THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF MECHANCIAL AND NUCLEAR ENGINEERING

DESIGN, CONSTURCTION, AND TESTING OF AN INSTUMENTENTED MULTI-JOINTED ARM AND COMPUTER SOFTWARE FOR MEASUREMENT AND VISUALIZATION OF 3D COORIDANATES ON LARGE OBJECTS

KEITH A. BOURNE

Spring 2004

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree in Mechanical Engineering
with honors in Mechanical Engineering

Approved: _____ Data: _____
Dr. Sean Brennan
Thesis Supervisor

Approved: _____ Data: _____
Dr. Domenic A. Santavicca
Honors Adviser in Mechanical Engineering

Table of Contents

Section 1: Abstract

A seven joint armature with rotary encoders at each joint was designed, constructed, and tested. The objective was to build a device that could be used to create 3D computer models of vehicles for use in visualization of vehicle dynamics and demonstration purposes. Two system types were evaluated: an instrumented armature, and a differential GPS based system. Both systems were evaluated for practicality, the inherent error in the system, and system cost. An instrumented armature was determined to be more practical from a usability and cost standpoint.

The mechanical potions of the device were machined on campus, and all sensors were off-the-shelf hardware. Data processing was accomplished by a DSP used in conjunction with a laptop. Matlab code controlled data capture and included programs to create 3D meshes from vehicle point clouds. Testing of the device showed satisfactory operation of the mechanical, electronic, and software portions of the system. RMS displacement error was measured at 1.49 inches. Also, a field test showed that the device was capable of creating a 3D model of one side of a car. This device will be used hereafter to measure vehicles and roadway profiles at the Pennsylvania Transportation Institute test track.

Section 2: Introduction

Purpose of this Work:

Vehicle dynamics and control is an important field of study that affects the everyday lives of many people. Understanding the motion of vehicles and the effects of various control schemes has been gained through a combination of real-world tests and analytical dynamic models. Data from these tests or models are numerical in nature, taken from instruments and models that are ultimately mathematical. Consequently, the information about vehicle dynamics can be inconvenient to visualize. The solution, of course, is to create a graphical vehicle model, and use that model to visually describe vehicle dynamics. Creating an accurate vehicle model within a computer, one that may be set in any desired position and animated, is the topic concern of this project.

There are several ways in which a vehicle model may be created: A model can be based upon original vehicle specifications or CAD files. However, such information can be difficult to obtain, perhaps impossible if the vehicle of interest is a recent model. A model could also be created through the work of a 3D computer graphic artist, but this method is slow and the model ultimately would only be as accurate as the perception. Thus, for a reasonably accurate model of a vehicle, one obtained without the need to query manufactures for information, a obvious solution is to directly measure the vehicle, the approach taken in this study.

Existing Tools and Methods:

There are multiple tools available for making detailed measurements of objects for use in constructing 3D models. Perhaps the most intuitive tools, from the standpoint

of a mechanical engineer, are digitizing arms that are instrumented such that the position of a point on the arm can be readily found. The object of interest is digitized by moving the device along its surfaces. However, commercial products based on this premise are often mounted to the tops of tables and tend to be limited in the size of objects they can scan. One system that is an exception, produced by Romer Cimcore, makes use of a mobile arm that locks into predrilled cups set in a concrete floor, and can achieve accuracies of +/- 0.002 inches (Romer Cimcore). However even this device is limited in the height it can measure. It is also impractical from academic research because it requires the creation of a measurement facility. Thus, evaluation of a larger scale version of a mechanical measurement device is investigated in the project.

Alternative systems in the market exist to measure large objects such as vehicles that make use of laser scanners. For example, the FARO Laser Tracker system can measure points within 100 feet of the device with accuracies measured in micrometers (Faro Technologies Inc., Laser Tracker). These systems also tend to have high data acquisition rates. However, such laser-based systems are also notoriously expensive. For the application described in this paper - visualization as opposed to reverse engineering - such accuracy is unnecessary.

If extreme accuracy is not necessary additional options exist. A vehicle could be measured by processing several images captured at different known angles, for instance. Also, with the emergence of increasingly accurate differential GPS system, it is actually becoming somewhat practical to use these units to measure large objects. Because they are expensive and somewhat cumbersome, such units are less than ideal for the use proposed in this project.

Kinematics Linkage Overview:

Returning to mechanical systems as measurement devices, it should be noted that such units rely completely on kinematics on order to calculate the potion of the arm. A brief overview of kinematics is provided. In a kinematic measurement device, each rigid link can be considered to posses its own coordinate system. That is, a xyz system could be applied to, say; the end link of the device and the position of the end of the arm could be readily measured within this coordinate system. Finding the position of the end relative to a fixed point on the arm involves a series of coordinate transforms. That is, if the position of point A in Figure 2.1 is known relative to coordinate system 1, and coordinate system 1's orientation is known relative to coordinate system 2, and coordinate system 2's orientation is known relative to coordinate system 3, then a series of coordinate transforms may be found to get the position of point A relative to coordinate system 3. Specifically, a translation would be applied to shift the origin of system 3 over top of the origin of system 2, and then one or more rotations would be applied to make the axis of system 1 coincident with system 2. The same process would be repeated to transform from system 2 to system 1.

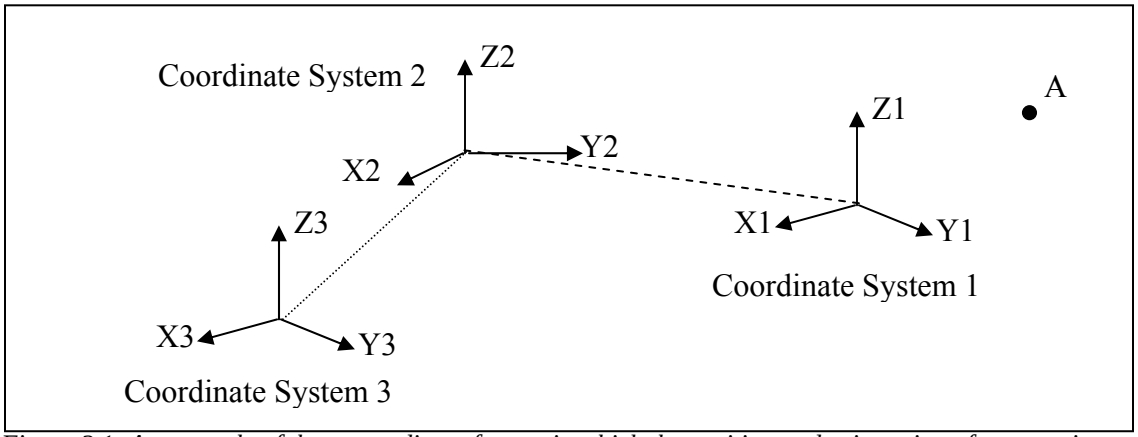


Figure 2.1: An example of three coordinate frames in which the position and orientation of any one is known relative to its adjacent frames.

In 3 dimensional space, the location of any point can be described by a vector $p = [x\ y\ z]^T$ and a rotation me be applied to that point by multiplying it by a 3x3 transformation matrix R where:

$p' = R * p$ Equation 2.1 (Ginsberg 68).

However, into order to include translation, a fourth column must be added to the transformation matrix. Also, so that translations can be combined through matrix multiplication of transformation matrices, a fourth row must be added as well. And, the position vector must be updated into the form: $p = [x\ y\ z\ 1]^T$. The translation matrix would be expressed as equation 2.2:

$$\mathbf{R}_{\mathbf{Trans}} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{\Delta X} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{\Delta Y} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{\Delta Z} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

Equation 2.2 (Ferguson 25)

Rotations about the x-axis, y-axis, or z-axis of a coordinate systems may be applied though use of the transformation matrices given as Equations 2.3 though 2.5. Note, These equations originated from Ginsberg but have been adapted to allow translations as per Ferguson.

$$\mathbf{R}_{\mathbf{RotX}} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{\cos(\theta_X)} & \mathbf{\sin(\theta_X)} & \mathbf{0} \\ \mathbf{0} & \mathbf{-\sin(\theta_X)} & \mathbf{\cos(\theta_X)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

Equation 2.3 (Ginsberg 58) (Ferguson 26)

$$\mathbf{R}_{\mathbf{RotY}} = \begin{bmatrix} \mathbf{\cos(\theta_Y)} & \mathbf{0} & \mathbf{-\sin(\theta_Y)} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{\sin(\theta_Y)} & \mathbf{0} & \mathbf{\cos(\theta_Y)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

Equation 2.4 (Ginsberg 58) (Ferguson 26)

$$\mathbf{R}_{\mathbf{RotZ}} = \begin{bmatrix} \cos(\theta_z) & \sin(\theta_z) & 0 & 0 \\ -\sin(\theta_z) & \cos(\theta_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equation 2.5: (Ginsberg 58) (Ferguson 26)

Successive coordinate transforms may be applied by applying the transformation equation successively. By performing several translation and rotations, complex geometry may be handled. Furthermore, in computer graphics, the same approach may be used to rotate a virtual solid object or cause it to translate though a virtual environment by applying the same coordinate transforms to all the vertices used to describe said object.

Overview of Meshing:

Whatever method is used to generate a cloud of data points, ultimately it is desirable to use those points to create the appearance of a solid object. This involves generating a mesh from the point cloud to define surface on the object. The mesh mathematically describes a way to connect the points with polygons such that the original geometry is closely represented.

The problem of meshing point clouds is difficult and studied form may years. Successful meshing algorithms exist, some of which are highly complex. The authors (Farin, G. et al 651) provide a good overview of meshing with regards to reverse engineering and (Remondino) provides a good overview of available meshing options.

In the limited context of this project, meshing is reduced to a 2D problem by revolving surfaces until they are viewed from an angle such that they are effectively flat.

A 2D Delaunay Tessellation is used to mesh the resulting2D surface. The tessellation is implemented by an existing Matlab function that determines triangles to connect a set of points without intersecting (The Mathworks, Matlab Documentation CD). Any triangles, produced during tessellation, that are outside user defined surface bounds are deleted. This method has the advantages of simplicity. Furthermore it does not require the use of additional commercial software and it performs reasonably well if surfaces are defined carefully.

Points on Graphic Visualization:

Once a cloud of points has been meshed, the meshed surface may be graphically displayed and animated. Multiple 3D packages exist that perform such functions, including Lightwave 3D, Maya 3D, and Truespace. Matlab also has some 3D graphics capability, although it is not a dedicated package and therefore lacks refined features.

Regardless of the package involved, 3D visualization involves positioning of a 3D object in a world coordinate system though a series of coordinate transforms, and then projecting these 3D objects onto a 2D plane for display on a computer screen (Ferguson)(Watt). Animation is accomplished by applying additional slightly different transformations to the mesh in each frame. Scripts, of inputs into transformation matrices control these animations.

Polygons within a mesh are generally shaded using flat shading, gouraud shading, or phong shading. Flat shading algorithms shade the entire surface of a polygon in the same way, producing objects with a faceted appearance. Gouraud shading calculates shading, as a function of light and surface properties at polygon vertices and interpolates

shading values elsewhere. Lastly, phong shading shades each pixel of a polygon based in a surface normal at that point and produces the best results, at the cost of computer power (Ferguson 128). Matlab has the ability to use any one of these shading algorithms, and selection of the appropriate algorithm can visually smooth out a rough surface created by somewhat inaccurate measurement.

Summary:

In summary, there exists a need to create 3D models of vehicles for use in visualization of dynamics models. The most consistent and accurate means of creating such models is to measure the vehicle of interest. Such measurements may be achieved using several systems: mechanical based systems, laser based systems, vision base systems, and even differential GPS based systems. This paper offers some consideration of a large-scale mechanical system and a differential GPS based system, although, a mechanical system was ultimately chosen. Meshing of clouds of 3D points is a subject of considerable work, however relatively simple meshing programs are available in the Matlab software used in this project. Furthermore, once an object is meshed, visualization can be carried out by several packages including Matlab, and animation can be handled by user generated scripts.

Section 3: Articulated Arm Mockup Test

Purpose:

To better understand and evaluate the feasibility of a "snake-like" digitizer arm configuration, a wooden mockup of a potential configuration was constructed and tested on a truck at the PTI (Pennsylvania Transportation Institute) test track. A primary goal was to determine if a single person could adequately manipulate such an armature, to place a stylus connected to the end of the mockup device against any point on a large vehicle parked at the facility.

Mockup Construction:

The mockup consisted of six, eight-foot long strips of wood (8' x 1.5" x 0.75"), which were connected end to end to form a 48' long snake-like armature. The first four strips, hereafter referred to as links 1 though 4, were joined to each other by drilling 5/16" holes through their ends and using ¼" bolts to connect the links together. The bolts simulated 1-DOF pinned joints. Also, links 4 though 6 were connected to each other via loops of rope run though 5/16" holes drilled through their ends. The loops of rope were intended to simulate 2-DOF joints. Because the rope was able to twist, if only to a limited degree, the actual joint showed some 3-DOF behavior.

Link 1 served as the connection to the base, and so that free end was only allowed to rotate. The free end of link 6 was attached to a mockup measurement stylus constructed from a wire coat hanger. The stylus could be bent into different shapes to find the one that worked best during testing. Near each end of link 4, sets of two swivel type caster wheels were affixed and were separated by about 2 inches. The caster wheels

were mounted to prevent the nuts from dragging on the ground and to evaluate the use of casters in the final design.

Ideally, the base would have been elevated and caster wheels would have been attached to link 2 as well. However, due to a lack of caster wheels and a suitable base, soda cans were attached to the bottom of said links instead. These cans were just the right height to keep all the links approximately level. Pieces of cardboard were attached to the bottoms of the cans to reduce the friction with the ground.

Figure 3.1: The mockup armature was assembled at the PTI test track. The arm was transported in a folded up state (upper left). Caster wheels were mounted to cross pieces, which were in turn mounted to the arm (upper right). Sodas cans, with cardboard bases, were taped to the bottom of the armature to keep the device level (lower left). The mockup was unfolded by pulling on the two links closest to the stylus (lower right).

Test Track Setup:

The day of the mockup's test, a full tractor and trailer were not available at the PTI test track. Instead, only a tractor was available. However, this was acceptable because the tractor's complex shape is the most difficult to measure of the tractor-trailer vehicle.

A full tractor-trailer could be about 52' long. Therefore, if the base of the arm was placed about ten feet away from the center of one side, then links 1-4 would be able to bring the remainder of the arm into range of any surface of one side of the tractor-trailer. To simulate this setup the base of the mockup was placed about 25' aft of the nose of the tractor and about 10' away from the side of the tractor.

Figure 3.2: The mockup armature sitting along side a tractor

Mockup Test Results:

The first thing examined, during the test, was the ability of the arm to extend from a folded position. By simply pulling on the end linkage, it was found that the entire armature unfolded itself without locking up at any point. The caster wheels worked well and did not cause the linkage to move unpredictably. Surprisingly, the improvised soda can supports also moved smoothly without tipping.

Next, the mockup arm was test for its ability to make contact with all points on a vehicle. Physical testing showed that the arm, terminated with a straight stylus, was able to touch all points along the side of the vehicle, as shown in Figure 3.3. However, points on the front of the vehicle were hard to reach with a straight stylus. In order to resolve this, the stylus was bent at a ninety-degree angle such that its tip extended perpendicular to the axis of the final link. With this modification, points on the front of the vehicle were easily reached by holding the final link parallel to the normal face of the front of the vehicle and moving the stylus in a line-by-line scanning pattern as shown in Figure 3.4. The bent stylus also made measurements of points on the side of the vehicle easier, although deep cavities, such as between the tires, became hard to reach or inaccessible.

Figure 3.3: The mockup was use to determine the feasibility of measuring points along the side of the tractor chassis (upper left) and the cab (upper right and lower left).

Figure 3.4: The arm is shown capable of measuring points along the front of the vehicle as well, provided that the stylus is bend to be at a ninety-degree angle to the axis of the link to which it is mounted.

Interestingly, it was found that during the scanability test only the first three links actually moved. The other links simply were kept in place by their own weight. This indicated that only three links would be required to measure a large vehicle, reducing the arm complexity to only two 2-DOF joints and one 1-DOF joint with five encoders total for the entire arm.

Next the ability of the armature to transverse the long the side of a tractor-trailer was tested. It was found that when pulling the end of the armature past its base the links tended to form the folded up configuration shown in Figures 3.5 and 3.6. This configuration was only transient and did not prevent transverse motion of the armature. However the force required to pull the armature out of its folded up configuration was much larger than in all previous motions and could potentially produce bending moments in the links.

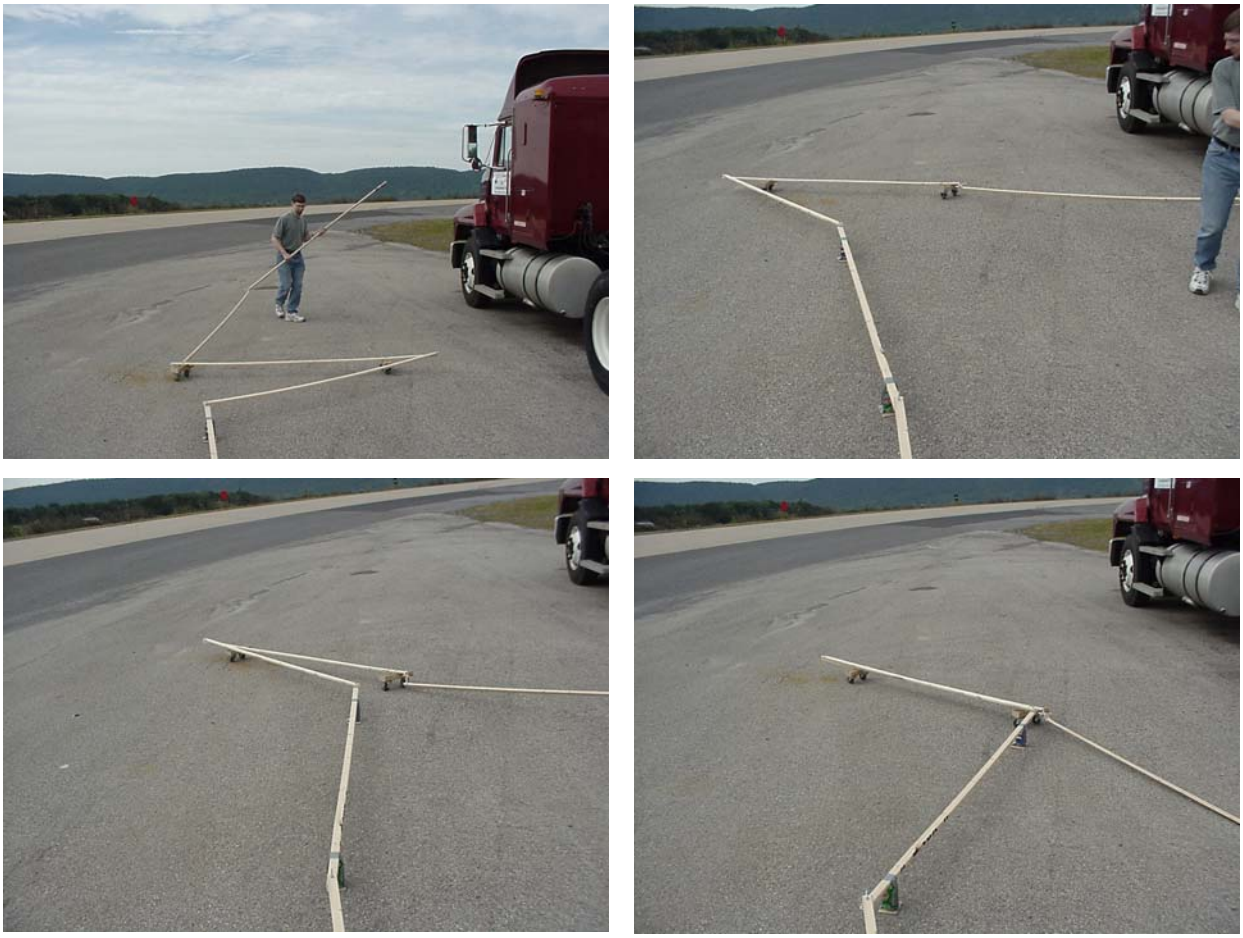


Figure 3.5: The end linkage of the armature is shown pulled towards its base in an effort to extend the armature in the opposite direction. During this motion the arm assumed a semi-folded-up configuration. The sequence of arm motion that achieved this configuration is shown, in order, upper left, upper right, lower left, and low right. By continuing to pull on the arm, this configuration unfolded itself and the arm was allowed to continue transversing.

Figure 3.6: As the end of the arm continued to be moved towards the armature base, the same folded up configuration, shown in Figure 5, appeared again, one joint closer to the base. Once, again, this configuration unfolded itself when continuous force was applied to the end linkage.

By the end of the test, it became apparent that the motion of the caster wheels over the rough pavement was producing significant vibrations in the remainder of the arm. In fact one the bolts used to attach a caster came loose and off the arm. Almost all of the other bolts were worked loose from their original finger-tight assembly. A future prototype clearly required provision for vibration in the design. For instance, lock washers or nylon lock nuts.

Conclusions:

- A light snake-like measurement arm is fairly easily to move by hand and fairly stable when not being moved.
- If the stylus is set perpendicular to the axis of the terminating linkage then in can be held parallel (and thus closer to) a vehicle being measured.
- Using caster wheels to support ground based linkages aids in moving a snake-line armature across the ground.
- Vibration imparted by the caster wheels rolling across a rough surface should be accounted for. This might be achieved by the addition of lock washers on bolts or nylon lock nuts.
- Three eight foot long linkages are adequate to measure the surfaces on one side of a tractor-truck as well as over half of the surfaces on the front of such a truck, provided that that the first linkage is connected to the base with at least a 1 degree of freedom joint and all other joints posses at least two degrees of freedom.
- Only the first three linkages of a long arm move significantly except when gross armature motions are made.

- Any design of a snake-like arm should take in account the “folded up” configuration that results when pulling the end linkage from one extended reach to another. Otherwise, pulling the arm linkage out of this configuration could induce significant moments that could bend the linkages.

Section 4: Preliminary Arm Error Evaluation

Preliminary Precision Analysis Overview:

This section is intended to evaluate the potential accuracy with which the location of the end of a multi-link armature, instrumented with angular encoders at it joints, can be found. Thus, in the first part of this section the arm configurations that are evaluated are described. Evaluation of expected error follows. Conclusions about the system are presented at the end of this section.

Evaluated Design:

The previous test of a wooden mockup armature showed that the entire side and front of a semi-truck tractor could be reached with only three 8' long links connected to each other with two approximately 2 degree of freedom (DOF) rope joints and connected to a base with a one 1-DOF joint. Furthermore, it was shown that caster wheels attached to the base link's end helped with movement and provided a means of keeping the base link level, provided that the ground was level. This simplest adequate configuration, represented in Figure 4.1, was chosen as an initial design to evaluate for the purpose of developing ballpark accuracy prediction.

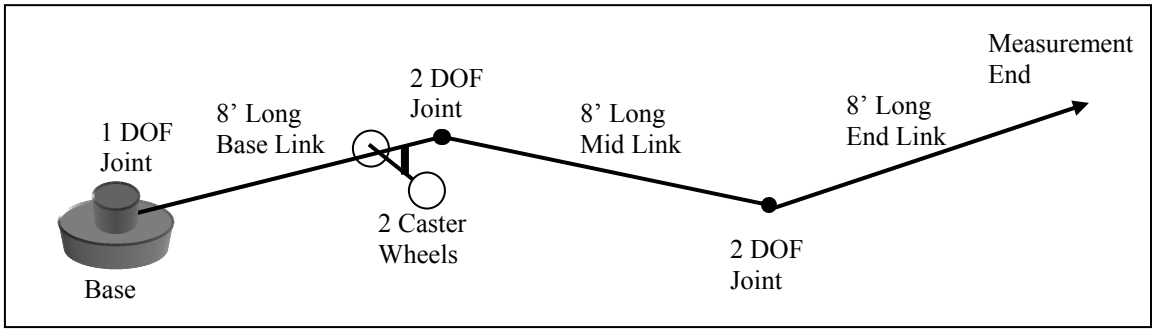


Figure 4.1: The simplest functional armature configuration found in mockup test is shown.

For simplicity of construction and modularization, it was decided that all joints in the arm would actually be 1-DOF pinned joints. 2-DOF joints would be simulated by connecting two 1-DOF joints such that the rotational axis of the joints would be arranged perpendicular to each other. Therefore, it was assumed that an arm, behaving similar to the arm depicted in figure 4.1 would make use of 5 joints, each with their own encoder.

Also, the links connecting the two 1-DOF joints, simulating a 2-DOF joint, would have to be sized such that both joints could experience a full range of motion without colliding. For a range of motion of about 315 degrees, about two feet of separation was estimated to be a good separation distance. The arm described is shown in Figure 4.2.

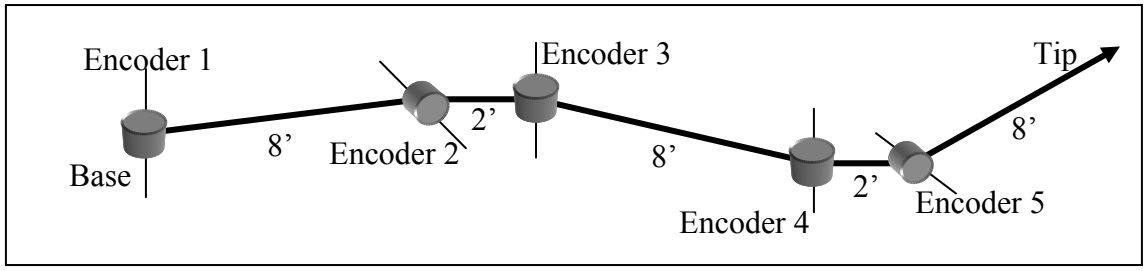


Figure 4.2: A model of a five joint armature design is shown in which each joint is allowed only one degree of freedom.

In addition to the configuration just described, as second configuration was also evaluated, which is shown in Figure 4.3. This new configuration simply possessed an additional 8' link at its base. Thus, by evaluating this configuration it would be possible to get an idea of how error should increases with arm length and complexity.

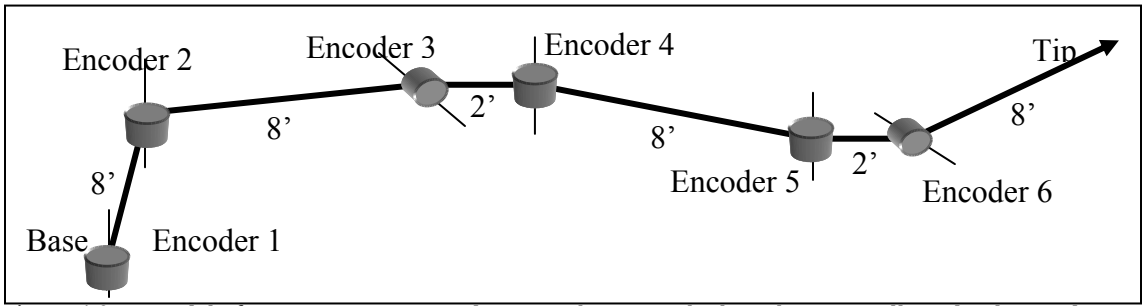


Figure 4.3: *A model of a six joint armature design is shown in which each joint is allowed only one degree of freedom. The addition base link was added to see the effects of increasing the arm's reach.*

Relative optical shaft encoders, produced by US Digital Corporation, were selected to instrument each joint. The highest resolution encoder available output 2048 counts per revolution though two channels (US Digital Corporation "E3 Optical Encoder Kit"). Through processing of the two signals the ultimate resolution of the encoder was 4 * 2048 = 8192 positions per rotation or 0.0439 degrees.

Assumed Errors:

Several sources of measurement error, due to arm construction and calibration were expected. Prior to constructing the device, and discovering the error first hand, estimates of expected errors were made. The individual expected errors were then combined to produce an estimate of overall measurement error, both random and bias. The error parameters evaluated were:

- Uncertainly in the length of each link
- Limited encoder resolution
- Twist of each link along the long axis
- Encoder bias introduced during calibration

The error values evaluated during the analysis of the 5-link configuration are given in Tables 4.1 and 4.2. Also, the error values evaluated during the analysis of the 6-link configuration are given in Tables 4.3 and 4.4 .

Table 4.1: Evaluated error sources a 5-link configuration consisted of encoder uncertainty, twist of links (assumed to be proportional to length), and error in measurement of link length.

Encoder Uncertainty (deg)	Twist Error (deg / ft)	Length Error (inch)
0.04395	0	0
0.04395	0.125	0
0.04395	0.25	0
0.04395	0	0.015625
0.04395	0.125	0.015625

Table 4.2: Evaluated Bias Errors in the 5-link configuration were assumed to consist of errors due to imperfect zeroing, of each joint during calibration.

Encoder Bias Error (deg)
0
1
0.5
0.25

Table 4.3: Evaluated error sources in a 6-link configuration consisted of encoder uncertainty, twist of links (assumed to be proportional to length), and error in measurement of link length.

Encoder Uncertainty (deg)	Twist Error (deg / ft)	Length Error (inch)
0.04395	0	0
0.04395	0.125	0.015625
0.04395	0.25	0.015625

Table 4.4: Evaluated Bias Errors in the 6-link configuration were assumed to consist of errors due to imperfect zeroing, of each joint during calibration.

Encoder Bias Error (deg)
0
1
0.5
0.25

Error Analysis:

For a system, with n inputs each having a known error, the worst case uncertainty of the system due to propagation of error is given by equation 4.1, and the sum of squares (RSS) uncertainty is given by equation 4.2 (Wheeler 160).

$$w_R = \sum_{i=1}^{n} \left| w_{xi} \frac{\partial R}{\partial x_i} \right|$$ *Equation 4.1 Max Uncertainty*

$$w_R = \left[\sum_{i=1}^{n} \left(w_{xi} \frac{\partial R}{\partial x_i} \right)^2 \right]^{1/2}$$ *Equation 4.2: RSS Uncertainty*

Therefore, in order to analyze the error in a measurement arm configuration the kinematical equations describing the system must first be derived, and from those equations the partial derivatives with respect to each variable, for which a uncertainly

estimate exists, must be calculated. Used in conjunction with the estimated uncertainties, equations 4.1 and 4.2 may then be used.

However, the derivation of kinematical equations of an arm of even three links, let alone five or six is a non-trivial task. Therefore, Matlab was used to derive the kinematical equations, in symbolic form, from a series of coordinate transforms and translations. The coordinate transformation for one 1DOF joint is explained in Figure 4.4.

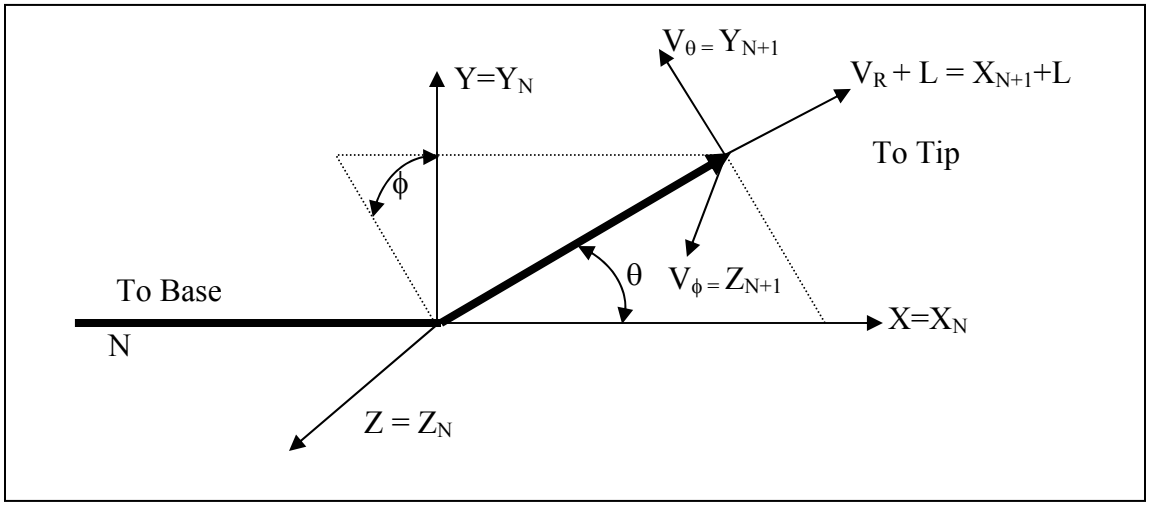


Figure 4.4: Transformation from a coordinate system on $Link_{N+1}$ into a coordinate system on $Link_N$ is shown. For a linkage of N joints were the end link is link N and the base is link 0, N transformations would be required to get from end link space to base space.

