *****/

```

```

set_mode(int dath, int dat1)
{
    int    add;        /* stores the command address */
    add=0101;          /* set the command address */

    outp(PC1,~add);
    outp(PA2,~dat1);
    outp(PB2,~dath);
    handshake();

    outp(PC2,STROBE);
    outp(PC2,RESET);

    add = 005;
    outp(PC1,~add);
    outp(PA2,~dat1);
    outp(PB2,~dath);
    handshake();

    outp(PC2,STROBE);
    outp(PC2,RESET);

    /* reset the ports */
    outp(PC1,RESET);
    outp(PA2,RESET);
    outp(PB2,RESET);
}

/*****
    set_port(): This subroutine the coarse and fine words to the
                corresponding addresses.
*****/
set_port(int addc, int addf, int cdat1, int cdath, int fdat1, int fdath)
{
    /*****
    send coarse read word to air bearing.
    see page 32, MPACS SYSTEM INTERFACE DEFINITION
    (BCD VERSION) for the bit pattern
    *****/
    outp(PC1,~addc);
    outp(PA2,~cdat1);
    outp(PB2,~cdath);
    handshake();

    outp(PC2,STROBE);
    outp(PC2,RESET);

    /*****
    send fine rate word to airbearing
    *****/
    outp(PC1,~addf);
    outp(PA2,~fdat1);
    outp(PB2,~fdath);
    handshake();

    outp(PC2,STROBE);
    outp(PC2,RESET);

    /*****
    clear address and data output ports
    *****/
    outp(PC1,RESET);
    outp(PA2,RESET);
    outp(PB2,RESET);
}

```

/\*

## CALIB2.C

written by: Hanspeter Schaub  
 Date: January 26th, 1994  
 Organization: Texas A&M University  
 Aerospace Department

This program finds the relationship between voltages and position angle of the 1-D PSD using Chebyshev polynomials. Position the airbearing at 180 degrees and then mount the sensor on the center of the airbearing. The laser is sitting on a steady foundation aiming at the sensor. The airbearing will then turn the sensor to zero the output voltage and find the corresponding angle. This is done using a Newton method root solver. Then the airbearing rotates in the positive and negative direction in 2 degree increments and stores the corresponding voltages. All angles and voltages are stored in a data file.

```

*****/

#include <stdio.h>
#include <math.h>
#include <bios.h>
#include "datbrd2.h"
#include "airbear.h"
#include "cheby.h"
#include <string.h>

float check();
float voltage3();
float zero_PSD();

/*
  define the macros
*/
#define V_TOLERANCE 0.0001 /* voltage tolerance considered zero */
#define N_SAMPLES 600 /* # of sample voltage readings */

float func(float); /* function to interpolate the data points */

double xx[1000],yy[1000]; /* arrays for the data points */
int num; /* number of data points obtained */

main()
{
  /* declare all variables used */
  char key,
        *ch1,*ch2;
  int i,j,k,dum,
      poly, /* # of poly. coefficients */
      dist, /* distance from target to PSD */
      counter, /* array position counter */
      psd, /* number of PSD to be calibrated */
      min, /* stores the number of data points in
             the negative direction */
      max, /* stores the number of data points in
             the positive direction */
      flag; /* test flag for calibration direction */
  float pos, /* the current airbearing position */
        v, /* voltage of PSD */
        pos0, /* zero voltage position */
        DEGREE, /* calibration step size in degrees */
        error, /* gives a measure of the calibration
                 error, no actual unites */
        v1; /* temporary variables */
  double *a, /* pointer to poly. coef. array */

```



```

        conv,          /* deg to rad conversion faktor */
        *x,            /* array for voltages */
        *y,            /* array for positions */
FILE    *fp,*fp2;      /* file pointer to store all measurements */

x = (double *) calloc(5000,sizeof(double));
y = (double *) calloc(5000,sizeof(double));
ch1 = (char *) calloc(15,sizeof(char));
ch2 = (char *) calloc(15,sizeof(char));

configure();           /* configures the airbearing and the
                        two MetraByte cards */

posmode();
pos = read_pos();
set_pos(pos);
setup2821();           /* configures the DT2821 board */

/*****
Sensor information input
*****/
conv = 3.14159265359/180.;
printf("\nEnter PSD number to be calibrated: ");
scanf("%d",&psd);
printf("enter distance to target: ");
scanf("%d",&dist);
sprintf(ch1,"psd%d%d.dat",psd,dist);
sprintf(ch2,"coef%d%d.dat",psd,dist);
fp = fopen(ch1,"w");
fp2 = fopen(ch2,"w");
printf("enter order of Chebyshev poly.:");
scanf("%d",&poly);
printf("enter degree step size:");
scanf("%f",&DEGREE);
printf("PSD data is written to file %10s\n",ch1);
printf("coefficients stored in %10s\n",ch2);

/*****
Zeroing the PSD Voltage
*****/
i=0;
while (i!=1) {
    pos0 = zero_PSD(psd);
    printf("is this zero ok? (y=1;n=0) ");
    scanf("%d",&i);
}

/*****
Calibration process
*****/
printf("\n\nStarting Calibration:\n\n");
i=1;counter=0;
flag = 1;

/* press any key to stop calibration process */
while(_bios_keybrd(_KEYBRD_READY)==0) {
    pos = pos0 + i*DEGREE*flag;
    pos = check(pos);
    set_pos(pos);
    check_pos(pos);

    /* delay to let PSD voltage catch up */
    dum=0;for (k=1;k<10000;k++) dum=sin(k);

    v = voltage3(0+(psd-1)*2);      /* read the position voltage */
    v1 = voltage3(1+(psd-1)*2);     /* read the signal strength */
    if (v1 > 0.004 && fabs(v) <= MAX) {
        fprintf(fp,"%15.10e    %15.10e\n",v,tan((pos-pos0)*conv));
        printf("pos=%10.5f v=%10.5f\r",pos-pos0,v);
        counter++;
    }
}

```

```

        y[counter] = tan((pos-pos0)*conv);
        x[counter] = v;
        i++;
    }
    else {
        if (flag < 0) {
            min = i-1;
            break;
        }
        flag = -flag;
        max = i-1;
        i = 1;
    }
}

/*****
    Curve fitting
*****/
num = min+max;

/* store the data points in ascending voltage order */
for (i=1;i<=min;i++) {
    xx[min-i] = x[max+i];
    yy[min-i] = y[max+i];
}
for (i=1;i<=max;i++) {
    xx[min+i-1] = x[i];
    yy[min+i-1] = y[i];
}

a = (double *) calloc(poly+1,sizeof(double));

cheby_fit(-MAX,MAX,a,poly,func);

error = 0;
for (i=0;i<num;i++) {
    error += fabs(yy[i]-cheby_eval(-MAX,MAX,a,poly,xx[i]))*DEGREE;
}

fprintf(fp2,"%15.9f\n",pos0);
fprintf(fp2,"%d\n",poly);
for(i=0;i<poly;i++) {
    printf("coef[%1d] = %20.15e\n",i,a[i]);
    fprintf(fp2,"%20.15e\n",a[i]);
}
printf("error -> %f\n",error);

set_pos(pos0);
check_pos(pos0);

offmode();
free(a);
free(x);
free(y);
free(ch1);
free(ch2);
fclose(fp);
fclose(fp2);

return 0;
}

/*
    check():    Takes a float n and rounds it off to the fourth digit
*/
float check(n)
float n;
{
    int long num;
    num = (n*10000. + .5);

```

