Open Loop Compliance Model of a 6 DOF Revolute Manipulator to Improve Accuracy Under Load

By

Mark W. Abbott

Thesis submitted to the faculty of Virginia Polytechnic Institute and State University in fulfillment of the requirements for the degree of

Master of Science

in

Mechanical Engineering

Approved:

Dr. Robert Sturges, Chair

Dr. Charles Reinholtz

Dr. Donald Leo

Dr. William Saunders

April 23, 2002
Blacksburg, Virginia

Open Loop Compliance Model of a 6 DOF Revolute Manipulator to Improve Accuracy Under Load

By
Mark W. Abbott

Abstract

Robotic accuracy has long been limited by the compliance of the manipulator. Whether links under bending loads or backlash in gear trains and stretching of belts, the resulting compliance causes a loss of accuracy at the end-effector. Previous research has investigated accuracy of ideally stiff manipulators from many different points of view; however, an overall compliant modeling technique has not been formulated in the literature. This thesis presents a general technique to develop a compliant model for a general six-degree manipulator with the intent of reducing end-effector error for precision manufacturing.

Experimental and theoretical work was performed on an American Robot Merlin six-degree of freedom robot. The solution technique assumes each link of the manipulator is subject to stiffnesses in three directions, that is, in the direction of motion, laterally and torsionally. Each of the three stiffnesses is assumed constant, but unknown. Three experimental regimes were established, each covering a successively larger region of the workspace, and 243 data samples were taken within each regime. Samples were taken at twenty-seven data points under nine known loads for each of the first two regimes and at nine locations under twenty-seven loads in the third regime. An OPTOTRAK 3020 non-contact distance-measuring system was used to gather data from twelve sensors for each trial. The results were transformed into three displacements and three rotations of the end-effector. A regression algorithm solved for the unknown stiffnesses of the compliant model based on the measured experimental deflection.

Results show that for loads ranging between zero and 445 N, the deflection of the end-effector is predicted within fifteen percent of experimental results for most data points. Furthermore, a load set between zero and 111 N (the stated lift capacity of the

manipulator) predicts end point position with an error of less than one-half a millimeter for all tested points.

This research provides a technique to quantify the compliance of a general manipulator and develops a model capable of being implemented with open-loop position control with known compliance.

Table of Contents

Table of Figures

Table of Tables

Chapter 1: Introduction

Demands for improved tolerance and consistency, combined with requirements for throughput, result in many manufacturing engineers turning to robotics to provide assembly operations. The automotive industry adopted robots to handle welding and other assembly-related issues, while the computer industry relies almost exclusively on robotic technologies to fabricate and populate circuit boards. While the above-mentioned applications require good positional accuracy, under known loads, the use of robots in variable load environments has only recently begun to be realized. This research provides a method for using the inherent compliance of a manipulator to improve open-loop accuracy and allow controlled force transmission throughout the workspace. This technique is developed in a general sense, but motivated by the need to replace hand-layup in the manufacture of composite materials.

Requirements for stronger, lighter and more versatile materials continue to drive technology for material fabrication. Composite materials are composed of at least two mutually insoluble materials [1], are used in a variety of applications from aircraft fuselages to bridges to children's toys. The combination of materials, particulate or fiberous, when surrounded by a matrix created by a second material can result in "high strength and stiffness combined with light weight" [1]. Composites are available in different forms, including individual fibers, tow, or tape, either preimpregnated or designed to have resin added at the point of use. A critical part of manufacturing composite material structures is the lay-up process. The remainder of this chapter will introduce the overall problem encountered in laying-up composite materials and then provide a brief synopsis of the organization of the entire thesis.

Composite lay-up can be a very manually intensive operation as each individual strip of composite material is positioned. This is similar to covering a surface with strips of paper-mache, but not being able to overlap the edges or leave gaps between them. In comparison to hand lay-up processes, industry (e.g. [4]) has developed a class of machines to provide automated tape and tow lay-up capabilities for a limited range of surface features. Accuracy, force and application speed significantly affect the overall

material properties of the composite in both machine and hand lay-up operations. As each additional composite structure is applied, the position and orientation, relative to the existing composite matrix, is critical to guarantee a completely void and lap-free product. Throughout the lay-up process, the composite material must be compressed to force voids from the matrix. Both the magnitude and orientation of the force must be controlled to successfully compress the applied composite. Furthermore, in many applications, heat is applied to aid the adhesion or curing and requires control of the lay-up speed to guarantee desired heat transfer rates and nip-point temperatures [5]. This thesis strives to develop a method to control the applied forces and motions described in the above operations.

Current robotic composite lay-up machines have the ability to start, stop, and cut up to thirty-two individual composite strands simultaneously [4]. However, this class of machines does not tightly control the applied force and resulting placement along the trajectory, resulting in inconsistencies in the compression and lap of the composite. Partially addressing this issue, research at Virginia Tech has investigated thermoplastic composite manufacturing of cylindrical shapes [5]. This technique utilizes a specially designed CNC lathe, with a controlled air-pressure applicator tool and a constant radius roller to provide a constant compaction force. This design results in high quality cylindrical thermoplastic composite parts.

The research presented in this thesis will address thermoset composites, and more specifically a general method to calibrate a generic industrial manipulator to exploit the built in compliance to provide constant force along a desired path. Although the initial target product is a flat coupon, the generalized manipulator should be able to process convex, concave and other surfaces.

The goal of this thesis is to develop a method for characterizing the compliance of a generalized manipulator. Experimental results will be used with a computer model to attempt to find the compliances of the manipulator. It is hoped the overall error between the models can be driven below one millimeter for the expected range of loads.

Chapter 2 of this thesis will discuss previous research in the area of manipulator compliance and then, based on the history provide a clear motivation for the research. The manipulator's kinematic modeling, with an emphasis on the subject manipulator, is

discussed in Chapter 3. This discussion includes forward and inverse kinematics as well as the relations between the applied forces and joint loading. The experimental setup is presented in Chapter 4, including the loading methods used, the physical system and measurement system. Furthermore, the mathematical transformations needed to relate the coordinate systems and the quality of the data recorded are detailed. Chapter 5 will address the optimization problem including a discussion of the algorithm used and the formulation of the objective function. Chapter 6 details the data collected and the trends in the collected data, while Chapter 7 presents the results obtained from the optimization. In conclusion, Chapter 8 restates the initial goals and summarizes the results. Chapter 9 and Chapter 10 include the references and the appendices respectively.

Chapter 2: Literature Review

The majority of this thesis looks forward, providing a new technique to characterize the compliance of a manipulator. However, before delving into new material it is necessary to present the preceding works in which this work is based. Previous research relating to compliant manipulators can be broken down into several areas. The first area covers research on manipulators that are designed to use built-in compliances to perform their duties. In order to properly investigate compliance, the second area addresses accuracy of robotic manipulators. Although not the main topic of this thesis, the discussion on accuracy is included as it is a prerequisite for compliance research. Robotic repeatability will be presented as a subset of accuracy research and finally research on compliance of stiff manipulators will be discussed.

2.1 Compliant Manipulators

Compliant manipulators can be separated into two classes depending on their intended mode of attaining positional accuracy. The first class of manipulators can attain positions based on known compliant deflections in the manipulator. The second class includes many industrial robots and assumes an infinitely stiff manipulator. These stiff manipulators are discussed in the final section of this chapter. The first class of manipulators is well suited for situations where weight is critical. One such example is a COBRA nuclear service robot, which is used beneath the tube sheet in a nuclear steam generator [21].

Joe Calkins presented a method to compensate for static deflection of a COBRA nuclear service robot [21]. This technique introduces a deflection operator that premultiplies the transformation matrices for each joint thereby modifying the Denavit-Hartenburg parameters. The deflection operator is based on either a cantilever beam or torsional stiffness member for each link in the manipulator. The loads are reduced to individual link loads and an iterative procedure tries to converge the deflected link position and orientation. Furthermore, a Hooke and Jeeves optimization routine was used to optimize the stiffness parameters based on experimental measurements under known loads. This technique reduced positional error in a COBRA nuclear service robot, an

extremely compliant robot, from nearly 38 mm to less than 1 mm. This work provides a good basis for implementing optimization to solve for constants relating a model and experimental results.