In the transformation, shown in Figure 4.4, the long axis of each link is coincident with a link fixed coordinate system in which the length of the link is added. The mathematical representation of this transformation is shown in Equation 4.3. The Matlab program "MakePositionEquation.m" applied one of this transformations, in sequence, for each joint. This program is given as Program A.1 in Appendix A.

$$\begin{bmatrix} X_N \\ Y_N \\ Z_N \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta)\cos(\varphi) & \cos(\theta)\cos(\varphi) & -\sin(\varphi) \\ \sin(\theta)\sin(\varphi) & \cos(\theta)\sin(\varphi) & \cos(\varphi) \end{bmatrix} \begin{bmatrix} X_{N+1}+L \\ Y_{N+1} \\ Z_{N+1} \end{bmatrix} = R_{Rot} \begin{bmatrix} X_{N+1}+L \\ Y_{N+1} \\ Z_{N+1} \end{bmatrix}$$

Equation 4.3: Spherical to Rectangular transform

Once the equations describing the location tip of the arm were known, in Matlab symbolic form, they were used as inputs into an additional program with took the partial derivative with respect to each variable and output the resulting formulas in a Matlab diary file. The program used to accomplish this, "ShowWeightEquation.m" is also given in Appendix A as Program A.2. The partial derivative equations were then entered into a program that used the equations to compute overall max error and RSS error based on error component input as per Equations 4.1 and 4.2. This program, "QFindPrecision.m" is given as Program A.3 in Appendix A. Lastly, bias error was estimated by pasting the kinematic equations into the program "QFindBias.m", which is given as Program A.4 at the end of this section. This program calculated the position of the arm's tip for all possible positions in which the joints were rotated to integer multiples of 45 degrees. A bias error estimate was added to a second calculation of each position, and the difference in position between a biased and unbiased output was taken as the bias error. The max bias error was the largest calculated difference.

Preliminary Precision Analysis Results:

Using the programs described, the max error and RSS errors in measurements made by both arm configurations were calculated. The expected total error, resulting from cumulative internal errors, in the 5-joint configuration is shown in Figure 4.5. And, the expected error due to encoder bias, for the same configuration, is shown in Figure 4.6. Also, the precision error inherent in measurements made by the six joint configuration, shown in Figure 4.3, was calculated and plotted in Figure 4.7.

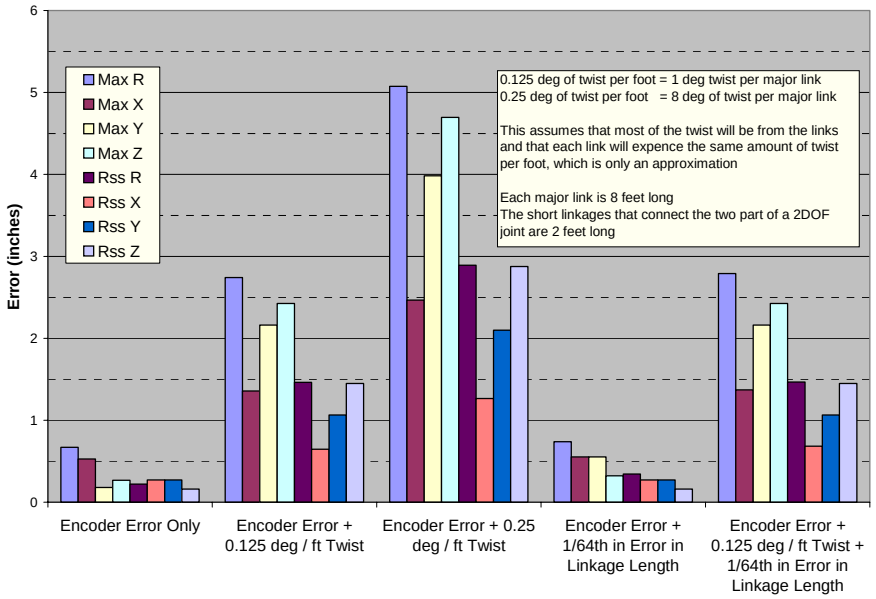


Figure 4.5: The total precision errors in measurements made by the 5-encoder arm were calculated for several assumed internal errors and plotted. Error is given in the x direction, y direction, z direction, and as displacement error R..

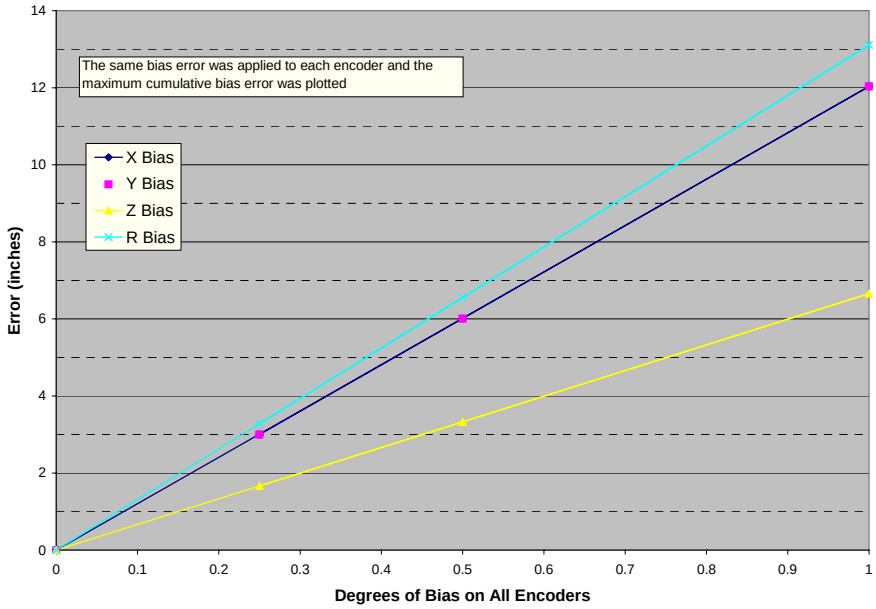


Figure 4.6: The bias error in measurements made by the 5 encoder arm are shown to present a linear trend where bias is proportional to max error. bias is given in the x direction, y direction, z direction, and as displacementr R.

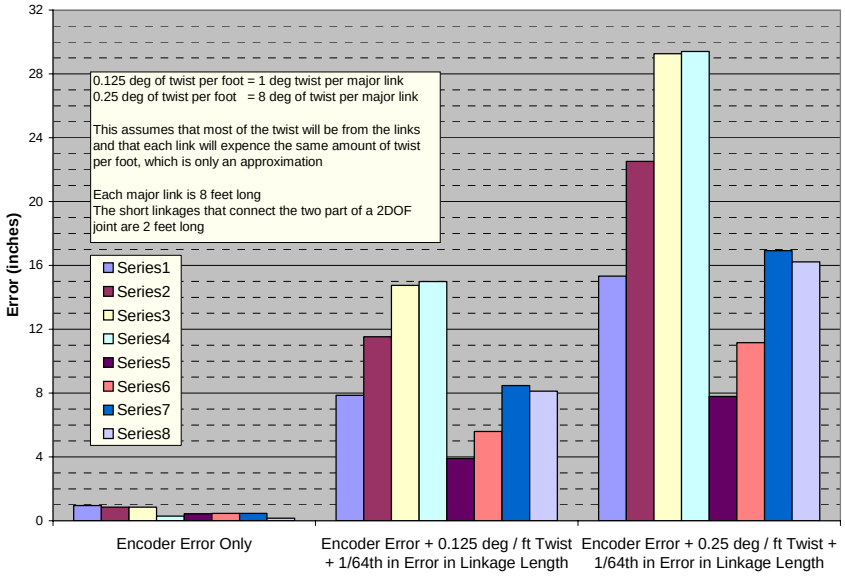


Figure 4.7: The bias error in measurements made by the 6 encoder arm are shown to present a linear trend where bias is proportional to max error. Error is given in the x direction, y direction, z direction, and as displacement error R.

Overall, these figures show that the largest source of expected precision error to be due to twist in each link. In particular if a link close the base experiences twist, a large rise in error results, as evidenced by the near 23 inch increase in expected RSS displacement error when an additional 8 foot link, connected to the base, is subjected to 0.25 degrees of twist per foot.

Conversely, the effect of adding an additional encoder appears to be minimal, increasing RSS displacement error R error by 0.2 inches when the additional encoder is close to the base. Nor does uncertainly in the length of each length appear to have a very large effect.

Preliminary Precision Analysis Conclusions:

- Twist in the links of the arm, particularly links close to the base, has a large effect on precision error.
- The effect, on precision error, of adding more encoders is minimal, and may be an acceptable tradeoff if more joints decrease twist in any link close to the base.
- Error due to uncertainty in the length any link is minimal. Thus, inordinate amounts of effort should not be spent trying to calibrate the length of each link or, alternatively, measure the length of each link.
- Bias error in the arm increases linearly with uncertainty in the calibration of each joint. Thus some procedure should be developed to calibrate each joint as accurately as possible.

Section 5: GPS Based System Evaluation

Potential GPS Based System Overview:

Given the large amounts of potential error calculated in an arm based system, it became desirable to evaluate a second possible system in which error propagation would not have such a profound effect. Furthermore, potential budget availability raised the possibility of using a high-resolution differential GPS at the Pennsylvania Transportation Institute test track. The intent of this unit is primarily to track the motion of instrumented vehicles driving along the track. Also, there is a close connected between motion tracking, and the ability to graphically visualize such motion, so a possibility existed that the unit could be shared between both tasks.

For both detailed vehicle position measurements and vehicle shape measurements, a differential GPS is required in order to achieve acceptable resolution. Under conditions where the number of satellite signals being received and the spacing of the satellites are typical, a regular GPS unit may only be able to determine a position to within 20 meters (Javed 28). By contrast a differential GPS (DGPS) unit increases its resolution by taking advantage of additional radio signals sent from a nearby base station of known latitude and longitude. Since the PTI test track has been thoroughly surveyed, much of the work required to setup such a station is already completed. The, a DGPS unit can achieve resolutions on the order of 1 to 2 centimeters (Corrsys Datron)(Javad) depending on environmental conditions and the unit selected.

Compared to regular GPS systems or the previously considered measurement arm, a DGPS system would have been very expensive. For instance, a DPGS from Javad, with 1 cm resolution can cost over $10,000 for a single receiver / base station unit,

and up to $20,000 for a moderate capability model. More involved systems, such at the RT 3003 from Datron-Corrsys (Datron-Corrsys) offers dual antennas for a resolution of 2cm and heading information good to 0.1 degree, but this system costs nearly $99,000.

Even high-resolution DGPC devices can produce poor measurements if the device is not positioned such that it can receive clear signals from several satellites. Satellite signals bouncing off the ground and nearby elevated objects would also produce interference, which might only partially be remedied through the use of filtering and a specially designed antenna (Javad).

An additional issue with a DGPS system is that short of buying two systems, or a single dual-antenna system, heading information would not be available. These are both costly options. Heading information is necessary because GPS systems measure position relative to the phase center of their antennas, which is generally close to the physical center of the antenna (Javad). The physical bulk of an antenna would insure that that actual location of points on a vehicle would not coincide with the DGPS measured point location. For instance, in the case of the Jarad MarAnt Antenna, the physical housing is 14cm x 14 cm x 7.4 cm, a size that would introduce uncertainty in contact measurements of 7 cm along the device's horizontal plane if heading is unknown.

Additional instrumentation could be used to find the orientation of the device. An inclinometer could be used to find the angle of incline of the device and either a gyrocompass or a magnetic compass could be used to find the device's heading. However, gyrocompasses tend to drift and magnetic compasses can be affected by the presence of large amounts of metal. This prevents usage around vehicles. Thus, a new

system was considered in which the location of the tip of a rod attached to the DPGS could be found without using a second antenna. This system is described below.

Single Antenna / DGPS Unit Concept:

A single antenna device would consist of a differential GPS unit antenna, and accompanying DGPS unit, attached to a rod of known dimension and any convenient shape. The tip of the rod would be made of a material such as hard rubber that would not slide easily on smooth surfaces. The DGPS antenna would be mounted on the other end. Output from the DGPS unit would be sent to a portable digital signal processor, which would analyze the output.

The system would take advantage of the fact that the accuracy of a DGPS unit is known to within a manufacturer provided confidence level. Therefore, the actual location of the measurement tip can be said to exist within a sphere surrounding the coordinates measured by the DGPS. The distance between the measured point and DGPS antenna is constant and easy to measure, therefore, the location of the tip must exist within a spherical shell surrounding the DGPS antenna (Figure 5.1). This shell would have a mean radius equivalent to rod length. The inner radius of this spherical shell would be the mean radius minus the accuracy of the DGPS unit and the outer radius would be the mean radius plus the accuracy of the DGPS unit.

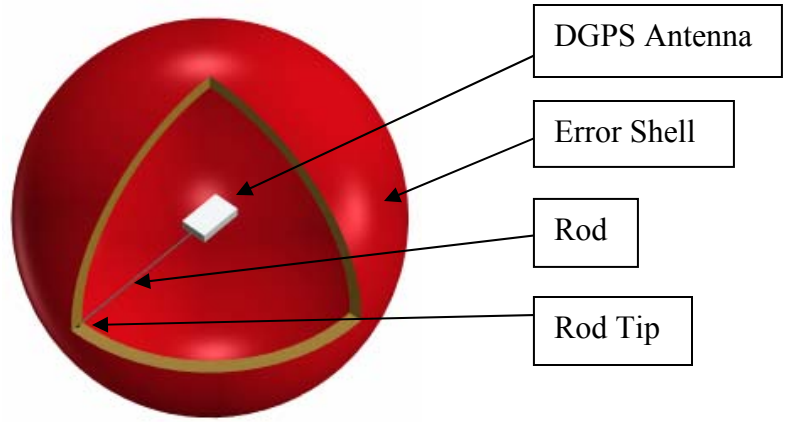


FIGURE 5.1: All possible locations for the tip of a rod, of any shape, attached to the GPS unit would exist within a spherical shell of known inner radius, outer radius, and center.

The location of the rods' tip within the shell would be found by swinging the device in two perpendicular arcs while maintaining contact between the tip of the rod and the surface being measured. This would cause the shell of possible points surrounding the device to move as well. However, the shell would be rotating around the measurement point. Thus, by calculating the location of multiple shells and calculating the location at which the most shell overlaps occur, the approximate location of the rod tip can be found. This concept is highlighted by the intersecting 2D cross sectional rings shown in Figure 5.2.

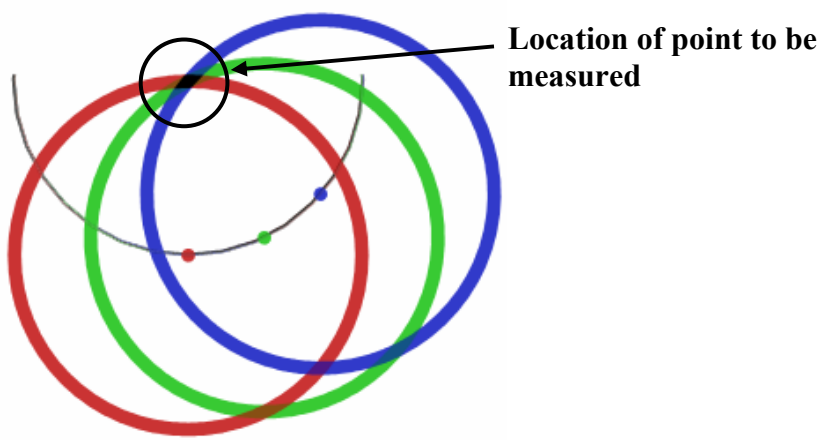


FIGURE 5.2: As the GPS Unit is moved in an arc about a point to measured, the shells of possible locations, shown here in cross section, would intersect to create a region in which the measured point exists, to a given confidence level.

The angular extent of the arcs would affect the uncertainty in the measurement. A larger angle would allow more shells to overlap, creating a smaller region in which the rod's tip should exist. For instance, an angle of about 120 degree would allow the size of the region to be no larger that the sphere of uncertainly originally introduced by the GPS device itself.

Single Antenna / DGPS Unit Computer Test:

In order to test the previous concept of finding points at the end of a rod, via an attached DGPS unit, several programs that are included in Appendix B were written. Of these programs, Program B.1 "TestFindWandTip.m" generates simulated output data from a DGPS, with a simulated normally distributed error. Program B.3 "FindWandTip.m" calculates the most likely contact point for the rod given the rod's length, the amount of expected error from the DPGS unit, and the simulated data. The program works by creating a 3D matrix of bins representing volumes of space. Anytime the error shell overlaps a bin, the bin's value is incremented. The bin with the largest count was deemed the most likely volute to contain the tip of the rod. The process is repeated multiple times, decreasing the volume of each bin and zooming in on the likely tip location.

Following several simulations, Program B.7 "MakeHistogram.m" was used to plot the effects of error in the DGPS on error in the ultimate measurement given varying lengths of the arc through which the unit was swung. These plots, shown in Figure 5.3 though 5.5, indicate that the error from the system will be similar to the error from the DGPS provided that the device is swung though at least perpendicular two arcs of at least

80 degrees. In particular, Figure 5.5 shows that error up to 20 cm might occur if an arc of only 60 degrees is used. Errors in the 1 meter range during a 60 cm swing were also predicted, but this prediction was due to the algorithm mistaking the shell intersection opposite the rod's tip for the rod's tip during the program's first iteration. A different program may not experience the same problem.

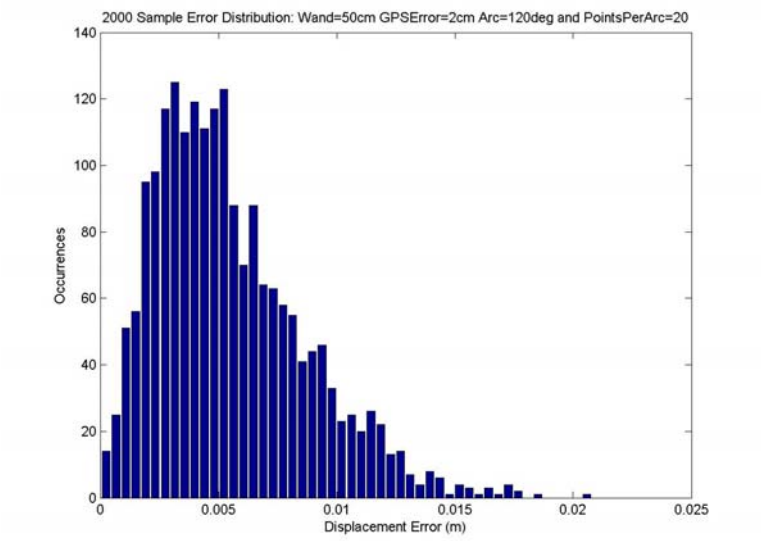


Figure 5.3: An error histogram of 2000 trials of the DPGS / rod system is shown in which DGPS error is assumed normally distributed at 2 cm with 2-sigma certainty. The simulated rod was 50 cm long and each of the two arcs were 120 degrees in length.

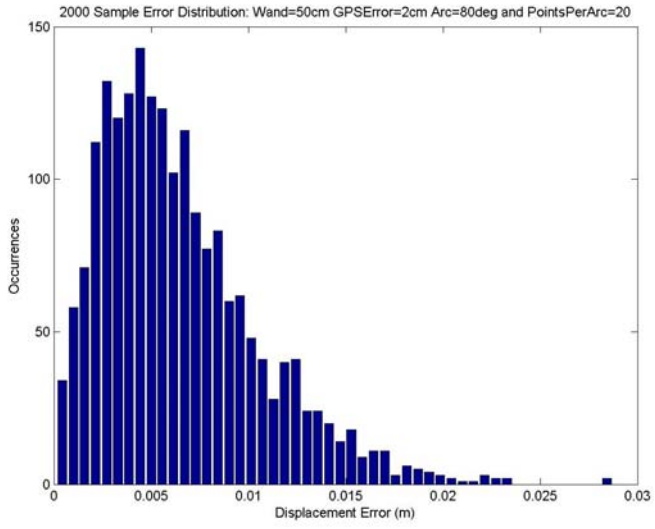


Figure 5.4: An error histogram of 2000 trials of the DPGS / rod system is shown in which DGPS error is assumed normally distributed at 2 cm with 2-sigma certainty. The simulated rod was 50 cm long and each of the two arcs were 80 degrees in length.

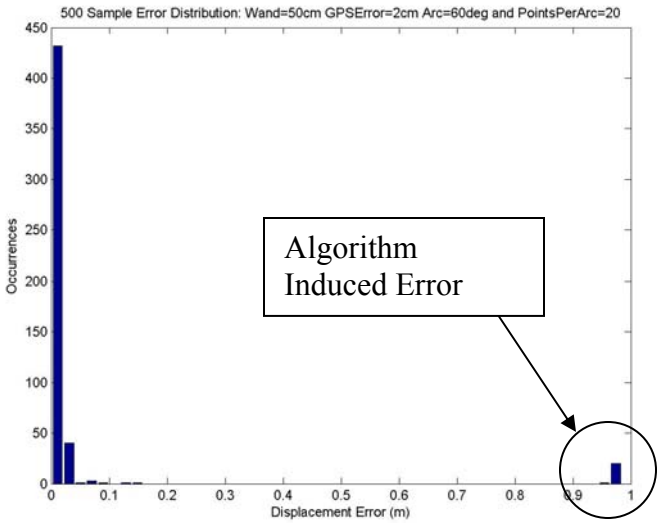


Figure 5.5: An error histogram of 500 trials of the DPGS / rod system is shown in which DGPS error is assumed normally distributed at 2cm with 2-sigma certainly. The simulated rod was 50 cm long and each of the two arcs were 60degrees in length.

Conclusions About DGPS Based System:

- A DGPS system can archive accuracies of 1 to 2 centimeters.
- A single DGPS system is very expensive ($10,000 +)
- In order to not introduce error due to the geometry of a DGPS antenna, it is necessary to know the heading of the device.
- DGPS devices capable of determining heading are considerably more expensive the regular DPGS devices.
- The location of a measured point can be found by mounting the antenna on the end of a rod and swinging the antenna though two perpendicular 80-degree arcs. This method does not introduce much additional error if the arc are of at least 80 degrees. However, this would require such an action to be repeated for every point captured and thousands of points – and arcs – may be required to fully capture the geometry of a large vehicle.

Decision About System Type:

The increased accuracy of a DGPS unit relative to large a measurement arm was offset somewhat by the inconvenience of having to physically swing the device though multiple arcs every time a point was to be measured. However, the ultimate deciding factor was budget availability. Despite the initial possibility of buying a DGPS unit, funding did not become available to buy a unit within the time frame of this project. This facilitated selection of the mechanical arm design for the remainder of the project.

Section 6: Mechanical Design

Overview:

After preliminary studies, it was decided that the measurement device should consist of a multi-linked instrumented arm, as was the original concept. Multiple pinned one-degree-of-freedom joints would allow motion. The joints would be designed to be interchangeable. Three long links, of approximately five feet each, would provide for reach. The length of five feet was chosen to eliminate some twist error inherent in a longer arm, as calculated in the previous section. All other joints were to be connected by short links ranging from 12 to 24 inches in length.

Initially, a five joint arm with three long links and two short links was planned and even partially constructed. However, this configuration was found to be unwieldy after partial assembly and mock operation; furthermore, twist seemed likely to occur in the links, a problem which the preliminary error analysis showed to be highly undesirable. Thus, a seven joint design was adopted, and is the configuration discussed in this section. This configuration includes five caster wheels that aid in the movement of certain links that generally lay on the ground during arm operation.

In addition to the arm linkage itself, a base stand was designed and built which serves as a portable anchor to which one end of the arm is fixed. The base has a tripod configuration in which the height of each leg is adjustable via a screw peg, and each peg ends in a sharp point that can dig into the ground. Provision to hang weight from the base, to increase its stability, is included.

Additional hardware includes a machined plate for calibrating the device, and a rubberized tip that is used to touch surfaces without marring them.

Material and Hardware List:

- 6’ foot long Al 6061-T6 2” x 3” rectangular tube with ¼” wall thickness, qty 1
- 5’ foot long Al 6063-T5 1.5”square tubes with 0.125” wall thickness, qty 3
- 2’ foot long Al 6063-T5 1.5” square tubes with 0.125” wall thickness, qty 2
- Inner ring and roller assemblies for ¾ inch shaft (McMaster-Carr # 5709K13), qty 14
- Roller bearing journals for ¾ inch shaft (McMaster-Carr # 5709K53), qty 14
- 0.75” diameter mild steel round stock, length ~ 40”
- ¾” ID 0.125” thick washers, qty 7
- ¾ - 10 nuts, qty 7
- 2.5” ¼ - 20 bolts, qty 72
- 1” ¼ - 20 bolts, qty 26
- 3.5” ¼ - 20 bolts, qty 2
- ¼ - 20 lock nuts, qty 100
- Quarter inch zinc plated washers, qty 200
- 2” swivel style four-hole-mount caster wheels, qty 5
- ¼” diameter mild steel round stock, length 3’
- 7” long ½” diameter carriage bolts, qty 3
- ½” diameter Wing nuts, qty 6
- Mountable bubble levels, qty 2
- 1.5 x 1.5 x 2 aluminum block, qty 1
- ¼” Aluminum plate (shop scrap)
- US Digital E3 rotatary encoders, qty 7
- US Digital E3 encoder mounting plates, qty 7
- ½” nuts, qty 3
- Disk shaped metal weights ~ 10 lbs each, qty 3
- 3” long ½” bolts, qty 3
- ½” washers, qty 9
- ½” 4 – 40 screws, qty 28 (in addition to screws included with encoders)
- 4 – 20 nuts, qty 28
- #6 flat washers, qty 28

- ½ spiral wrap, qty ~ 25 feet
- ¼ spiral wrap, qty ~ 10 feet
- Electronics case, qty 1

General Purpose Joint Components:

Each one-degree-of-freedom joint consisted of two halves. Both halves were machined from Al 6061-T6 2" x 3" rectangular tubing with a ¼" wall thickness. The material was selected because of its low weight, high strength, and the ease with which it could be machined.

The half shown in Figure 6.1 served to hold a steel shaft in place via a 0.001-inch interference fit between the two ¾ inch bores, on opposing sides of the part, and the shaft. Six 0.257-inch holes allowed links, which slid within the tubing, to be aligned and secured. Six of these joint halves were produced.

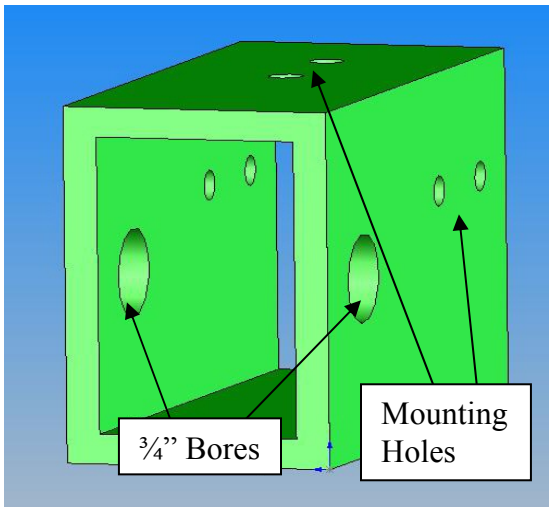


Figure 6.1: A rendering of the joint half into which a shaft was press is shown.

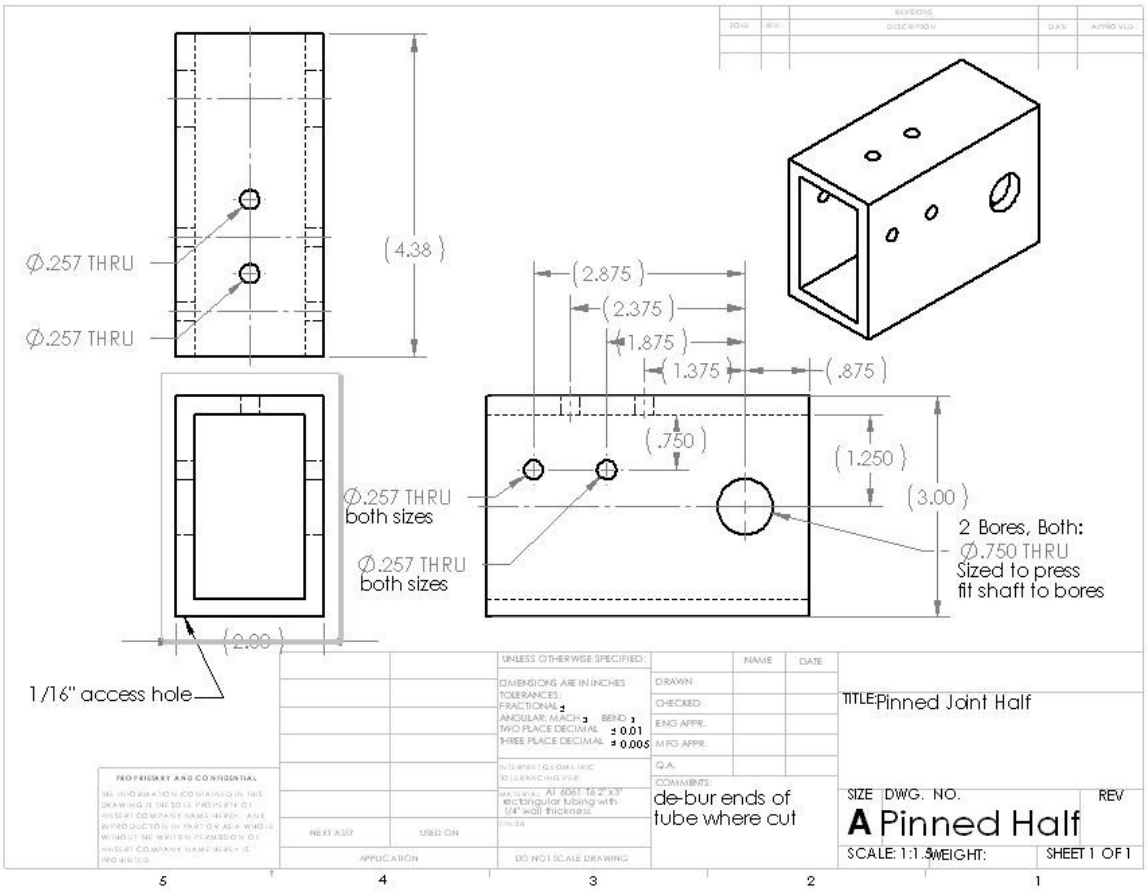


Figure 6.2: The specification for machining of the joint half into which a shaft was pressed is shown.

Each joint assembly is held together with a ¾"diameter mild steel shaft. The elastic stiffness of steel was desired to decrease flex in the joint. Mild steel was selected because a large amount of strength is unnecessary. A machinist set the exact shaft diameter such that the inner ring of a ¾" tapered roller bearing could be slid onto the shaft by hand. Additionally, a ¾ - 10 thread was machined into one end, and the opposing end was pressed into the part shown in Figure 6.1.

Using this shaft design, a force is applied to each bearing, using a nut, along the shaft axis until all play is removed from the bearing, thus increasing arm accuracy. Seven of these shafts were produced.

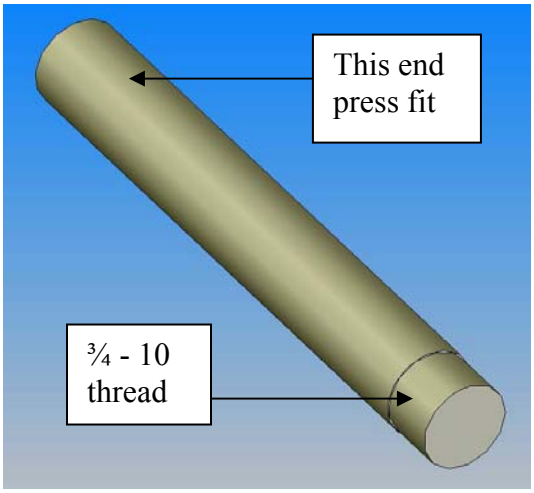


Figure 6.3: A ¾ - 20 thread was cut into the end of a mild steel shaft, with the diameter set to fit within the roller bearing.