```

    n = num/10000.;
    return n;
}

/*
   voltage3(): Takes N_SAMPLES voltage readings from channel
               and averages them.
*/
float voltage3(channel)
int channel;
{
    float a = 0.;
    int i;
    for (i=0;i<N_SAMPLES;i++) {
        a += read_ad(channel);
    }
    a = a/N_SAMPLES;
    return a;
}

/*
   zero_PSD(): Subroutine to zero the PSD voltage
*/
float zero_PSD(psd)
int psd;
{
    float pos,pos1,pos0,v,v1,inc;

    printf("\nzeroing the PSD voltage.\n\n");
    pos = read_pos(); /* read initial airbearing position */
    v = voltage3(0+(psd-1)*2); /* read initial PSD voltage */
    printf("pos=%10.5f v=%10.5f\n",pos,v);

    /* move airbearing by +1 degree and measure new angle and voltage */
    pos1 = pos+1.;
    pos1 = check(pos1); /* check for round off errors */
    set_pos(pos1);
    check_pos(pos1);
    v1 = voltage3(0+(psd-1)*2);
    printf("pos=%10.5f v=%10.5f\n",pos1,v1);
    inc = 1./(v1-v);

    /* press any key to interrupt this newton method iteration */
    while(!_bios_keybrd(_KEYBRD_READY)==0) {
        pos = pos1 - inc*v1;
        set_pos(check(pos));
        check_pos(pos);
        v=(voltage3(0+(psd-1)*2)+voltage3(0+(psd-1)*2)
          +voltage3(0+(psd-1)*2))/3.;
        printf("pos=%10.5f v=%10.5f\n",pos,v);
        if (fabs(v)-V_TOLERANCE) {
            break;
        }
        inc = (pos-pos1)/(v-v1);
        v1 = v;
        pos1 = pos;
    }
    return pos;
}

/*
   func(): This function linearly interpolates between the measured
           data points and return a float value. used by cheby_fit().
*/
float func(float x)
{
    float a;
    int first=0;

```

```

if (x < xx[0]) {
    a = (yy[1]-yy[0])/(xx[1]-xx[0])*(x-xx[0]) + yy[0];
}
else if (x > xx[num-1]) {
    a = (yy[num-1]-yy[num-2])/(xx[num-1]-xx[num-2])*(x-xx[num-1])
        + yy[num-1];
}
else {
    while (x >= xx[first+1]) {
        first++;
    }
    a = (yy[first+1]-yy[first])/(xx[first+1]-xx[first])*(x-xx[first])
        + yy[first];
}
return a;
}

```

/\*

## CHEBY.H

Author: Hanspeter Schaub  
 Date: March 29th, 1994  
 Organization: Texas A&M University

The following functions perform a Chebyshev polynomial best fit.  
 cheby\_fit(): Calculates the chebyshev coefficients c[] for the  
 range [a,b] up to order n. The curve is given as  
 a function.

chevy\_eval(): Evaluates the actual point on a curve given the  
 value x in [a,b] and the chebyshev coefficients c[].

```

*****/
#include <stdlib.h>
#include <math.h>

```

```

#define PI 3.141592653589793
#define MAX 8.

```

```

float cheby_eval();

```

/\*

cheby\_fit(): Takes a function func and performs a n-th order chebyshev  
 polynomial curve fit on it on the bound (a,b). The  
 chebyshev coefficients are stored in c[].

```

*/
void cheby_fit(float a, float b, double c[], int n, float (*func)(float))
{

```

```

    int k, j;
    double fac,          /* 2/n */
           bpa,          /* (b+a)/2 */
           bma,          /* (b-a)/2 */
           *f,
           y,            /* x mapped into [-1,1] */
           sum;

```

```

    f = (double *) calloc(n, sizeof(double));

```

```

    bma = .5*(b-a);
    bpa = .5*(b+a);
    for (k=0; k<n; k++) {
        y = cos(PI*(k+0.5)/n);
        f[k] = (float) (*func)(y*bma+bpa);
    }

```

```

    fac = 2.0/n;
    for (j=0; j<n; j++) {
        sum = 0.;
        for (k=0; k<n; k++) {
            sum += f[k]*cos(PI*j*(k+0.5)/n);
        }
        c[j] = fac*sum;
    }

```

```

    free(f);
}

```

/\*

cheby\_eval(): Takes the chebyshev coefficients c[] up to an order  
 order of n and evaluates the f(x).

```

*/
float cheby_eval(float a, float b, double c[], int m, float x)
{
    float d=0.0,

```

```

        dd=0.0,
        sv,
        y,
        y2;
int      j;

if ((x-a)*(x-b) > 0.0) {
    printf("x not in range in routine cheby_eval\n");
    exit(1);
}
y2 = 2.0*(y-(2.0*x-a-b)/(b-a));
for (j=m-1;j>=1;j--) {
    sv = d;
    d = y2*d-dd+c[j];
    dd = sv;
}
return y*d-dd+0.5*c[0];
}

```

/\*

## CONFIG.C

written by: Hanspeter Schaub  
 Date: February 4th, 1993  
 Organisation: Texas A&M University  
 Aerospace Departement

This program will be establish the configuration of the 3-D Laser Scanning System. For each eye, it will take three angle measurements of three known points and extract the position and the orientation of the eye. The three fixed points should be located on a vertical line on the center of the airbearing an equal distance apart. Each target will be successively illuminated with a laser beam.

\*\*\*\*\*/

```
#include <stdio.h>
#include <math.h>
#include <bios.h>
#include <stdlib.h>
#include "datbrd2.h"
#include "cheby.h"

float voltage();

#define N_SAMPLES    1000
#define MAX          8.      /* maximum psd voltage allowed */

main()
{
    int      i,j,k,
             fa,          /* focal adjustment number */
             N[2];        /* # of chebyshev coefficients */
    double   angle[4][2], /* stores three angles for each SU */
             dum,         /* dummy double value */
             beta,gamma,  /* temp. variables */
             e1,e2,       /* temp. variables */
             error,       /* offset error */
             dist,        /* distance to target */
             scale1,scale2, /* weighing factors for focus adjustment */
             b,           /* distance between two fixed points */
             m[2],        /* slope of the lines */
             X[2],Z[2],    /* position of eye */
             delta0[2],    /* tangent of the zero line angle */
             **coef,       /* array for least squares coefficients */
             x[4],z[4],    /* measured coordinates of the three points */
             v[2];        /* PSD voltage */
    char     c,            /* character pointer to file name */
             *fp,          /* File pointer */
             *f2;          /* File pointer to setup file */

    setup2821();

    /*****
     * read the Chebyshev calibration coefficients
     *****/
    coef = calloc(2,sizeof(double *));
    ch = (char *) calloc(15,sizeof(char));

    for (j=0;j<=1;j++) {
        coef[j] = calloc(5,sizeof(double *));
        for (k=0;k<5;k++) {
            sprintf(ch,"coef%d%d.dat",j+1,k+4);
            fp = fopen(ch,"r");
            fscanf(fp,"%lf",&dum);
            fscanf(fp,"%d",&N[j]);
            coef[j][k] = (double *) calloc(N[j]+1,sizeof(double));
            for (i=0;i<N[j];i++) {
```