2.2 Manipulator Accuracy

Manipulator accuracy in the sense of this thesis refers to open loop accuracy. A variety of research has been completed on this topic and therefore will not be investigated independently in the research of this thesis. However, it is important to understand the accuracy problem, since the study of compliance is ultimately limited by robotic accuracy. The following sections provide an overview of some of the important research done in this area.

In 1983, Derby used a computer graphics system (GRASP) to accommodate deflections based on elementary beam theory [10]. This technique was presented as a fast way to calculate the loading a robotic arm would see and to quantify the possible modes of failure. Five years later, in conjunction with Bodur, Derby incorporated joint clearances as well as inertial forces into the model [11]. This model was able to predict a deflected positional accuracy within one percent (about 1 mm error) of experimental tests, although it neglected rotation errors at the joint. Bodur's and Derby's work is representative of many early works which attempt to correlate different models and loading with experimental loads and deflections. This research presented validation of the plausibility of using beam theory for manipulator bending modes.

In addition, in 1983, Mooring argued for a kinematic representation presented by Suh and Radcliffe to identify errors when links are slightly skew [13]. When joint axes are slightly skew from parallel, the common normal can be located far from the manipulator using the Denavit-Hartenburg representation. The common normal defines the location of the coordinate system in this representation. As the "degree of misalignment decreases, the coordinate system location approaches infinity" [13]. The Suh and Radcliffe representation utilizes unit vectors to define the "direction of the rotation axis, a rotation angle about the rotation axis, and a point through which the rotation axis passes" [22]. This technique develops well-conditioned transformation

matrices that are not subject to the ill conditioning of the Denavit-Hartenburg transformation matrices when axes are slightly skew. Their research shows a single degree of error in the shoulder joint can result in more than twenty-five mm of error at the tool tip for a Unimation PUMA 600 6-DOF manipulator. Mooring proposed a technique to calculate the "significant kinematic parameters" by taking measurements of the robotic end-effector relative to the base of the robot at different locations. In this technique, each joint is moved, independently, through a known angle and a new position is recorded. The computer code is then modified to consider the true motion. Mooring's research provides numerical quantification of the possible error a manipulator may see due to compliant deflections. Furthermore, he presents an alternate to the Denavit-Hartenburg representation to avoid the singularities present when two assumed parallel axes are actually slightly skew.

In 1984, Mooring, with Tang, went on to propose an improved method for analyzing the accuracy losses due to joint misalignment without requiring expensive 3-D positioning hardware [14]. The manipulator was placed in a series of fixtures that ensured a set of known joint angles in the manipulator. Based on these known positions, the research provides a method to identify the orientation and axis position errors for a 6-DOF robot. Experimentally, this method was able to compensate for position errors and rotation errors to "within acceptable limits" [14]. Mooring's research with Tang began pioneering work on solving the accuracy problem without the need for expensive 3-D hardware.

Whitney investigated a calibration procedure for serial manipulators in 1986 [20]. The manipulator model included both geometric parameters, e.g. link lengths, joint encoder offsets and the relative orientation of consecutive axes, and non-geometric parameters, e.g. compliance, gear transmission errors, and backlash. Based on theolodite measurements, and a least squares numerical search algorithm, Whitney was able to solve for the unknown model parameters. Experimental tests show reductions in positional error of roughly one order of magnitude.

In 1990, Lin, Chiang and Cui developed a model incorporating both link deflection and joint rotation [12]. Based on Whitney's research, they presented five areas

of accuracy error; backlash, gear transmission, joint drive compliance, cross coupling of joint rotations and base motion. Their research used energy methods to more accurately calculate angular deflections, and went on to compare the functionality between Timoshenkoi beam theory and Castigliano's second theorem, resulting in improved performance from Castigliano's theory since it included joint rotations. Results were calculated using a custom two-link planar mechanism and showed to be in agreement with experimental results to within 0.25 mm. The alternate formulation using energy methods provided an important alternate to standard beam theory used by many researchers.

All of the research on accuracy provides techniques for modeling the mechanical aspects of a manipulator in such a way that the errors can be predicted and compensated. While some approach compliance of specific aspects of the manipulator, none of the above research attempts to apply a compliant model to the entire manipulator.

2.3 Manipulator Repeatability

While robotic manipulators have long been quantified by their repeatability, it is necessary to understand how repeatability is defined and to have at least one method to calculate it. In 1990, Caenan and Angue presented a method for robotic identification and calibration in order to define the limit of accuracy, also known as repeatability, of a manipulator [19]. The technique involves identifying both geometric and non-geometric parameters, including a modification for slightly skew axes with the Denavit-Hartenburg representation. A definition for the identification model and Jacobian matrix are presented and final experimental results showed a limit of accuracy of 0.55 mm for the subject manipulator, "comparable to the limit of the robot". It is not feasible to expect a robotic manipulator, compliant or otherwise, to perform acceptably beyond the limit of accuracy.

2.4 Compliance for Stiff Manipulators

In comparison to compliant manipulators, which are designed to utilize a significant deflection under load, stiff manipulators are idealized to have no deflection

under load. However, this is rarely the case, and compliant deflections contribute to the positional errors seen by so-called stiff manipulators.

In 1994, research was completed using modal analysis at a quasi-static frequency to breakdown static compliances [15]. Experimentation on a fixed-configuration beamlike structure and an industrial robot demonstrated the power of the method. The quasi-static compliance breakdown consists of two steps, namely, the experimental quasi-static measurement and the compliance reduction based on the measured data. The experiment involved applying a sinusoidal force to the system well below the first mode and measuring the accelerations on each member. The examples also illuminated some stringent requirements for the motion sensors. Due to the low frequencies applied, the data from the accelerometers had to account for gravity. Each link was modeled as a long symmetric beam characterized by eight compliances and each joint was modeled as either a torsional or a first order spring, whether rotational or prismatic, respectively. Based on an experiment on a KUKA 6-DOF robot, the theoretical compliance matched statically measured compliances within five percent.

Henry Stone developed a technique based on an S-Model to define link parameters [16]. The S-Model is similar to the Denavit-Hartenburg Model, but includes two additional components. During the prototype application, this method was used to improve performance on a set of PUMA 560 robots. Each manipulator was calibrated and with the arm signature applied was driven to one, two, and three-dimensional touch tasks. The one-dimensional test reduced the error between sixty-five and ninety percent. The two-dimensional tests reduced errors an average of seventy percent, while the three-dimensional tests reduced the overall errors roughly sixty-eight percent on average.

In 1996, Shetty and Ang presented a compliance model utilizing a 6-axis force-torque sensor mounted to the manipulator's wrist and an algorithm to modify the motion of the manipulator. The scheme presented allowed the operator to specify the center of compliance and the appropriate compliances for the task. The force torque sensor measured the interaction between the manipulator and the work piece and the trajectory was modified in real time to ensure the compliances were maintained. The experimental

implementation showed “satisfactory results and hence verified the feasibility of the scheme”.

In 2000, Gong, Yuan, and Ni presented their work on achieving the stringent accuracy requirements required by robotic measurements systems by compensating for nongeometric thermal and compliance issues as well as geometric errors [17]. An error model was developed to combine geometric errors, position-dependent compliance errors and time-variant thermal errors. A laser tracker was used to calibrate the above errors by an inverse calibration method. Experimental trials show that by applying “this methodology the mean residual error of an industrial robot [could be] reduced from 1.2 to 0.11 millimeters”.

While many of the techniques mentioned provide for final errors in the neighborhood of five percent, many require constraints on the directions of applied loads or the area of the workspace in which the model is valid. The goal of this thesis is to provide a technique that can operate throughout the workspace under reasonable loads. The theoretical basis of the research is provided in Chapter 3.