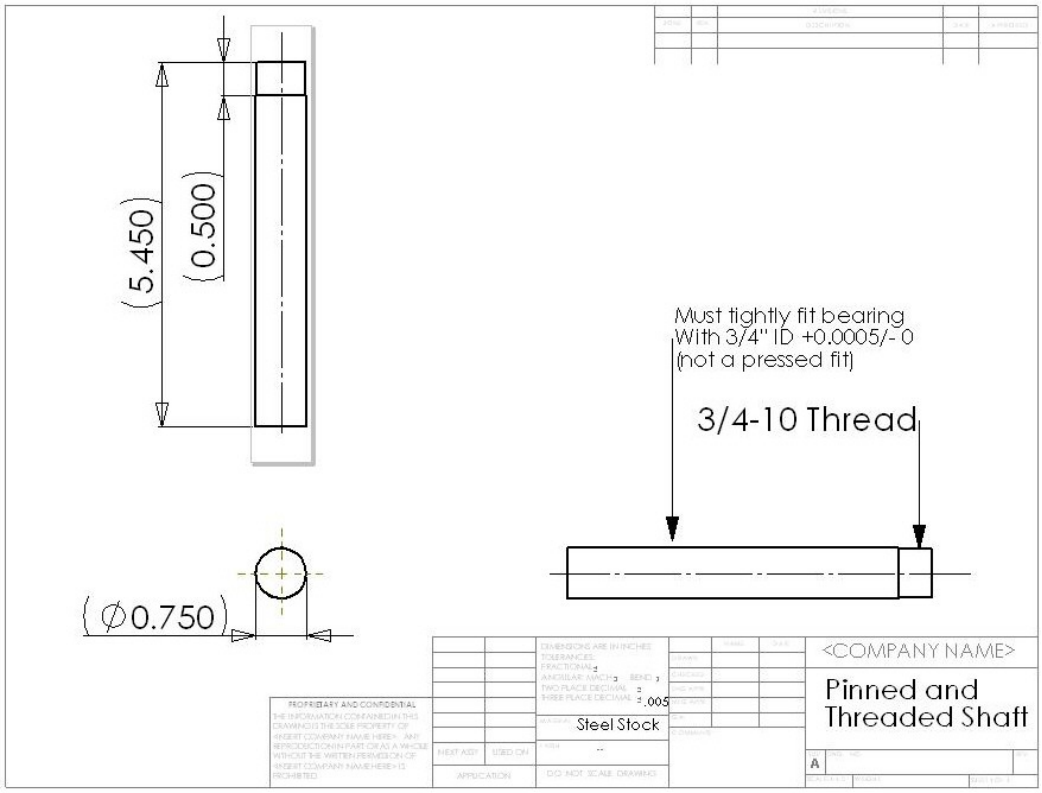


Figure 6.4: The specifications for machining of threaded shaft.

The second half of each joint, shown in Figure 6.5, was designed to house each rotary encoder. 1.781" bores on each side of the part hold bearing journals that are press fit with 0.001" interference. Four 0.125" holes are arranged near one journal to provide a means of fixing the mounting plate for each encoder, and quarter inch holes around the

opposing bore provide access to the housed encoder for joint assembly and maintenance. Seven of these joint halves were machined.

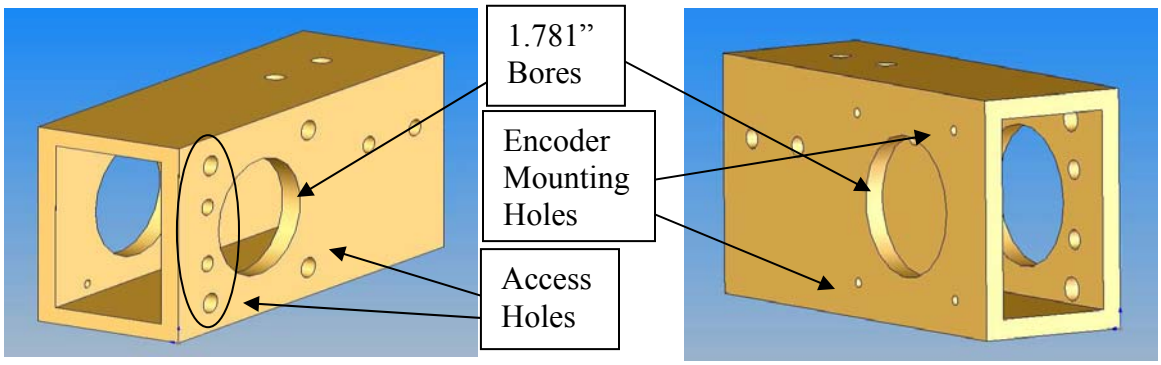


Figure 6.5: The joint half housing each encoder shown from two perspectives.

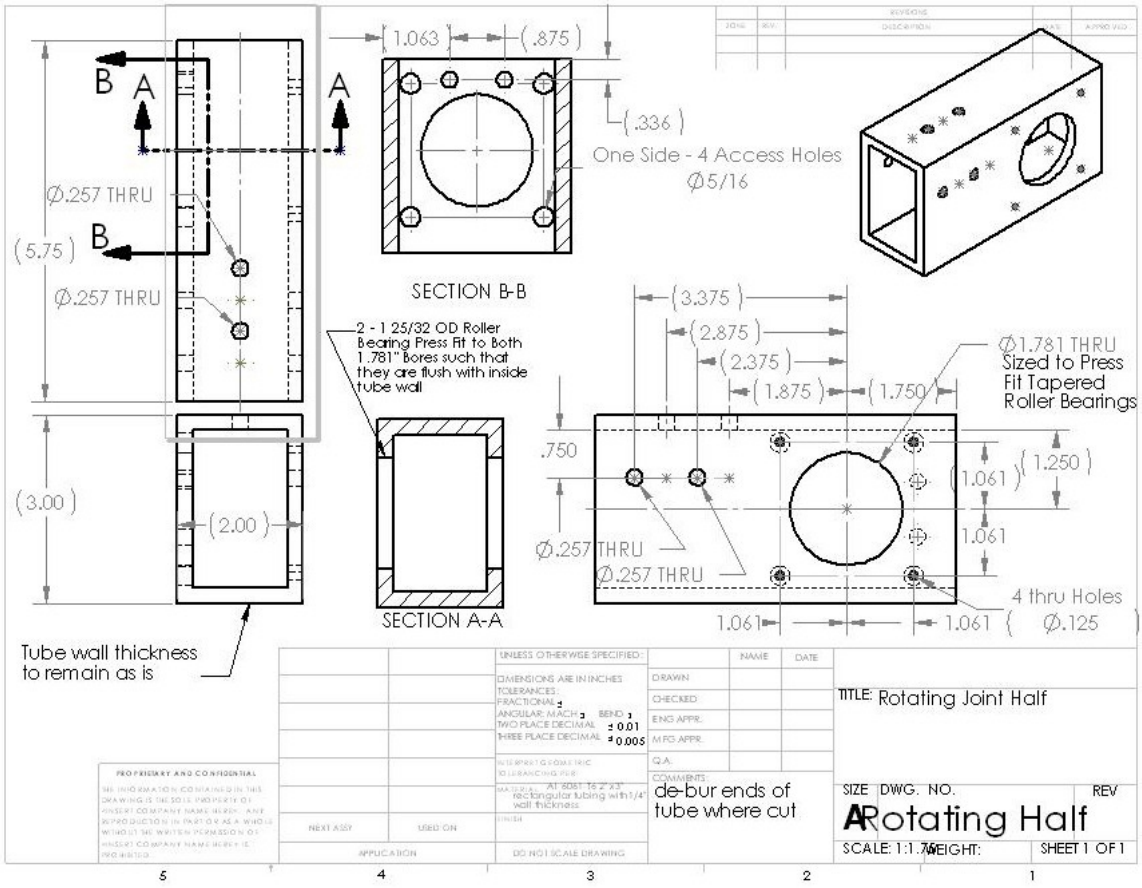


Figure 6.6: Specification for machining the rotating half of each of joint.

General Purpose Joint Assembly:

After the components shown in figures 6.1,6.3, and 6.5 were completed, 6 joints were assembled as shown in Figures 6.7 through 6.9.

The mounting plate for the US digital encoders is secured to the inside of each joint half two with ½" 4-40 machine screws and nuts. #6 Washers were set the plate slightly above the inner wall of the joint half, allowing clearance for the bearings that fit in the journals.

Two roller bearing inner ring and roller assemblies are set into the two journals. The ¾" shaft, pressed into the other joint half, runs through the inner diameter of each bearing, and the rotary encoder hardware. A 1/8" thick spacer washer is placed between the two halves to provide for spacing, and entire assembly is held together by a ¾ - 10 nuts screwed into the end of the shaft. The orientation of joint half number 2 can be opposite the orientation shown in Figure 6.7 without any complication in joint assembly or operation.

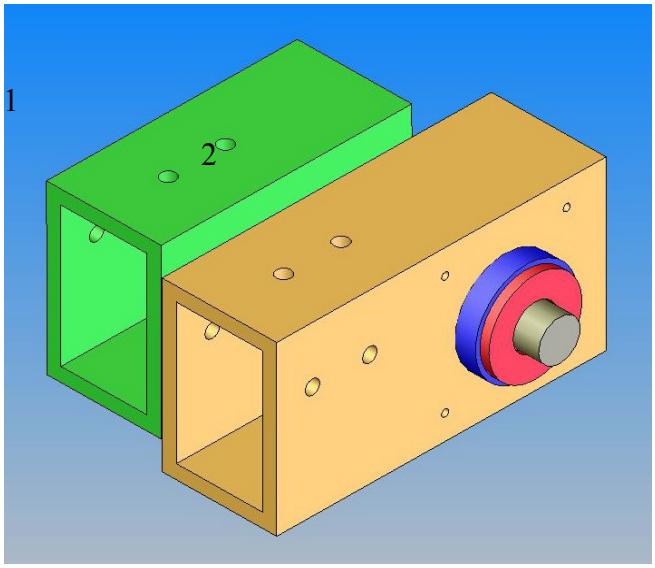


Figure 6.7: A general-purpose joint assembly is shown above.

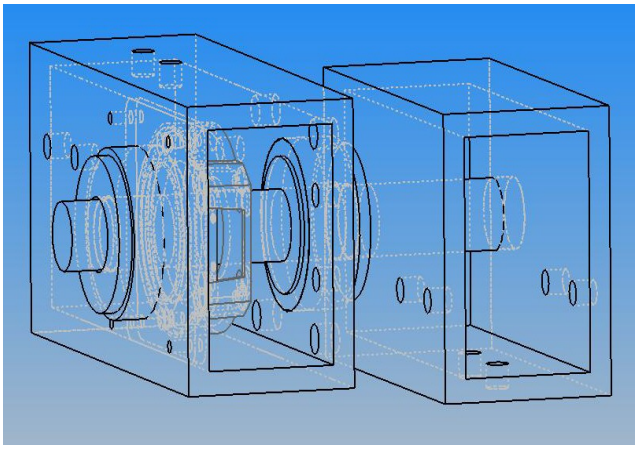


Figure 6.8: A second view of the general purpose joint assembly.

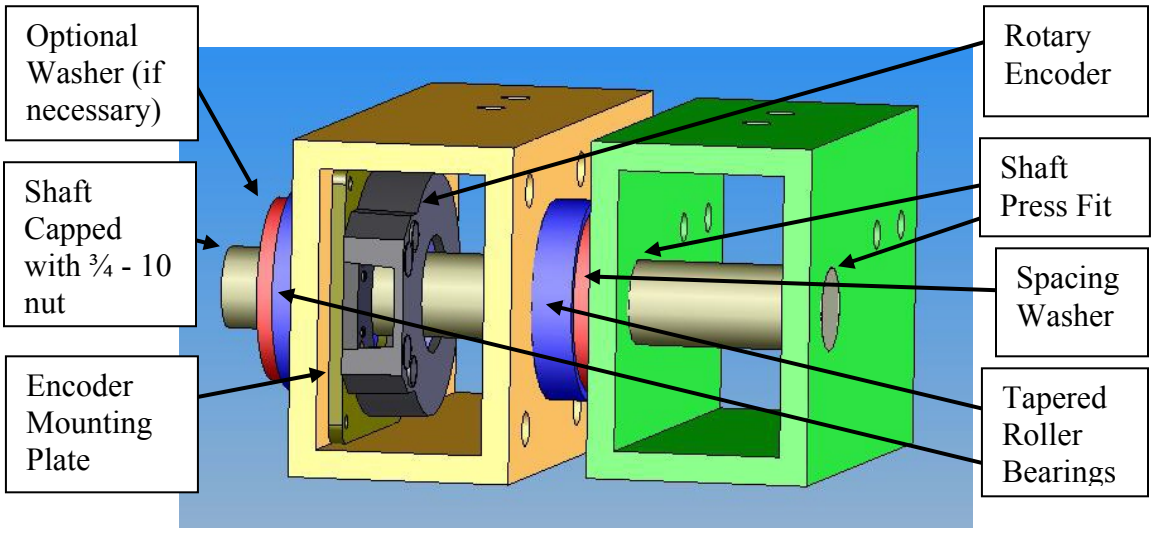


Figure 6.9: A third view of the general-purpose joint assembly. The orientation of joint half on the left can be opposite the orientation shown in the figure, i.e. the encoder can be on the opposite side, without adding any complication to joint assembly or operation.

Base Joint Components:

Six of the joints are identical. However, the base joint was designed such that half of the joint was different, as shown in Figure 6.10 left. The shaft in the base joint is pressed into the two opposing bores such that the threaded end faces away from the four quarter inch holes, as shown in Figure 6.10 right. The four holes are intended to provide a means of mounting the joint half to any convenient surface. Assembly of the base joint is exactly like the assembly of the six other joints. Only one of these base joint halves was produced.

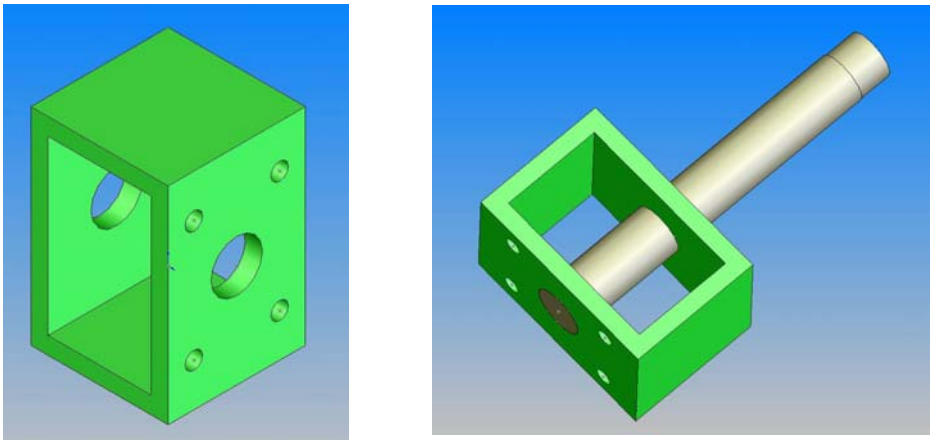


Figure 6.10: The Base joint half is shown on the left. The joint half – shaft assembly is shown on the right.

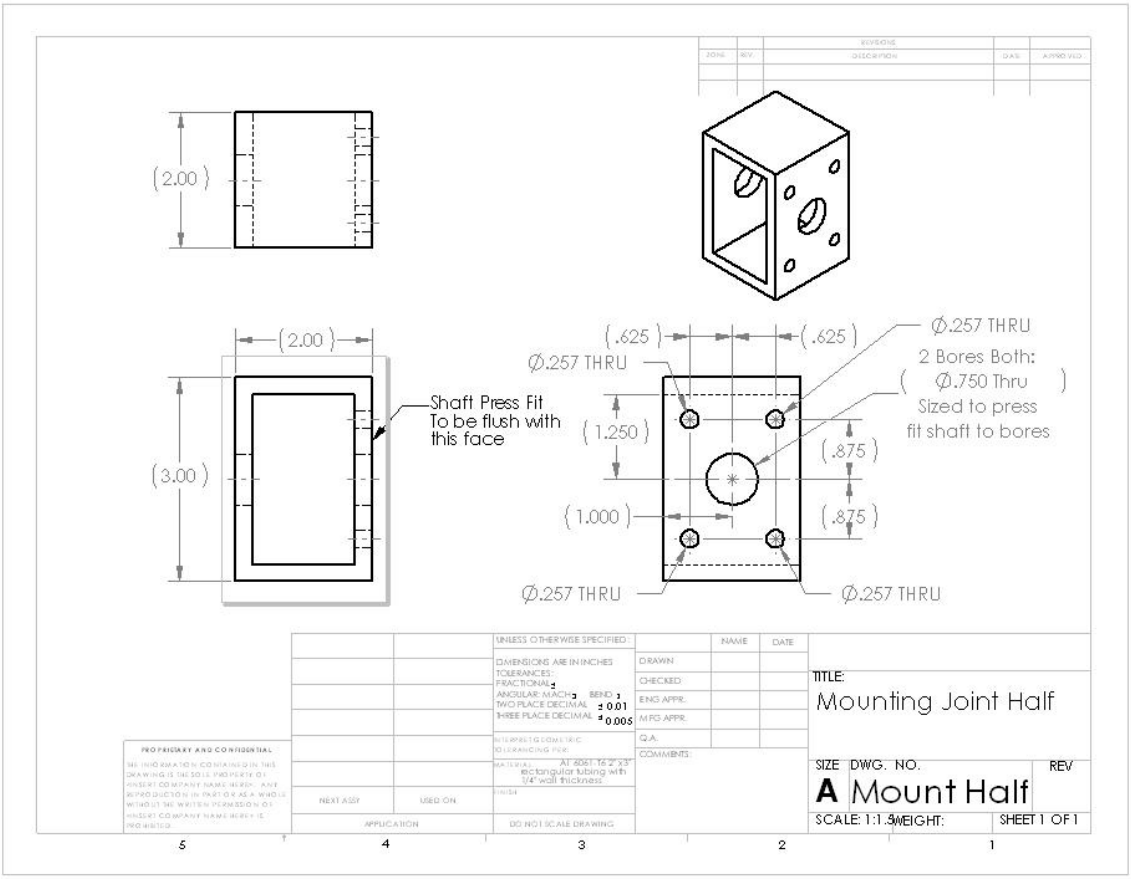


Figure 6.11:The specifications for machining the base joint half.

Connecting Links:

The links were designed to connect the joint into a snake-like arm. The links are machined from square Al 6063-T5 1.5" tubes with 0.125" wall thickness. Square tubing is used because it offers good resistance to twisting while also providing perpendicular mounting surfaces that simplify the process of mounting joints at ninety degrees to each other. Because of the presence and configuration of the joints, the links in the arm should not experience much load. Hence a weaker aluminum alloy than the 6061-T6 was used in the joints in order to reduce costs while still maintaining the weight benefits of aluminum.

The outer diameter of the linkage tubing should have permitted each link to snuggly slide into the tubes from which the joints were constructed. However, the inner diameter of the delivered tubing was found to be about 0.015" less that the outer diameter of the links. So to compensate, a small amount of metal was shaved from the side of each link as shown in Figure 6.12. Also, mounting holes are drilled in the ends of each link so that they mate with the holes drilled in each joint.

The lengths of each link, measured with a meter stick accurate to 0.5 mm (0.02"), are as follows:

Link 1: 12.04" (Base Link):
Link 2: 60.07"
Link 3: 6.58"
Link 4: 8.19"
Link 5: 15.80"
Link 6: 60.02"
Link 7: 60.02" (End Link):

Because a good deal of fitting is required during assembly, the lengths provided are from measurements taken after assembly.

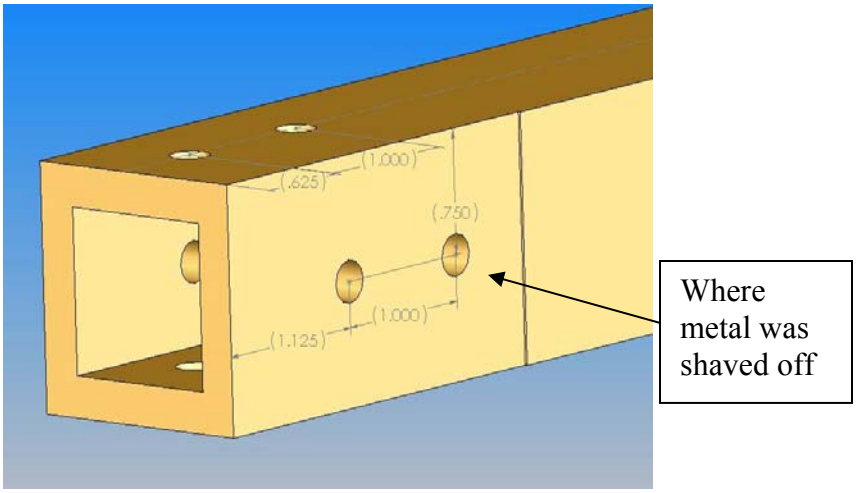


Figure 6.12: End of a typical link. About 0.015" was removed from part one face.

Calibration Plate:

To aid in the calibration of the arm, a plate was machined from ¼" aluminum plate which could be pressed against the top of the each joint assembly, forcing it into its most extended configuration. The slots in the plate were included to allow for the bolts extending from to top of each joint. One side of the plate was machined such that half of the surface was offset 0.015" from the other. This allows the calibration surface of opposing joint halves to be offset by 0.015". The offset is required because of unanticipated variation in the wall thickness of opposing sides of the joint tubing.

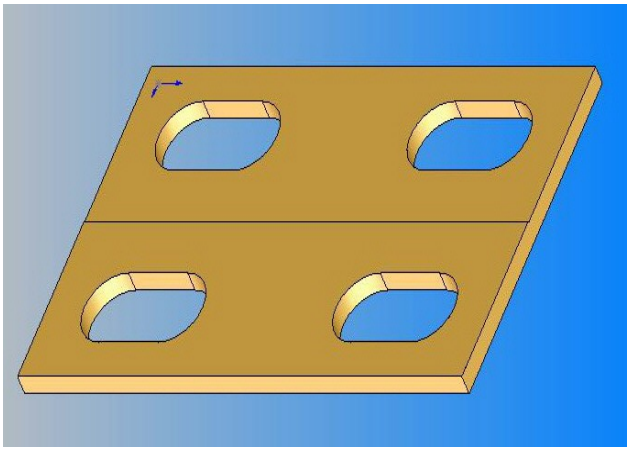


Figure 6.13: A joint calibration plate is shown with a 0.015" offset machined into it along with one inch wide slots meant to accommodate bolts protruding from each joint.

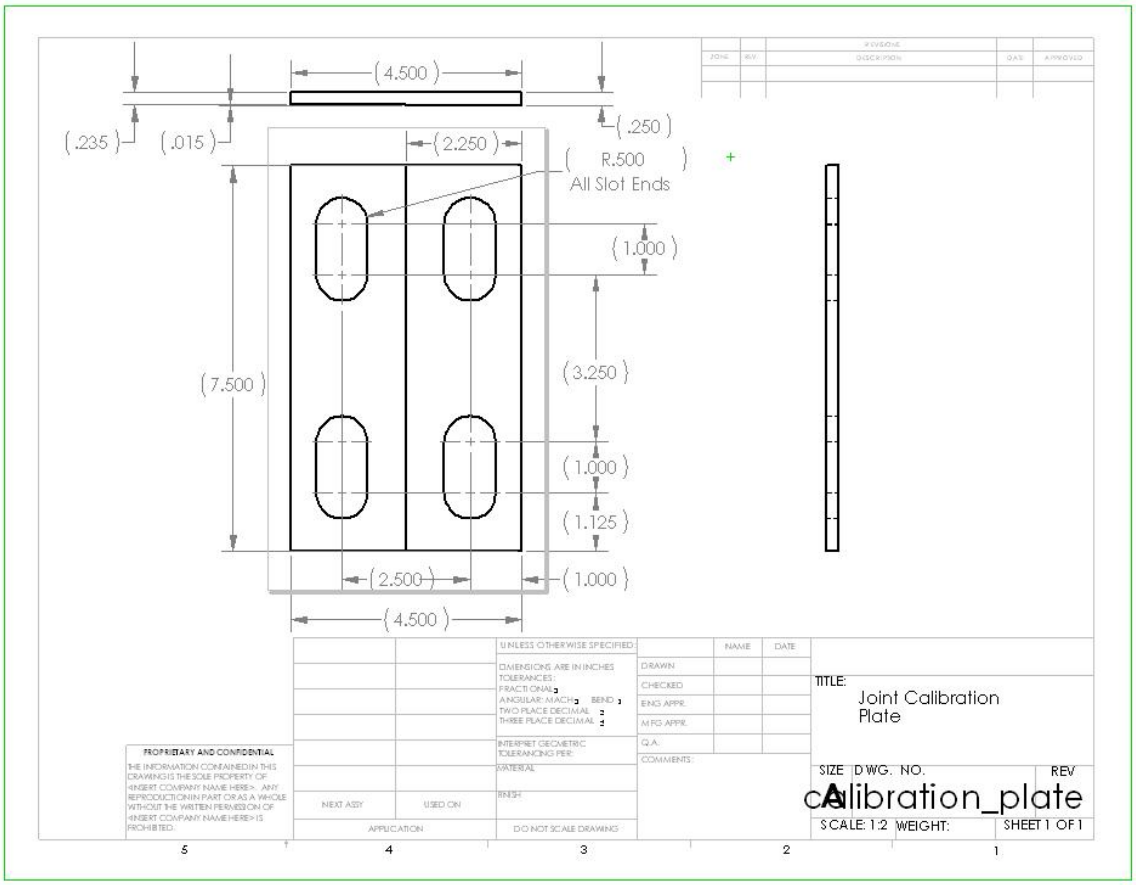


Figure 6.14: The specifications for machining a joints calibration plate.

Arm Base Assembly:

A dedicated base for the arm, shown in Figures 6.15 and 6.16 was constructed for use in most measurements. The base consists of three quarter inch thick AL 6061-T6 legs extending from a hub. The hub is made of three plates of aluminum. Half-inch bolts are set in holes in each leg allowing weights to be hung underneath the assembly, increasing its stability. Each leg is terminated by a half inch threaded spike machined from carriage bolts, to prevent the base from sliding. The spikes also can be adjusted up or down to level the base.

The base is not designed and machined with precision and is actually made of scrap material to save time and cost. The dimensions of the base are not expected to affect measurements made by the arm, provided the base remains stationary. Thus, should it become desirable to use the base in conjunction with another measuring system in the future, for instance, GPS unit or surveying equipment, an updated design may be necessary.

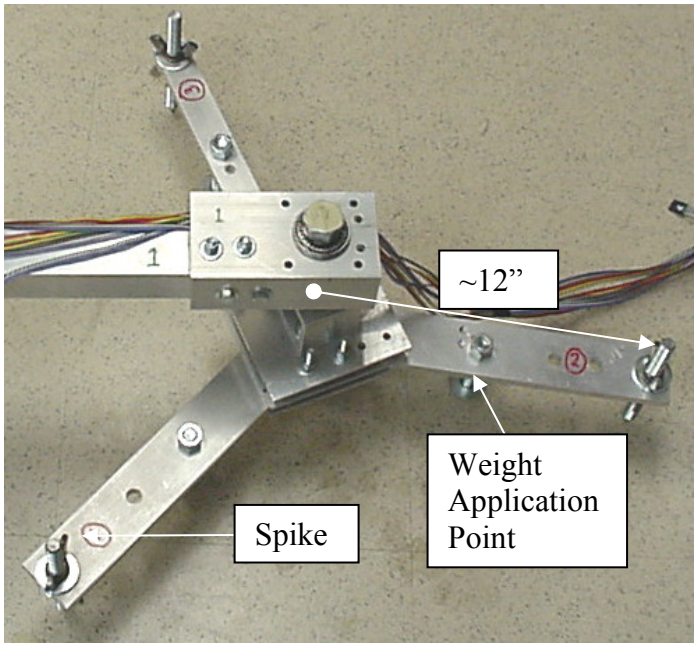


Figure 6.15: Top view of arm base.

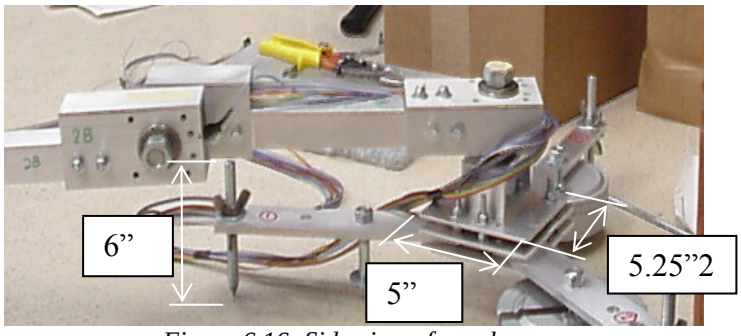


Figure 6.16: Side view of arm base.

Measurement Stylus:

A stylus is shown in Figure 6.17 is attached to the end of the arm to provide a convenient contact point between the arm and an object being measured. The stylus consists of a quarter inch diameter piece zinc plated steel rod hammered into a quarter inch hole in an aluminum plug, machined to fit within and cap the tubing of the end link. Two 0.257" holes are drilled through the plug and the tube wall to provide a means of securing the plug with bolts. A rubberized coating is added to the rod to minimize scratching of any delicate measured surfaces like the painted metal of a car body.

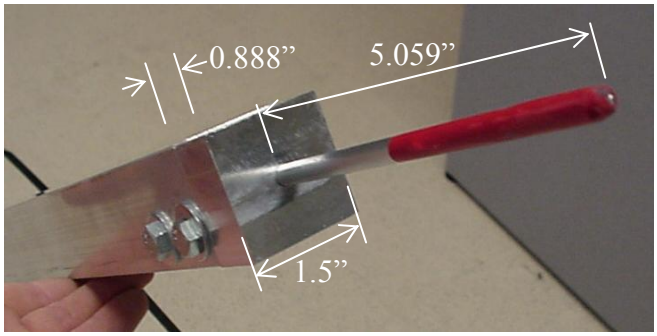


Figure 6.17: The stylus at the end of the digitizer arm is shown dimensioned.

Caster Wheel Supports:

The results of testing a wooden mockup arm revealed that only two links of the arm should lift off the ground and any time. Thus, the remaining five links are supported by swiveling caster wheels. These exist to increase the ease with which the links are moved and to eliminate wear on the links and joints. By trial and error, it was found that five 2" caster wheels are required. Four of these wheels are mounted in sets of two, as shown in Figures 6.18 and 6.20, and one of these wheels is mounted by itself, as shown in figure 6.19. The mounting surfaces are scrap aluminum plates, which are bolted to links 2, 4, and 5.

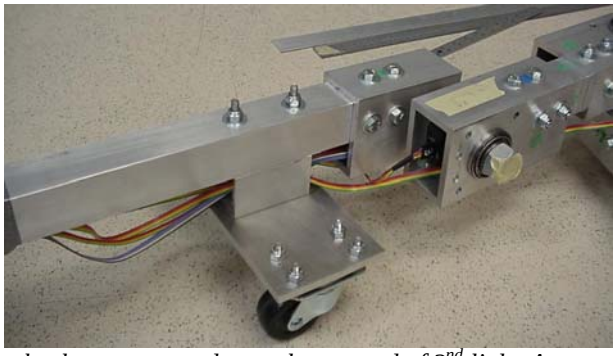


Figure 6.18: Dual caster wheels are mounted to a plate at end of 2nd link. A spare piece of tubing is used as a spacer to elevate the link off the wheels by 1.5 inches.

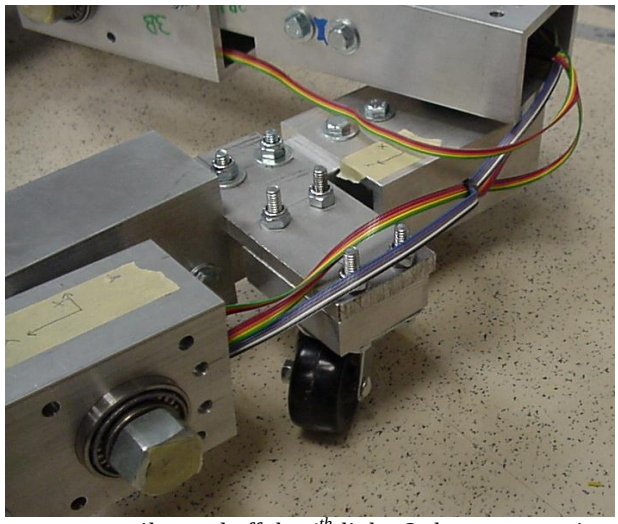


Figure 6.19: As single caster are cantilevered off the 4th link. Only one caster is used because two casters, one on each side of the joint, would have interfered with the motion of the fourth joint.

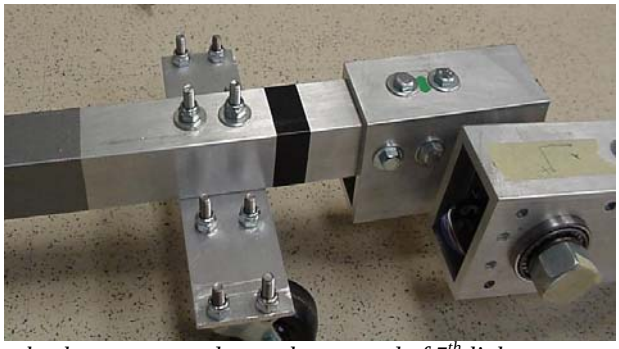


Figure 6.20: Dual caster wheels are mounted to a plate at end of 5th link.