```

        fscanf(fp,"%lf",&coef[j][k][i]);
    }
    fclose(fp);
}

/*****
    take angle measurements of three fixed points
*****/
printf("enter the distance b in mm:");
scanf("%lf",&b);
printf("position 1 is the highest, position 3 the lowest!\n\n");

f2 = fopen("setup.dat","r");
for (i=1;i<=3;i++) {
    v[0] = v[1] = 0.;
    for (j=0;j<3;j++) {
        printf("place laser on position %ld\n",i);
        while(!_bios_keybrd(_KEYBRD_READY)==0) {
            printf("str1 = %10.6f str2 = %10.6f\n",voltage(1),
                voltage(3));
            for (k=1;k<=3000;k++) {
                dum = 0;
            }
            c = getch();
            printf("\n");
            v[0] += voltage(0);
            v[1] += voltage(2);
        }
        v[0] = v[0]/3.;
        v[1] = v[1]/3.;
    }

    /* measure angle for both psd eyes */
    for (j=0;j<=1;j++) {

        /* find focal adjustment number */
        fscanf(f2,"%lf",&dist);
        fa = 0;
        k = 4;
        while ((dist > k+1) && (k < 8)) {
            k++;
            fa++;
        }
        if (dist > 8.) {
            dist = 8.;
            fa--;
        }
        scale1 = k+1.-dist;
        scale2 = dist - k;

        /* calibrate voltage to the tangent of the angles */
        angle[i][j] = scale1*cheby_eval(-MAX,MAX,coef[j][fa],N[j],v[j]) +
            scale2*cheby_eval(-MAX,MAX,coef[j][fa+1],N[j],v[j]);
        printf("angle%d = %f\n",j,atan(angle[i][j]));
    }

    printf("took position %ld measurements.\n\n",i);
}
fclose(f2);

/*****
    calculate position and orientation of each PSD eye
*****/
fp = fopen("config.dat","w");
for (j=0;j<=1;j++) {
    e1 = b/(angle[1][j]-angle[2][j])*(1.+angle[1][j]*angle[2][j]);
    e2 = b/(angle[2][j]-angle[3][j])*(1.+angle[2][j]*angle[3][j]);
    printf("e1 = %f\n",e1);
    printf("e2 = %f\n",e2);
}

```



```

X[j] = (4.*b*b*(e1+e2))/(4.*b*b+(e1-e2)*(e1-e2));
Z[j] = (2.*b*(e2*e2-e1*e1))/(4.*b*b+(e1-e2)*(e1-e2));
delta0[j] = (X[j]/Z[j]-angle[2][j])/(1.+X[j]/Z[j]*angle[2][j]);
fprintf(fp,"%16.10f %16.10f %16.10f\n",X[j],Z[j],delta0[j]);
printf("PSD %ld\n",j+1);
printf("X=%fmm Z=%fmm delta0=%fdeg\n",X[j],Z[j],atan(delta0[j])*180./
3.14159265359);

}
fclose(fp);

/*****
find vertical offset from center point measurement
*****/
for (i=1;i<=3;i++) {
    for(k=0;k<=1;k++) {
        m[k] = (1.-delta0[k]*angle[i][k])/(delta0[k]+angle[i][k]);
        x[i] = (m[0]*X[0] - m[1]*X[1] + Z[1] - Z[0])/(m[0]-m[1]);
        z[i] = Z[0] + m[0]*(m[1]*(X[0]-X[1])+Z[1]-Z[0])/(m[0]-m[1]);
        printf("point %ld: x = %10.5f mm z = %10.5fmm\n",i,x[i],z[i]);
    }
    error = sqrt(pow(x[1]-x[2],2)+pow(x[3]-x[2],2));
    printf(" vertical offset: %10.5f\n",error);

    for (i=0;i<=1;i++) {
        for (k=0;k<=5;k++) {
            free(coef[i][k]);
        }
        free(coef[i]);
    }
    free(coef);
    free(ch);

    return 0;
}

/*
voltage(): Takes N_SAMPLES voltage readings from channel
and averages them.
*/
float voltage(channel)
int channel;
{
    float a = 0.;
    int i;
    for (i=1;i<=N_SAMPLES;i++) {
        a += read_ad(channel);
    }
    a = a/N_SAMPLES;
    return a;
}

```

/\*

## DATBRD2.H

written by: Hanspeter Schaub  
 Date: March 4th, 1994  
 Organization: Texas A&M University

This header file contains code to communicate with the Data Translation 2821 board.

setup2821() - configures the 2821 board to use one D/A port and zeros the voltage. Also sets up the DMA access routines.  
 set\_voltage() - commands a certain voltage at the X D/A port  
 read\_ad() - reads the analog input from channel c unfiltered  
 filter() - filters an array by averaging each point about the point before and after  
 terminate2821() - terminate the ATLAS channels

\*\*\*\*\*/

```
#include <stdio.h>
#include <conio.h>
#include "atldefs.c"
#include "atlerrs.c"
#include <stdlib.h>
#include <malloc.h>
```

/\*  
 define the macros

```
*/
#define ADCSR 0x240
#define THRCR 0x24E
#define DACSR 0x246
#define DADAT 0x248
#define SUPCSR 0x24C
#define CHANCSR 0x242
#define ADDAT 0x244
#define N_BUFFERS 1
#define BUFFER_SIZE 30
#define scan_count 2
#define n_scans 15
#define n_sens 2
#define BUFFER 100
```

```
extern int SAMPLES=15;
float *buffer;
int position;
```

AL\_CONFIGURATION configuration;

```
int channels[16]; /* channel scan list */
int gains[16]; /* gains for channels */
int *get_buffer (); /* get a usable buffer (see ATLEXSUB) */
int bufnum[N_BUFFERS]; /* storage for buffer numbers */
int *buffers[N_BUFFERS]; /* array of pointers to data buffers */
```

/\*  
 configure the D/A output port and zero it

```
*/
setup2821()
```

```
{
    int i,
```

```

        timeout;
        float
        rate;

        outpw(SUPCSR,0x0021); /* set bit 5: DAC initialize
                               /* set bit 1: board initialize */
        outpw(DACSR,0x0100); /* set bit 8: Single channel select */
        outpw(DADAT,2048); /* send zero volts to d/a channel */
        outpw(SUPCSR,0x0080); /* set bit 7: DAC Single Conversion */

        buffer = (float *) calloc(BUFFER,sizeof(float));
        position = 0;

        al_initialize(); /* Initialize the ATLAB subroutines */
        al_select_board(1); /* Select board 1, the first unit */
        al_reset(); /* Perform a reset on the device */
        dump_configuration(); /* Display the current unit configuration */

        /*
        define the channels and gains
        */
        for (i=0;i<=3;i++) {
            channels[i] = i;
            gains[i] = 1;
        }

        /*
        Set the A/D parameters
        */
        al_setup_adc(INTERNAL_TRIGGER+INTERNAL_CLOCK,scan_count,
            channels,gains);

        rate = 150 * 1e3;
        al_set_frequency(rate); /* set frequency rate */

        /*
        Allocate some data arrays, declare them to ATLAB, and link them
        onto the Buffer Transfer List.
        */
        for (i=0; i<N_BUFFERS; i++) {
            if ( buffers[i] = get_buffer ( BUFFER_SIZE ) == NULL )
                printf ("ERROR -- cannot allocate buffer %d\r\n", i );
            else {
                al_declare_buffer ( &bufnum[i], buffers[i], BUFFER_SIZE );
                al_link_buffer ( bufnum[i] );
            }
        }

        al_set_timeout (3);
    }

    /*
    set_voltage(): command a voltage v from D/A port x
    */
    set_voltage(double vol)
    {
        int v;
        v = (vol/5.*2048);
        /* check bounds of demanded voltage */
        if (v>2047) v=2047;
        if (v<-2048) v=-2048;

        outpw(SUPCSR,0x0020); /* set bit 5: DAC initialize */
        outpw(DADAT,2048+v); /* send the voltage */
        outpw(SUPCSR,0x0080); /* set bit 7: Single DAC conversion */
    }

```