Chapter 3: Theoretical Progression

Accurate control of a robotic manipulator is dependent on the model of the robot. Therefore, it is critical to develop a model that is as accurate as possible with regard to the distances, offsets, angles and other mechanical properties of the physical manipulator. Furthermore, modeling a robot to accurately control its endpoint positions under load requires an effective characterization of the compliances in each link and joint. To maintain consistency with other research, the Denavit-Hartenburg (DH) Representation is implemented to model the robot structure [3]. Although Craig [2] provides the historical representation, a slightly modified representation presented by Spong and Vidyasagar [3] is used. Spong's and Vidyasagar's representation uses a different index numbering scheme that provides a more straightforward calculation of the Jacobian matrix. The next several sections discuss the kinematic representation used in this work.

The kinematics are broken down into two separate sections. The first section, the forward kinematics, solves the problem of finding the position and orientation of the end-effector given joint angles. The inverse kinematics solves the reverse problem of finding valid joint angles given the position and orientation of the end-effector. The chapter will then present the technique used to solve for joint torques and the basic formulation of the compliant model.

3.1 Forward Kinematics

Throughout this research, a six-degree of freedom (6-DOF) American Robot Merlin manipulator was used for experimentation. This robot was counter-weighted for a maximum net lift capacity of 111 N at any point in the workspace. The first task in modeling the manipulator is defining the coordinate frames based on the Denavit-Hartenburg representation. Assigning frames is a structured process, best explained by a series of steps, as follows.

1. Locate and label the axis of each z-axis according to the motion of the joint
    a. For a revolute joint the z-axis is along the axis of rotation
    b. For a prismatic joint the z-axis is along the direction of motion

2. Locate the origin of each frame where the common normal to z_i and z_{i-1} intersects z_i.
3. Establish x_i through origin i and in the direction of $z_{i-1} \times z_i$.
4. Establish the end-effector frame.
5. Create the Denavit-Hartenburg Table, as follows
    a. a_i = distance from z_{i-1} to z_i along positive x_i
    b. α_i = angle from z_i to z_{i+1} about positive x_i
    c. d_i = distance from x_{i-1} to x_i along positive z_i
    d. θ_i = angle from x_{i-1} to x_i about positive z_{i-1}

Figure 3.1 shows the selected coordinate frames for the Merlin robot and Table 3.1 includes the associated DH Table. Throughout this thesis, the first three joints may be referred to as joints 1, 2 and 3 or the waist, shoulder and elbow joints respectively. References to the wrist refer collectively to joints 4, 5 and 6.

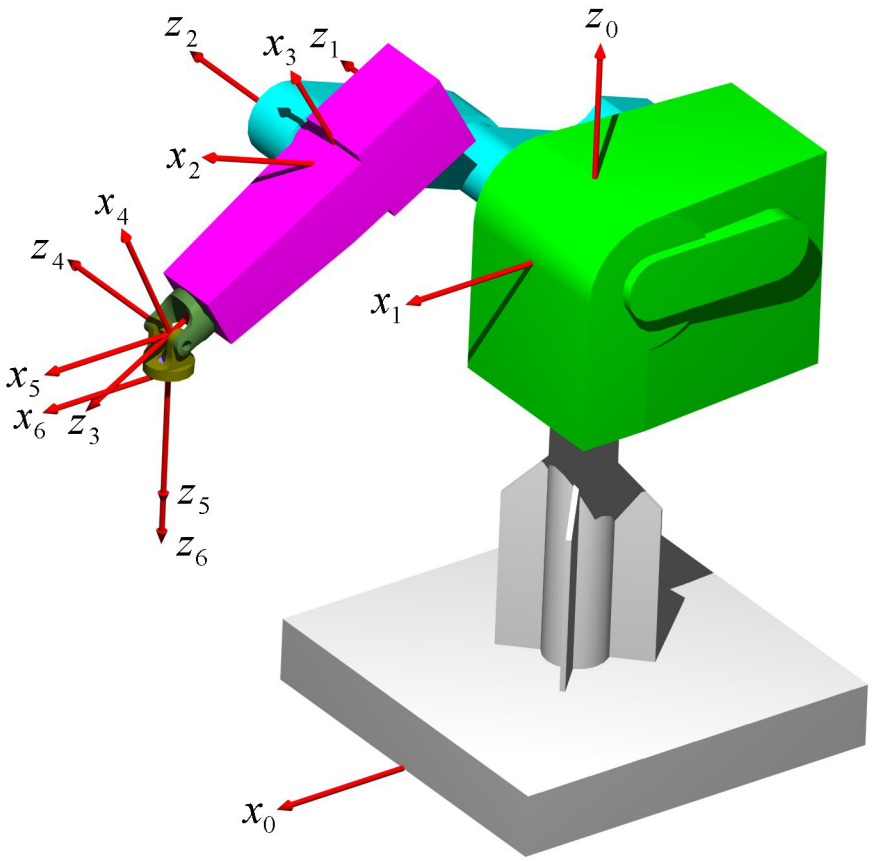


Figure 3.1 Denavit-Hartenburg Coordinate Frames

Table 3.1 Denavit-Hartenburg Table

i	a_i	α_i	d_i	θ_i
1	0	+90	d_1	θ_1^*
2	a_2	0	d_2	θ_2^*
3	0	+90	0	θ_3^*+90
4	0	-90	d_4	θ_4^*
5	0	+90	0	θ_5^*
6	0	0	d_6	θ_6^*

Forward kinematics determines the unique position and orientation of the end-effector, given the joint angles. The forward kinematics solution is derived directly from the DH table. Equation 3.1 defines the transformation matrix from frame *i – 1* to frame *i*. Equation 3.2 is the general method for creating transformation matrices relative to the base frame. Table 3.2 shows the computed transformation matrices from one coordinate frame to the next, as well as a transformation between the sixth frame and the robot's built-in tool frame. All transforms in Table 3.2 are derived based on Equation 3.2, Equation 3.1 and Table 3.1.

Equation 3.1

$$A_i = T_{i-1}^{i} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equation 3.2

$$T_0^n = A_1 A_2 \ldots A_n$$

Table 3.2 DH Transformation Matrices

$T_0^1 = \begin{bmatrix} \cos(\theta_1) & 0 & \sin(\theta_1) & 0 \\ \sin(\theta_1) & 0 & -\cos(\theta_1) & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$T_1^2 = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & a_2\cos(\theta_2) \\ \sin(\theta_2) & \cos(\theta_2) & 0 & a_2\sin(\theta_2) \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
$T_2^3 = \begin{bmatrix} -\sin(\theta_3) & 0 & \cos(\theta_3) & 0 \\ \cos(\theta_3) & 0 & \sin(\theta_3) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$T_3^4 = \begin{bmatrix} \cos(\theta_4) & 0 & -\sin(\theta_4) & 0 \\ \sin(\theta_4) & 0 & \cos(\theta_4) & 0 \\ 0 & -1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
$T_4^5 = \begin{bmatrix} \cos(\theta_5) & 0 & \sin(\theta_5) & 0 \\ \sin(\theta_5) & 0 & -\cos(\theta_5) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$T_5^6 = \begin{bmatrix} \cos(\theta_6) & -\sin(\theta_6) & 0 & 0 \\ \sin(\theta_6) & \cos(\theta_6) & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
$T_6^7 = \begin{bmatrix} \cos(\theta_7) & -\sin(\theta_7) & 0 & a_7\cos(\theta_7) \\ \sin(\theta_7) & \cos(\theta_7) & 0 & a_7\sin(\theta_7) \\ 0 & 0 & 1 & d_7 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$T_6^T = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

One particular transformation matrix of interest is the transform between the zeroeth and sixth frames. This transform relates the tool plate of the end-effector with the ground location of the robot and is shown in an expanded form in Table 3.3 along with the notation convention used to compress trigonometric terms.