Full Arm Assembly:

The full joint-link-base-stylus assembly is shown in Figure 6.21. In the configuration shown, the first and fourth joints provide for rotation about the z-axis.

Meanwhile, joints six and seven provide motion within the x-y plane as well as elevation. Joints two, three, and six allowed the arm to move over uneven terrain without any of the beams becoming cantilevered or twisted. Joint three also relieves bending moments from the two end links when the entire arm is moved by pulling on said links, perpendicular to their long axes.

The caster wheels mounted to the third, fourth, and fifth links provide even support for the arm. Such support not only provides for smooth operation but also prevents large twisting forces from being transferred to the base. Thus, the 35 lbs of weight hung from the base is adequate to prevent the assembly from moving.

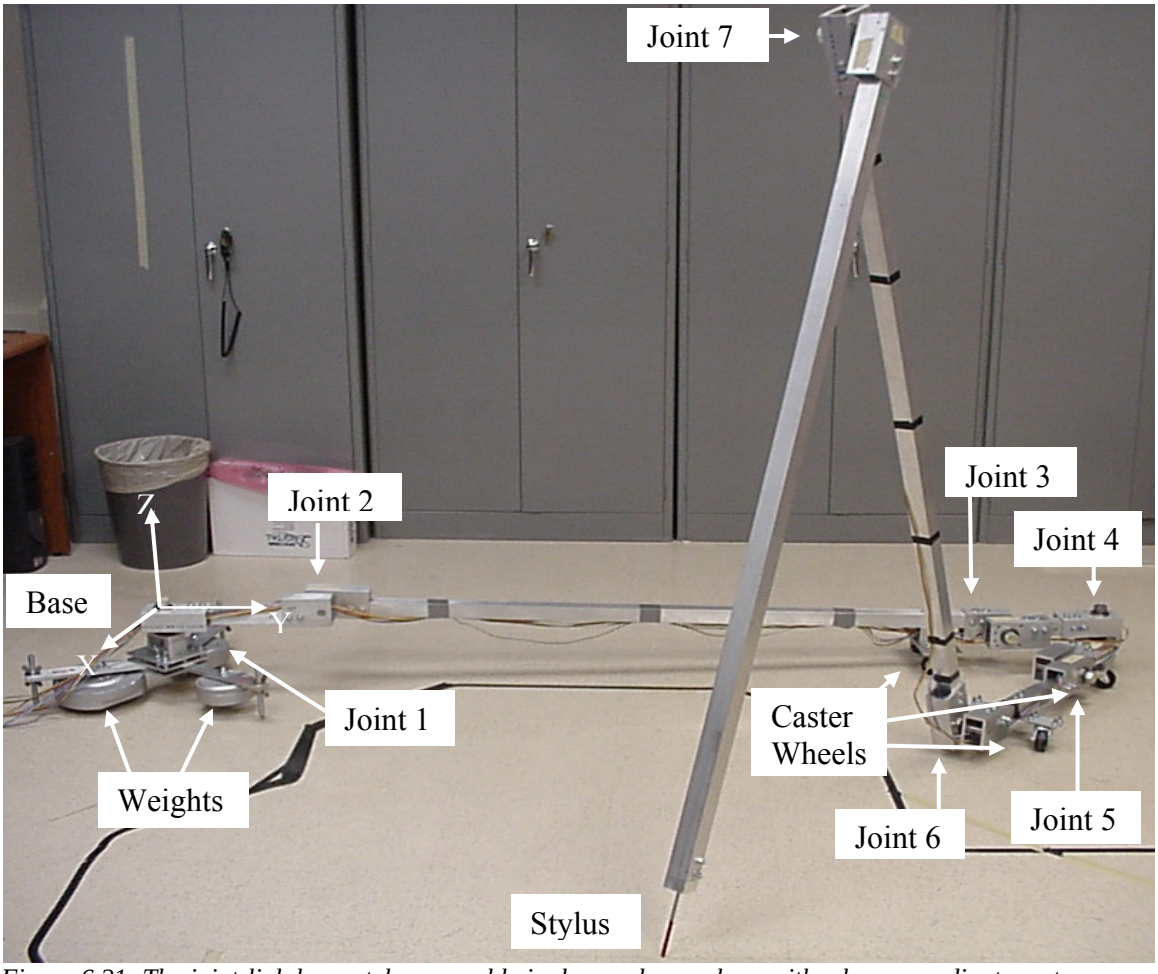


Figure 6.21: The joint-link-base-stylus assembly is shown above, along with a base coordinate system.

Section 7: Electrical Design

Electrical Design Overview:

The mechanical portion of the digitizer arm, described in the previous section, essentially converts a position in space into the rotational positions of seven shafts located within the joints of the device. Sensors and electronics are required to further convert these shaft positions into conventional Cartesian coordinates. This section intends to provide an overview of the electronics required to accomplish the mechanical to electrical convertion. This section is not intended as a detailed map by which each electronics module used in the digitizer arm may be understood and/or recreated; as such documentation exists elsewhere (US Digital Corporation. “E3 Optical Encoder Kit”) (Brennan).

Sensors:

Each digitizer arm joint is instrumented with a US Digital E3 incremental rotary optical encoder. The encoder is customized to have a high-resolution optical disk with 2048 counts per revolution, an index channel, a shaft through hole, and a mounting plate. Detailed specifications for this unit are provided by US Digital Corporation (US Digital Corporation. “E3 Optical Encoder Kit”).

Each encoder includes 5 pin outs from its onboard electronics module.

1. Ground Input
2. Index Output: set high when the index position of the encoder is sensed
3. Output Channel A: set high or low based on incremental disk marks
4. +5 VDC Input
5. Output Channel B: set high or low based on incremental disk marks

In operation, the unit can determine a single index position of the shaft to which it is mounted. It can also determine the direction of rotation of the shaft was being turned via quadrature. It has an effective resolution of 2048 * 4 = 8192 positions per revolution approximately 0.0439 degrees.

Wiring:

Ribbon cable connects each encoder to a base computer. That is, five separate wires were run in each ribbon cable for a total of 45 wires present at the base of the device. The wires were kept as short as practical in order to reduce signal degradation due to wire impedance and electromagnetic interference.

Additionally, all wire bundles are covered with spiral wrap in order to protect them from wear. The need to protect the wires as such became very apparent when, during the electrical/software debug phase of arm assembly, wear in the insulation of one wire caused a short circuit that destroyed an encoder electronics module.

Sensor Signal Processing:

Each encoder simply outputs TTL pulses on three channels when marks on its optical disk are sensed by the built in electronics module. Therefore a decoder module is necessary. Under different circumstances, a decoder device many have been purchased or custom built. However, a device already existed in the lab, which could perform this function. Specifically, a custom built digital signal processor with I/O boards capable of interpreting encoder.

The internal workings of the DSP and I/O boards are beyond the scope of this project, and explanation is unnecessary as the device is self-contained. Thus, documentation of the DSP is not included in this write-up. Furthermore, the DSP system itself is a work in progress so formal documentation of the DSP and the source code does not exsit. Information on the device can be found by contacting Dr. Sean Brennan at Penn State University (Brennan).

On the two I/O boards stacked on the DSP, channels 1, 3, 4 and 5 of each encoder were connected directly to the decoder module. The index channel of each encoder was connected to general digital I/O pins 2 through 8 on the bottom I/O board, where pin 2 was connected to encoder 1, pin 3 was connected to encoder 2, etc. Lastly a RS-232 communication port is used to interface the DSP with a personal computer. The entire unit is, enclosed within a protective box purchased for this project, shown in Figure 7.1.

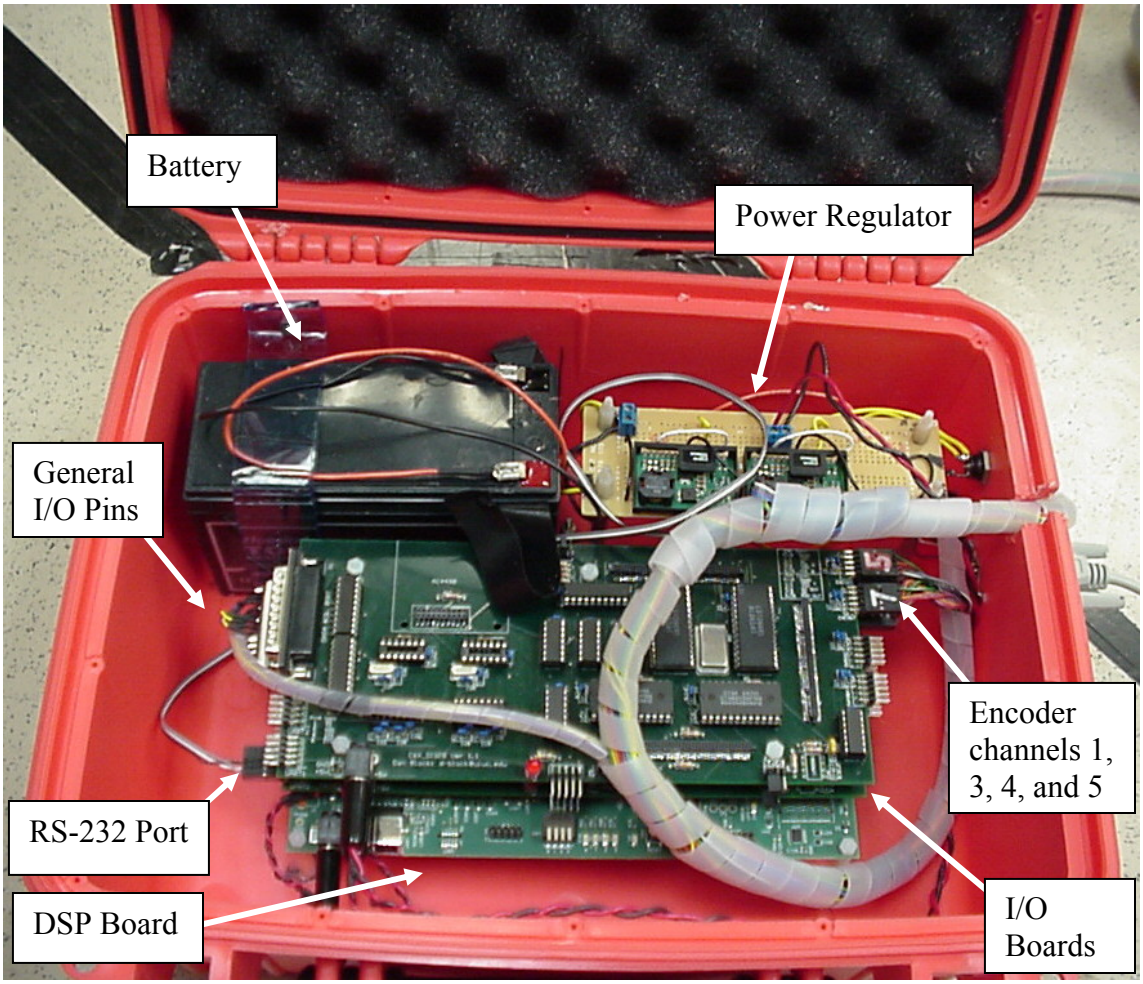


Figure 7.1: The DSP unit used to interpret encoder signals is shown (bottom), along with its power supply (top). Encoder lines are shown wrapped with spiral wrap, and the corresponding connectors are labeled.

Computer Interface:

The DSP and accompanying I/O boards are only used to read in and interpret encoder data, based on serial port input. This input is provided via a PC running Matlab, which served as the ultimate interface between the digitizer arm's sensors and the user. Three Matlab routines were written by John Cameron to allow Matlab to exert limited control over the DSP and to read data from the DSP. These Matlab programs were written specifically for this project, and they are not documented elsewhere, so they are included in appendix D as Programs D.1 though D.3.

Each Matlab program executes a specific command when called from the Matlab command line or from a subroutine. Specifically:

readEncoders.m : Instructs the DSP to read the position of each encoder and return the data to Matlab where it is output, in radians, as a seven entry vector

endCal.m : Instructs the DSP to internally zero the output from each joint when a pulse from its index channel is sensed.

startCal.m : Instructs the DSP to stop zeroing each joint when a pulse from its index channel is sensed. This option is available to prevent accidental zeroing due to electrical interference.

In the final user interface designed for this project, the programs listed above were not often called from the command line, but rather called as subroutines from a data capture program called *CaptureData.m*.

System Overview:

In summary the electronics of the digitizer arm consisted of seven optical rotary encoders that sent signals to a DSP which in turn decoded these signals. The DSP communicates with a PC via RS-232 protocol, running out of Matlab. A diagram of the data paths described is shown in Figure 7.2.

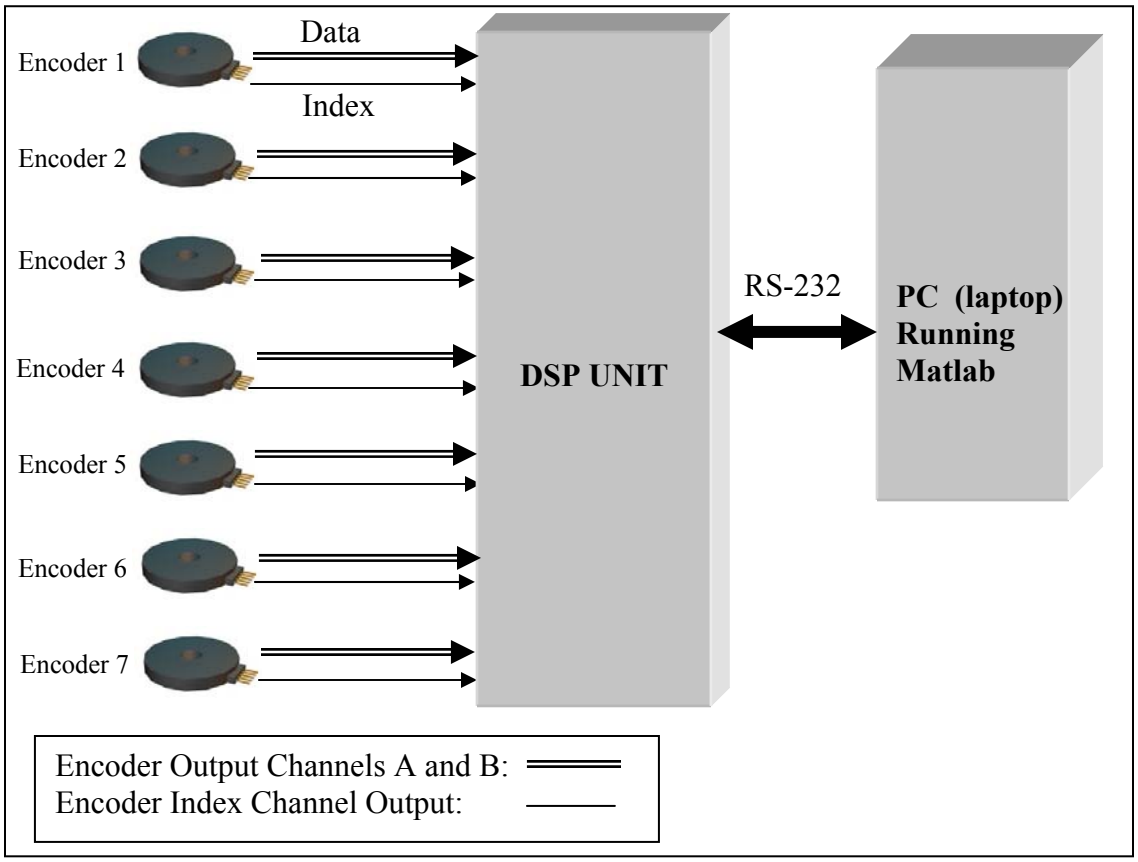


Figure 7.2: The overall data path of the digitizer arm is shown above.

Section 8: Arm Kinematics

Purpose of Kinematic Analysis:

The digitizer arm produced for this project is intended to measure the locations of points in 3D space based solely upon arm geometry and joint angles. In order to accomplish this, conventions for reporting joint angles and arm geometry are first established. Next, the kinematics of the arm are analyzed in order to produce a measurement equation and a sequence of sequential mathematical operations, for the position of the arm's tip. Explanations of geometric convention and the kinematic formulas for the position of the arm's tip are given in the following section.

Angle Conventions Coordinate Systems:

Coordinate systems are established at each joint so that kinematic analysis can simply consist of a series of coordinate transforms. Figure 8.1 shows the coordinate frames applied to the half of each joint closest to the end of the arm.

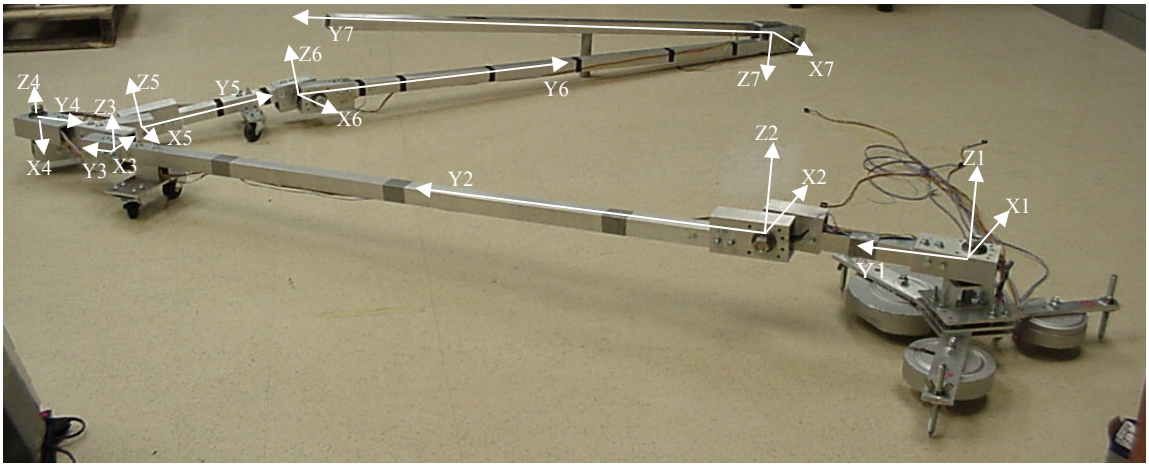


Figure 8.1: Coordinate systems were applied to the half of each joint closest to the end of the arm.

The coordinate convention was chosen so that the y-axis of each system is always co-linear with the axis of the link closer to the arm's tip, and is pointed away from the

joint closer to the arm's base. Also, the x-axis is chosen to be the axis of rotation in joints that tended to allow vertical motion (joints 2, 3, 5, 6, and 7) and the z-axis is the axis of rotation in joints that tended to allow motion in the horizontal plane (joints 1 and 4).

The zero position of each joint is chosen to be the position in which the joint is most extended. The angles reported by each joint are taken as positive if the joint half, closer to the arm's tip, rotated in a manner consistent with a right-handed rotation about the joint's axis of rotation. For example, the angle reported by joint 7 in Figure 8.2 would be negative pi / 4 radians.

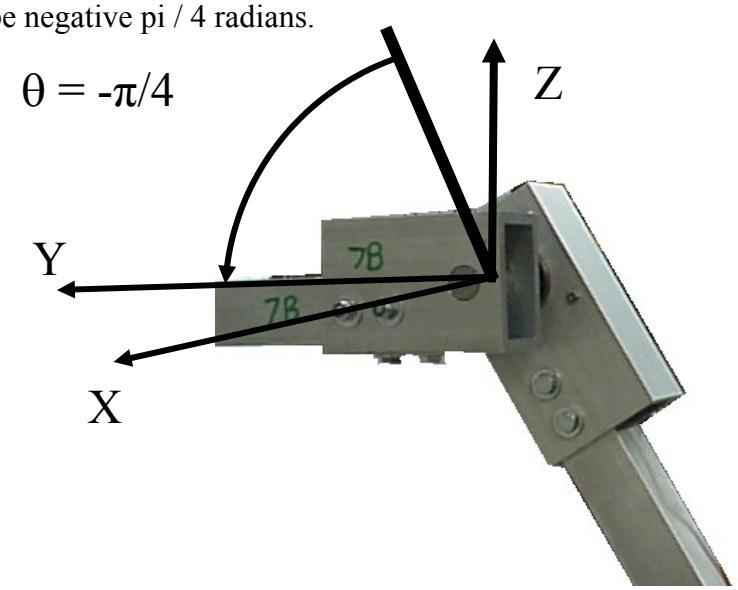


Figure 8.2: An example angular output from the 7th joint from the base is given. In this example the output would be -π/4 radians.

As can be seen in Figure 8.1, the joint halves containing each encoder were not necessarily located such that the established angle convention would be consistent with the raw output of the encoder, i.e. an encoder might read decreasing counts when measuring increasing angles . Multipliers, given in Table 8.1, were applied to all angular measurements to correct the sign on rotation. The corrected angles are hereafter used as inputs into the kinematic analysis.

Joint	Multiplier
1	1
2	-1
3	1
4	-1
5	-1
6	1
7	-1

Table 8.1: The listed multipliers were applied to angular readings from each encoder in order to maintain a consistent sign convention.

The exact positioning of each coordinate system relative to its host joint is given in Figures 8.3 though 8.9. Rotation of a joint is represented by the rotation of a coordinate system, attached to the link closer to the end of the arm, onto a coordinate system attached to the opposing link. However, as can be seen in Figures 8.3 though 8.9, positive rotation of each frame is in the opposite direction as positive rotation of each physical joint, as given by the convention above. A second multiplier of (–1) is applied to all angles, prior to kinematic analysis to correct for this, providing the values θ_1 though θ_7.

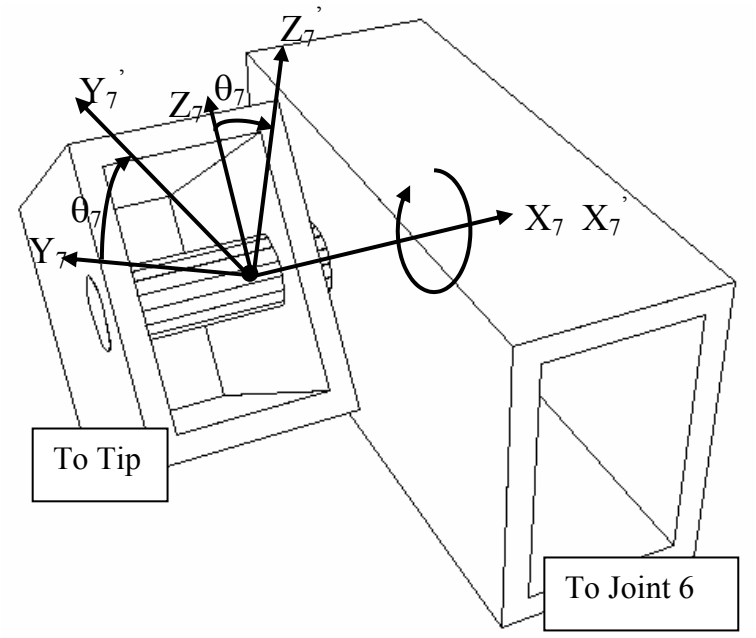


Figure 8.3: A coordinate system is shown coincident with the center of joint 7's shaft and the inner wall of the joint half closer to the arm's tip. Rotation is about the x-axis of the applied coordinate system.

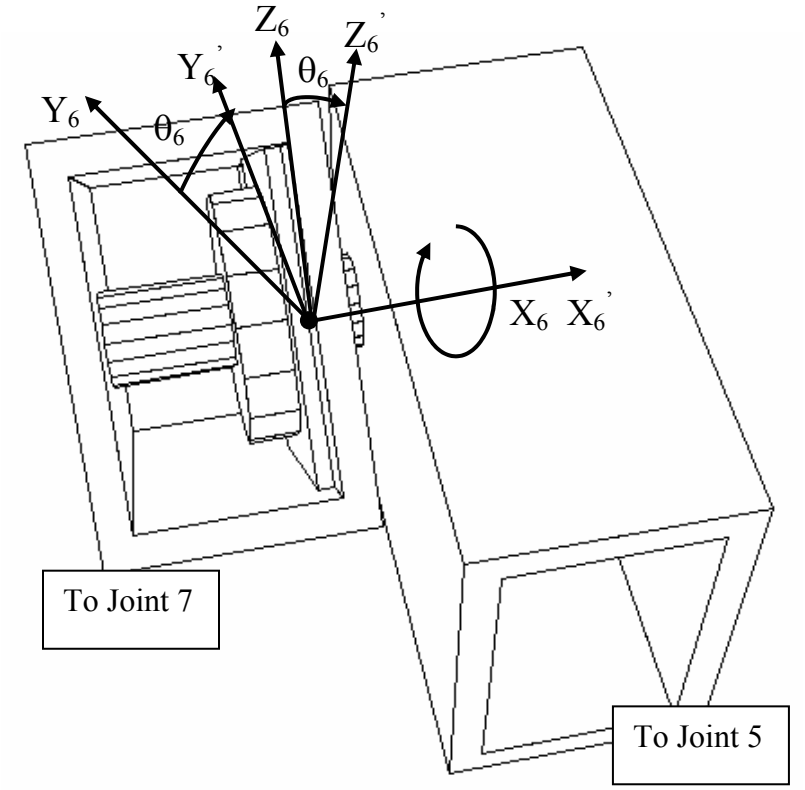


Figure 8.4: A coordinate system is shown coincident with the center of joint 6's shaft and the inner wall of the joint half closer to the arm's tip. Rotation is about the x-axis of the applied coordinate system.

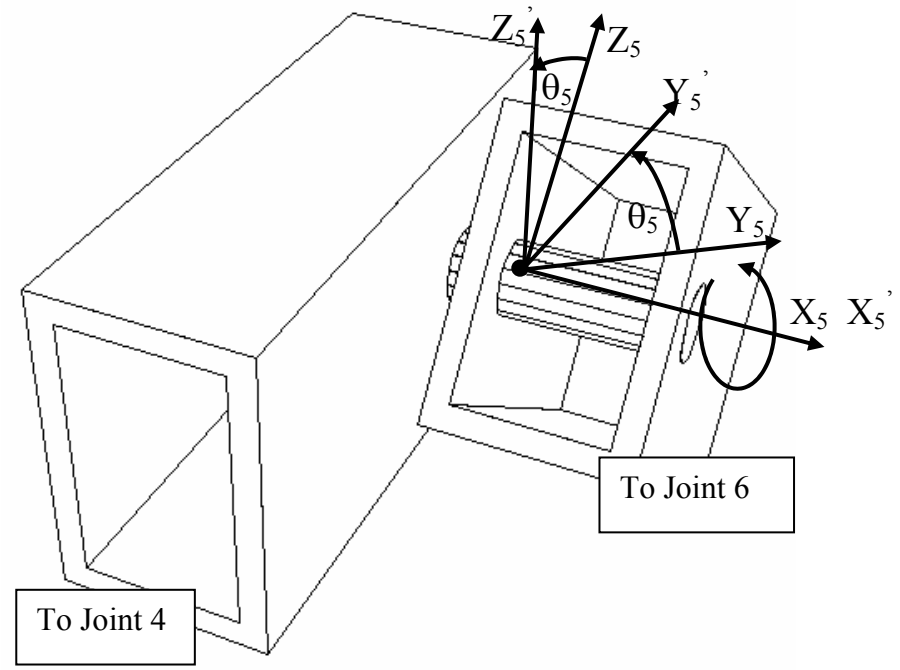


Figure 8.5: A coordinate system is shown coincident with the center of joint 5's shaft and the inner wall of the joint half closer to the arm's tip. Rotation is about the x-axis of the applied coordinate system.

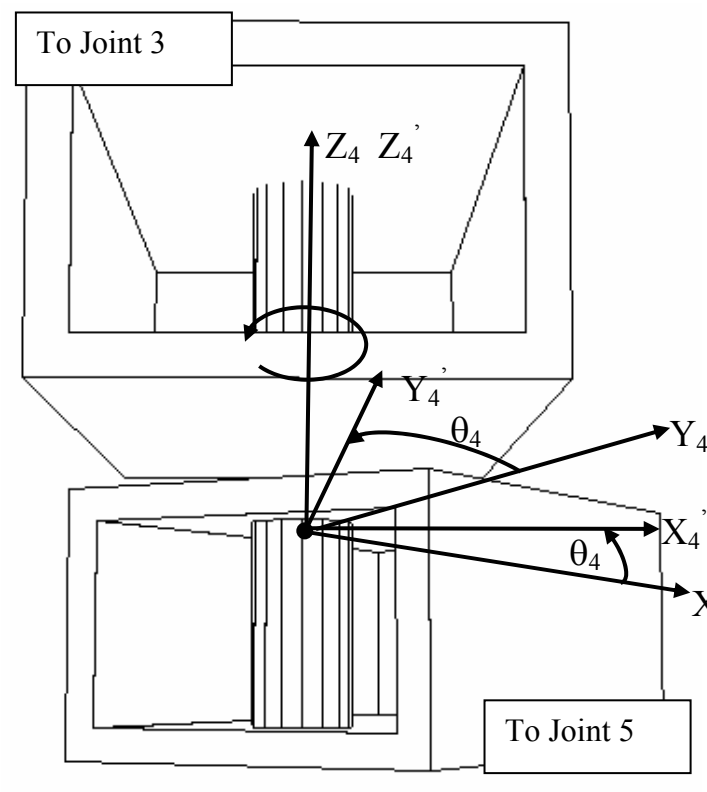


Figure 8.6: A coordinate system is shown coincident with the center of joint 4's shaft and the inner wall of the joint half closer to the arm's tip. Rotation is about the z-axis of the applied coordinate system.

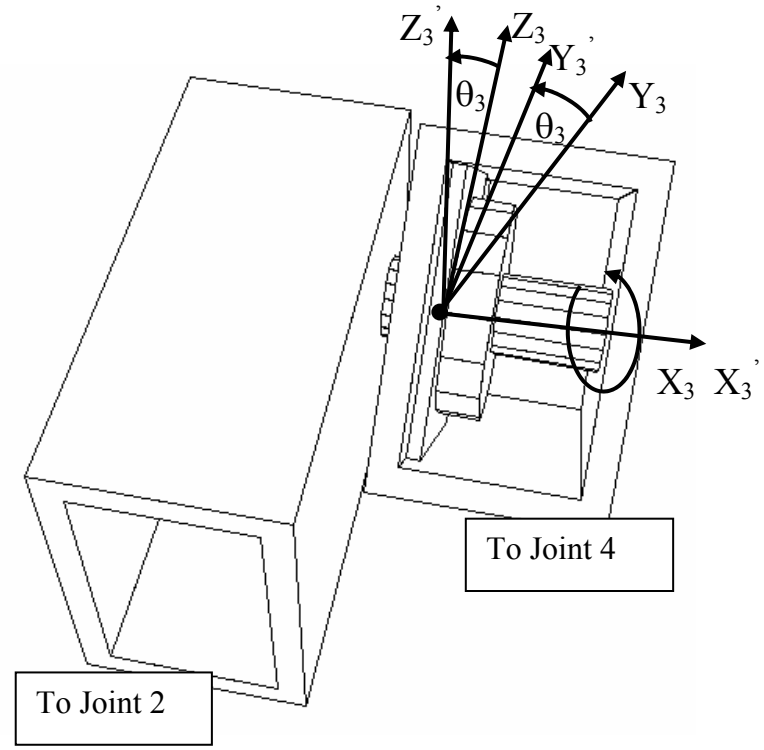


Figure 8.7: A coordinate system is shown coincident with the center of joint 3's shaft and the inner wall of the joint half closer to the arm's tip. Rotation is about the x-axis of the applied coordinate system.

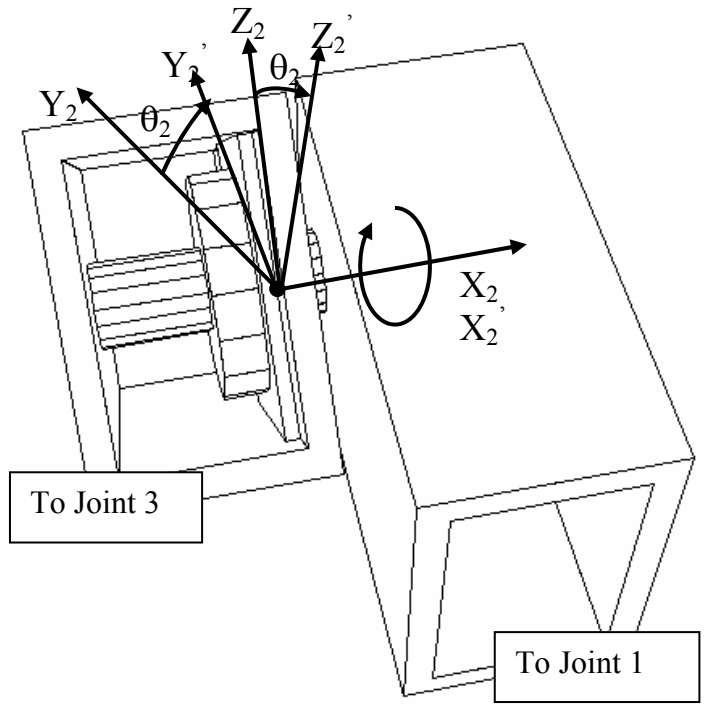


Figure 8.8: A coordinate system is shown coincident with the center of joint 2's shaft and the inner wall of the joint half closer to the arm's tip. Rotation is about the x-axis of the applied coordinate system.