```

/*
    read_ad():   reads the analog input from channel and returns a integer value
                  between -10 and 10 using the ATLAB routines.
*/
float read_ad(channel)
int channel;
{
    float a;
    unsigned value;

    al_adc_value(channels[channel],gains[channel],&value);

    a = ((int)value-2048)/204.8;

    return a;
}

/*
    read_ad2():  reads the analog input from channel and returns a integer value
                  between -10 and 10.
*/
double read_ad2(int channel)
{
    double a;

    outpw(SUPCSR,0x2240); /* set bit 13: Clear DMA Done
                           set bit 9: use buffer A
                           set bit 6: A/D initialize */
    outpw(CHANCSR,0x8000); /* set bit 15: LLE (load list enabled)
                           set bits 3-0: 0000 Number of RAM entries
                           */
    outpw(ADCSR,0x0200+channel); /* set bit 9: A/D clock enable
                                  set bits 5-4: 00 (gain select for
                                  voltage range -10 to 10)
                                  set bits 3-0: 0000 channel select */
    outpw(CHANCSR,0x0000); /* clear bit 15: LLE */
    outpw(SUPCSR,0x0010); /* set bit 4: Preload Multiplexer */

    /* monitor the MUXBUSY (bit 8 of ADCSR) */
    while ((inpw(ADCSR) & 256) != 0) {}

    outpw(SUPCSR,0x0008); /* set bit 3: Software Trigger */

    /* monitor A/D DONE (bit 7 of ADCSR) */
    while ((inpw(ADCSR) & 128) != 128) {}

    a = ((int)inpw(ADAT)-2048)*10./2048.;
    return a;
}

/*
    terminate2821():  Terminate the 2821 controls.
*/
terminate2821()
{
    int i;

    outpw(DADAT,2048); /* send zero volts to d/a channel */
    outpw(SUPCSR,0x0080); /* set bit 7: DAC Single Conversion */

    for (i=0;i<N_BUFFERS;i++) {
        free(buffers[i]);
    }
    free(buffer);

    al_terminate(); /* terminate atlab channels */
}

```

```

/*
    filter():    Takes an array of MAX_NUM size and filters the first SIZE
                  elements by averages each element with
                  the +/- SMOOTH elements.
*/
filter(a, size, smooth)
float *a;
int size;
int smooth;
{
    float *b;
    int count,pos,i;

    b = (float *) calloc(size+2,sizeof(float));

    /* filter b and store result in a */
    for (pos=0;pos<size;pos++) {
        b[pos] = 0.;
        count = 0;
        for (i=pos-smooth;i<=pos+smooth;i++) {
            if (in_buffer(i,size)) {
                b[pos] += a[i];
                count++;
            }
        }
        b[pos] = b[pos] / count;
    }

    /* copy b into a */
    for (i=0;i<size;i++) {
        a[i] = b[i];
    }

    free(b);
}

/*
    in_buffer():    Makes sure the K is within 0 and SIZE
*/
int in_buffer(k,size)
int k,size;
{
    if ((k < 0) || (k>=size)) {
        return 0;
    }
    return 1;
}

```

```

/*
 *      EULERANGLES.H
 */

/* header file containing subroutines which construct the transformation
   matrixes for the Euler angles.
   all rotations are made according to the right hand rule!
*/

/* declare the subroutines defined in this header */
struct matrix3 xrot(double angle);
struct matrix3 yrot(double angle);
struct matrix3 zrot(double angle);

/*
 * Listing of the subroutines
 */

/*
 * xrot():Returns a direction cosine matrix for a rotation about the x-axis
 */
struct matrix3 xrot(double angle)
{
    struct matrix3 m;
    m.r1.x = 1.0;
    m.r1.y = m.r1.z = m.r2.x = m.r3.x = 0.0;
    m.r2.y = m.r3.z = cos(angle);
    m.r2.z = sin(angle);
    m.r3.y = -sin(angle);

    return m;
}

/*
 * yrot():Returns a direction cosine matrix for a rotation about the y-axis
 */
struct matrix3 yrot(double angle)
{
    struct matrix3 m;
    m.r2.y = 1.0;
    m.r2.x = m.r2.z = m.r1.y = m.r3.y = 0.0;
    m.r1.x = m.r3.z = cos(angle);
    m.r1.z = -sin(angle);
    m.r3.x = sin(angle);

    return m;
}

/*
 * zrot():Returns a direction cosine matrix for a rotation about the z-axis
 */
struct matrix3 zrot(double angle)
{
    struct matrix3 m;
    m.r3.z = 1.0;
    m.r3.x = m.r3.y = m.r2.z = m.r1.z = 0.0;
    m.r2.y = m.r1.x = cos(angle);
    m.r2.x = -sin(angle);
    m.r1.y = sin(angle);

    return m;
}

```

/\*

## RECON.C

written by: Hanspeter Schaub  
 Date: May 10th, 1994  
 Organisation: Texas A&M University  
 Aerospace Engineering Department

Due to the sensitive nature of the configuration program, this reconfiguration program was developed. Three known points are measured and then compared to the known solution. The error is used along with the sensitivity matrix to calculate an improved sensor configuration.

\*\*\*\*\*/

```
#include <stdio.h>
#include <math.h>
#include <bios.h>
#include <stdlib.h>
#include "datbrd2.h"
#include "cheby.h"

float voltage();
void LU();

#define N_SAMPLES 1000
#define MAX 8. /* maximum psd voltage allowed */

main()
{
    int    i,ii,j,k,n,
           fa,          /* focal adjustment number */
           poly,        /* order of inverse matrix */
           N[2];        /* # of chebyshev coefficients */

    double X[2],Z[2],   /* position of eye */
           delta0[2],   /* tangent of the zero line angle */
           dum,
           error,        /* average of x,z errors */
           ***coef,     /* array for least squares coefficients */
           dist[2],     /* distance to target */
           scale1,scale2, /* weighing factors for focus adjustment */
           **m,         /* slope of the lines */
           **v,         /* PSD voltage2 */
           *xr,*zr,     /* real position coordinates */
           *x,*z,       /* measured position coordinates */
           **angle,     /* stores three angles for each psd eye */
           **matrix,    /* matrix used during calculations */
           **a,         /* sensitivity matrix */
           *dx2,        /* position error vector */
           *dc,         /* Transpose(A).dx */
           alpha,       /* configuration parameter error vector */
           s,sp;        /* iteration scale */

    char   c,
           *ch;
    FILE   *fp;          /* character pointer to file name */

    setup2821();

    /*****
       allocate memory for arrays
       *****/
    printf("enter # of points: ");
    scanf("%d",&poly);
    m = calloc(poly+1,sizeof(double *));
    v = calloc(poly+1,sizeof(double *));
```

```

angle = calloc(poly+1,sizeof(double *));
for (i=0;i<poly+1;i++) {
    m[i] = (double *) calloc(2,sizeof(double));
    v[i] = (double *) calloc(2,sizeof(double));
    angle[i] = (double *) calloc(2,sizeof(double));
}
xr = (double *) calloc(poly,sizeof(double));
zr = (double *) calloc(poly,sizeof(double));
x = (double *) calloc(poly,sizeof(double));
z = (double *) calloc(poly,sizeof(double));

poly = 2*poly;
A = calloc(poly+1,sizeof(double *));
for (i=0;i<=poly;i++) {
    A[i] = (double *) calloc(6+1,sizeof(double));
}
matrix = calloc(6+1,sizeof(double *));
for (i=0;i<=6;i++) {
    matrix[i] = (double *) calloc(6+1,sizeof(double));
}
dx = (double *) calloc(poly+1,sizeof(double));
dx2 = (double *) calloc(6+1,sizeof(double));
dc = (double *) calloc(6+1,sizeof(double));