Table 3.3 Base to End-Effector Transformation Matrix

$T_0^6 = A_1A_2A_3A_4A_5A_6$	$r_{11} = c_5c_6(s_1s_4 - c_1c_4s_{23}) - c_6s_5c_1c_{12} + s_6(c_4s_1 + c_1s_4s_{23})$
	$r_{12} = -c_5c_6(c_1s_4 + c_1c_4s_{23}) - c_6s_5s_1c_{23} + s_6(-c_1c_4 + s_1s_4s_{23})$
	$r_{13} = c_5c_6c_4s_{23} - c_6s_5s_{23} - s_6s_4c_{23}$
	$r_{21} = c_6(c_4s_1 - c_1s_4s_{23}) - c_5s_6(c_1c_4 - c_1c_4s_{23}) + c_1c_{23}s_5s_6$
$T_0^6 = \begin{bmatrix} r_{11} & r_{21} & r_{31} & d_x \\ r_{12} & r_{22} & r_{32} & d_y \\ r_{13} & r_{23} & r_{33} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$r_{22} = c_6(-c_1c_4 + s_1s_4s_{23}) + c_5c_6(c_1s_4 + s_1c_4s_{23}) + s_5s_6s_1c_{23}$
	$r_{23} = -c_6(s_4c_{34}) - c_5c_6c_4c_{23} + c_{23}c_5s_6$
	$r_{31} = s_5(s_1s_4 - c_1c_4s_{23}) - c_1c_5c_{23}$
	$r_{32} = -s_5(s_1s_4 + c_4s_1s_{23}) + c_5s_1c_{23}$
	$r_{33} = c_5s_{23} + s_5c_4c_{23}$
	$d_x = a_2c_1c_2 + d_2s_1 + c_1c_{23}(d_4 + d_6c_5) + d_6s_5(s_1s_4 - c_1c_4s_{23})$

	$d_y = a_2 c_2 s_1 - d_2 c_1 + s_1 c_{23}(d_4 + d_6 c_2) - d_6 s_5 (c_1 s_4 + c_4 s_1 s_{23})$ $d_z = d_1 + a_2 s_2 + d_4 s_{23} + d_6 c_5 s_{23} + d_6 s_5 c_4 c_{23}$
Where:	$c_{12} = c_1 c_2 - s_1 s_2 = \cos(\theta_1 + \theta_2)$ $s_{12} = c_1 s_2 + c_2 s_1 = \sin(\theta_1 + \theta_2)$

For this research, two different formulations of the forward kinematics were developed. The first assumed the DH coordinates remained constant. This code therefore models the manipulator assuming the joints are infinitely stiff. The second code allows specification of all the DH coordinates, that is, α, θ, *d*, and *a*. This technique is used later in the work to generate the deformed manipulator shape, by allowing modified DH variables to be used. The forward kinematic matrices can be developed to allow specification of the DH variables by generalizing the DH table with variables and building the transforms as shown in Equation 3.1 and Equation 3.2. MATLAB codes to compute the forward kinematics based on both techniques are included in the appendix, and are named Merlin_FK.m (10.2.3) and forwardk.m (10.5.1), respectively. In order to fully quantify the problem, it is necessary to solve both the forward and inverse kinematics, the following sections present the inverse kinematic derivation.

3.2 Inverse Kinematics

Inverse kinematics solves for sets of joint angles to reach a given position and orientation of the end-effector in the workspace. For a general six-degree of freedom manipulator, like the Merlin, eight joint angle solutions exist for a single point in space. As the robot configuration approaches mechanical limits or singularities in the workspace, the total number of valid solutions reduces. The Merlin's eight solutions result from a combination of the elbow-up versus elbow-down configurations, a forward and backward alignment of the waist joint and a wrist reversal. For a six degree of freedom manipulator with a spherical wrist, it has been shown that the inverse kinematics can be solved in closed-form if the z-axes for the final three joints intersect at the wrist center [8]. This condition allows for kinematic decoupling of the inverse kinematics problem into two segments.

This research solves the inverse kinematics following the graphical solution of Spong and Vidyasagar for the arm, and adopts their spherical wrist solution. Initially the wrist and arm problems are decoupled by backing the tool point down to the wrist center. Equation 3.3 shows the location of the wrist center, *p*, where d_6 is the distance along z_5 to the frame on the tool plate, d_x, d_y, and d_z are the components of the end-effector position and r_{xx} are defined in Table 3.3.

Equation 3.3

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} d_x - d_6 r_{13} \\ d_y - d_6 r_{23} \\ d_z - d_6 r_{33} \end{bmatrix}$$

Decoupling allows solution of the inverse position kinematics, including the first three joints, to be solved independent of the wrist joints. However, the converse is not true, as the wrist solution relies on a successful arm solution.

3.2.1 Inverse Position Kinematics

The arm solution is based on a graphical procedure as shown in Figure 3.2 and Figure 3.3. Figure 3.2 is a top view of the arm of the robot, and Figure 3.3 shows a planar view of links 2 and 3 of the robot arm.

Theta 1 is calculated based on Figure 3.2 by assigning α and β as shown in Equation 3.4 and Equation 3.5.

Equation 3.4

$$\alpha = ATAN2(p_y, p_x)$$

Equation 3.5

$$\beta = ATAN2(d_2, s)$$

The equivalent arm length *s* is derived based on right triangles and shown in Equation 3.6.

Equation 3.6

$$s = \sqrt{p_x^2 + p_y^2 - d_1^2}$$

The two solutions, included in Table 3.4, are then solved for by summing angles about the triangles and solving for the unknown values.

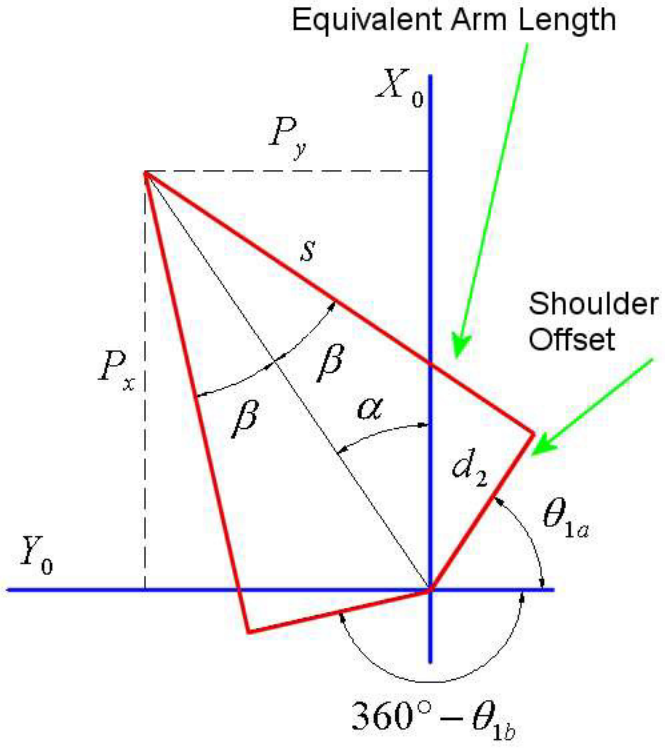


Figure 3.2 Inverse Position Kinematics, Top View of Arm

Based on Figure 3.3, both theta 2 and 3 can be determined. Using the law of cosines, Equation 3.7, and the triangle formed by the two links and their respective hypotenuse theta 3 is solved for explicitly as shown in Equation 3.8.

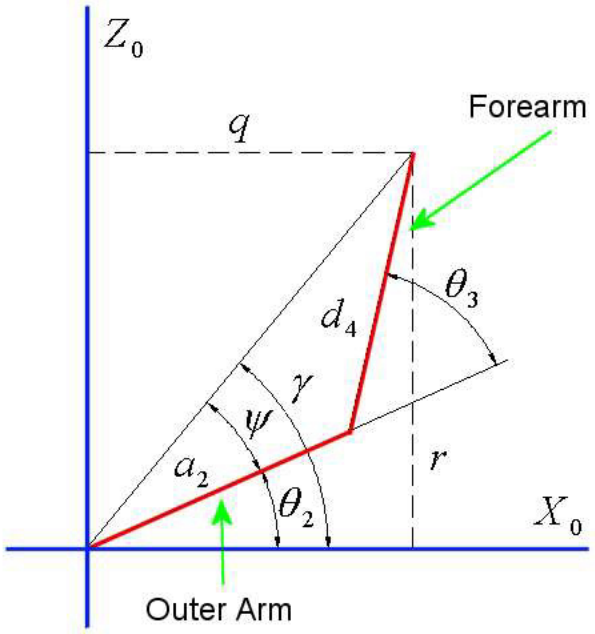


Figure 3.3 Inverse Position Kinematics, Planar View of Arm

Equation 3.7

$$q^2 + r^2 = a_2^2 + d_4^2 - 2a_2 d_4 \cos(180 - \theta_3)$$

Equation 3.8

$$\theta_3 = ATAN2\left(\pm\sqrt{1 - D_3^2}, D_3\right);\ D_3 = \frac{q^2 + r^2 - a_2^2 - d_4^2}{2a_2 d_4}$$

Theta 2 can be obtained by finding angles γ and ψ. Again the law of cosines, as shown in Equation 3.9, is used to determine ψ, Equation 3.10.