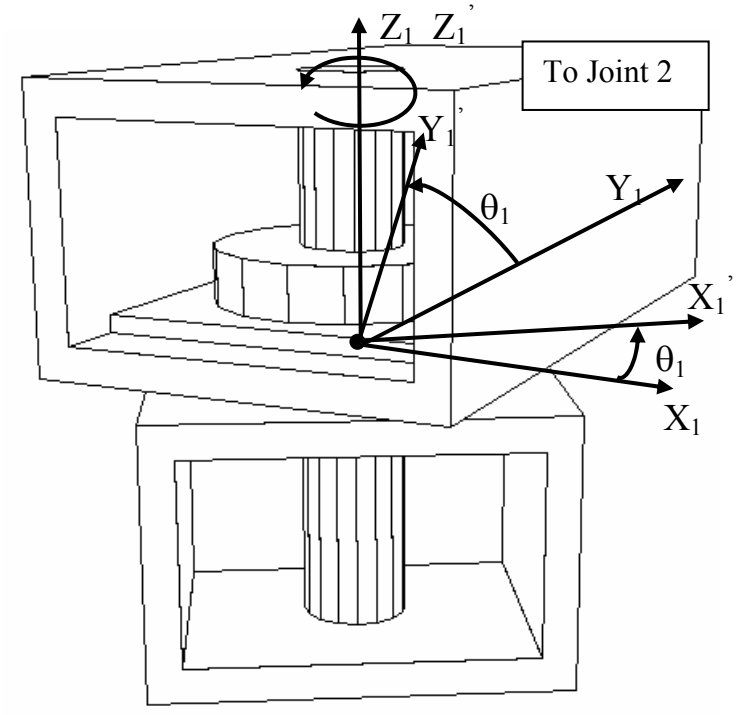


Figure 8.9: A coordinate system is shown coincident with the center of joint 1's shaft and the inner wall of the joint half closer to the arm's tip. Rotation is about the z-axis of the applied coordinate system.

Geometry:

The geometry of the arm was described by the relative position each of coordinate system relative to the adjacent coordinate system closer to the end of the arm. Describing the geometry is this way allowed link dimensions to be mathematically treated as coordinate system translations. The dimensions of each link, given in the coordinate system of the joint attached to it, are shown in Table 8.2.

Link #	X (inches)	Y (inches)	Z (inches)
1	-0.75	14.09	1.25
2	-0.983	62.07	-1
3	1.04	9.077	-1.009
4	-1.286	9.69	-0.25
5	-0.258	17.789	-1
6	1.045	62.512	1
7	-2.266	66.712	-0.5

Table 8.2: The geometry of each link is given relative to coordinate system of the joint, attached to the link that is closer to the arm's base. Note that these values were computed by measuring the final dimensions of each link (which were not machined very accurately), and using those measurements in conjunction with the specified dimensions of each joint (which were machined accurately).

Kinematic Equations:

The kinematic analysis of the arm consisted of a series of body fixed rotations and space fixed translations at each joint as well as a set of translations and rotations at the arm's base. The formula used to apply each individual translation and rotation is given as Equation 8.1 where R is a 4x4 transformation matrix and Position is a 4 x 1 matrix in which the first 3 entries are a x, y, z coordinates and the fourth entry is set to 1.

Position = R * Position

Equation 8.1: Translations and rotating are applied to a 4x1 position vector, by matrix multiplication, with a 4x4 transformation matrix R.

The equation for a space fixed translation transformation matrix is given as Equation 8.2. Equations 8.3, 8.4, and 8.5 provide the transformation matrices for body fixed rotations about the x-axis, y-axis, and z-axis respectably.

$$\mathbf{R}_{\mathbf{Trans}} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{\Delta X} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{\Delta Y} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{\Delta Z} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

Equation 8.2: A 4x4 transformation matrix is shown in which delta x, y, and z are translations in the x, y , and z directions respectably. (Ferguson 25)

$$\mathbf{R}_{\mathbf{RotX}} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{\cos(\theta_X)} & \mathbf{\sin(\theta_X)} & \mathbf{0} \\ \mathbf{0} & \mathbf{-\sin(\theta_X)} & \mathbf{\cos(\theta_X)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

Equation 8.3: A 4x4 transformation matrix is shown for the application of body fixed rotations about the x-axis. (Ginsberg 58) (Ferguson 26)

$$\mathbf{R}_{\mathbf{RotY}} = \begin{bmatrix} \mathbf{\cos(\theta_Y)} & \mathbf{0} & \mathbf{-\sin(\theta_Y)} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{\sin(\theta_Y)} & \mathbf{0} & \mathbf{\cos(\theta_Y)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

Equation 8.4: A 4x4 transformation matrix is shown for the application of body fixed rotations about the y-axis. (Ginsberg 58) (Ferguson 26)

$$\mathbf{R}_{\mathbf{RotZ}} = \begin{bmatrix} \mathbf{\cos(\theta_Z)} & \mathbf{\sin(\theta_Z)} & \mathbf{0} & \mathbf{0} \\ \mathbf{-\sin(\theta_Z)} & \mathbf{\cos(\theta_Z)} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

Equation 8.5: A 4x4 transformation matrix is shown for the application of body fixed rotations about the z-axis. (Ginsberg 58) (Ferguson 26)

The application of multiple translations and rotations was handled by applying equation 8.1 in conjunction with equations 8.2, 8.3, 8.4, or 8.5, and using the output from that operation as the input for the next operation. For example to find the potion of the arms tip relative to joint 6, Equation 8.2 is applied to a position vector of $[0\ 0\ 0\ 1]^T$ translate from the arm's tip to the origin of coordinate system 7. Equation 8.3 is then

applied to the new position vector to rotate from coordinate system 7 to coordinate system 7'. Lastly, Equation 8.2 is applied, once again, to translate, along joint 6, to the origin of coordinate system 6.

Table 8.3 lists the sequence of transformations applied to a position vector in order to find the location of a point on the on the arm's end link relative to the base coordinate system $X_1Y_1Z_1$' shown in Figure 8.9, where the original position vector is given as Position = $[0\ 0\ 0\ 1]^T$.

Table 8.3: The sequence of transformations listed in this table, from top to bottom, were applied to find the location of the arm's tip relative to base coordinate system $X_1Y_1Z_1$'. The multiplier of –1 on each joint angle is included as a reminder that coordinate frame rotation is the opposite direction as physical joint rotation, as per the established convention given above.

Num	Purpose	Type	Magnitude		
1	Link 7 Geometry	Translation	X= -2.266	Y= 66.712	Z= -0.5
2	Joint 7 Twist	Rotation about y-axis	0 radians		
3	Joint 7 Angle	Rotation about x-axis	-1 * Theta 7		
4	Link 6 Geometry	Translation	X= 1.045	Y= 62.512	Z= 1
5	Joint 6 Twist	Rotation about y-axis	0 radians		
6	Joint 6 Angle	Rotation about x-axis	-1 * Theta 6		
7	Link 5 Geometry	Translation	X= -0.258	Y= 17.789	Z= -1
8	Joint 5 Twist	Rotation about y-axis	0 radians		
9	Joint 5 Angle	Rotation about x-axis	-1 * Theta 5		
10	Link 4 Geometry	Translation	X= -1.286	Y= 9.69	Z= -0.25
11	Joint 4 Twist	Rotation about y-axis	0 radians		
12	Joint 4 Angle	Rotation about z-axis	-1 * Theta 4		
13	Link 3 Geometry	Translation	X= 1.04	Y= 9.077	Z= -1.009
14	Joint 3 Twist	Rotation about y-axis	0 radians		
15	Joint 3 Angle	Rotation about x-axis	-1 * Theta 3		
16	Link 2 Geometry	Translation	X= -0.983	Y= 62.07	Z= -1
17	Joint 2 Twist	Rotation about y-axis	0 radians		
18	Joint 2 Angle	Rotation about x-axis	-1 * Theta 2		
19	Link 1 Geometry	Translation	X= -0.75	Y= 14.09	Z= 1.25
20	Joint 1 Twist	Rotation about y-axis	0 radians		
21	Joint 1 Angle	Rotation about z-axis	-1 * Theta 1		

Additionally, the transformations listed in Table 8.4 were applied to the position of the arm's tip in the $X_1Y_1Z_1$' base coordinate system. This transformation sequence served the purpose of finding the tip position relative to some fixed reference on the

floor, as determined by geometry parameters also given in Table 7.3 . Knowing the position of the tip relative to a floor reference was desired because it would help with calibration, and because it would prevent the arm from reading the floor plane as being negative, something that could be counter intuitive to a user receiving real time feedback from the arm. As such, the Height parameter in Table 8.4 is set to the height that coordinate system $X_1Y_1Z_1$' is above the floor, typically around 7 inches. The other values in the table are set such that the zero coordinate of the system is at some known position on the ground.

Num	Purpose	Type	Magnitude
22	Direction base is pointing	Rotation about z-axis	0 radians
23	Leveling of base joint	Rotation about y-axis	0 radians
24	Leveling of base joint	Rotation about x-axis	0 radians
25	Location and height of base joint	Translation	X = 0 Y = 0 Z = Height

Table 8.4: The sequence of transformations listed in this table, from top to bottom, were applied to the location of the tip, relative to the base coordinate system $X_1Y_1Z_1$' in order to find the location of the tip relative to some point on the floor.

Section 9: Joint Assembly

Joint Assembly Overview:

The construction of the scanner arm required seven pinned one degree of freedom encoder joints to be machined and assembled. Assembly of these joints was somewhat complicated, and therefore a record of the joint assembled is provided. This record should aid in the construction of any additional joints and aid in the disassembly of any joints if such action is required.

Joint Assembly Walkthrough:

1. The encoder half, with two journals pressed into it, is laid out such that its six screwdriver access holes face up. Small #6 flat spacer washers are then placed over each encoder mounting plate attachment hole.
2. The encoder mounting plate is positioned over its attachment holes, and ½" 4-40 screws are placed in the holes using a magnetic screwdriver inserted through four of the access holes.
3. A small wrench is used to secure 4-40 nuts onto the ends of each screw just inserted.
4. A 1/8 inch spacer waster and a bearing assembly is slid over the shaft of the mating joint half.

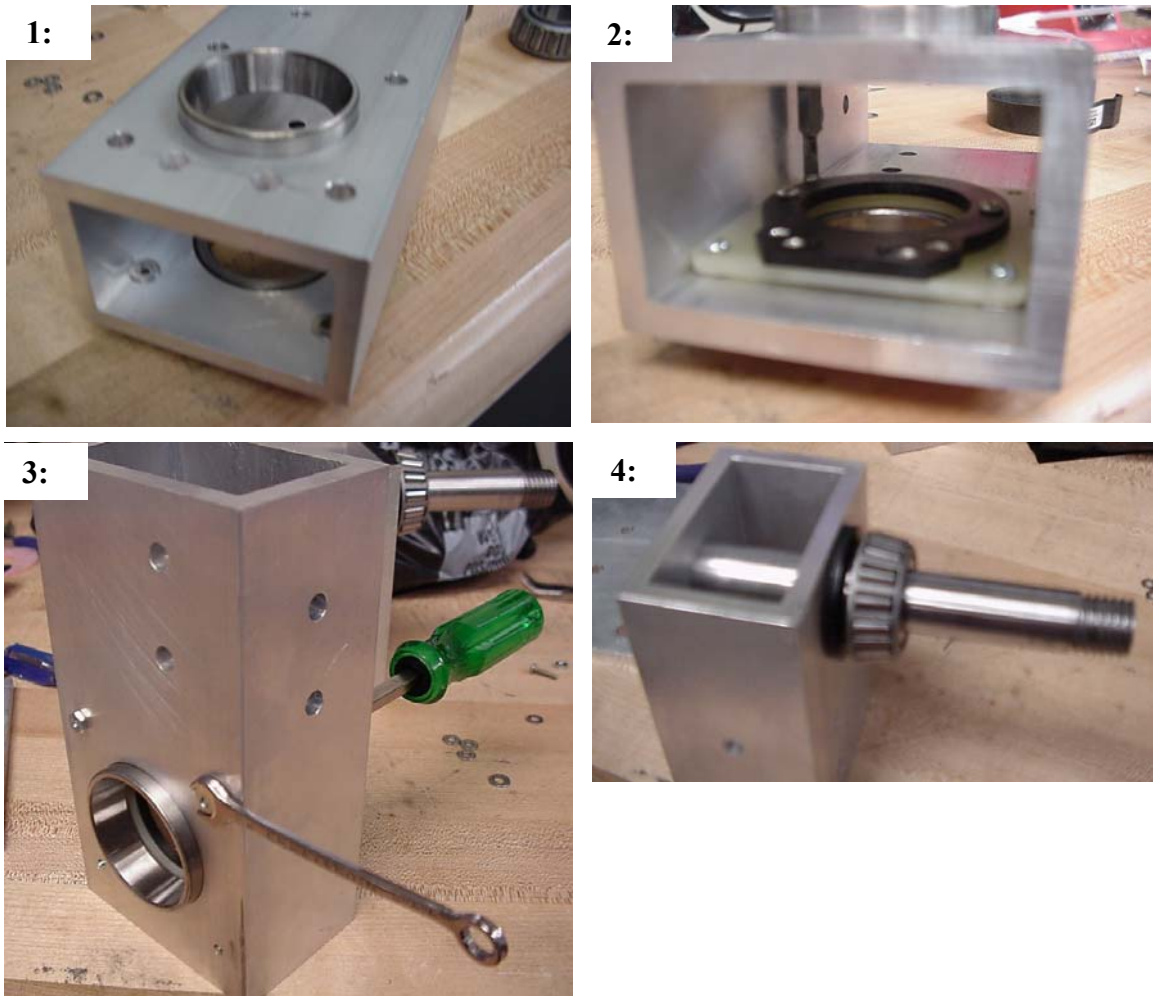


Figure 9.1: The encoder mounting plate is shown being installed in one half of a joint, and the other half of the joint is shown being prepared for assembly.

5. The joint half, possessing the journals, is *partially* slid over the ¾ inch shaft.
6. The encoder disk / hub assembly is gently grasped at it edges and positioned at the end of the shaft. An index finger, placed though the hole in the hub, holds the assembly in position while the two joint halves are brought partially together.
7. The encoder's outer housing is slid over the shaft after the hub / disk assembly.
8. As the two halves are brought completely together, the small hex-head tool supplied with the encoder is used to move the hub/disk assembly so that it does not press against the wall of the joint half.

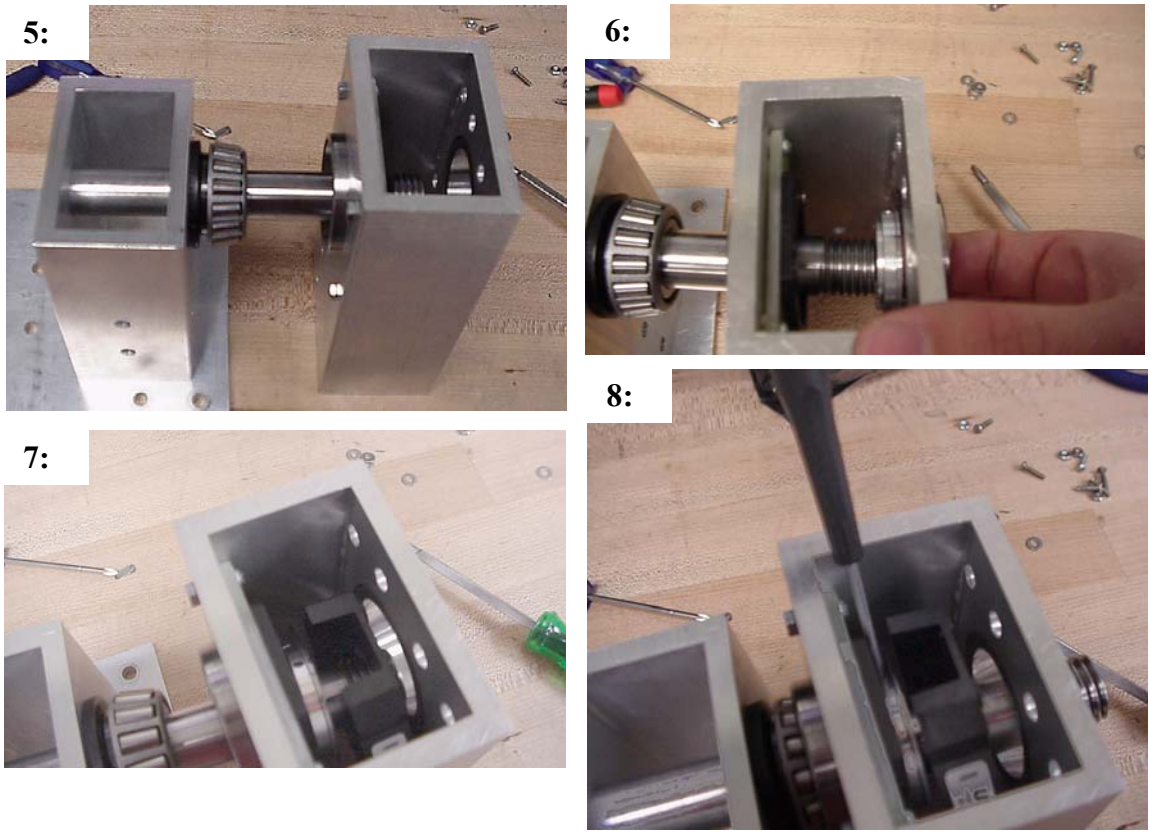


Figure 9.2: The partial installation of an encoder disk is shown.

9. A second bearing is slid onto the end of the shaft, and a ¾ - 10 nut is screwed onto the threaded portion of the shaft to hold the joint together.
10. A pair of needle nose pliers was used to set the spacer tool, provided with the encoder, around the smaller diameter of the encoder disk hub. Note that the spacer is applied from the *far* side of the housing. This is because a small tab exists on the near side of the mounting plate that will interferes with the spacing tool.
11. The hub is pressed as close to the mounting plate as possible, archiving the proper disk spacing.
12. Using the supplied hex-head tool, the two setscrews in the encoder disk hub are tightened down until the disk / hub assembly is secured.

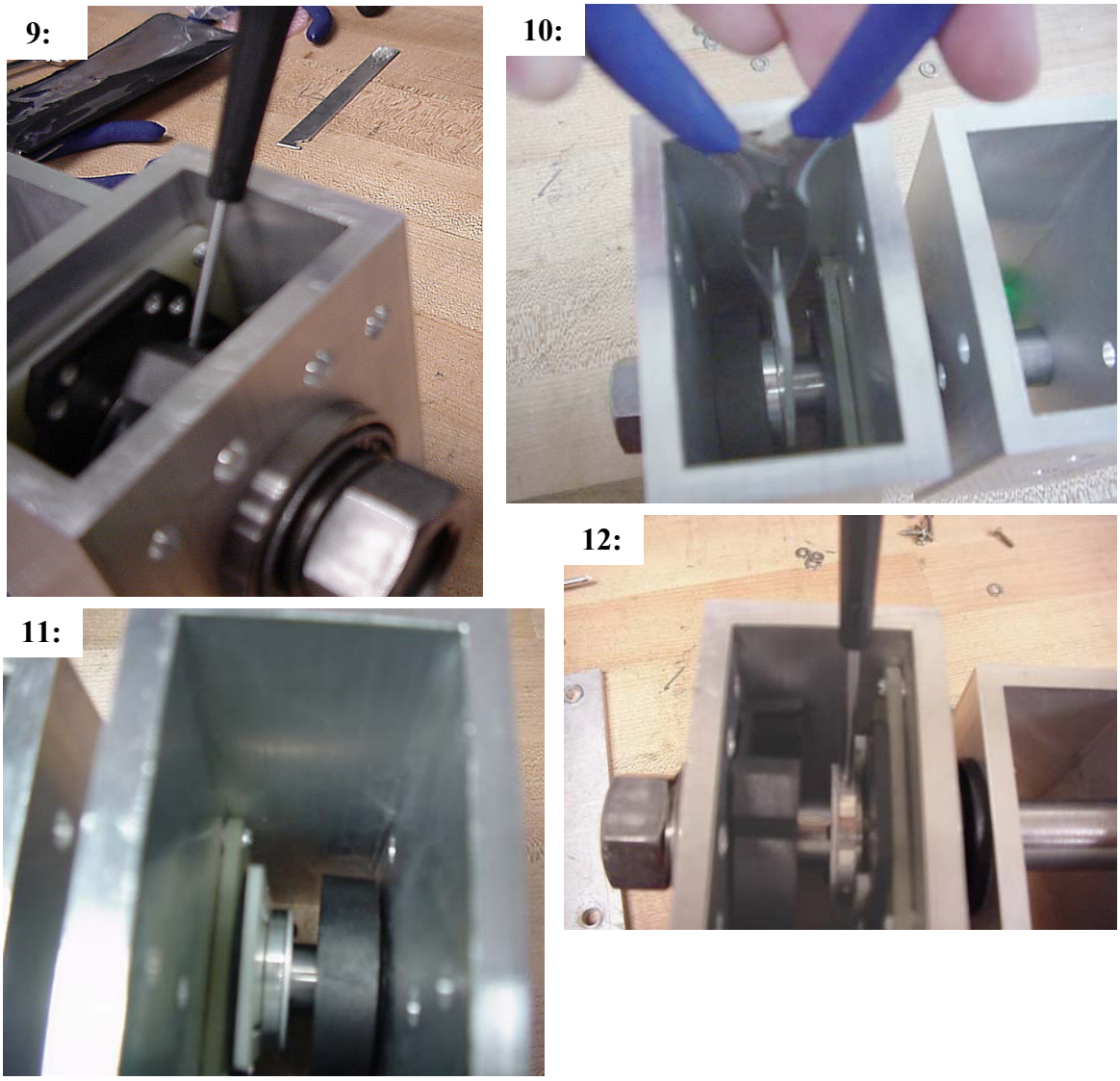


Figure 9.3: Finally positioning of a encoder disk is shown.

13. A small Philips head screwdriver is used to screw two of the pan head screws, supplied with the encoder, far enough into the encoder' electronics module to just barely emerge on the other side. During this operation, and all other operations in which the electronics module is handled, a grounding strip must be worn.
14. The module is positioned on the encoder mounting plate such that two small tabs in the plate fall into two divots in the bottom of the module. The module is then

attached to the encoder mounting plate via the Philips head screws. Two of the access holes are used to reach the screws with a non-magnetic screwdriver.

15. The encoder housing is slid into place over the assembly.
16. Two additional screws, supplied with the encoder, are used to secure the encoder hosing in place, completing the joint assembly.

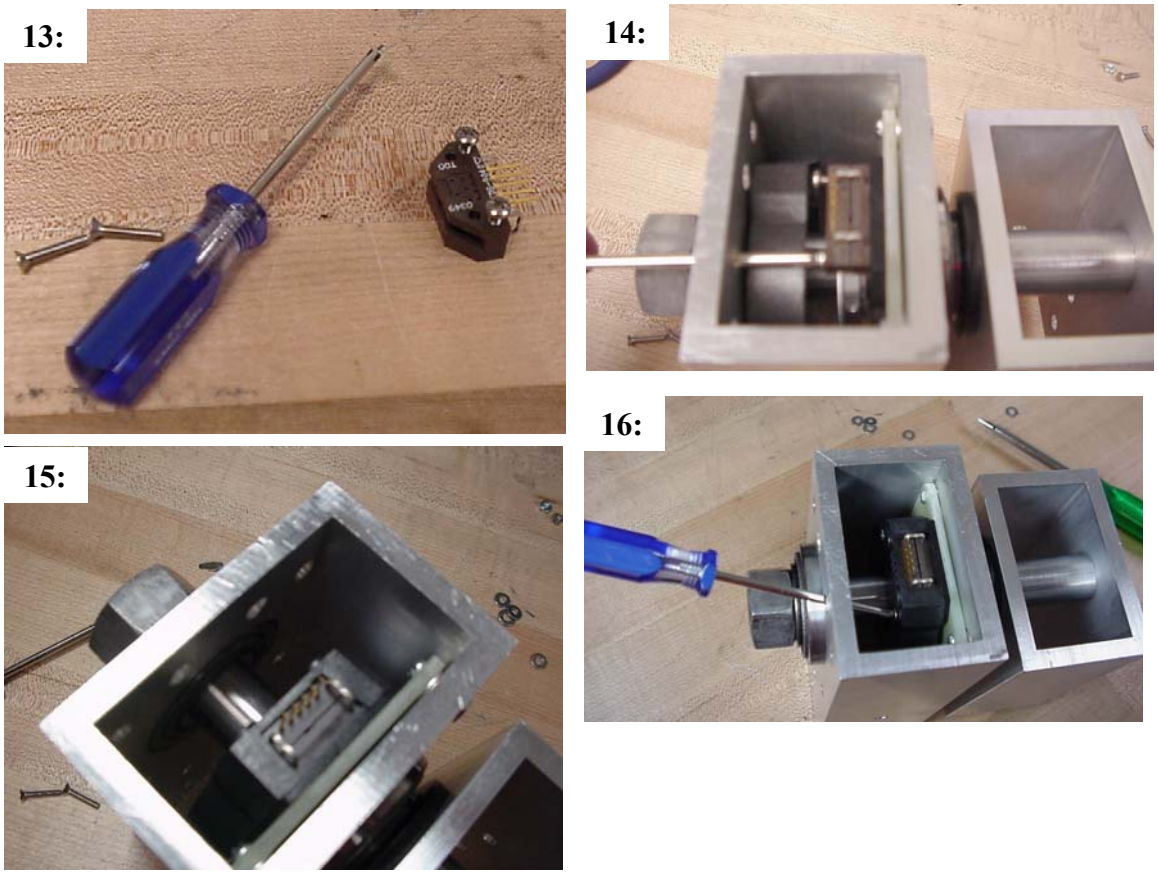


Figure 9.4: Installation of an encoder electronics module is shown along with the means of securing the cover on an encoder

Section 10: Digitizer Arm Software Overview

Software Purpose:

During the course of the digitizer arm project, it became clear the some sort of software would be required to keep track of, manipulate, and display the data collected by the device. Currently, several commercial packages exist for displaying 3D data and still others exist specifically for dealing with point cloud data generated by various scanning devices. However, in the interest of the flexibility, it was decided that an open source software package would be created for use with the arm. Because of its ease of use, and existing 3D functionally, the Matlab programming environment was chosen for this purpose.

Software Functionality:

- Control of digitizer arm digital signal processor (DSP) during operation
- Querying of encoder angular position information
- Converting imported angular data a points in 3-dimentional space
- Storage and generation of point cloud
- Organization of data based on user input
- Graphical display of point cloud and relevant user data
- Meshing of point cloud
- Display, and script based animation, of 3D scenes

Required Software / Software Compatibility:

The digitizer arm software was written in Matlab Version 6.5.0.1924 Release 13 running on the Window 98. The digitizer arm software makes use of no additional Matlab toolboxes or third party plug-ins. Hence the software should run correctly on any

system using the named software, or later versions. This is, of course, not a guarantee, and it's up to the user to confirm proper operation of the software, particularly if it's run on a different version of Matlab.

Required Files:

The Digitizer Arm Software consists of the following files, containing code that is either explicitly called from the command line or which is used as a subroutine. All of these programs must exist in the Matlab Search Path in order for the software to work properly.

Check4Holes.m	EditScene.fig
CutExtraPoly.m	EditScene.m
DefineBounds.m	EditVertex.fig
EditPoly.m	EditVertexm
MeshGroup.m	InitializeScene.m
MeshSurface.m	InitializeScript.m
OrientSurface.m	InitializeWorkspace.m
RefineBounds.m	Local2World.m
Vertor2Rotation.m	ReadScript.m
Animate.m	CaptureAngles.m
DisplayAxis.m	CaptureData.fig
DisplayCamera.m	CaptureData.m
DisplayHoleEdges.m	GetArmGeometry.m
DisplayLight.m	GetTipPos.m
DisplaySurface.m	readEncoders.m
DrawScene.m	StartCal.m
EditObject.fig	
EditObject.m	

Data Storage:

Prior to capturing data with the digitizer arm, it is important to understand the data hierarchy in which point cloud data, and all data generated from it, is stored. The following section provides an overview of said data hierarchy, as well as how it is actually implemented using Matlab variables. Also, the following section provides, the definitions to several terms used extensively in the remainder of this software write-up.

Terms and Definitions:

Object: A term used to describe grouping of data within a scene.

Scene: A collection of vertices, polygons, surfaces, and groups that form a virtual environment.

Point: A location in space, described by an x, y, and z coordinate, that describes the captured position of the tip of the digitizer arm at a location of interest.

Vertex: One of many points in space connected by polygons to describe a 3D object. For example, a minimum of 8 vertices would be required to describe a cube.

Polygon: A multi-sided planner facet of a 3D surface, described by connecting multiple vertices with a plan bounded by lines drawn between vertices. In the digitizer arm software, all polygons are three sided, and thus are described by three vertices. Hence, a minimum of 12, three sided, polygons would be required to describe a cube.

Surface: A collection of polygons that share vertices and collinear bounding edges. In this software, surfaces are restricted to 3D constructs that can be

collapsed into a 2D construct without loss of connectivity information. Hence, a minimum of 6 surfaces would be required to describe a cube.

Group: A collection of surfaces, lights, and cameras that are grouped together. Each group has it's own local coordinate system in which objects contained in the group can be described.

Render: The process of converting numerical data, describing a scene, into an image.

Light: A source of illumination in a virtual scene which determines how polygons will be shaded.

Camera: A virtual camera, existing in a scene, that describes the viewpoint from which the scene is rendered. In the digitizer arm software, multiple cameras may exist in a scene, but only on may be in use at a time.

Transformation: A mathematical operation in which coordinate systems are moved though 3D space.

Rotation: A transformation in which everything in a coordinate system is revolved about an axis if that system.

Translation: A transformation in which everything in a coordinate system is move linearly relative to a parent coordinate system.

Global Coordinate System: A coordinate system in which all objects in a scene exists.

Local Coordinate System: A coordinate system existing within the global coordinate system.

Parent: In a data hierarchy, a parent is a high-level data object that owns a lower level on. All operations done to a parent object affect its children.

Child: In a data hierarchy, a child is a lower level data object owned by a parent.

Active / Inactive: Data objects may be active or inactive. An active object will be graphically displayed when a scene is rendered. An inactive object will not be visible.

Data Hierarchy:

All data stored by the digitizer arm software is organized in a data hierarchy. In this hierarchy, lower-level data objects, such as polygons, are used to construct higher-level data objects such as surfaces. Each data object is stored in its own array. References to other arrays establish the actual hierarchy. For instance, a surface object would posses a list of polygons included in that surface. Furthermore, each data object possesses several attributes, dependent on the type of object, that are unique to the object, the color of a surface for instance.

All data objects of the same type are stored in a signal global variable, containing the arrayed data of each existing object of that type. The types of data objects are as follows.

Point Data Object:

The most fundamental data object stored by the digitizer arm software is a point. As given in the definition section, a point is the location of the tip of the digitizer arm when it was in contact with something of interest and position data was captured. A point is described by an x, y, and z coordinate in the global coordinate system. In the Matlab workspace, point data is stored in the "Point" global array given in Table 10.1.

Raw Angle Data Object:

The raw data object simply stores the angles of each encoder corresponding to each point in the Point data object. In the Matlab workspace, point data is stored in the "RawAngle" global array given in Table 10.1.

Vertex Data Object:

In the digitizer arm software, a vertex is a point in space that describes the corners of a three-sided triangle (See Figure 10.1). The vertex itself is described by an x, y, and z coordinate in the local coordinate system of the group the vertex belongs to. In the Matlab workspace vertex data is stored in the "Vertex" global array given in Table 10.1.

Polygon Data Object:

In the digitizer arm software, a polygon is a three dimensional triangle whose corners are described by three vertices as shown in See Figure 10.1. In the Matlab workspace, polygon data is stored in the "Poly" global array as lists of the three vertex indices that make up each polygon, as given in Table 10.1.