/*****
    read the Chebyshev calibration coefficients
*****/
coef = calloc(2,sizeof(double *));
ch = (char *) calloc(15,sizeof(char));

for (j=0;j<=1;j++) {
    coef[j] = calloc(5,sizeof(double *));
    for (k=0;k<5;k++) {
        sprintf(ch,"coef%d%d.dat",j+1,k+4);
        fp = fopen(ch,"r");
        fscanf(fp,"%lf",&dum);
        fscanf(fp,"%d",&N[j]);
        coef[j][k] = (double *) calloc(N[j]+1,sizeof(double));
        for (i=0;i<N[j];i++) {
            fscanf(fp,"%lf",&coef[j][k][i]);
        }
        fclose(fp);
    }
}

/*****
    read old configuration parameters
*****/
fp = fopen("config.dat","r");
for (i=0;i<2;i++) {
    fscanf(fp,"%lf",&X[i]);
    fscanf(fp,"%lf",&Z[i]);
    fscanf(fp,"%lf",&delta0[i]);
}
fclose(fp);

/*****
    enter coordinates of the positions
    to be measured.
*****/
printf("enter coordinates of:\n");
for (i=0;i<poly/2;i++) {
    printf("x[%d]=",i+1);
    scanf("%lf",&xr[i]);
    printf("z[%d]=",i+1);
    scanf("%lf",&zr[i]);
}

/*****
    Take voltage measurements

```



```

*****/
printf("\n");
for (i=0;i<poly/2;i++) {
    for (j=0;j<2;j++) {
        v[i][j] = 0.;
    }
}
for (k=0;k<3;k++) {
    for (i=0;i<poly/2;i++) {
        printf("\nplace laser spot on x=%7.2f z=%7.2f\n",xr[i],zr[i]);
        printf("PRESS ANY KEY TO TAKE MEASUREMENTS\n");
        while(_bios_keybrd(_KEYBRD_READY)==0) {}
        c = getch();
        for (j=0;j<2;j++) {
            v[i][j] += voltage(j*2);
        }
    }
}
for (i=0;i<poly/2;i++) {
    for (j=0;j<2;j++) {
        v[i][j] = v[i][j]/3.;
    }
}

alpha = 1.;
error = 10.;
while (error > .0001) {
    /*****
    take angle measurements and
    calculate the positions
    *****/
    for (i=0;i<poly/2;i++) {
        dist[0] = dist[1] = 6.;
        for (j=0;j<1;j++) { /* loop for focus iteration */
            /* measure angle for both psd eyes */
            for (ii=0;ii<2;ii++) {
                /* find focal adjustment number */
                fa = 0;
                k = 4;
                while ((dist[ii] > k+1) && (k < 8)) {
                    k++;
                    fa++;
                }
                if (dist[ii] > 8.) {
                    dist[ii] = 8.;
                    fa--;
                }
                scale1 = k+1.-dist[ii];
                scale2 = dist[ii] - k;

                /* calibrate voltage to the tangent of the angles */
                angle[i][ii] = scale1*cheby_eval(-MAX,MAX,coef[ii][fa],N[ii],v[i][ii]) + scale2*cheby_eval(-MAX,MAX,coef[ii][fa+1],N[ii],v[i][ii]);

                m[i][ii] = (1.-delta0[ii]*angle[i][ii])/
                    (delta0[ii]+angle[i][ii]);
            }
        }

        /* calculate measured position */
        x[i] = (m[i][0]*X[0]-m[i][1]*X[1]+Z[0])/(m[i][0]-m[i][1]);
        z[i] = Z[0]+m[i][0]*(m[i][1]*(X[0]-X[1])+Z[1]-Z[0])/
            (m[i][0]-m[i][1]);

        /* update distance estimate in inches */
        for (ii=0;ii<2;ii++) {
            dist[ii] = sqrt(pow(x[i]-X[ii],2)+pow(z[i]-Z[ii],2))/25.4;
        }
    }
}

```

```

        printf("x%d=%8.3f z%d=%8.3f ", i, x[i], z[i]);
    }
    printf("\n");

    /*****
    Calculate position error vector
    *****/
    error = 0.;
    for (i=0; i<poly/2; i++) {
        j = 1+i*2;
        dum = xr[i] - x[i];
        error += fabs(dum-dx[1+i*2]);
        dx[1+i*2] = dum;
        dum = zr[i] - z[i];
        error += fabs(dum-dx[2+i*2]);
        dx[2+i*2] = dum;
    }

    /*****
    Calculate Sensitivity Matrix
    *****/
    for (i=0; i<poly/2; i++) {
        s = pow(sin(atan(angle[i][0])+atan(delta0[0])), 2);
        sp = pow(sin(atan(angle[i][1])+atan(delta0[1])), 2);
        A[1+2*i][1] = m[i][0]/(m[i][0] - m[i][1]);
        A[1+2*i][2] = -1./(m[i][0] - m[i][1]);
        A[1+2*i][3] = (x[i]-X[0])/(m[i][0] - m[i][1])/s;
        A[1+2*i][4] = -m[i][1]/(m[i][0] - m[i][1]);
        A[1+2*i][5] = 1./(m[i][0] - m[i][1]);
        A[1+2*i][6] = (X[1]-x[i])/(m[i][0] - m[i][1])/sp;
        A[2+2*i][1] = m[i][0]*m[i][1]/(m[i][0] - m[i][1]);
        A[2+2*i][2] = -m[i][1]/(m[i][0] - m[i][1]);
        A[2+2*i][3] = ((z[i]-Z[0])/(m[i][0]-m[i][1])-(z[i]-Z[0])/m[i][0])/s;
        A[2+2*i][4] = -m[i][0]*m[i][1]/(m[i][0] - m[i][1]);
        A[2+2*i][5] = m[i][0]/(m[i][0] - m[i][1]);
        A[2+2*i][6] = (Z[0]-z[i]-m[i][0]*X[0]-X[1])/(m[i][0] - m[i][1])/sp;
    }

    /*****
    Calculate dx2 = Transpose(A).dx
    *****/
    for (i=1; i<=6; i++) {
        dx2[i] = 0.;
        for (j=1; j<=poly; j++) {
            dx2[i] += A[j][i]*dx[j];
        }
    }

    /*****
    Calculate matrix = Transpose(A).A
    *****/
    for (i=1; i<=6; i++) {
        for (j=1; j<=6; j++) {
            matrix[i][j] = 0.;
            for (k=1; k<=poly; k++) {
                matrix[i][j] += A[k][i]*A[k][j];
            }
        }
    }

    /*****
    Solve linear set of equations using Cholesky LU decomposition
    *****/
    LU(6, matrix, dx2, dc);

    /*****
    Adjust old configuration parameters
    *****/
    for (i=0; i<2; i++) {
        X[i] += alpha*dc[1+i*3];
    }

```

```

        Z[i] -= alpha*dc[2+i*3];
        delta0[i] -= alpha*dc[3+i*3];
    }

}

printf("\n\nNew configuration:\n");
for (i=0;i<2;i++) {
    printf("X[%ld]= %f Z[%ld]= %f delta0[%ld]= %f\n",i,X[i],i,
        Z[i],i,delta0[i]);
}
printf("Do you want to store this result? (Yes=1;No=0)  ");
scanf("%ld",&i);
if (i==1) {
    fp = fopen("config.dat","w");
    for (j=0;j<2;j++) {
        fprintf(fp,"%16.12f %16.12f %16.12f\n",X[j],Z[j],delta0[j]);
    }
    fclose(fp);
    printf("Result stored.\n");
}
else {
    printf("Result not stored.\n");
}