Equation 3.9

$$d_4^2 = q^2 + r^2 + a_2^2 - 2\sqrt{q^2 + r^2}\, a_2 \cos(\psi)$$

Equation 3.10

$$\psi = ATAN2\left(\pm\sqrt{1 - D_2^2}, D_2\right);\ D_2 = \frac{d_4^2 - q^2 - r^2 - a_2^2}{-s\sqrt{q^2 + r^2}\, a_2}$$

Based on the geometry of the arm in Figure 3.3, γ can be determined directly and is included as Equation 3.11. The difference in γ and ψ is the angle theta 2, as shown in Equation 3.12.

Equation 3.11

$$\gamma = ATAN2(r, q)$$

Equation 3.12

$$\theta_2 = \gamma - \psi$$

Table 3.4 displays the graphically derived solution of the inverse position problem. Solutions are shown as pairs with the only difference being the wrist orientation.

Table 3.4 Inverse Kinematic Position Solutions

Solution #1 and #2

$$\theta_1 = ATAN2(P_y, P_x) + ATAN2(d_2, s)$$

$$\theta_2 = ATAN2(r, +q) - ATAN2\left(+\sqrt{1 - D_2^2}, D_2\right)$$

$$\theta_3 = ATAN2\left(+\sqrt{1 - D_3^2}.D_3\right)$$

Solution #3 and #4

$$\theta_1 = ATAN2(P_y, P_x) + ATAN2(d_2, s)$$

$$\theta_2 = ATAN2(r, +q) - ATAN2\left(-\sqrt{1 - D_2^2}, D_2\right)$$

$$\theta_3 = ATAN2\left(-\sqrt{1 - D_3^2}.D_3\right)$$

Solution #5 and #6

$$\theta_1 = ATAN2(P_y, P_x) - ATAN2(d_2, s) + 180°$$

$$\theta_2 = ATAN2(r, -q) - ATAN2\left(+\sqrt{1 - D_2^2}, D_2\right)$$

$$\theta_3 = ATAN2\left(+\sqrt{1 - D_3^2}.D_3\right)$$

Solution #7 and #8

$$\theta_1 = ATAN2(P_y, P_x) - ATAN2(d_2, s) + 180°$$

$$\theta_2 = ATAN2(r, -q) - ATAN2\left(-\sqrt{1 - D_2^2}, D_2\right)$$

$$\theta_3 = ATAN2\left(-\sqrt{1 - D_3^2}, D_3\right)$$

Defined Variables

$$q = \sqrt{p_x^2 + p_y^2 - d_1^2}$$

$$s = \sqrt{p_x^2 + p_y^2 - d_1^2}$$

$$D_2 = \frac{d_4^2 - q^2 - r^2 - a_2^2}{-2\sqrt{q^2 + r^2}\, a_2}$$

$$D_3 = \frac{q^2 + r^2 - a_2^2 - d_4^2}{2a_2 d_4}$$

3.2.2 Inverse Orientation Kinematics

The wrist solution assumes a given tool orientation, and predetermined values for joint angles 1, 2 and 3. Knowing these values allows the formulation of rotation-only equations (Equation 3.13) where the right-hand side is completely known and the left-hand side is only a function of the three unknown wrist angles.

Equation 3.13

$$R_3^6 = \left(R_0^3\right)^T R_T$$

Equation 3.13, with known values inserted, results in Equation 3.14 and Equation 3.15, in matrix form.

Equation 3.14

$$R_3^6 = \begin{bmatrix} c_4c_5c_6 - s_4s_6 & -c_4c_5s_6 - s_4c_6 & c_4s_6 \\ s_4c_5c_6 + c_4s_6 & -s_4c_5s_6 + c_4c_6 & s_4s_5 \\ -s_5c_6 & s_5s_6 & c_5 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Equation 3.15

$$\left(R_0^3\right)^T R_T = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

By equating terms in Equation 3.14 and Equation 3.15 with Table 3.4, the solutions for the unknown angles are found, as shown in Table 3.5.

Table 3.5 Inverse Orientation Kinematics Solutions

Solutions #1 - #4
$\theta_4 = ATAN2(b_{23}, b_{13})$
$\theta_5 = ATAN2\left(\sqrt{1-b_{33}^2}, b_{33}\right)$
$\theta_6 = ATAN2(b_{32}, -b_{31})$

Solutions #5 - #8
$\theta_4 = ATAN2(b_{23}, b_{13}) + \pi$
$\theta_5 = ATAN2\left(-\sqrt{1-b_{33}^2}, b_{33}\right)$
$\theta_6 = ATAN2(b_{32}, -b_{31}) + \pi$

Complete solutions are formed by superposition of the inverse position and orientation solutions. Solutions of the forward and inverse kinematics form the basis of the computer model of the manipulator. The formulation at this stage represents an ideal, or infinitely stiff, manipulator. Under any load and in any orientation it is assumed the links remain straight and there is no deflection or loss in the joints. Generalized solutions to the inverse kinematics can generate the solutions in any order depending on the method used to solve the equations. Complete forward and inverse kinematics were verified against each other with MATLAB code included in appendix section 10.5.16 .

3.2.3 Inverse Kinematic Root Tracking

Implementation of the above inverse kinematic technique has an added feature that the roots track relative to the closure of the manipulator arm. That is, if the roots are

maintained in the same order, a certain solution will always be an elbow-up, no wrist reversal solution. Therefore, if the said solution does not exist within mechanical limits, then the manipulator is not capable of reaching the position and orientation with an elbow-up, no wrist reversal orientation. The results from the IK_Merlin.m (10.2.4) MATLAB code are formatted to report results based on this technique.

The following example of the kinematic input and output uses a test code designed to verify both the forward and inverse kinematics against each other. A set of joint angles is specified in the forward kinematic solver to generate the corresponding rotation and positions vectors that are in turn passed to the inverse kinematics. The inverse kinematics then return up to eight solutions including the solution used to seed the problem. Table 3.6 shows the seed angles used and Table 3.7 displays the results from the inverse kinematic solver. In this case, the first solution corresponds to the solution used to seed the problem.

Table 3.6 Inverse Kinematic Solution Seed Angles

Joint	Angle(deg)
1	0
2	40
3	25
4	0
5	25
6	0

Table 3.7 Inverse Kinematic Solutions

	Solutions (degrees)							
Joint	#1	#2	#3	#4	#5	#6	#7	#8
1	0	0	0	0	120	120	120	120
2	40	40	64.91	64.91	115.1	115.1	140	140
3	25	25	-25	-25	25	25	-25	-25
4	0	180	0	180	180	360	180	360
5	25	-25	50.09	50.09	50.09	-50.09	25	-25
6	0	180	0	180	60.03	240	60.03	240

It is evident the solutions are grouped in pairs, with only the wrist reversal distinguishing the solutions. Furthermore, these results are consistent with other solutions regarding the closure of the arm. That is, solution #1 will always be the forward, elbow-up solution without the wrist reversal. Table 3.8 shows the tracking of the individual roots from the inverse kinematic solver.

Table 3.8 Inverse Kinematic Root Tracks

	Solution			
Orientation	**#1**	**#2**	**#3**	**#4**
Waist	Forward	Forward	Forward	Forward
Elbow	Up	Up	Down	Down
Wrist	Forward	Reversed	Forward	Reversed

	#5	#6	#7	#8
Waist	Backward	Backward	Backward	Backward
Elbow	Up	Up	Down	Down
Wrist	Forward	Reversed	Forward	Reversed

3.3 Transformation Matrices

With forward and inverse kinematics firmly established, the notion of building transforms needs to be clarified. In many cases, the transforms necessary between

coordinate frames are the default matrices built in the forward kinematics. However, in other situations, a coordinate system is built from experimental data and the transformation between this experimental frame and another frame does not come directly from Denavit-Hartenburg coordinate frames. The following technique is used throughout the computer code to generate necessary transformations.