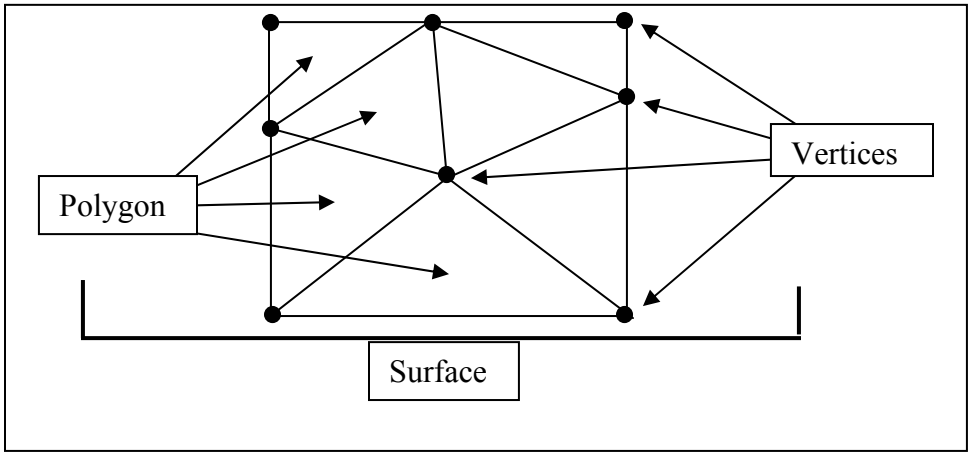


Figure 10.1: A depiction of the relationship between Vertices, Polygons, and Surfaces

Surface Data Object:

A surface object consists of one or more polygons that are connected such that they share common vertices and polygon edges, forming a sheet as shown in Figure 10.1. In the digitizer arm software, there is a limitation on the shape of surfaces. Specifically, a surface must be able to be projected onto a 2d plane without any portion of the surface overlapping, i.e. the surface must be describable with an arbitrarily orientated elevation map. The restriction exists due to the simplicity of the meshing algorithm included in the software. If meshing is not desired, or a markedly improved meshing algorithm, possibly a commercial algorithm, is implemented in the future, this restriction would no longer apply. Figures 10.2 and 10.3 give examples of acceptable and unacceptable surfaces, while Figure 10.4 gives an example of how to break a cylindrical object up into surfaces. In the Matlab workspace surface data is stored in the "Surface" global array of structures, and given in Table 10.1.

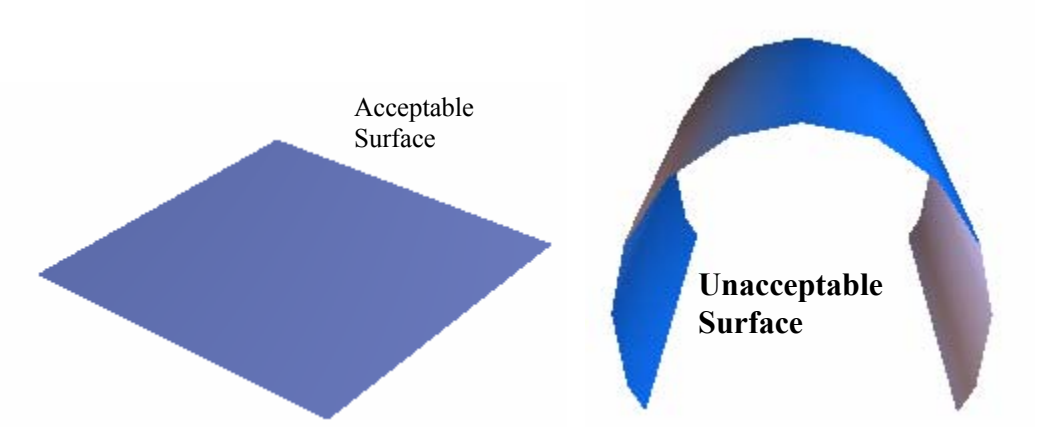


Figure 10.2: Examples of an acceptable and an unacceptable surface. The unacceptable surface curves underneath itself and cannot be described by a elevation map.

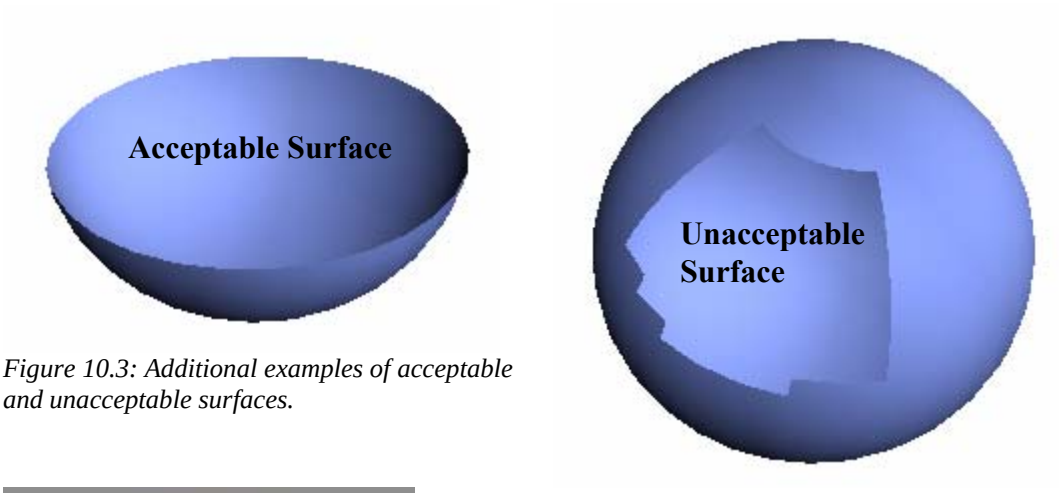


Figure 10.3: Additional examples of acceptable and unacceptable surfaces.

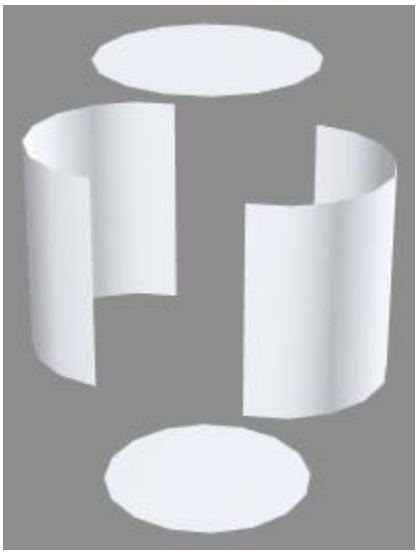


Figure 10.4: The decomposition of a cylinder into acceptable surfaces is shown. Four surfaces are required for such a decomposition.

Light Data Object:

A light data object, like its name implies, represents a light source in a scene. Like in the real world, light objects within a scene control how objects are colored and shaded. Light objects may consist of point lights (like a light bulb) or directional lights (like light coming from the sun). Light position specified in a local coordinate system, directionality, and color may be specified in this data structure. In the Matlab workspace surface data is stored in the "Light" global array of structures, and given in Table 10.1.

Camera Data Object:

A camera object describes a virtual camera used to view a scene. This camera has a position in a local coordinate system, a target in the same coordinate system, and an angle of viewing, somewhat like the zoom on a camera. Multiple cameras may exist in a scene, but only one may be in use or "active" at a given time. In the Matlab workspace surface data is stored in the Camera global array of structures, and given in Table 10.1.

Group Data Object:

Group objects are the highest-level objects in the digitizer arm software's data hierarchy. A group may have as its child: a surface, a light, a camera, or another group. The unique quality about groups is that each one possesses its own local coordinate system in which its children exist. Furthermore, if a group exists within another group, its coordinate system may be translated in the parent coordinate system and it may be rotated about any one of its axis multiple times in a series of body fixed rotations. In this way objects contained within a group, and thus within its coordinate system, may be moved trough space without the need to change the actual position of points, vertices, lights, or cameras in their own data structures.

Additionally all objects, excluding the Render object, are children of either the World group or the Trash group. The world group describes an absolute global coordinate system that never changes. It can be thought of as the Earth in many cases. All objects in the world group may be displayed and manipulated by the various programs included in this software.

By contrast, the Trash group is the equivalent of the trash bin in most operating systems. Anytime an object is "deleted" its data is not actually erased, but instead pointers to the data are eliminated from the hierarchy headed by World group and placed in the trash group. As such, objects in the trash group cannot be manipulated by functions that effect data in the World group hierarchy. Furthermore, objects in the Trash group cannot be displayed when a scene is rendered.

Lastly, it should be noted that when a surface is present in a group, it is a useful practice to include all surfaces in that group that would form a single inflexible solid object. In this way a group can also be thought of as a physical construct that can be moved relative to another object (when it's the child of a said object), or which can have another movable object attached to it (when it's the parent of said object). This physical way of thinking about groups can expedite model building and animation.

Render Data Object:

Unlike the other previously mentioned data objects types, of which there may be multiple data objects, there is only one render data object. The render data object controls what information is displayed when the scene is rendered as well as how that information is displayed. This variable is stored in the global "Render" array struct, which is explained in Table 10.1. The results of changing the Render Data Object are discussed in a later section on controlling the display of scene information using the included "EditScene" GUI.

Calibrate Data Object:

Also, like the Render Data Object, there is only one calibrate data object. The calibrate object stores all information pertaining to the calibration of the digitizer arm. This variable is stored in the global "Calibrate" array struct, which is explained in Table 10.1.

Organization of Objects:

Terms and definitions aside, perhaps the best way to explain the function of each object and its place in the data hierarchy is with an example. Say that it is desirable to construct a 3D rendition of a van, which can be placed in a scene, perhaps containing a road or some buildings. And, say that it is desirable to be able to position the wheels of the vehicle independent of each other except that they should remain attached to the body of the vehicle at all times. Such a scene could be constructed using the hierarchy shown in Figure 10.5.

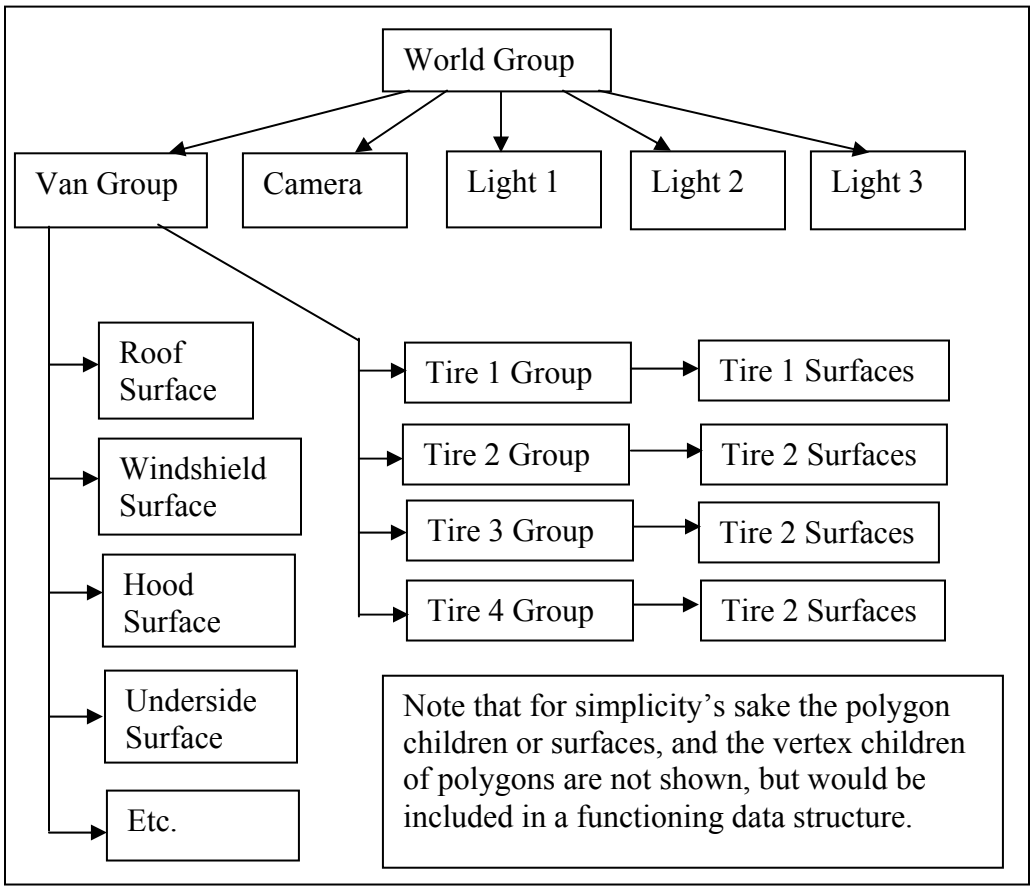


Figure 10.5: An example scene data hierarchy containing a van, a camera, and three lights.

Using the data structure described in Figure 10.5, a van could be translated in any direction within the world group or rotated about its own x, y, or z local axis to simulate roll, pitch, or yaw. Furthermore, the tires, existing as children of the van group, would follow the van group's motion automatically. Actions such as turning the wheels could be accomplished by rotating the wheel group of interest. The wheel group could also be translated within the van group to simulate the actions of the van's shocks.

A graphical rendition of this scene, created using the 3D graphics package, Truespace 4.5 is shown in Figure 10.6.

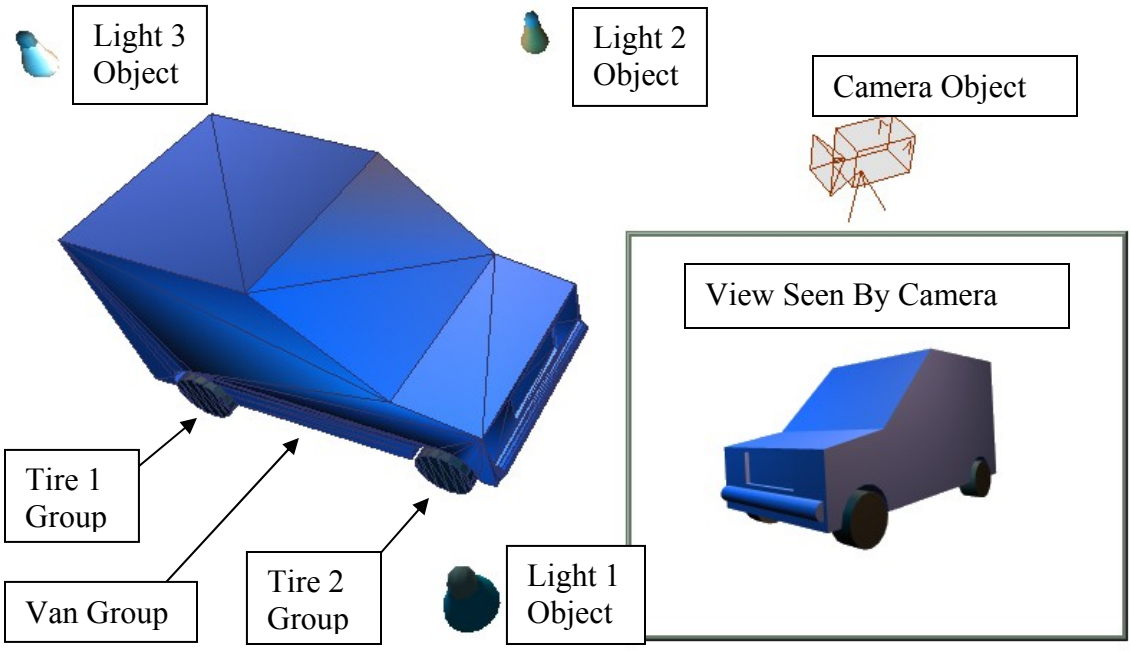


Figure 10.6: Example scene containing a group objects and its children. The data hierarch describing this scene is shown in Figure 10.5.

Matlab Variables

The following Table 10.1 provides an overview of the variables stored in the Matlab workspace during regular operation of the digitizer arm software.

Variable	Storage Type	Attributes	Data Type	Description	Data Format
Point	Numeric (N x 3) Array	N/A	[Nx3 double]	Point x y z Coordinates. One point per row.	[x1, y1, z1] [x2, y2, z2] [etc…]
Vertex	Numeric (N x 3) Array	N/A	[Nx3 double]	Vertex x y z Coordinates. One vertex per row.	[x1, y1, z1] [x2, y2, z2] [etc…]
Poly	Numeric (N x 3) Array	N/A	[Nx3 double]	Vertex row indices, forming a 3 vertex polygon. One polygon per row.	[vert 1, Vert2, Vert3] [vert 1, Vert2, Vert3] [etc.]
Surface	1D Array of struct arrays	Name	string	User supplied surface name.	['name']
	1 struct array per object	IsActive	logical	Indicates if the surface is active.	[true / false]
	of Surface type	PointList	[1xN uint16]	List of all points in the surface by their indices in the Point variable.	[P1, P2, P3, etc]
		EdgePoints	[1xR cell [NxM uint16]]	Each cell contains the indices of points that for a boundary to the surface. In each cell this is a n x m array with n sequences of up to m points that are known to be connected when forming a boundary. Zeros are used to fill the array when there are not m points in a row.	{ [P1, P2, P3] , [P1, P2, 0] , [P1, 0, 0] , [P1, P2, P3] } { [P1, P2, P3, P4] , [P1, 0, 0, 0] , [P1, P2, P3, 0] }
		BoundsList	[1xR cell [1xN uint16]]	Each cell contains a list of ordered vertex indices that describe a complete boundary to the surface when connected in order.	{[v1, v2, v3, v4, etc.] ,[v1, v2, v3, v4, etc.] ,[etc.]}
		PolyList	[1xN uint16]	List of all polygons contained in the surface by their indices in the Poly variable.	[Poly1, Poly2, Poly3, Poly4, etc.]
		FaceColor	string	Color of surface may be 'cyan', 'magenta', 'yellow', 'red', 'green', 'blue', 'white', or 'black'.	['color']
Light	1D Array of struct arrays	Name	string	User supplied light name.	['name']
	1 struct array per object	IsActive	logical	Indicates if the light is active.	[true / false]
	of Light type	Color	string	Color of light may be 'cyan', 'magenta', 'yellow', 'red', 'green', 'blue', 'white', or 'black'.	['color']
		Position	[1x3 double]	If a local point light, describes position of light in parent group coordinate system. If a infinite light, is a vector describing direction of light.	[x, y, z]
		Style	string	Indicate if light is local point light, infinite directional light without a origin	['local'] or ['infinite']

Table 10.1: Important Variables Stored to Matlab Workspace

Variable	Storage Type	Attributes	Data Type	Description	Data Format
Camera	1D Array of struct arrays	Name	string	User supplied camera name	['name']
	1 struct array per object of Camera type	IsActive	logical	Indicates if the camera is active. Only one camera may be active (in use) at a time.	[true / false]
		Position	[1x3 double]	Position of camera in parent coordinate system.	[x, y, z]
		Target	[1x3 double]	Position of what the camera is aimed at in the parent coordinate system.	[x, y, z]
		ViewAngle	double	Angle, in degrees, describing the size of the cone of vision extending from the camera object. ViewAngle must be between 0 and 180 degrees.	[ViewAngle]
Group	1D Array of struct arrays	Name	string	User supplied group name	['name']
	1 struct array per object	IsActive	logical	Indicates if the group is active	[true / false]
	of Group type	Parent	Uint16	Index of parent group in group variable. The world group and trash groups use a zero here.	[ParentGroupIndex]
		ShowAxis	logical	Indicates if the local axes of the group will be displayed when the scene is rendered.	[true / false]
		GroupList	[1xN uint16]	List of indices, in the group variable, of all children groups of the group in question.	[G1, G2, G3, etc]
		SurfaceList	[1xN uint16]	List of indices, in the surface variable, of all surface children of the group.	[S1, S2, S3, etc]
		LightList	[1xN uint16]	List of indices, in the light variable, of all light children of the group.	[L1, L2, L3, etc]
		CameraList	[1xN uint16]	List of indices, in the camera variable, of all camera children of the group.	[C1, C2, C3, etc]
		Alpha	double	Sets all surface in group to be opaque (value of 1), transparent (value of 0), or semi-transparent (value between 1 and 0).	[alpha]
		SetupTrans	[1x3 double]	A translation applied to the group coordinate system to position said system.	[x, z, z]
		SetupRot	[2x3 double]	A set of three rotations applied to group coordinate system to postion said system. The first row gives the magnitude of each rotation in radians. The second row give the axis of rotation for each rotation where xaxis=1, yaxis=2, and zaxis=3.	[rot1, rot2, rot3] [axis1, axis2, axis3]

Table 10.1 (Continued): Important Variables Stored to Matlab Workspace

Variable	Storage Type	Attributes	Data Type	Description	Data Format
Group (Continued)	1D Array of struct arrays. 1 stuct array per object of group type.	OffsetTrans	[1x3 double]	A2nd translation applied to the group coordinate system to animate said system.	[x, y, z]
		OffsetRot	[2x3 double]	A second set of three rotations applied to group coordinate system to animate said system. The first row gives the magnitude of each rotation in radians. The second row give the axis of rotation for each rotation where xaxis=1, yaxis=2, and zaxis=3.	[rot1, rot2, rot3] [axis1, axis2, axis3]
		HoleEdgeList	[1xN uint16]	List of all vertices, by vertex index, that bound a hole found in the mesh formed by all the surfaces in a group.	[v1, v2, v3, v4, etc]
Render	struct array	Lighting	string	Indicates type of lighting. Valid options are 'none', 'flat' 'gouraud' and 'phong'.	['phong']
		BackgroundColor	string	Color of scene backdrop may be 'cyan', 'magenta', 'yellow', 'red', 'green', 'blue', 'white', or 'black'.	['color']
		AxisScaling	double	Scales local axis, if displayed. A value of larger than one scales the axis up, a less than one scales the axis down, and a value of 1 does not scale the axis.	[1]
		AxisThickness	double	Provides thickness of local axis, if displayed.	[1]
		Projection	string	Indicate type of project used by rendered. Valid options are 'orthographic' or 'perspective'	orthographic'
		ShowPoints	logical	Displays all points, in active surfaces, when set to true	[true / false]
		ShowVertex	logical	Displays all vertices, in active surfaces, in scene when set to true.	[true / false]
		ShowPolygons	logical	Displays all polygons, in active surfaces, when set to true.	[true / false]
		TextSize	double	Sets size of labeling test in graphic window. Size is specified in points.	[12]
		TextColor	string	Color of labeling text may be 'cyan', 'magenta', 'yellow', 'red', 'green', 'blue', 'white', or 'black'.	['color']
		LabelPoints	logical	Labels all points, currently displayed, with their row indices in the global Point variable when set true.	[true / false]

Table 10.1 (Continued): Important Variables Stored to Matlab Workspace

Variable	Storage Type	Attributes	Data Type	Description	Data Format
Render (Continued)	struct array	LabelVertex	logical	Labels all vertices, currently displayed, with their row indices in the global Vertex variable when set true.	[true / false]
		LabelPolygons	logical	Labels all polygons, currently displayed, with their row indices in the global Poly variable when set true.	[true / false]
		LabelSurface	logical	Labels all surfaces, currently displayed, with their indices in the global Surface variable when set true.	[true / false]
		LabelLight	logical	Labels all lights, currently displayed, with their indices in the global Light variable when set true.	[true / false]
		LabelAxis	logical	Label all axis when "x", "y", "z", and name of group the axis belongs to when set true.	[true / false]
		PointSize	double	Set the size of the marker used to represent points.	[4]
		VertexSize	double	Set the size of the maker used to represent vertices.	[4]
		VertexColor	string	Color of vertex marker may be 'cyan', 'magenta', 'yellow', 'red', 'green', 'blue', 'white', or 'black'.	['color']
		PointColor	string	Color of point marker may be 'cyan', 'magenta', 'yellow', 'red', 'green', 'blue', 'white', or 'black'.	['color']
		ShowGrid	logical	Displays a grid along three plans normal to the global coordinate system, intersecting at its origin when set to true.	[true / false]
		ShowBounds	logical	Displays bounds calculated for each displayed surface, if set to true.	[true / false]
		ShowHoles	logical	Displays any vertices bounding any holes found in a series of surfaces comprising a group.	[true / false]
		ShowEdges	logical	Causes all displayed polygons to be outlined by their three edges if set to true. Otherwise only polygon surface is rendered.	[true / false]
RawAngle	Numeric (N x 7) Array	N/A	[Nx7 double]	Encoder angles for each point in the point variable. One angles set per row.	[angle 1, angle 2 …] [angle 1, angle 2 …] etc.

Table 10.1 (Continued): Important Variables Stored to Matlab Workspace

Variable	Storage Type	Attributes	Data Type	Description	Data Format
Calibrate	struct array	Zero	[1x7 double]	Contains the offset to be added to all incoming angles in order to zero the joint.	[offset 1, offset 2, offset 3, … offset 7]
		Known	[Nx3 double]	The known position of a point corresponding to points n the MeasAngle and MeasPos variables	[x1, y1, z1] [x2, y2, z2] [etc…]
		MeasAngle	[Nx7 double]	The angles of each encoder measured when a point, of known location, was touched.	[angle 1, angle 2, anlge 3, … angle 7]
		MeasPos	[Nx3 double]	The measured position of a point of known location, corresponding to the "known" variable.	[x1, y1, z1] [x2, y2, z2] [etc…]
		WayPoint	[Nx3 double]	Contain the coordinates of waypoints, lock the base into a known coordinate system.	[x1, y1, z1] [x2, y2, z2] [etc…]
		TouchPoint	[Nx3 double]	Contains the coordinates of points that were measured by touching a waypoint after moving the base.	[x1, y1, z1] [x2, y2, z2] [etc…]
		BaseTransfrom	[4x4 double]	A transformation applied to account for the location of the base relative to some fixed point in the room	[a b c d] [e f g h] [l j k l] [m n o p]
		Geometry	[1x34 double]	Contains the geometry of the arm used by the kinematic equations.	[geo1, geo 2, geo 3, geo 4, … geo 34]

Table 10.1 (Continued): Important Variables Stored to Matlab Workspace

Section 11: Digitizer Arm Software Instructions

Getting Started:

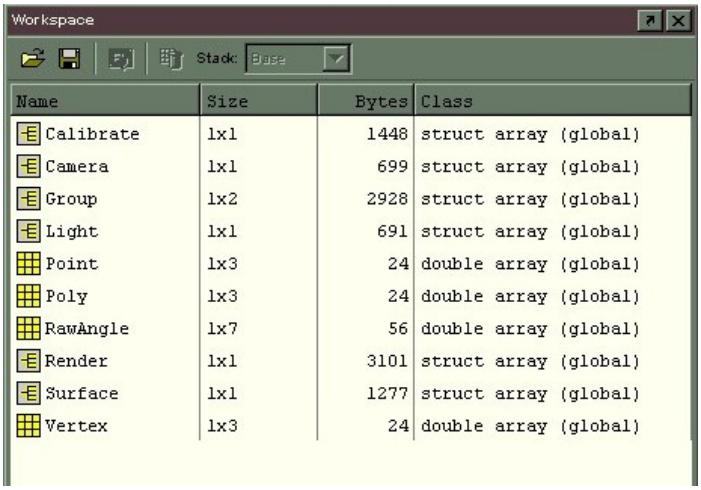
Prior to running the scanner arm software, Matlab version 6.5 or higher must be installed. Matlab is produced by Mathworks incorporated and must be purchased separately. The files required to running the scanner arm software must also be included on the Matlab file search path. The easiest way to do this is to copy the software to a convenient directory, and use the Matlab menu bar to select:

file > set path > add with subfolders > directory containing all scanner arm software files

If data collection is intended, make sure the serial cable from the arm's DSP unit is attached to COM 1 port on the PC or laptop. Initialize the software by typing:

> InitializeWorkspace

This will set up all required workspace variables. A newly initialized workspace is shown in Figure 11.1



WARNING! Initializing the workspace will also delete any data currently existing in the workspace.

Figure 11.1: Initialized Matlab workspace, required for operation of scanner arm software

Setting Up a Data Structure:

The next step is to set up the data structure in which subsequent data will be stored: points, vertices, polygons, etc. A GUI may be called to assist in this action by typing:

> EditObject

The GUI will appear is shown in Figure 11.2.

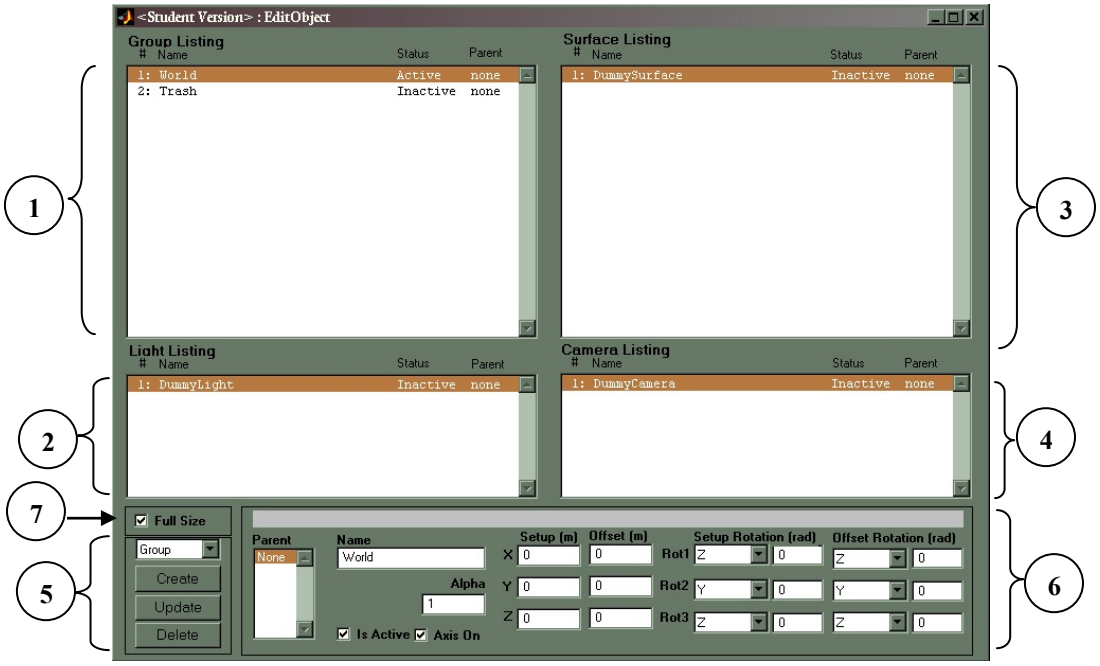


Figure 11.2: GUI for use in building and editing data structure. Each of seven grouping of controls are label with a number.

1. **Group list / selection menu**: This menu displays all existing groups names, group status, and group parent. Clicking on items in this menu selects them for editing, but only one item may be selected at a time.
2. **Light list / selection menu**: Similar to menu 1 except it displays all lights in a scene.
3. **Surface list / selection menu**: Similar to menu 1 except it displays all surfaces in a scene.

4. **Camera list / selection menu**: Similar to menu 1 except it displays all cameras in a scene.
5. **Action menus**: This popup menu controls the type of object to be view / edited. The selections are: group, surface, light, and camera. The selected object type can be selected from one of the selection menus (1 – 4). The "create" button will create a new object of the selected type. The "update" button will apply any changes made to the object using the edit menus, and the "delete" button will delete an object.

> Note: internally the pointers to deleted objects are simply overwritten and new pointers are made to the trash group. The data itself are not destroyed.

6. **Object information display / edit menu**: This menu changes depending on whether a group, surface, light, or camera is being edited.
7. **GUI size control**: Controls whether the GUI is shown full sized on minimized like in Figure 11.3.

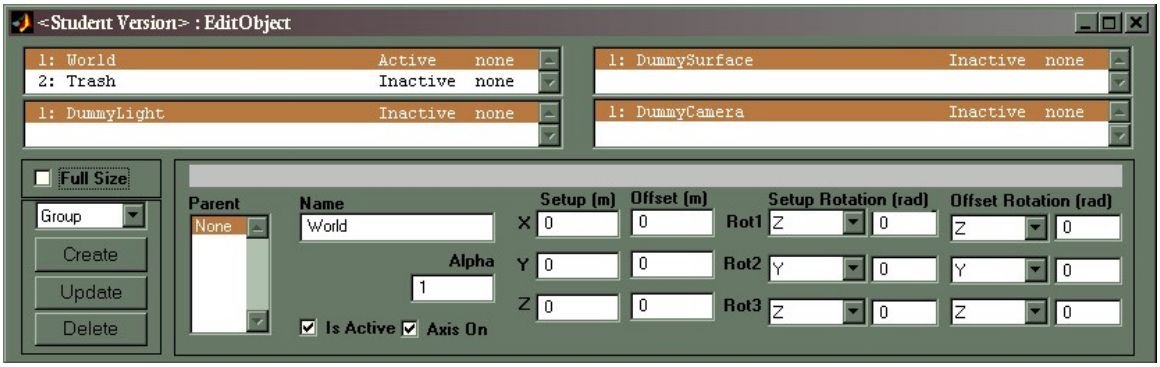


Figure 11.3: Minimized GUI for use in building and editing the data structure.

Within the GUI, any number of data objects (groups, camera, lights, and surfaces) may be created by selecting the desired object type in the popup menu and clicking the "create" button. Also, by changing options in the edit menus (discussed below) a user may configure an object.