/*****
    free pointers
*****/
for (i=0;i<=1;i++) {
    for (k=0;k<5;k++) {
        free(coef[i][k]);
    }
    free(coef[i]);
}
free(coef);
free(ch);
for (i=0;i<=poly;i++) {
    free(A[i]);
}
for (i=0;i<=6;i++) {
    free(matrix[i]);
}
free(matrix);
free(A);
free(dx);
free(dx2);
free(dc);
free(x);
free(z);
free(xr);
free(zr);
for (i=0;i<=poly/2;i++) {
    free(m[i]);
    free(v[i]);
    free(angle[i]);
}
free(m);
free(v);
free(angle);

return 0;
}

/*
    voltage(): Takes N_SAMPLES voltage readings from channel
               and averages them.
*/
float voltage(channel)

```

```

int channel;
{
    float a = 0.;
    int i;
    for (i=1;i<=N_SAMPLES;i++) {
        a += read_ad(channel);
    }
    a = a/N_SAMPLES;
    return a;
}

/* LU():
subroutine uses a LU decomposition to solve the linear set of
equations, to save memory the the L and U matrix are stored in
the augmented A matrix */
void LU(int n,double **a,double *b,double *alpha)
{
    int i,j,k,m,ii,ipl,jml,iml;
    double **A,sum,max,ab;

    /* allocate memory */
    A = calloc(n+1,sizeof(double *));
    for (i=0;i<=n;i++) {
        A[i] = (double *) calloc(n+2,sizeof(double));
    }

    /* make augmented a matrix */
    for (i=1;i<=n;i++) {
        A[i][n+1] = b[i];
        for (j=1;j<=n;j++)
            A[i][j] = a[i][j];
    }

    /* search first column of matrix for largest element */
    m=1;
    max=fabs(A[1][1]);
    for (i=2;i<=n;i++) {
        if (fabs(A[i][1]) > max) {
            max=fabs(A[i][1]);
            m=i;
        }
    }

    if (max == 0) {
        printf(" no unique solution exists \n");
        exit(1);
    }

    if (m != 1) {
        for (j=1;j<=n+1;j++) {
            max = A[m][j];
            A[m][j] = A[1][j];
            A[1][j] = max;
        }
    }

    /* calculate first row of the new matrix */
    for (j=2;j<=n+1;j++)
        A[1][j] = A[1][j]/A[1][1];

    /* calculate the ith column from A[i][i] to A[n][i] */
    for (i=2;i<=n;i++) {
        j=i;
        for (ii=j;ii<=n;ii++) {
            sum=0.0;
            jml=j-1;
            for (k=1;k<=jml;k++)
                sum = sum + A[ii][k]*A[k][j];
            A[ii][j] = A[ii][j] - sum;
        }
    }
}

```

```

if (i != n) {
    /* search for largest A[ii][j] value from A[i][i]
       to A[n][i] */
    m=i;
    max=fabs(A[i][i]);
    ip1=i+1;
    for (ii=ip1;ii<=n;ii++) {
        ab = fabs(A[ii][i]);
        if (ab > max) {
            max = ab;
            m = ii;
        }
    }

    if (max == 0.0) {
        printf(" no solution m=%d\n",m);
        exit(1);
    }

    if (m != i) {
        for (j=1;j<=n+1;j++) {
            max = A[m][j];
            A[m][j] = A[i][j];
            A[i][j] = max;
        }
    }

    /* calculate the new elements of the ith row from A[i][i+1]
       to A[i][n+1] */
    ip1 = i+1;
    for (j=ip1;j<=n+1;j++) {
        sum = 0.0;
        iml = i-1;
        for (k=1;k<=iml;k++)
            sum = sum + A[i][k]*A[k][j];
        A[i][j] = (A[i][j]-sum)/A[i][i];
    }
}

/* solve for alpha by back substitution */
alpha[n] = A[n][n+1];
for (i=1;i<=n-1;i++) {
    sum = 0.0;
    for (j=n-i+1;j<=n;j++)
        sum = sum + A[n-i][j]*alpha[j];
    alpha[n-i] = A[n-i][n+1] - sum;
}

for (i=0;i<=n;i++) {
    free(A[i]);
}
free(A);
}

```

/\*

## SCAN.C

written by: Hanspeter Schaub  
 Date: February 9th, 1993  
 Organisation: Texas A&M University  
 Aerospace Departement

This is the main 3-D scanning program. It performs profile scans of an object on an airbearing and then rotates the airbearing slightly. The "PSD eye" calibration and configuration data are read in from previously established data files. (see CALIB.C and CONFIG.C)

```

*****/

#include <stdio.h>
#include <math.h>
#include <bios.h>
#include <stdlib.h>
#include <string.h>
#include <datbrd2.h>
#include "airbear.h"
#include "vector.h"
#include "euler.h"
#include "cheby.h"

/*****
    define the macros
*****/
#define MAX      8.
#define MIN_SIGNAL 0.004
#define MAX_SCAN 300

float check();

main()
{
    int    i,j,k,ii,jj,kk,
           fa,
           N,
           FIRST,
           poly[2],
           smooth,
           time,
           N_scan;
    float  pos0,
           pos,
           dum,dum1,
           inc,
           scale1,scale2,
           sl,s2,
           dist[2],
           theta,
           motor,
           vol[2],
           *vl,*v2,
           scan_angle;
    double **coef,
           alpha[2],
           m[2],
           X[2],Z[2],
           R0,
           delta0[2];
    char   c,
           *ch;
    struct vector r1,
               r2,
               r,
               /* focus adjustment factor */
               /* # of profile scans to be performed */
               /* flag indicating first profile point */
               /* # of chebyshev coef. */
               /* smoothing factor of the digital filter */
               /* counter for elapsed time */
               /* # of points per profile scan */
               /* initial airbearing position */
               /* actual airbearing position */
               /* airbearing motion increment */
               /* weighing factors for focus adjustment */
               /* stores signal strengths */
               /* distance to target in inches */
               /* angle from initial position */
               /* motor voltage */
               /* stores the current PSD voltages */
               /* arrays to store the PSD voltages */
               /* scan angle to be performed */
               /* poly. coef. for PSD calibration */
               /* incidence angles */
               /* slope of incident light */
               /* arrays for PSD position */
               /* radius of circular camera flight path */
               /* array for PSD orientation */
               /* stores the output file name */
               /* camera frame vector of obs. point */
               /* inertial vector of obs. point */
               /* inertial position of camera */

```

```

        rc; /* camera frame vector of camera */
struct matrix3 C; /* camera orientation matrix */
FILE *fp,*fp2;

/*****
    read the Chebyshev coefficients
*****/
ch = (char *) calloc(15,sizeof(char));
coef = calloc(2,sizeof(double *));
for (j=0;j<=1;j++) {
    coef[j] = calloc(5,sizeof(double *));
    for (k=0;k<5;k++) {
        sprintf(ch,"coef%d%d.dat",j+1,k+4);
        fp = fopen(ch,"r");
        fscanf(fp,"%lf",&R0); /* read in an unused variable */
        fscanf(fp,"%d",&poly[j]);
        coef[j][k] = (double *) calloc(poly[j]+1,sizeof(double));
        for (i=0;i<poly[j];i++) {
            fscanf(fp,"%lf",&coef[j][k][i]);
        }
        fclose(fp);
    }
}