In order to solve for the transformation matrix relating two coordinate frames, it is necessary to know the position and orientation of one frame in the other. It is assumed for this explanation that the unknown frame is known in the relative frame, and shown in Figure 3.4.

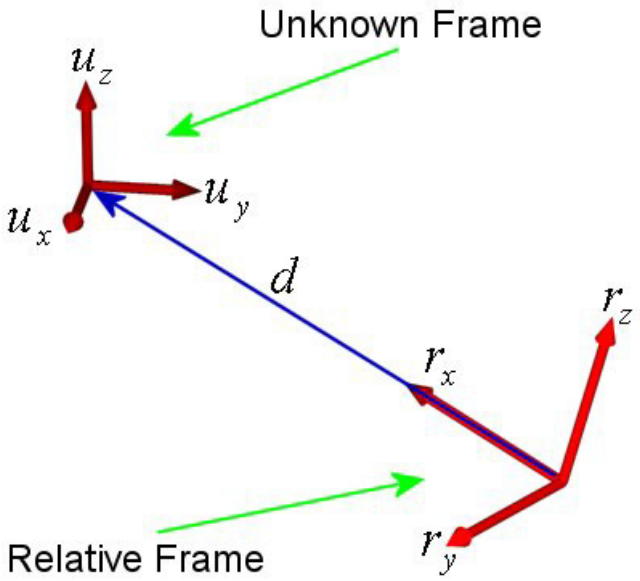


Figure 3.4 Transformation Matrix Example

Based on the unit vectors for the axes of the unknown frame in the relative frame, and the position of the origins relative to one another, the transformation is assembled as shown in Equation 3.16. The leading superscript denotes the frame of reference, and the three unit vectors, $\hat{X}$, $\hat{Y}$ and $\hat{Z}$, are the projections of the three axes of the unknown frame in the relative frame. The vector d is the distance from the origin of the relative frame to the origin of the unknown frame, defined in the relative frame.

Equation 3.16

$$T_{relative}^{unknown} = \begin{bmatrix} {}^{rel}\hat{X} & {}^{rel}\hat{Y} & {}^{rel}\hat{Z} & {}^{rel}d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.4 Joint Torques and Manipulator Jacobians

In order to quantify the compliance of the robot, it is necessary to convert all external loads to joint torques. Individual joints are modeled as torsional springs in this research. Although the primary stiffness of interest is about the axis of rotation, the links may also exhibit a significant compliance both laterally and torsionally. Each of these stiffnesses, as shown in Figure 3.5, are represented in the compliance model.

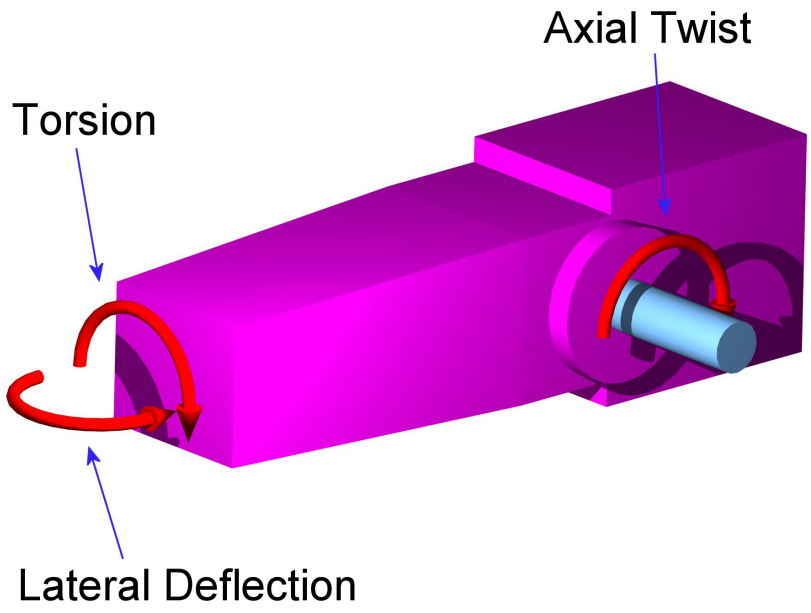


Figure 3.5 Modeled Joint Stiffnesses of Manipulator Arm

A computer compliance model has been developed to characterize an idealized robotic manipulator (10.5.2). The model is written in MATLAB and solves the forward and inverse kinematics for a 6-DOF American Robot Merlin manipulator. Furthermore,

the forward kinematic module computes the Jacobian. In the model, joint torques are applied to joint stiffnesses, and a joint rotation is developed, resulting in deflection. In order to solve the model, the joint torques must first be solved.

3.4.1 Loads and Joint Torques

Joint torques are related to applied loads through a manipulator Jacobian matrix. The following steps present a method to determine joint torques from both external and internal loads.

The Jacobian can be defined column-wise based on the motion of the joint, where Equation 3.17 is used for a revolute joint and Equation 3.18 is used for a prismatic joint [3].

Equation 3.17

$$ {}^{0}J_i = \begin{bmatrix} z_{i-1} \times (o_n - o_{i-1}) \\ z_{i-1} \end{bmatrix} $$

Equation 3.18

$$ {}^{0}J_i = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix} $$

Each of these equations assume o_k is the vector from the origin o_0 to the origin o_k for any k and the leading superscript ^{0} denotes the frame in which the Jacobian is defined (in this formulation, it is always the zero frame). It may be noted that once the forward kinematics are calculated, the above Jacobian definition is easily computed. That is, z_i is simply the first three elements of the third column of T_0^i, while o_i is given by the first three elements of the fourth column of T_0^i.

3.4.1.1 Generalized Torque Generation Method

In general, the Jacobian transpose provides a direct mapping between a force and the corresponding joint torques. However, the Jacobian must be defined in the same frame as the load being applied. Depending on how the Jacobian is generated (if not by the method above) it may be necessary to transform the Jacobian between two frames.

The rotation of a Jacobian from one frame to another is shown in Equation 3.19, and derived in Craig's *Introduction to Robotics* [2]. The two rotation matrices are standard three-by-three coordinate rotations, while the zeros are three-by-three matrices of zeros to fill out the six-by-six matrix.

Equation 3.19

$$ {}^{A}J = \begin{bmatrix} R_B^A & 0 \\ 0 & R_B^A \end{bmatrix} {}^{B}J $$

In the case of transforming a Jacobian in the sixth frame back to the zeroeth frame, the rotations are simply those defined by the transformation from the zeroeth frame to the sixth frame.

3.4.1.2 Loads at the End-Effector

The standard Jacobian, based on the above formulation, is defined in the zeroeth frame. Therefore, it is straightforward to determine the joint torque components due to an external force (defined in the zeroeth frame) applied to the end-effector. The matrix multiplication to generate the torques is shown below in Equation 3.20, where τ denotes a torque, F_i denotes a force, and J_i is a Jacobian column as defined above.

Equation 3.20

$$ \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \\ \tau_4 \\ \tau_5 \\ \tau_6 \end{bmatrix} = \begin{bmatrix} J_1 & J_2 & J_3 & J_4 & J_5 & J_6 \end{bmatrix}^T \begin{bmatrix} F_x \\ F_y \\ F_z \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} $$

3.4.1.3 Weight of Links and Body Forces

All terrestrial manipulators must carry loads applied externally and the weight of the manipulator's links. When considering the body forces and link weights of a manipulator, it is necessary to have a generalized method in order to guarantee flexibility

in the model. A generalized formulation allows for easier application and modification on both the robot of interest as well as other manipulators. The weight of each link is assumed to act through the center of gravity of the link, however the technique presented does not assume the location of the center of gravity. Therefore, in order to move the link weight to a new location, the center of gravity frame must be relocated.