Ultimately, the goal is to create a fully described scene. However, a scene may be edited at any time and it is not necessary to fully describe every object prior to collecting data. An example of the data structure of a partially completed scene, one including a van, is shown in Figure 11.4.

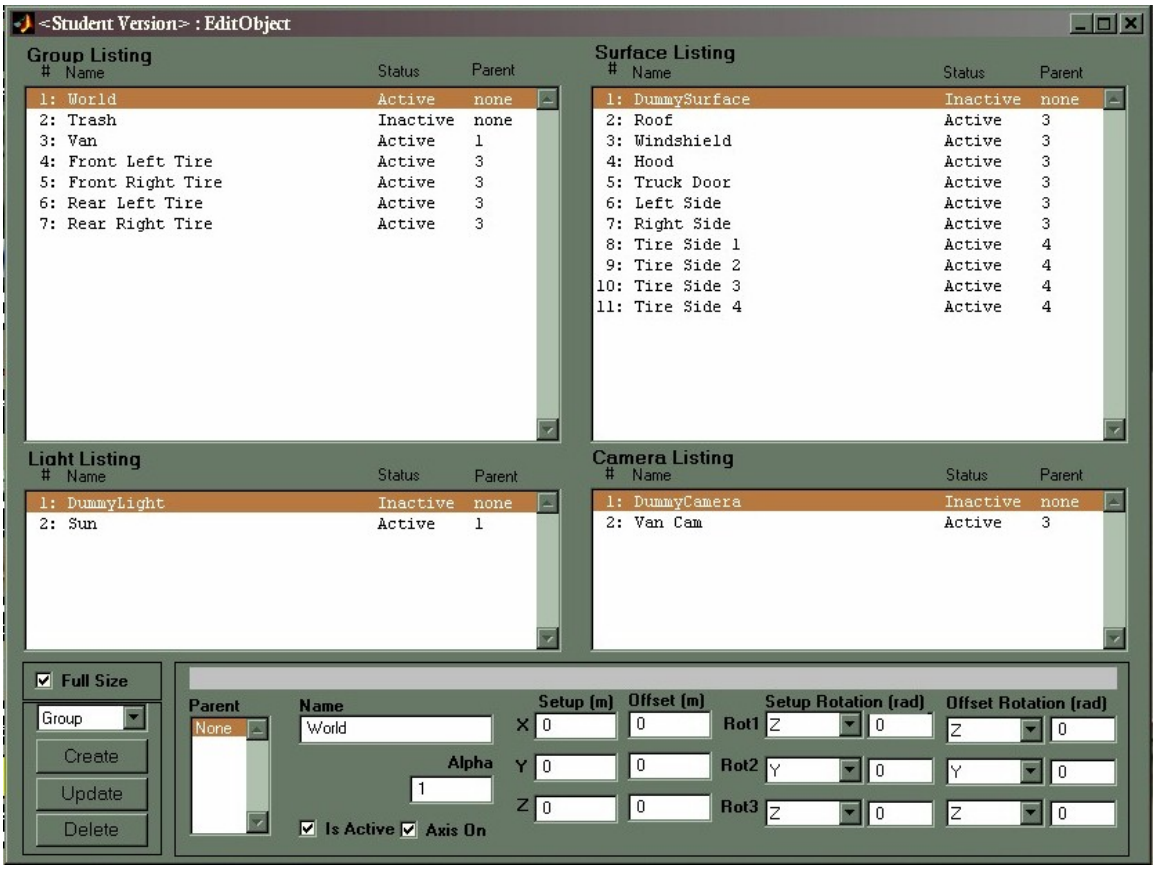


Figure 11.4: An example Scene Data Structure

In order to create a scene the edit menu must be employed (Item 6 in Figure 11.2). The layout of this menu and available options depend on which object type selected. The object to be affected depends on which item in the corresponding listbox is selected (Items 1 – 4 in Figure 11.2).

Group Edit Menu:

If a group object is selected in the popup menu, the edit options shown in Figure 11.5 will become available.

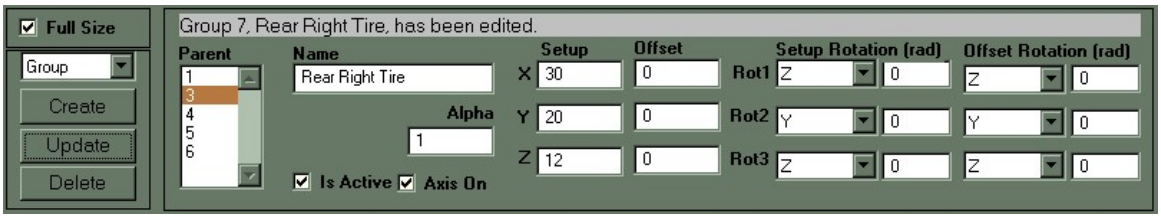


Figure 11.5: Group object display and edit menu.

Within the group edit menu grouping, there are user controls and a gray message window at the top of the grouping. Any changes made to the controls will only take effect if and when the "Update" button is pressed. The function of each control is as follows.

- **Parent Selection Listbox**: This listbox displays the numbers of the groups any of which can be selected as a parent for the currently edited group. A parent group applies its properties to all of its children. The number selected by default is the number of the object's current parent.
- **Name Text Box**: This is the name of the group. Any name fitting within the box may be entered.
- **Alpha Text Box**: The value in alpha box controls the level of transparency of all surfaces contained in the group. If this value is set to 1 all surfaces in the group will be completely opaque, and if this value is set to 0 all surfaces in the groups will be transparent. Values between 0 and 1 will result in semi-transparency.

- **Is Active Checkbox**: If this checkbox is checked, the edited group is active. Only active groups will be used when the scene is rendered. An inactive group, and all its children, will appear to not exist in a scene that is rendered.
- **Axis On Checkbox**: If this checkboxes is checked a visible xyz axis will represent the location and orientation of the group within a scene when it is rendered.
- **Setup X, Y, and Z Edit Boxes**: The xyz values input into these three boxes will cause the edited group to translate, by the input amount, within the parent group coordinate system. This translation is used to assemble a static scene from separate scanned objects allowing, for example, a tractor measured at one location to be attached to a trailer measured at another.
- **Offset X, Y, and Z Edit Boxes**: The xyz values input into these boxes apply a translation within the parent group, after the setup translation. This translation is intended for used in animation and should not be used to setup a static scene.
- **Setup Rotation Rot1, Rot2, and Rot3 Popup Menus**: The objects in each group are capable of completing three body-fixed rotations about the group's own axes for scene setup purposes. Rot1 is performed first, and is a body-fixed rotation about the axis specified in the Rot1 popup menu. Similarly, Rot2 and Rot3 are the second and third body fixed setup rotations applied to all objects in the group, and the Rot2 and Rot3 popup menus specify the axes of rotation. By default Eulerian rotations are specified.
- **Setup Rotation Rot1, Rot2, and Rot3 Edit Boxes**: Each of these boxes specify the magnitude, in radians, of the body fixed rotations specified in the Setup Rotation popup menus.

- **Offset Rotation Rot1, Rot2, and Rot3 Popup Menus**: Similar to the setup rotation popup menu except the specified rotations are intended for animation purposes and not scene setup. These rotations are applied *after* the scene setup rotations. Uses of these rotations include animating the pitch and yaw of a vehicle or the rotation of a vehicle's tires.
- **Offset Rotation Rot1, Rot2, and Rot3 Edit Boxes**: These three boxes specify the magnitude, in radians, of the boy-fixed rotation specified in the Offset Rotation popup menus.

Surface Edit Menu:

If a surface object is selected in the popup menu, the edit options shown in Figure 11.6 will become available.

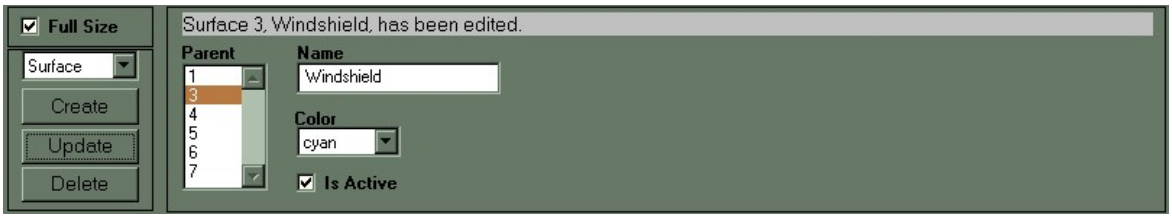


Figure 11.6: Surface object display and edit menu.

Within the surface edit menu grouping there are four user controls as well as a gray message window at the top of the grouping. Any changes made to the controls will only take effect if and when the "Update" button is pressed. The function of each control is as follows.

- **Parent Selection Listbox**: This listbox displays the numbers of the groups, which can be selected as a parent for the edited surface. The number selected by default is the object's current parent.
- **Name Text Box**: This is the name of the surface. Any name that can fit within the box may be entered.
- **Color Popup Menu**: This menu specifies the color of the rendered surface. Currently, valid colors include: yellow, magenta, cyan, red, green, blue, white, and black. However, future additions to the code may allow additional colors.
- **Is Active Checkbox**: If this checkbox is checked, the edited surface is active. An active surface will be shown when the scene is rendered. An inactive surface will not be drawn in a scene.

Light Edit Menu:

If a light object is selected in the popup menu, the edit options shown in Figure 11.7 will become available.

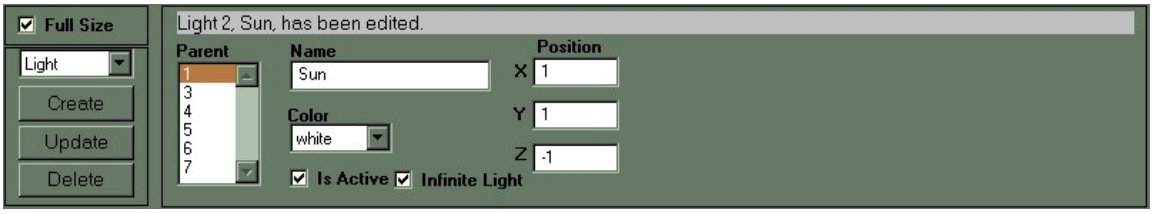


Figure 11.7: Light object display and edit menu.

Within the light edit menu grouping there are 8 user controls as well as a gray message window at the top of the grouping. Any changes made to the controls will only take effect if and when the "Update" button is pressed. The function of each control is as follows:

- **Parent Selection Listbox**: This listbox displays the numbers of the groups which can be selected as a parent for the edited light. The object's current parent is selected when the edit menus are first displayed.
- **Name Text Box**: This is the name of the light. Any name that can fit within the box may be entered.
- **Color Popup Menu**: This menu specified the color light emitted. Currently valid colors are: yellow, magenta, cyan, red, green, blue, white, and black. However, future additions to the code may allow additional colors.
- **Is Active Checkbox**: If this checkbox is checked, the edited light is active. An active light will cast illumination within a rendered scene. An inactive light will not be rendered in a scene.
- **Infinite Light Checkbox**: If this checkbox is checked light will be directed in the direction specified by the xyz vector in the position edit boxes, much like light from a distant light source such as the sun. Otherwise the light will appear as a point source, like a light bulb, at the location specified in the position edit boxes.
- **Position Edit Boxes**: The values in these boxes alternatively specify the direction of the light or its location within its parent group's coordinate system.

Camera Edit Menu:

If a camera object is selected in the popup menu, the edit options shown in Figure 11.8 will become available.

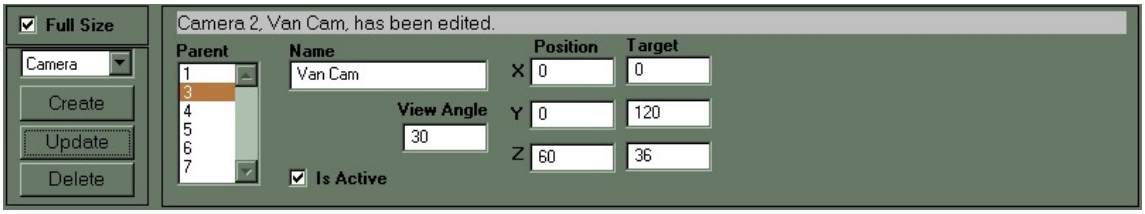


Figure 11.8: Camera object display and edit menu.

Within the group edit menu grouping there are 10 user controls as well as a gray message window at the top of the grouping. Any changes made to the controls will only take effect if and when the "Update" button is pressed. The function of each control is as follows:

- **Parent Selection Listbox**: This listbox displays the numbers of the groups, which can be selected as a parent for the edited camera. The object's current parent is selected when the edit menus are first displayed.
- **Name Text Box**: This is the name of the camera. Any name that can fit within the box may be entered.
- **View Angle Edit Box**: The value in this box, specified in degrees, determines the field of vision of a camera, and can be used like the zoom feature on a camera. Valid angles are between, but not including, 0 degrees and 180 degrees.
- **Is Active Checkbox**: If this checkbox is checked, the edited camera is active. Only one camera may be active at a time. The active camera will determine the view of the scene rendered.
- **Position Edit Boxes**: The values in these boxes specify the location of a camera within its parent group's coordinate system.

- **Target Edit Boxes**: The values in these boxes specify the location of the point in space the camera is looking at within its parent group's coordinate system.

Capturing Data:

Once the data structures of a scene have been tentatively assembled data may be captured. In order to active the data capture GUI shown in Figure 11.9, type:

> CaptureData

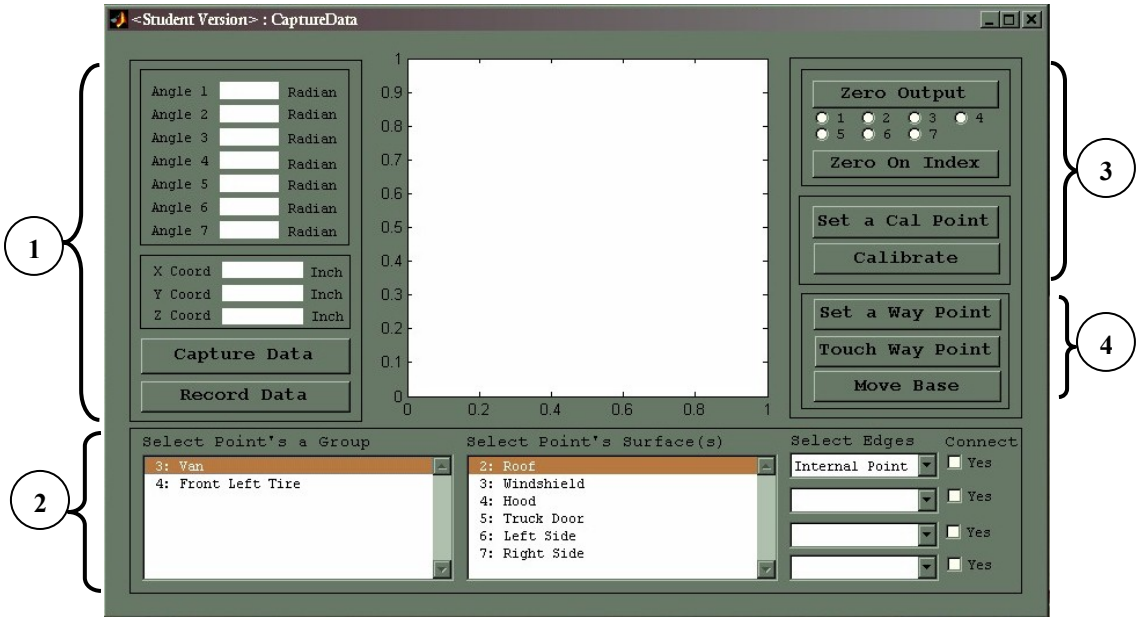


Figure 11.9: The GUI used to control the measurement arm during data capture

In the group capture GUI there are 4 groups of controls. The control groups are the:

1) Data input controls / display (Figure 11.10)
2) Data organization controls (Figure 11.11)
3) Calibration controls (Figure 11.13)
4) Base position controls (Figure 11.14)

Data Input Controls / Display:

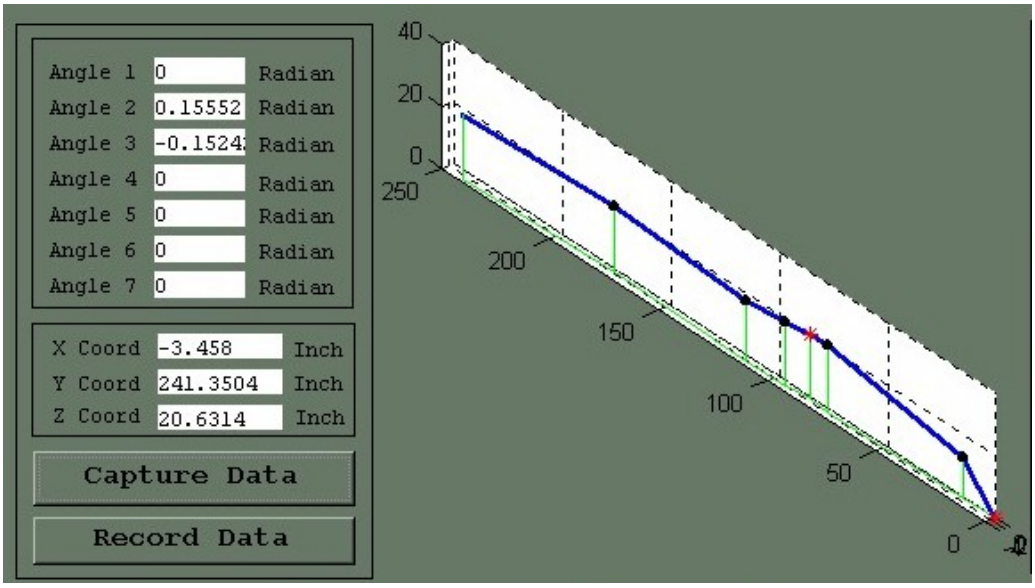


Figure 11.10: Data Input Controls and Information Display.

The data input controls / display provides all relevant information about the arm's current configuration and the position of its tip. The upper left text box grouping shows the angle of each joint, in radians. By the notation of the software, angle 1 is the angle of the base joint and angle 7 is the angle of the joint nearest the tip of the arm. The xyz coordinates of the tip of the arm are given in inches.

To the right of the raw numerical output from the arm is a scaled graphical representation of the arm's current configuration. In this display the arm's links are shown in blue. Joints whose axis of rotation ordinarily coincides with the normal of the floor are shown as red asterisks, and joints whose axis of rotation ordinarily aligned with the plane of the floor are shown as black dots. Green lines project from the representation of the arm to a datum plane, making the graphic easier to interpret.

The two buttons in the GUI control data input and storage. When the Capture Data button is pressed, the arm will query the position of each encoder ten times a

second, and display this information in the outputs just described. At other times the most recently captured data will be displayed instead. The record data button is pressed every time a point is to be stored. Immediacy after pressing the button the point is written saved in the Matlab data structure. Selections made via the data organization control determine where the data will be stored.

Data Organization Controls:

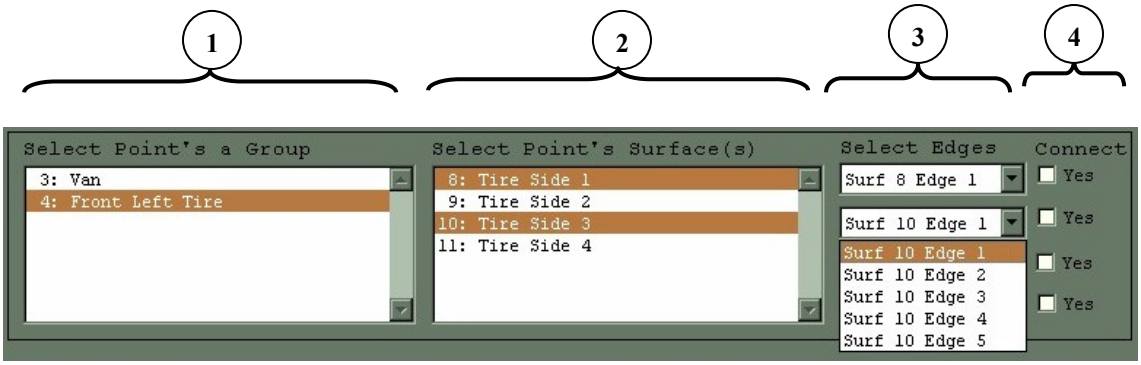


Figure 11.11: Data Organization Control Menus and Options.

The data organization controls determine where to store captured points. Possible location are queried from the Data structures, created with the EditObjects command.

In this GUI, the left list box (1) displays the names and identification numbers of each group that contains a surface. The group in which a point is to be stored should be selected.

In the center list box (2), all surfaces within the selected group are listed. All surfaces in which the point is to be stored should be highlighted. That is, if a point is to be captured which falls within the bounds of one surface, only that surface would be selected in this list box. However, if a point to be captured boarders multiple surfaces, each of these surfaces should be selected. This in turn designate the point as a surface

edge. Multiple surfaces may be selected by holding down the control key and up to four surfaces may be selected.

The "selected edges" group of popup menus (3) allows the user to enter information about points that double as the edges of surfaces. Any surface may have use up to five distinct edges, as shown in Figure 11.12. The selected edges popup menus allow the user to specify which edge of each surface a point will be stored to.

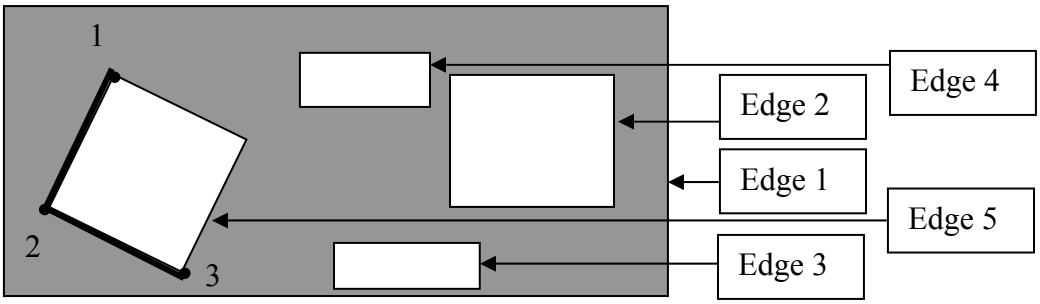


Figure 11.12: Example of Surface Edges

The "connect points" checkbox grouping (4) allows the user to specify whether = sequentially recorded points should be connected together in a chain to from at least part of an edge. For instance, say point 1 on Figure 11.13 is captured and stored to surface 1, and then the connect checkbox for surface1 is checked. If points 2 and 3 were recorded with the checkbox still on, then points 1, 2, and 3 would be connected. These connected points would form part of the edge of the surface

Note: when the user does not explicitly state that edge points are connected, the computer will assume that each point is connected to the point nearest to it.

Calibration Controls:

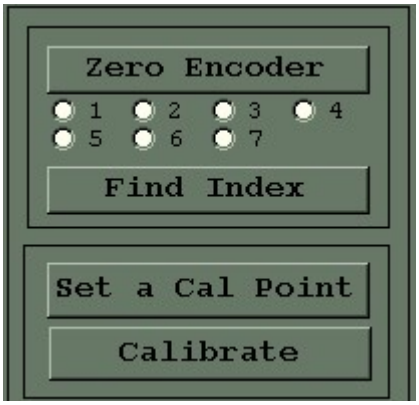


Figure 11.13: Calibration Controls

The calibration controls have three functions:

1. Zero the output of the encoders at a desired position.
2. Two find the index position, which cases the DSP to zero its output.
3. Perform a calibration based on the measurement of known points with the arm.

In order to zero the output of an encoder, simply select the radio button corresponding to the encoder joint and press "Zero Encoder" while the arm is capturing data. A message box will ask the user to confirm that the encoder should actually be zeroed. The act of zeroing will be accomplished by storing an offset value in the "Calibrate.Zero" variable in the Matlab workspace. This variable can be saved or loaded for later sessions.

The "Find Index" button commands the DSP, reading the encoders to zero encoder output sent to the PC whenever an encoder index pulse is received. This allows the arm remember its calibration even after power is removed.

The "Set a Cal Point" button and "Calibrate" buttons are for use in a calibration algorithm in which points of known location are measured. The feature is still under

development. The "Set Cal Point" button will, when data is being captured, store the angles or each joint and prompt the user for the actual coordinates, measured independently, of the captured point. These values will be stored in the "Calibrate. Measured" and the "Calibrate.Known" variables respectively. The "Calibrate" button will activate an algorithm that attempts to eliminate bias error from the measurements.

Base Position Controls:

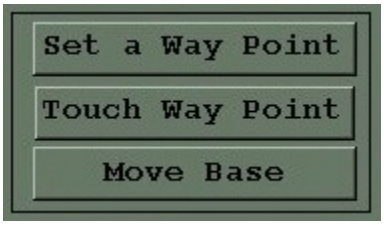


Figure 11.14: Base Translation Controls.

Due to the limited reach of the measurement arm, it may be desirable to scan part of a large object, move the arm's base, and then scan the rest of the object. This can be accomplished by measuring at least four "waypoints" with the arm prior to moving the base. The waypoints are recorded by pressing the "Set a Way Point" button while data is being captured.

After the arm has been moved, the same four waypoints can be measured by pressing the "touch waypoint" button while data is being captured.

Pressing the "move base" button will compute a 4 x 4 transformation matrix required for coinciding waypoints to be in the same physical location. This transformation, stored in the Calibrate.BaseTransform variable, will be applied to all subsequent measurements. However, this feature is still under development and has not been tested.

Meshing a Scene:

After points have been captured, they may be viewed using the "EditScene" GUI. Meshing is the process by which selected captured and record data points are converted into vertices and polygons. This creates a skin or shell representing the physical boundary of a scanned object. Thus, in order for a scanned object to appear as a solid and not a point cloud it must first be meshed. In order to mesh a group type:

> MeshGroup("Group Number")

Where "Group Number" is the identification number of the group to be meshed, as listed in the EditScene GUI.

The program will then list, on the command line, all surfaces in the group. It's status, if a surface has already been meshed, and there are too few points on the surface for it to be meshed (a surface must contain at least three points in order to be meshed).

Based on the presented information, the user will then be asked if meshing should continue. If the meshing process is continued, a copy of the Matlab workspace prior to meshing will be stored to the file "PreGroupMesh.Temp.mat" in the Matlab work directory, providing a backup in case something goes wrong.

The user would then be asked if bounds should be established for the surface. That is, should the edge information input via the CaptureData GUI be used. If this information is not used, the surface may come out oddly shaped and adjacent surfaces may not be connected. If this information is used, then the bounds of each surface will be displayed graphically.

If bounds are to be established, the user will be prompted for the minimum distance between edge vertices on each surface. That is, if a surface's edge is defined with four points, each 10 units apart, and a minimum distance between edge vertices of 1 unit is specified, the software will create 8 new vertices, equally spaced, between each point. This is desirable in order to maintain to avoid a large number polygons, from a large number of internal points, from terminating at a relatively small number of edge points, which can distort the geometry of the meshed object.

Next, the program prompts the user on whether each surface included in the group should be meshed. Then, as each specified surface is meshed, the new mesh will be displayed graphically.

After the meshing process, the software will prompt the user to check for holes in the mesh. The mesh contained in a group is said to have a hole in it if it does not fully enclose a volume. For instance, a mesh of a sphere that appeared as Figure 11.15, when rendered, would be said to have a hole in it.

Figure 11.15: Mesh of a sphere with a hole in it.

Of course not all groups contain a fully enclosed volume to start with, for instance, a group containing only a "road" surface. Also, even in situations where a fully enclosed object does have a hole in it, this will not prevent the object from being

displayed. Rather, a hole in an object's mesh will only prevent CAD operations such as calculating the enclosed volume of the object.

The last user prompt will ask if the mesh should be retained. If the user responds that the mesh should NOT be retained (perhaps because something is wrong with it) the Matlab workspace will be restored to its state prior to when meshing began via the file saved at the beginning of the meshing operation.

To illustrate the meshing process, consider the group of four surfaces, artificially generated, shown in Figure 11.16. It this figure, each vertex (intersection of lines) will serve as a simulated data point captured by the arm. For this group, the test inputs and outputs from the program would be as follows:

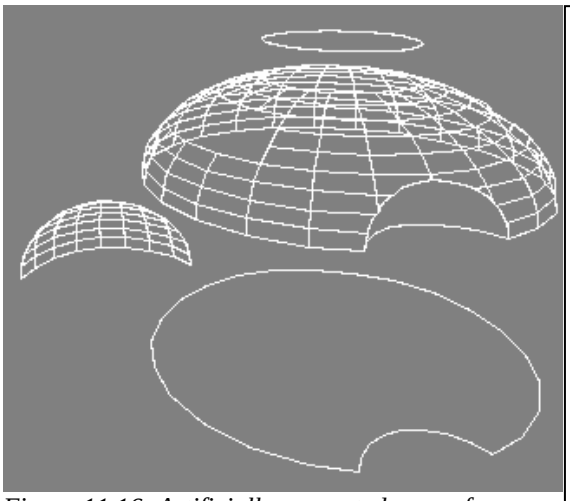


Figure 11.16: Artificially generated group for meshing

```
4 meshable surface(s) were found.
4 surface(s) raise no warnings
    Surface 2, Front
    Surface 3, Top
    Surface 4, Bottom
    Surface 5, Main
Continue with meshing? (1=yes, 2=no): 1

Saving workspace to temp file "PreGroupMesh.Temp.mat"

Warning: Surface bounds define how surfaces
are joined.  Restablishing bounds of
existing meshes can cause holes in mesh.
Remesh all surfaces in group to avoid this.

Est. surface bounds? (1=yes, 2=no): 1
Creating surface bounds from edge points.

Specify min dist between bounding verticies
of listed surface. Specify large distance to
avoid bound refinement

Surface 2 Front : 10
Surface 3 Top : 10
Surface 4 Bottom : 10
Surface 5 Main : 10

Boundaries refined
Press return to continue.

Mesh surface 2, Front (1=yes, 2=no): 1
Mesh surface 3, Top (1=yes, 2=no): 1
Mesh surface 4, Bottom (1=yes, 2=no): 1
Mesh surface 5, Main (1=yes, 2=no): 1

Delaunay tessellation based meshing complete.

Check for holes in mesh? (1=yes, 2=no): 1

Hole(s) in mesh found.
Polygon edges bounding hole(s) written to group`s HoleEdgeList.

Meshing Complete. (1=Keep Mesh, 2=Discard): 1
Done
```

During the meshing process, the bounds of the surfaces would be graphically displayed, as shown in Figure 11.17. After the meshing process, the meshed surface would be displayed as shown in Figure 11.18. Any holes in the mesh would be highlighted in red, as shown in Figure 11.19.

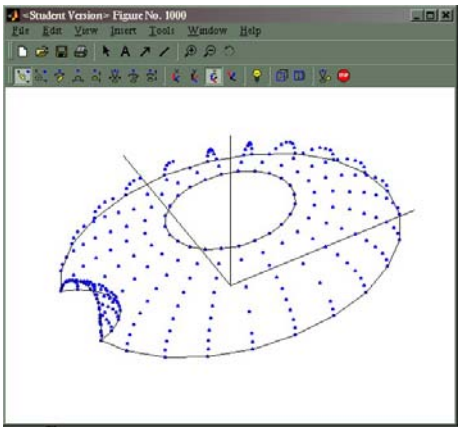


Figure 11.17: Bounds of an example group of surface being meshed

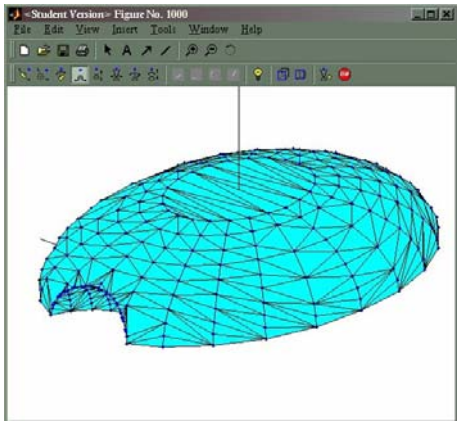


Figure 11.18: Mesh generated for an example group of surfaces

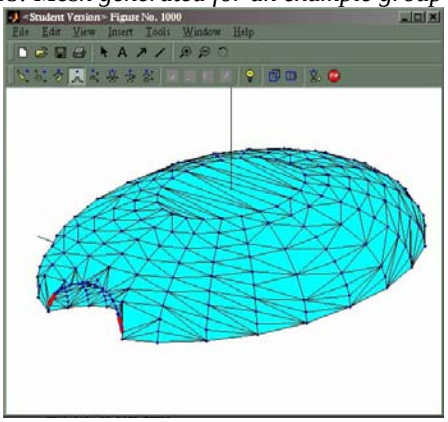


Figure 11.19: Holes (shown in red) in an example mesh

Displaying a Scene:

Once data has been captured by the CaptureData GUI and / or a group of surfaces have been meshed with the MeshGroup command, the captured / generated data my be displayed in a variety of ways via the EditScene GUI shown in Figure 11.20. To activate this GUI, type:

> EditScene

The function of each control in the GUI is as follows:

- **Recreate Scene Button**: Clicking on this button causes the scene to be rendered in a figure window. The location of the camera in the scene will dictate what is being looked at.
- **Update Scene Button**: Clicking on this button cause the scene to be updated. That is, any change made to the scene, including those made with this GUI, will occur but the view will not reset to the view specified by the camera if the 3D tools in the Figures menu were used to change the view. See the Matlab documentation on how to use the 3D controls in a Figure.
- **Project Popup Menu**: This lets the user specify whether the scene should be drawn using orthographic or perspective projection.