/*****
    read the PSD position,orientation
*****/
fp = fopen("config.dat","r");
for (j=0;j<=1;j++) {
    fscanf(fp,"%lf",&X[j]);
    fscanf(fp,"%lf",&Z[j]);
    fscanf(fp,"%lf",&delta0[j]);
}
fclose(fp);

/*****
    calculate the camera parameters
*****/
rc.x = 0.5*(X[0]+X[1]);
rc.y = 0.;
rc.z = 0.5*(Z[0]+Z[1]);
R0 = rc.x;

/*****
    configure airbearing and read position
*****/
configure();
pos0 = read_pos();
posmode();
set_pos(pos0);

/*****
    configure the 2821 data board
*****/
setup2821();
set_voltage(.15); /* warm up motor */

/*****
    enter scan information
*****/
printf("The current airbearing position is: %10.6f degrees\n\n",pos0);
printf("enter scan angle in degrees:");
scanf("%f",&scan_angle);
printf("enter angle between profile scans:");
scanf("%f",&inc);
N = ((int) scan_angle/inc) + 1;
printf("enter smoothing filter factor:");
scanf("%d",&smooth);
printf("enter output file name: ");
scanf("%s",ch);

```

```

ch = strcat(ch, ".dat");

FIRST = 0;
while(FIRST != 1) {
    printf("enter scanning motor voltage:");
    scanf("%f", &motor);
    set_voltage(motor);
    printf("Is this speed ok? (Y=1/N=0)*");
    scanf("%id", &FIRST);
}

printf("\n\nThis run will scan from %10.6f deg to %10.6f deg.\n",
    pos0, pos0+scan_angle);
printf("A total of %d profile scans will be performed.\n", N);
printf("Output will be stored in file:%s\n", ch);

printf("\n\nTurn off any lights!");
printf("\nPRESS ANY KEY TO START SCANNING.\n\n");
while (_bios_keybrd(_KEYBRD_READY)==0) {};
c = getch();

/*****
    Start the 3-D scanning operation
*****/
printf("\nScanning has begun.\nPress any key to interrupt\n");
fp = fopen("dummy.dat", "w");
i = 0;
pos = pos0;
v1 = (float *) calloc(MAX_SCAN+1, sizeof(float));
v2 = (float *) calloc(MAX_SCAN+1, sizeof(float));

printf("STATUS: %6.2f%%\r", i*100./N);
while ((i<N) && (_bios_keybrd(_KEYBRD_READY)==0)) {
    fprintf(fp, "%14.9f\n", (float) (pos-pos0)*3.14159265359/180.);

    /*****
        Profile Scan
        *****/
    j = 0;
    FIRST = 1;
    time = -32000;
    while (time < MAX_SCAN) {
        v1[j] = read_ad(0);
        v2[j] = read_ad(2);
        s1 = read_ad2(1);
        s2 = read_ad2(3);
        if ((fabs(v1[j])<MAX) && (fabs(v2[j])<MAX) &&
            (s1>MIN_SIGNAL) && (s2>MIN_SIGNAL)) {
            /*if ((fabs(v1[j])<MAX) && (fabs(v2[j])<MAX)) {*/
                j++;
                if (FIRST) {
                    time = 0;
                    FIRST = 0;
                }
            }
            time++;
        }

    /*****
        Move airbearing to new position
        *****/
    i++;
    pos = pos0 + i*inc;
    pos = check(pos);
    if (pos >= 360.) pos -= 360.;
    else if (pos < 0.) pos += 360.;
    set_pos(pos);
    printf("STATUS: %6.2f%%\r", i*100./N);

    /*****

```



```

        filter voltages by central averaging
        *****/
        filter(v1,j,smooth);
        filter(v2,j,smooth);

        /******
        store voltages in file "dummy.dat"
        while airbearing is moving
        *****/
        fprintf(fp,"%d\n",j);
        for (k=0;k<j;k++) {
            fprintf(fp,"%14.8f %14.8f\n",v1[k],v2[k]);
        }
        check_pos(pos);
    }

    fclose(fp);
    free(v1);
    free(v2);

    /******
    turn scanning motor off
    *****/
    set_voltage(0.);

    /******
    Postprocessing of voltages into 3D points
    *****/
    printf("\n\nperforming postprocessing\n");
    fp = fopen("dummy.dat","r");
    fp2 = fopen(ch,"w");
    for (i=0;i<N;i++) {
        printf("STATUS: %6.2f%%\r",i*100./N);
        /******
        find inertial camera position and orientation
        *****/
        fscanf(fp,"%f",&theta);
        fscanf(fp,"%d",&N_scan);
        C = zrot(-theta);
        r.x = cos(theta)*R0;
        r.y = sin(theta)*R0;
        r.z = rc.z;

        for (j=0;j<N_scan;j++) {
            fscanf(fp,"%f",&vol[0]);
            fscanf(fp,"%f",&vol[1]);

            /******
            loop twice trough position
            calculation to adjust for focus
            *****/
            for (ii=0;ii<2;ii++) {
                if (ii==0) dist[0] = dist[1] = 6.;

                for (k=0;k<=1;k++) {
                    /* find focus adjustment number */
                    fa = 0;
                    jj = 4;
                    while ((dist[k] > jj+1) && (jj < 8)) {
                        jj++;
                        fa++;
                    }
                    if (dist[k] > 8.) {
                        dist[k] = 8.;
                        fa--;
                    }
                    scale1 = jj+1.-dist[k];
                    scale2 = dist[k]-jj;
                }
            }
        }
    }

```

```

/* find tangent of PSD incidence angle */
dum = cheby_eval(-MAX,MAX,coef[k][fa],poly[k],vol[k]);
dum1 = cheby_eval(-MAX,MAX,coef[k][fa+1],poly[k],vol[k]);
alpha[k] = scale1*dum + scale2*dum1;
/* find the slope */
m[k] = (1.-delta0[k]*alpha[k])/(delta0[k]+alpha[k]);
}

/* calculate position in camera frame */
r1.x = (m[0]*X[0] - m[1]*X[1] + Z[1] - Z[0])/(m[0]-m[1]);
r1.y = 0.;
r1.z = Z[0] + m[0]*(m[1]*(X[0]-X[1])+Z[1]-Z[0])/(m[0]-m[1]);

/* update distance estimate in inches*/
for (k=0;k<2;k++) {
    dist[k] = sqrt(pow(X[k]-r1.x,2)+pow(Z[k]-r1.z,2))/25.4;
}

/******
find light position vector in inertial coordinate frame
******/
r2 = add(r,Mdot(C,sub(r1,rc)));
fprintf(fp2,"%16.10f %16.10f %16.10f\n",r2.x,r2.y,r2.z);
}

fclose(fp);
fclose(fp2);
for (i=0;i<2;i++) {
    for (k=0;k<5;k++) {
        free(coef[i][k]);
    }
    free(coef[i]);
}
free(coef);
free(ch);

printf("\n\nSCAN PROCESS COMPLETED.\n\n");

return 0;
}

/*
check(): Takes a float n and rounds it off to the fourth digit
*/
float check(n)
float n;
{
    int long num;
    num = (n*10000. + .5);
    n = num/10000.;
    return n;
}

```

```

/*
 *      VECTOR.H
 */

/* header file defining all the vector structures and operations */

/* declare the structures used */
struct vector {
    double x;
    double y;
    double z;
};

struct matrix3 {
    struct vector r1;
    struct vector r2;
    struct vector r3;
};