Briefly, this technique involves locating coordinate frames at the center of gravity of each link. This coordinate frame is oriented with respect to gravity, but located with respect to a parent coordinate frame, which should be fixed relative to the robot. A load is then applied in the coordinate frame and converted back to the torques of the joint through a Jacobian built in the same frame. The following steps will layout the process for computing joint torques due to link weights.

3.4.1.3.1 Step 1. Creating the Center of Gravity (CG) frame

Each link weight is assigned a corresponding CG coordinate system with the same orientation as the parent frame. The parent frame is defined as the primary frame for the link. For example, if the weight of link 2 is being considered, then frame 2 is the parent frame for the 2CG frame, as shown in Figure 3.6. The center of the gravity frame is shown offset from the parent frame for clarity, however this is not necessarily typical.

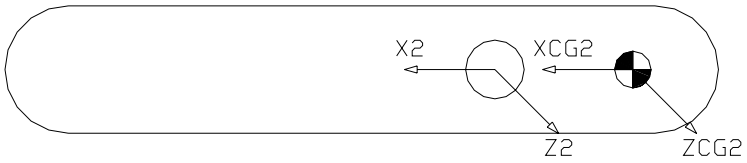


Figure 3.6 Example of CG Coordinate Frame

3.4.1.3.2 Step 2: Creating the Transformation from Parent to CG

Once the center of gravity frames are located, the next step is to create the transform back to the parent frame. By definition, this transformation will have an identity matrix for the rotation component and the displacement vector will be defined from the parent to the CG frame. The basic form is shown below in Equation 3.21. In the notation shown below, *CGn* denotes that the coordinate frame is a center of gravity frame

as well as which parent frame it is associated. In later examples, *n* will be dropped for simplicity once the parent frame is obvious.

Equation 3.21

$$T_n^{CGn} = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.4.1.3.3 Step 3: Creating the Transformation from 0 to CGn

Ultimately, the problem is solved in the zeroeth frame, therefore it is necessary to create a transformation between the *CGn* frame and the zeroeth frame. This transformation, shown as Equation 3.22, is simply the multiplication of the transforms from the zeroeth frame to the parent frame and the parent frame to the *CGn* frame. The transformation from the zeroeth frame to the parent frame is defined through the solution of the forward kinematics.

Equation 3.22

$$T_0^{CGn} = T_o^n T_n^{CGn}$$

3.4.1.3.4 Step 4: Definition of y-axis in weight frame

In order to control the direction in which the weight force is directed, it is necessary to create a frame along which the weight can be oriented. The technique of steps four through six present one technique to create a generic orthonormal frame for use as a weight frame. The y-axis of the weight frame is chosen to be oriented with gravity, and in this case, gravity acts along the negative z-axis of the zeroeth frame. Accordingly, the y-axis of all weight frames is given as Equation 3.23.

Equation 3.23

$${}^0\hat{y}_w = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

3.4.1.3.5 Step 5: Definition of z-axis in weight frame

Based on the y-axis definition, both the x- and z-axes must be defined. The z-axis of the weight frame is defined as the cross product between the x-axis of the CG frame and the y-axis of the weight frame. In the event these two axes are not perpendicular, dividing by the absolute value of the cross product will ensure the result remains a unit vector. If the y-axis of the weight frame and the x-axis of the CG frame are collinear, the z-axis of the CG frame can be substituted. Equation 3.24 and Equation 3.25 show the solutions for both cases.

Equation 3.24

$$ {}^{0}\hat{z}_{w} = \frac{{}^{0}\hat{x}_{CG} \times {}^{0}\hat{y}_{w}}{\left| {}^{0}\hat{x}_{CG} \times {}^{0}\hat{y}_{w} \right|} $$

Equation 3.25

$$ {}^{0}\hat{z}_{w} = \frac{{}^{0}\hat{z}_{CG} \times {}^{0}\hat{y}_{w}}{\left| {}^{0}\hat{z}_{CG} \times {}^{0}\hat{y}_{w} \right|} $$

3.4.1.3.6 Step 6: Definition of the x-axis in the weight frame

The x-axis for the weight frame is defined by the y- and z-axes in the weight frame to create a right-handed frame as shown in Equation 3.26.

Equation 3.26

$$ {}^{0}\hat{x}_{w} = {}^{0}\hat{y}_{CG} \times {}^{0}\hat{z}_{CG} $$

3.4.1.3.7 Step 7: Transformation from zero frame to the weight frame

As with Step 3, it is necessary to create a transformation between the previous coordinate frame and the zeroeth frame. The rotation component of the transformation is simply based on the previously defined three unit vectors while the vector from frame zero to the weight frame is the same as the vector from frame zero to frame CG.

Therefore, the displacement component of the transformation can be taken directly from T_0^{CGn} as defined in step 3, and the complete matrix is shown as Equation 3.27.

Equation 3.27

$$T_0^w = \begin{bmatrix} \begin{bmatrix} & & \\ & R_0^w & \\ & & \end{bmatrix} & \begin{bmatrix} T_0^{CG}(1,4) \\ T_0^{CG}(2,4) \\ T_0^{CG}(3,4) \end{bmatrix} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{bmatrix}$$

3.4.1.3.8 Step 8: Extraction of vectors for Jacobian Matrices

In order to formulate the Jacobian matrices, certain quantities must be extracted from the transformation. Based on the generalized Jacobian development, the **o** and **z** vectors can be easily extracted from the complete transformation matrices as shown as Equation 3.28.

Equation 3.28

$$T_0^w = \begin{bmatrix} - & - & \begin{bmatrix} \\ z_w \\ \end{bmatrix} & \begin{bmatrix} \\ o_w \\ \end{bmatrix} \\ - & - & & \\ - & - & & \\ - & - & - & - \end{bmatrix}$$

3.4.1.3.9 Step 9: Building the Jacobian

In order to build the Jacobian, only the columns relating to actual mechanical joints are considered. This may cause some difficulty since it does not obviously include the offsets between the CG and parent frames. However, the formulation of the Jacobian shows the o_n term which defines a vector between the zeroeth frame and the weight or CG frame. This vector includes the offsets in the formulation. The Jacobian is built column-wise with the definitions presented previously.

For the weight on the forearm of the Merlin robot, the Jacobian would include 3 columns, corresponding to torques on joints 1, 2 and 3.

In summary, the joint torques are a result of the superposition of both internal and external loads. Those loads applied to the end-effector are easily transformed into joint

torques through the default Jacobian, assuming the loads are defined in the zero frame. In order to handle the effects of the internal weights of the manipulator, a series of coordinate transforms are established to locate and orient the weight with gravity. The Jacobian between the zero frame and the parent frame of the desired load is computed and used to solve for the torques on the joints affected by the weight. MATLAB code to perform the torque computations is included in appendix section 10.5.13 . Furthermore, torque verification results are included in appendix section 10.6.1 .

3.5 Compliance

In general, once the torques about the coordinate axes are known, a stiffness can be applied and a deflection developed. These deflections are then applied to the corresponding Denavit-Hartenburg coordinates (a_i, α_i, d_i, θ_i) which are in turn submitted to a generalized forward kinematics code to determine the deflected end-effector position and orientation. The details and caveats of this technique will be discussed in the following pages.

The initial step involves taking an applied torque and an assumed stiffness and calculating the deflection. For this research, all deflections are modeled as torsional springs. For revolute joints, this technique matches the motion of the joint exactly. However, for the lateral deflection, the assumption predicts all of the deflection occurs in the joint and the arm itself does not deflect. This is not as poor an approximation as it may seem at first, because when the manipulator construction is considered, the joint will see the largest torque, and the link itself is very stiff. Furthermore, it is assumed deflections are small. Therefore, all torques and stiffnesses are treated in the manner of Equation 3.29, if τ represents the applied torque, K is the joint stiffness about the axis the torque is applied about and θ is the deflection, in radians.