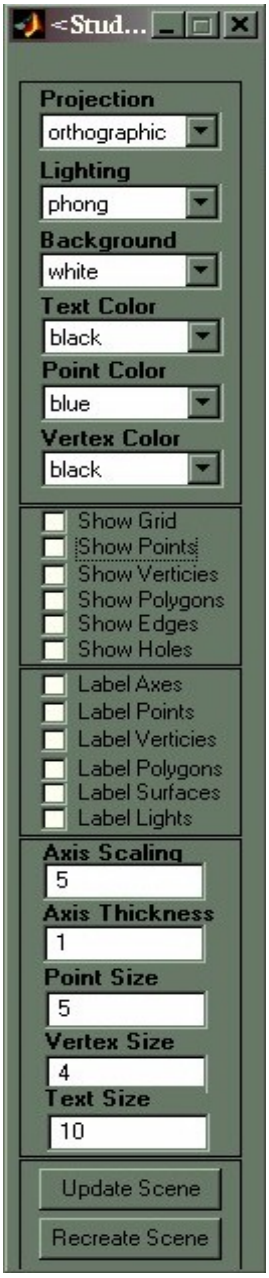


Figure 11.20: Edit Scene GUI

- **Light Popup Menu**: The menu specifies the type of lighting / shading used when rendering the scene. The options are "none', "flat", "gourand", and "phong". See the Matlab documentation for definitions of the various lighting schemes.
- **Background Popup Menu**: This menu allow the user to specify the background color of a scene.
- **Test Color Popup Menu**: This menu controls the color of any text draw on top of the scene.
- **Point Color Popup Menu**: This menu controls the color of any points, captured via the CaptureData GUI, present in the scene.
- **Vertex Color Popup Menu**: Controls the color of any vertices shown.
- **Show Grid Checkbox**: When checked draws a grid along the world xyz axis.
- **Show Vertices Checkbox**: when checked, draws vertices in the scene as dots.
- **Show Polygons Checkbox**: When checked draws any existing polygons in the scene.
- **Show Edges Checkbox**: When checked, draws the edges of polygons as a black line.
- **Show Holes Checkbox**: When checked, indicates any holes found in the mesh during the meshing operation. Holes will be highlighted in red.
- **Label Axis Checkbox**: When checked, labels each axis with x y z designator and the name of the group it belongs to.
- **Label Points Checkbox**: When checked, displays the index of each point, in the Point variable, next to each point.
- **Label Vertices Checkbox**: When checked, displays the index of each vertex, in the Point variable, next to each vertex.

- **Label Polygons Checkbox**: When checked, Displays the index of each polygon, in the Poly variable, in the center of said polygon.
- **Label Surface Checkbox**: When checked, Displays the number of each surface the center of each surface.
- **Label Lights Checkbox**: Displays the number of each light in the scene.
- **Axis Scaling Editbox**: Scales each exist display in a scene by a factor given in this box. Values less than 1 but greater than 0 shrink the axes while values greater than 1 increase the size of the axis by multiples of the input number.
- **Axis Thickness Editbox**: Controls the thickness of the lines used to represent axis. Values are in points.
- **Point Size Editbox**: Controls the size of dots used to represent points. Values are in points.
- **Vertex Size Editbox**: Controls the size of dots used to represent vertices. Values are in points.
- **Text Size Editbox**: Controls the size of text display on top of the rendered scene. Values are in points.

Editing Vertices:

Initially, when a surface is meshed, every vertex in the mesh has the same coordinates as the point that the vertex was generated from. These coordinate are relative to the world group coordinate system. However, this is not be desirable because vertex

locations must be given within the coordinate system of a parent group, other than the world group, in order for animation of surfaces made from these vertices to be possible.

The shift vertices GUI was created to address this problem. Using this GUI the coordinate of each vertex may be shifted within a selected coordinate system. That is, rather than applying a transformation to a coordinate system in which the vertices exist, the transformation is directly and permanently applied to the vertices themselves. Another way to look at it is that local axis can be move into position relative to an existing mesh.

The "Shift Vertices" GUI is shown in Figure 11.21. In this GUI the Edit Group popup control which frame the vertices are being shifted within. All other controls are similar to those found in the EditObject GUI except that every time a change is made the scene will automatically be redrawn and displayed.

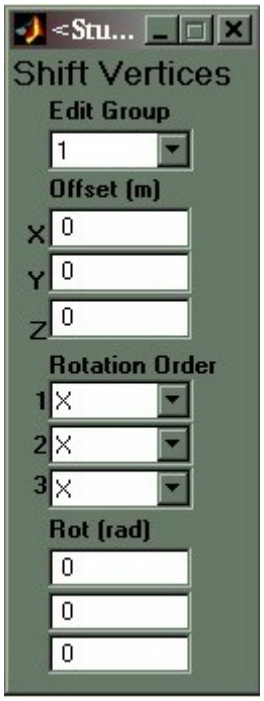


Figure 11.21: Edit Vertices GUI

Animating a Scene:

A scene may be animated once all objects of interest in a scene have been meshed, vertices have been shift into convenient positions relative to group axes, and setup translations and rotations have been input. The animation is controlled by a user generated to script stored in the Matlab workspace. In order to prepare a script for type, on the command line:

```
> InitializeScript
```

This will create two four column numeric arrays for each group and camera object in the scene. The array will be stored to variable in the format:

```
SCRIPT_CAMERA_<CameraNumber>_POS
SCRIPT_CAMERA_<CameraNumber>_TARGET
SCRIPT_GROUP_<GroupNumber>_ROT
SCRIPT_GROUP_<GroupNumber>_TRANS
```

These arrays control the values of each camera's position and target as well as each group's offset rotation and offset translation, respectively. Data should be entered into each array in the format:

```
[ Time 1 in seconds, value 1, value 2, value 3;
  Time 2 in seconds, value 1, value 2, value 3;
  ...
  Time N in seconds, value 1, value 2, value 3 ]
```

Where, at a specified time in seconds, value 1, value 2, and value 3 specify camera position, camera target, group offset rotation, or group offset translation. For instance the script:

```
SCRIPT_GROUP_3_TRANS  = [10          5        6        7]
```

would instruct group 3 to move, within it parent coordinate system, to x=5, y=6, and z=7 units at time = 10 seconds.

If multiple values are provided, each with their own time index, the software will linearly interpolate value for all animation frames that occur in-between. For instance the script:

```
SCRIPT_GROUP_3_TRANS  = [0      0     0     0;
                         10     3     4     0;
                         25     0     0     0]
```

would cause group 3 to move from coordinates x = 0, y=0, z = 0 to coordinates x = 3, y = 4, z = 0 at a rate of 2.5 units a seconds and to then return to its original position at 5 units per second. In animation terminology, frames during which the position is explicitly given are called key frames.

Note: The software will always linearly interpolate between key frames, but will NOT extrapolate prior to the first keyframe or after the last keyframe. Instead the scripted value at the first or last time step will be used, respectably.

Once the script has been completed, a Matlab movie file may be created by typing:

> "Any_Movie_Name_You_Like" = Animate

The "Animate" program with prompt the user for the time at which the animation should start, in seconds, and the time at which the animation should end, also in seconds. Lastly, the user will be prompted for the number of frames per second that should be animated.

Once the movie file has been created, it may be played back via the built Matlab function

> movie(movie_name, 1, frames per second)

Please note that the number of frames per second given the Animate and movie function must match or playback will appear either too fast or too slow.

Section 12: Arm Test and Measurements

Overview of Tests:

After construction and assembly of the digitizer arm, it was desirable to test the arm's accuracy and to test its ability to accurately measure a vehicle. To this end, the arm was first physically calibrated and its accuracy was evaluated by measuring points of known location. The arm was then used to measure the side of a car. This section details the calibration and testing of the digitizer arm.

Calibration:

Physical calibration of the digitizer arm consisted of two steps: first a Matlab program, (startCal included in Appendix E), instructed the arm's DSP to search for calibration pulses from the index channel of each encoder. Each joint was then moved until the index was found, indicated by corresponding LEDs on the DSP lighting up. At each index, the DSP zeroed the encoder at the position of the index. After all the index positions were found, calibration positions could be recorded by the device, and full calibration would not be necessary every time power to the DSP is removed. Rather, it would only be necessary to once again find the index positions.

Once all joint index positions were found, each joint was further calibrated by zeroing each joint at its most extended position. The zero position was found by pressing a machined calibration plate against the top of each joint, the side from which bolts protrude, when the joint was extended. The actually zeroing occurred within Matlab, which upon command within the CaptureData program, calculated the offset required to zero the joint at its current position and added that offset to all subsequent angles read in

from the calibrated joint. The "Calibrate" global structure, stored in the Matlab workspace by the digitizer arm software, was then saved so that the calibration could be retrieved by reloading the variable and finding each joint's index position.

In Lab Error Evaluation:

In order to evaluate the accuracy of the arm, the digitizer arm was set up within a lab, and the coordinates of twenty points within the lab were measured relative to the base of the arm. As shown in Figure 12.1, sixteen of these points were located on the floor of the lab and four points were elevated (the top of a filing cabinet). The x-y position of each point was found by noting that the floor tiles in the lab formed a grid with 12" increments, as measured with a metal ruler, and z coordinates were found with a meter stick marked with half-millimeter increments.

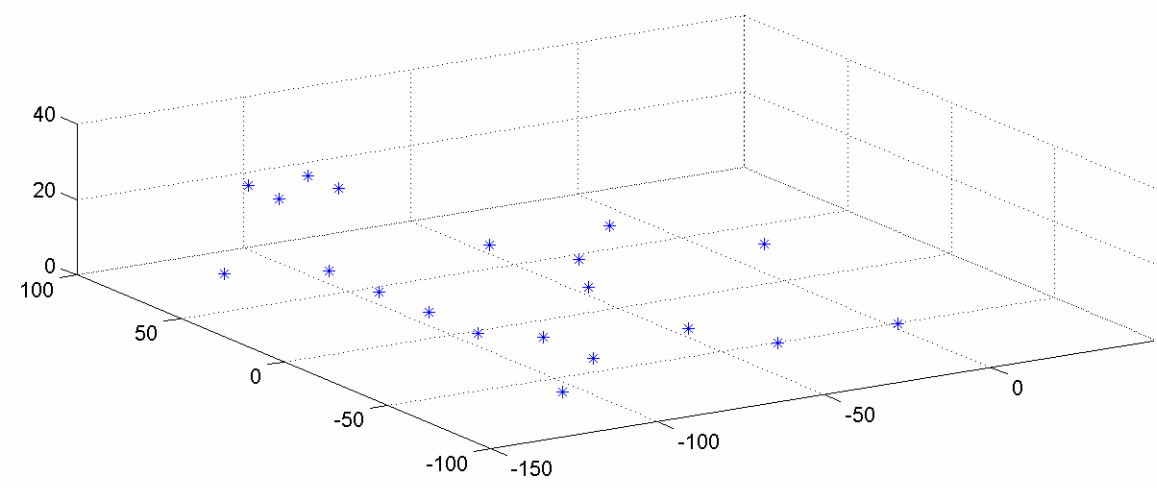


Figure 12.1: The coordinates of twenty test points, relative to the digitizer arm's base were measured using the floor tiles in the lab and a meter stick.

The physically calibrated digitizer arm was then used to measure the locations of all twenty points. A comparison of the two measurements, one using a meter stick and the other using the arm, is given in Table 12.1. Also, assuming the meter stick

measurements to be "true", the maximum measured errors and calculated RMS errors are given in Table 12.2.

Table 12.1: The positions of twenty points, as measured with a meter stick, given as "Known Positions", are shown in comparison to the locations of the same points measured by the digitizer arm. Taking the known position to be the true value, the error introduced by the arm was calculated for each coordinate along with the total displacement error.

Known Positions			Measured Positions			Error			
X (in.)	Y (in.)	Z (in.)	X (in.)	Y (in.)	Z (in.)	Delta X	Delta Y	Delta Z	Delta R
-105.32	-62.54	0.00	-106.65	-61.02	-0.20	-1.33	1.52	-0.20	2.03
-81.32	-38.54	0.00	-82.91	-37.50	-0.34	-1.59	1.04	-0.34	1.93
-81.32	-14.54	0.00	-82.71	-12.87	-0.16	-1.39	0.67	-0.16	1.55
-93.32	-2.54	0.00	-93.81	-1.25	-0.40	-0.49	1.29	-0.40	1.44
-93.32	21.46	0.00	-93.60	23.00	0.14	-0.28	1.54	0.14	1.57
-93.32	45.46	0.00	-94.55	46.52	1.27	-1.23	1.06	1.27	2.06
-93.32	69.46	0.00	-94.58	70.39	1.23	-1.26	0.93	1.23	1.99
-117.32	81.46	0.00	-118.21	81.48	1.00	-0.89	0.02	1.00	1.34
-45.32	69.46	0.00	-45.81	71.09	0.78	-0.49	1.63	0.78	1.87
-9.32	69.46	0.00	-9.04	71.07	0.88	0.28	1.61	0.88	1.86
14.68	33.46	0.00	14.46	33.41	-0.15	-0.22	-0.05	-0.15	0.27
2.68	-50.54	0.00	1.73	-50.34	-1.00	-0.95	0.20	-1.00	1.39
-33.32	-50.54	0.00	-34.20	-50.74	-0.52	-0.88	-0.20	-0.52	1.05
-45.32	-26.54	0.00	-45.79	-27.01	-0.20	-0.47	-0.47	-0.20	0.70
-45.32	21.46	0.00	-44.98	21.31	0.26	0.34	-0.15	0.26	0.45
-33.32	45.46	0.00	-33.34	45.24	-0.11	-0.02	-0.22	-0.11	0.24
-105.32	45.56	28.90	-104.78	46.19	29.65	0.54	0.63	0.75	1.12
-105.32	60.34	28.90	-105.39	60.78	29.73	-0.07	0.44	0.83	0.94
-123.32	60.34	28.90	-123.20	60.69	29.76	0.12	0.35	0.86	0.93
-123.32	45.46	28.90	-124.36	45.76	31.06	-1.04	0.30	2.16	2.42

Table 12.2: A summary of max and average error from Table 12.1 and RMS error is given.

	X (in)	Y (in)	Z (in)	R (in)
Max Error	1.59	1.63	2.16	2.42
Ave Error	0.69	0.71	0.66	1.36
RMS Error	0.84	0.9	0.83	1.49

As can be seem from this data, RMS error typically was less than an inch in any one direction and less than one and a half inches overall. Furthermore, error appeared to be slightly larger in the y-direction and have a positive bias in this direction. There also appeared to be a negative bias in the x-direction. These observations may be due to bias introduced during arm calibration or when the locations of the points were being measured by hand relative to the base of the arm.

Future work will concentrate on quantifying and eliminating such biases. Dr. Sommer III of the Penn State Mechanical Engineering Department has provided Matlab code that aids in calibrating segmented arms. This code is given in Appendix D as Programs D.9, D.10, and D.11. These programs allow bias error to be identified by regression. However, at the time of this write-up, additional work is necessary to test this code, and work is ongoing in this area.

The in lab test demonstrated that physically calibrated digitizer arm did introduce errors, but that these errors were not so large as to render collected data unusable. Even the largest measured error was still small relative to even a mid-sized car, being less than 2% of its length. The effect of such an error on a 3D model, intended for visualization purposes only, might not even be noticeable.

Field Data Capture:

Following in-lab testing of the arm, a field test was conducted in which the arm was used to measure the side of a car. These measurements were used to produce a 3D model of the car's side, which was visually compared to the actual vehicle. As such, the goal of the test was to evaluate operation of the device outside a lab, and to determine the quality of images produced with the arm.

The arm was carried to the parking lot, in one piece, by two people. The act of transporting the arm showed that it might be necessary to include the ability to disassemble the arm into two pieces in future versions of the device. The base of the arm was set about ten feet from the driver's side of the vehicle to be measured, as shown in Figure 12.2. This positioning allowed all points on the side of the vehicle to be reached

by the tip of the arm. Furthermore, about half of the hood, windshield, roof, and trunk of the car could also be reached.

Figure 12.2: During the test, the arm's base was positioned about ten feet from the side a car. This positioning allowed all points on the side of the car to be reached by the tip of the arm.

After positioning the arm, surfaces on the side of the car were identified and marked with masking tape. Data surfaces where then created in Matlab using the digitizer arm software. Initially, the car was broken up into many surfaces representing, on one side of the car: the front fender panel, hood, headlight, front bumper, front door panel, front glass, pillar between doors, rear door panel, rear body panel, rear tail light, rear bumper, trunk, roof, bottom, rear glass, front door window, and rear door window. However, the data capture interface proved very cumbersome to deal with for such a large number of surfaces. Therefore, it was decided to instead use a much smaller number of surfaces, specifically the:

left side body paneling
left front door window
left rear door window
bottom of the car
roof of the car
hood
front section of the car
trunk of the car
front windshield
rear windshield

Of these surfaces, sufficient points to create a mesh were only captured for: the left side body paneling, the left front door window, and the left rear door window.

After the surfaces were chosen, the boundaries of the left side body paneling, left front door window, and left rear door window were measured.

Overall, the arm and its interface performed well. However the current interface to input user data and to identify boundaries did prove inefficient and in need of improvement. It also was found that input errors were easy to make. This suggests the need for a more streamlined boundary input process in the future.

After boundaries were defined, the three surfaces whose boundaries were fully defined were filled in with additional interior points. Points were selected at locations that would help capture the shape of the side of the car. Using the current interface it proved easy to capture over a hundred points in about twenty minutes. The captured points are shown in Figure 12.3.

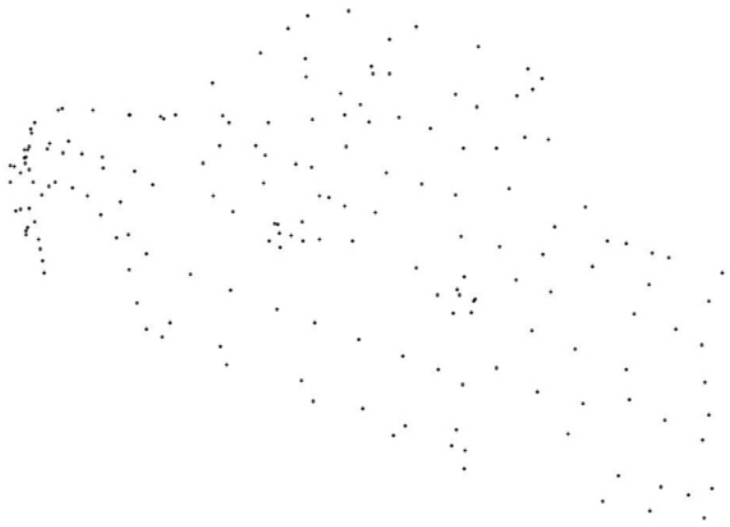


Figure 12.3: This point cloud of the driver side of a car captured with the digitizer arm.

Meshing Results:

After the data points for one side of a car were captured a mesh was created based upon these points via the digitizer arm meshing software. After some debugging of the associated software, the meshing process worked well, and no manual manipulation of the mesh was required to correct a poor mesh. For comparison, Figure 12.4 shows a photograph of the side of the car while Figure 12.5 shows the same side of the car represented as a mesh in Matlab.

Figure 12.4: A photograph is shown of the side of the car that was measured.

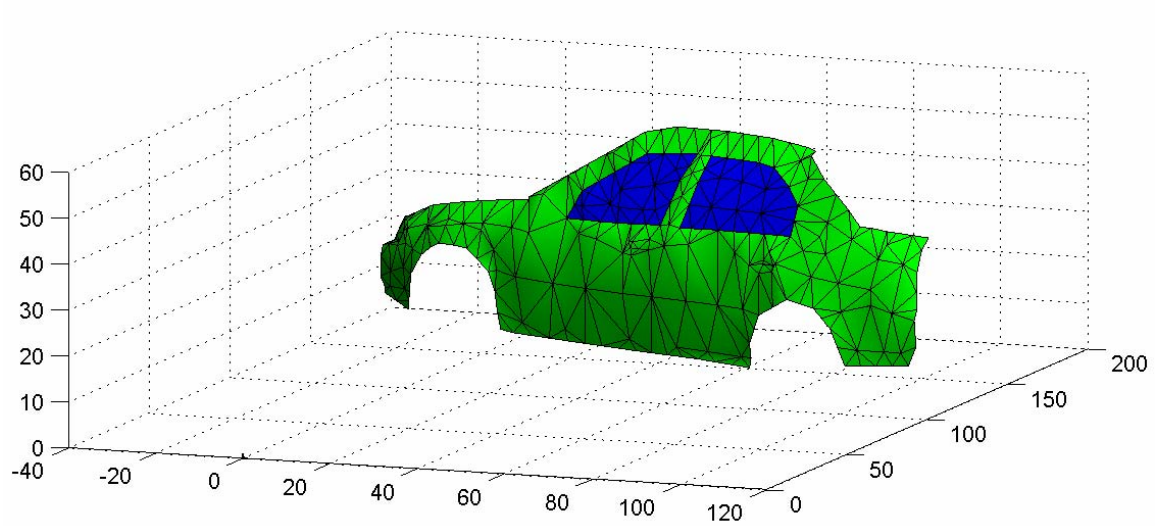


Figure 12.5: The mesh of the measured points, generated by the scanner arm software, is shown against a grid with units of inches. The figure shows that both the gross shape of the car, and smaller features, such as the flared section of metal around the wheel wells were captured.

Visual inspection of Figures 12.4 and 12.5 showed that the digitizer arm, and accompanying software was able to generate a visually acceptable 3D representation of the side of a car. Furthermore, error in measurement did not cause any major surface breakup. Figure 12.6 shows the rendered mesh using flat lighting in order to emphasize the effect of errors. This effect is sufficiently small that the use of phong lighting - which tends to have a smoothing effect during rendering - effectively eliminates much of the visual impact of error in measurement, as shown in Figure 12.7.

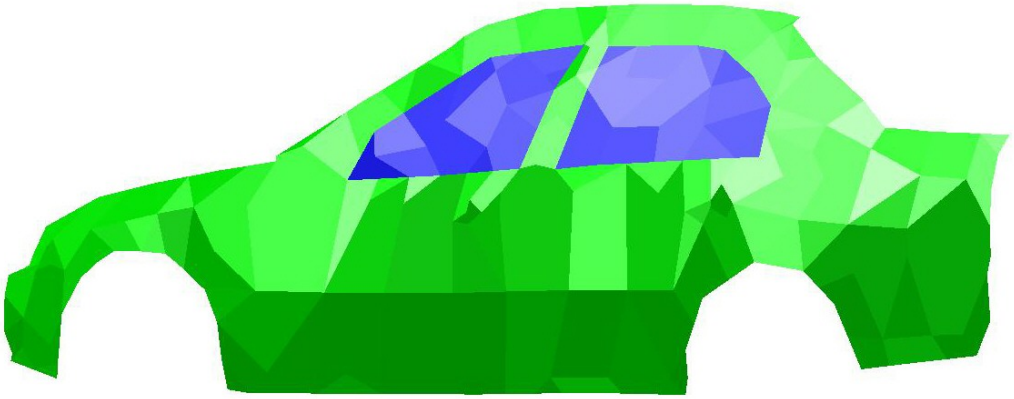


Figure 12.6: A 3D rendering of the side of a car is shown with flat lighting in order to emphasize the visual impact of error in measurement.

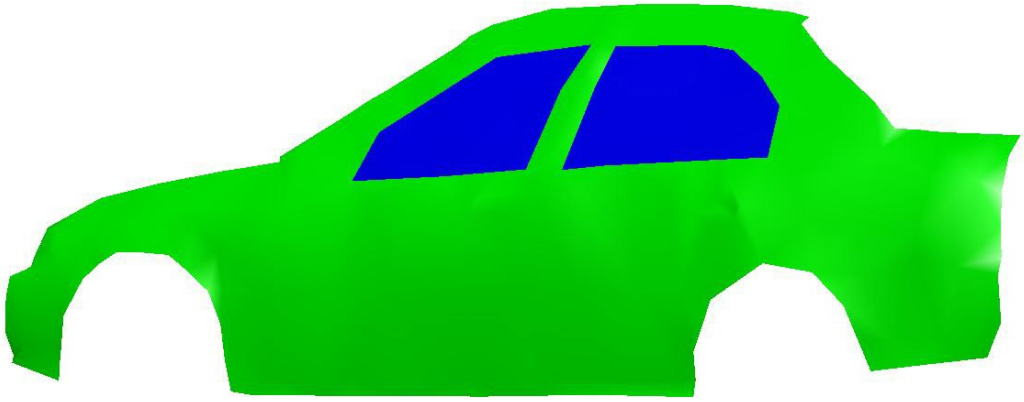


Figure 12.7: A 3D rendering of the side of a car is shown with phong lighting in order to demonstrate attenuation possible in the visual impact of error in measurement, using appropriate lighting.

Summary and Conclusions:

The intent of this project was to create a device capable of measuring vehicle features for visualization purposes, and in this regard, the digitizer arm has satisfied the design requirements. The snakelike-armature design used in this project was shown to function in a satisfactory manner. Specifically, the mechanical portions of the system have proven adequately functional and the electrical portions were reliable enough to function properly over a field test that lasted several hours. Also, the computer / DSP interface and accompanying Matlab software that controlled data input worked reasonably well, although there are still some software interface issues to address.

Testing showed that the seven-link armature behaves in a very controllable manner when handled by a user. The entire device was consistently positioned by solely handling the two links closest to its end, a feat easily accomplished by a single user. Furthermore, such handling does not involve large amounts of physical labor and is not very tiring.

The tip of the digitizer arm can touch most points in space within the physical reach of the arm, excluding points within about a one-foot radius of the center of the arm's base. The act of positing the tip in a desired location did posses a small learning curve; however, a couple hours of practice appear sufficient to enable a user to measure most points.

The mechanical concerns that arose during the wooden mockup test and preliminary error evaluation were clearly mitigated in the final design. The seven-joint configuration transverses well along level ground or slightly rough ground without causing any of the links to noticeably bend or twist, a significant early concern.

Additionally, the use of caster wheels under the links provided stable support for the bulk of the device. The base was very stable and, once set in place, could not be moved even when the end of the arm was pulled on when the intent of dragging the base along the ground. Lastly, vibration from moving on pavement did not cause any loosing problems, likely due to the use of lock nuts on all critical bolted joints.

Physical calibration using a calibration plate appears to have been surprisingly effective given its simplicity. Measured RMS displacement error after such calibration was 1.49 inches and the worst displacement error experienced was 2.42 inches. Implying that the effect of error on 3D models for visualization purposes will not be large given the scale of objects being measured. Furthermore, it appears likely that bias error is present in current measurements. Such error may eventually be accounted for and eliminated, thereby increasing the overall accuracy of the device.

The electrical equipment on board the arm proved reasonably reliable and rugged once it was enclosed in protective material. Specifically, the encoders housed in each joint did not show any problems despite their somewhat delicate construction, even when a joint was jarred from being dropped a short distance. Also, spiral wrap covering the wires helped prevent short circuits that caused early problems. Furthermore, the electrical systems showed no ill effect for being used outside for several hours during a hot day.

The user interface within Matlab provided adequate to capture a point cloud representation of the side of a car. The algorithms responsible for calculating the position of the arm's tip worked well. Also, monitoring of the data input process by a second individual sitting at a computer proved to be reasonably efficient, and a good way to

confirm that data was being captured correctly. However, the user interface did prove inefficient in defining boundaries around measured surfaces, an act that currently requires significant of user input. Modifications to the software are underway.

The software written to mesh the surfaces successfully meshed a point cloud. However, this meshing program currently requires significant user input attached to each data point, and eventually a commercial meshing package may replace the current hand-coded routines.

In any case, the digitizer arm and accompanying software is capable of creating a visually appealing model of one side of a vehicle. Measurement error is noticeable in 3D models created by the digitizer arm, but this visual impact is not very large. Furthermore, data processing can easily reduce the visual impact of measurement error. Therefore, once a full vehicle is measured it should be possible to fully manipulate it in a computer and create visually appealing scripted animations based on actual vehicle dynamics models.

Future Work:

Despite the progress that has been made at this point, future work remains. Specifically, the ability to move the base of the arm during measurement needs to be implemented in order to measure objects larger than the current reach of the arm. Movement of the base requires measuring the location of waypoints, i.e. points with a known location, on an object in order to back-calculate the location of the base at the new position. This will be the subject of future work, although a procedure for such a process has already been partially implemented.

The computer interface used to control the capture of data also needs to be streamlined. The current interface works, but is somewhat inefficient. Improvements such as better placement of GUI controls and more feedback to the user should improve data capture. A better way of capturing surface boundaries also needed to be implemented.

Future efforts to reduce error in the arm will likely focus on the removal of bias error. In particular, a means of removing bias by measuring points with known locations is planned. And, most of the code necessary for this functionality is already in place.

Lastly, a means of exporting 3D models into a common file format is planned. Such a feature will allow the models to be read by more powerful commercial 3D graphics packages and used to generate high quality images and animations. Likely file formats are DXF and/or VRML.

Despite the number of improvements left to be made to the arm, it is currently functioning well and its accuracy is satisfactory for visualization purposes. Furthermore, it has achieved these capabilities for a fraction of the cost of commercial equipment and software often used for measurement and reverse engineering with only a slight cost in accuracy. After the addition of software improvements that are planned in the near future, the digitizer arm will be in full use to creating 3D models of vehicles and roadways at the Pennsylvania Transportation Institute test track.

Section 13: References

Brennan Sean, Brennan Automation: Advancing Control Research and Education 24 Apr. 2004 <http://controlfreaks.mne.psu.edu>.

Corrsys Datron, Products | Sensors 2004 <http://www.corrsys-datron.com/prod_sensors.htm>.

Farin, G., Hoschek J, and M. –S. Kim. Handbook of Computer Aided Geometric Design. Elsevier Science B.V., 2002.

Faro Technologies Inc., Faro Scan Arm. 2004 <http://www.faro.com/Products/ScanArm.asp#>.

Faro Technologies Inc., Laser Tracker. 2004 <http://www.faro.com/Products/Laser_Tracker.asp#>

Ferguson, Stuart R. Practical Algorithms for 3D Computer Graphics. A K Peters, Ltd., 2001.

Ginsberg, Jerry H. Advanced Engineering Dynamics Second Edition. Cambridge University Press, 1998.

Hosaka, M. Modeling of Curves and Surface in CAD/CAM. Spring-Verlag Berlin Heidelberg, 1992.

Javad Positioning Systems, A GPS Tutorial: Basic of High Precision Global Positioning Systems. 1998 <http://www.javad.com/>

Kimura, Fumihiko, ed. Geometric Modeling: Theoretical and Computational Basis towards Advanced CAD Applications. International Federation for Information Processing, 2001.

Patrick, Marchand. Graphics and GUIs with MATLAB Second Edition. CRC Press LLC, 1999.

Remondino, Fabio. "From Point Cloud to Surface: The Modeling and Visualization Problem." International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XXXIV-5/W10.

Romer Cimcore. GridLOK System. 2004 <http://www.romer.com/lang/en/products_pdf/1005005644.pdf>

The Mathworks, Matlab Documentation CD. CD-ROM. The Mathworks, Inc., 2002.

The Mathworks. Learning MATLAB 6.5. The MathWorks Inc., 2002.

US Digital Corporation, EM1 & HEDS Transmissive Optical Encoder Modules 9 Mar. 2004 <http://www.usdigital.com/products/em1-heds/>.

US Digital Corporation, E3 Assembly Instructions. 24 Feb. 2004 <http://www.usdigital.com/products/e3/e3assem.shtml>.

US Digital Corporation, E3 Optical Encoder Kit. 24 Feb. 2004 <http://www.usdigital.com/products/e3/>.

Watt, Alan. 3D Computer Graphics Second Edition. Addison Wesley Publishing Company Inc., 1993.

Wheeler, Anthony J., and Ganji, Ahmad R. Introduction To Engineering Experimentation. Prentice Hall, 1996.