/* Declare all function defined */
struct vector add(struct vector, struct vector);
struct vectorsub(struct vector, struct vector);
struct vectormult(double, struct vector);
struct vector cross(struct vector, struct vector);
double dot(struct vector, struct vector);
struct matrix3dotT(struct vector, struct vector);
double mag(struct vector);
struct vectorMdot(struct matrix3, struct vector);
struct matrix3MdotM(struct matrix3, struct matrix3);
struct matrix3transpose(struct matrix3);
struct matrix3mmult(double, struct matrix3);
struct matrix3Madd(struct matrix3, struct matrix3);
struct matrix3Msub(struct matrix3, struct matrix3);
struct matrix3tilde(struct vector);
struct matrix3inverse(struct matrix3);
double det(struct matrix3);

/*
 *      DEFINING ALL VECTOR FUNCTIONS AND OPERATIONS
 */

/* subroutine to perform the dot product of two vectors
 */
double dot(struct vector v1, struct vector v2)
{
    return (v1.x*v2.x+v1.y*v2.y+v1.z*v2.z);
}

/* subroutine to calculate the cross product of two vectors
 */
struct vector cross(struct vector v1, struct vector v2)
{
    struct vector v;
    v.x = v1.y*v2.z - v1.z*v2.y;
    v.y = v1.z*v2.x - v1.x*v2.z;
    v.z = v1.x*v2.y - v1.y*v2.x;
    return v;
}

/* subroutine to add two vectors
 */
struct vector add(struct vector v1, struct vector v2)
{
    struct vector v;

    v.x = v2.x+v1.x;
    v.y = v2.y+v1.y;
    v.z = v2.z+v1.z;
    return v;
}

```

```

)

/* subroutine to subtract two vectors
*/
struct vector sub(struct vector v1, struct vector v2)
{
    struct vector v;

    v.x = -v2.x+v1.x;
    v.y = -v2.y+v1.y;
    v.z = -v2.z+v1.z;
    return v;
}

/* subroutine to multiply a scalar by a vector
*/
struct vector mult(double k, struct vector v1)
{
    struct vector v;
    v.x = v1.x*k;
    v.y = v1.y*k;
    v.z = v1.z*k;
    return v;
}

/* subroutine to do the outer dot product of two vectors v.vT
returns a 3x3 matrix */
struct matrix3dotT(struct vector a, struct vector b)
{
    struct matrix3 m;

    m.r1 = mult(a.x,b);
    m.r2 = mult(a.y,b);
    m.r3 = mult(a.z,b);

    return m;
}

/* subroutine to find the magnitude of a vector */
doublemag(struct vector a)
{
    return sqrt(dot(a,a));
}

/* subroutine to dot a vector with a 3x3 matrix */
struct vector Mdot(struct matrix3 m, struct vector a)
{
    struct vector r;

    r.x = dot(m.r1,a);
    r.y = dot(m.r2,a);
    r.z = dot(m.r3,a);

    return r;
}

/* subroutine to find the transpose of a 3x3 matrix */
struct matrix3 transpose(struct matrix3 m)
{
    struct matrix3 t;
    t.r1 = Mdot(m,(struct vector) {1.0,0.0});
    t.r2 = Mdot(m,(struct vector) {0.1,0.0});
    t.r3 = Mdot(m,(struct vector) {0.0,1.0});
    return t;
}

/* subroutine to perform the dot product between two 3x3 matrixes */
struct matrix3 MdotM(struct matrix3 m1, struct matrix3 m2)
{
    struct matrix3 a;

```

```

    m2 = transpose(m2);
    a.r1.x = dot(m1.r1,m2.r1);
    a.r1.y = dot(m1.r1,m2.r2);
    a.r1.z = dot(m1.r1,m2.r3);
    a.r2.x = dot(m1.r2,m2.r1);
    a.r2.y = dot(m1.r2,m2.r2);
    a.r2.z = dot(m1.r2,m2.r3);
    a.r3.x = dot(m1.r3,m2.r1);
    a.r3.y = dot(m1.r3,m2.r2);
    a.r3.z = dot(m1.r3,m2.r3);

    return a;
}

/* subroutine to multiply a constant times a matrix */
struct matrix3 Mmult(double k, struct matrix3 m)
{
    m.r1 = mult(k,m.r1);
    m.r2 = mult(k,m.r2);
    m.r3 = mult(k,m.r3);

    return m;
}

/* subroutine to add two matrices */
struct matrix3 Madd(struct matrix3 m1, struct matrix3 m2)
{
    struct matrix3 m;

    m.r1 = add(m1.r1,m2.r1);
    m.r2 = add(m1.r2,m2.r2);
    m.r3 = add(m1.r3,m2.r3);

    return m;
}

/* subroutine to subtract two matrices */
struct matrix3 Msub(struct matrix3 m1, struct matrix3 m2)
{
    struct matrix3 m;

    m.r1 = sub(m1.r1,m2.r1);
    m.r2 = sub(m1.r2,m2.r2);
    m.r3 = sub(m1.r3,m2.r3);

    return m;
}

/* subroutine to return a tilde matrix */
struct matrix3tilde(struct vector v)
{
    struct matrix3 m;

    m.r1.x = m.r2.y = m.r3.z = 0.0;
    m.r1.y = -v.z;
    m.r1.z = v.y;
    m.r2.x = v.z;
    m.r2.z = -v.x;
    m.r3.x = -v.y;
    m.r3.y = v.x;

    return m;
}

/* subroutine to calculate the determinante of matrix m */
double det(struct matrix3 m)
{
    double a;

    a = -m.r1.z*m.r2.y*m.r3.x+m.r1.y*m.r2.z*m.r3.x+m.r1.z*m.r2.x*m.r3.y -

```

```

        m.r1.x*m.r2.z*m.r3.y-m.r1.y*m.r2.x*m.r3.z+m.r1.x*m.r2.y*m.r3.z;
    }
    return a;
}

/* subroutine to return the inverse of the 3x3 matrix m */
struct matrix3inverse(struct matrix3 m)
{
    struct matrix3 a;

    a.r1.x = -m.r2.z*m.r3.y+m.r2.y*m.r3.z;
    a.r1.y = m.r1.z*m.r3.y-m.r1.y*m.r3.z;
    a.r1.z = m.r1.y*m.r2.z-m.r1.z*m.r2.y;
    a.r2.x = m.r2.z*m.r3.x-m.r2.x*m.r3.z;
    a.r2.y = m.r1.x*m.r3.z-m.r1.z*m.r3.x;
    a.r2.z = m.r1.z*m.r2.x-m.r1.x*m.r2.z;
    a.r3.x = m.r2.x*m.r3.y-m.r2.y*m.r3.x;
    a.r3.y = m.r1.y*m.r3.x-m.r1.x*m.r3.y;
    a.r3.z = m.r1.x*m.r2.y-m.r1.y*m.r2.x;
    a = Mmult(1./det(m),a);

    return a;
}

```

## VITA

Hanspeter Schaub is the son of Hanspeter and Margarita Schaub of Fuellinsdorf, Switzerland. Hanspeter received the Matura Typus-C from the Gymnasium Liestal in Switzerland in 1987. After fulfilling his military duties with the Swiss army he went to Texas A&M University in 1988. He earned a B.S. degree in aerospace engineering in spring of 1992. Staying with Texas A&M he studied under Dr. J.L. Junkins toward his master's degree with an emphasis in dynamics and controls. Hanspeter will continue on at Texas A&M University to work towards a doctorate degree. His permanent mailing address is Hanspeter Schaub, Giebenacherstrasse 73, 4414 Fuellinsdorf, BL, Switzerland.