Equation 3.29

$$\theta = \frac{\tau}{K}$$

Angular deflections are used to modify the DH coordinates directly. The success of this method is dependent on the initial selection of DH coordinate systems. For the

subject manipulator, all joints are revolute and therefore the z-axis of each coordinate frame is coincident with a joint axis. Therefore, rotations about z-axes will be applied directly to the theta DH coordinate, as θ_i measures the angle about z_{i-1} between x_{i-1} and x_i, as shown in Figure 3.7 [3]. Similarly, a rotation about an x-axis will be directly applied to the alpha DH coordinate, as α_i measures the angle between z_{i-1} and x_i about z_{i-1}. Rotations about y-axes do not directly apply to a DH coordinate in the applied frame. In most cases, the y-axis is coincident with the z-axis of the prior frame. In those cases, a rotation about the current y-axis can simply be added to the rotation of the prior z-axis. However, in certain situations, such as y_2 and y_6, the rotations are not directly applicable to the DH model. Figure 3.8 shows the limitation that arose for the y-axis rotations due to the choice of DH frames. As shown a torque about the y-axis in frame two cannot be modeled through the DH coordinates. The changes in length of a2 and d2 are easily computed, but the rotation does not translate into a DH coordinate. The loss of this torque will have a negligible effect on the effectiveness of the model because the z-axis of the zeroeth frame is nearly coincident and will compensate for any small arm deflections.

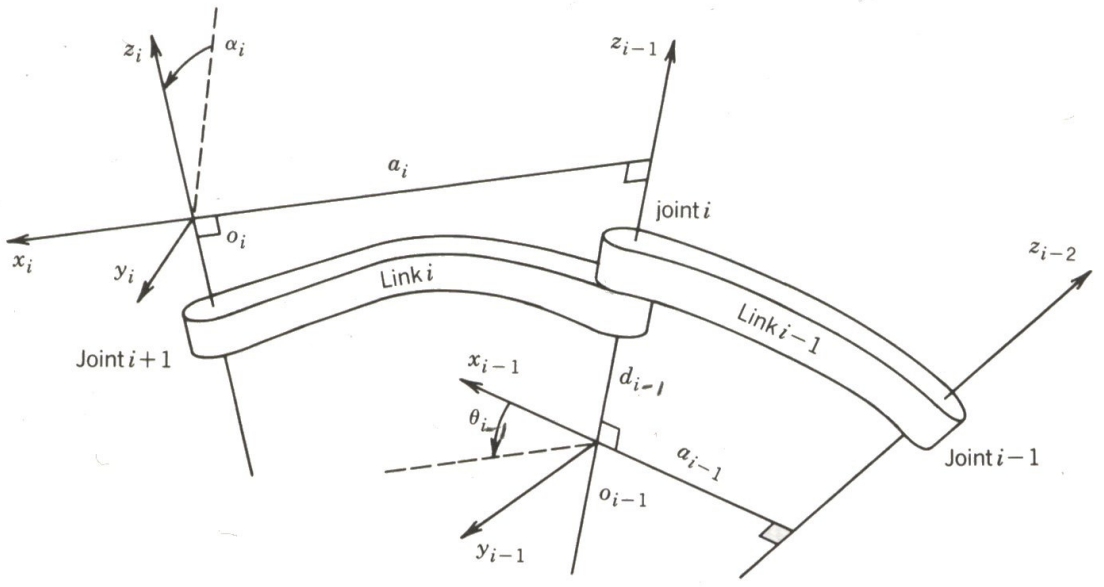


Figure 3.7 Denavit-Hartenberg Frame Assignment

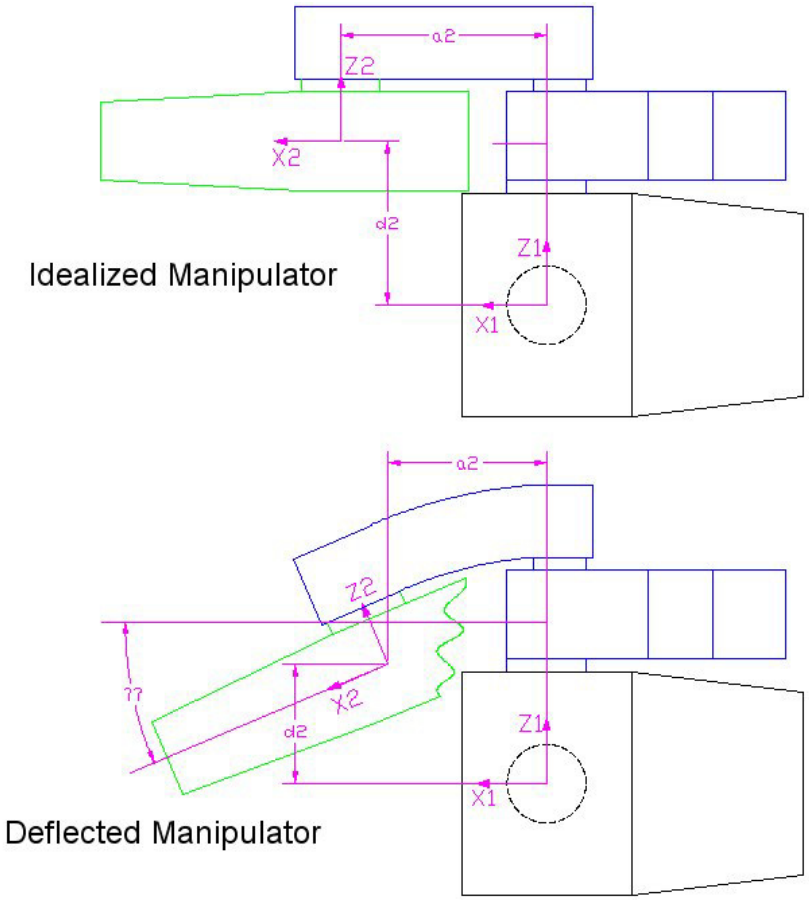


Figure 3.8 Y2 Unaccounted Deflection

The limitation with the sixth frame is a result of the sixth frames' orientation relative to the fifth frame. Since the sixth frame is not tied to a joint motion, the fifth frame corresponds to the sixth joint motion, the y-axis does not line up with any prior axis of motion. The error due to torques about the y-axis of the sixth frame is minimal for the scope of this research since the only torques about the axis are applied torques which were minimized during experimentation.

Once the DH coordinates have the modified angles inserted, the new end-effector position is determined by solving a fully general forward kinematics. This formulation of the forward kinematics allows specification of not only the theta angles, but also the alpha angles and the two displacements. For the scope of this research, only the two DH angles were modified, however the two distances are included for completeness. This new end-effector position and orientation are then compared with the experimental results.

In order to compare the results, it is necessary to compare the differences between the loaded positions to the unloaded positions. In this manner, the influence of any error in rationalizing the two coordinate systems (the measurement system and the robot) will be minimized. In order to compare the results, both the loaded and unloaded cases are reduced to three displacements and three rotations. The displacements fall along the robot's coordinate axes in the zeroeth frame, and the rotations are defined as the yaw, pitch and roll, about the corresponding ${}^0x$, ${}^0y$ and ${}^0z$ axes of the robot. In a method similar to that used in the inverse kinematics, the rotation part of each transformation matrix is equated with a yaw-pitch-roll rotation matrix, and the three angles are extracted. The formulation of the matrix is found in Spong and Vidyasagar's text [3] and shown in Equation 3.30.

Equation 3.30

$$R = \begin{bmatrix} c_\phi c_\theta & -s_\phi c_\psi + c_\phi s_\theta s_\psi & s_\phi s_\psi + c_\phi s_\theta c_\psi \\ s_\phi c_\theta & c_\phi c_\psi + s_\phi s_\theta s_\psi & -c_\phi s_\psi + s_\phi s_\theta c_\psi \\ -s_\theta & c_\theta s_\psi & c_\theta c_\psi \end{bmatrix}$$

Equating Equation 3.30 with the result matrices yields a trigonometric relation to find the roll, pitch and yaw values (ϕ, θ, ψ). The unloaded values are then subtracted from the loaded results for each load case. These results will be used to compare against experimental numbers to compute the objective function, as discussed in Chapter 5: .

This chapter has presented the kinematic model of the manipulator including assignment of DH coordinate frames, development of forward and inverse kinematics, formulation of the compliance model, and converting loads into torques. The optimization will be presented in Chapter 5 after the experimental problem and setup are discussed in Chapter 4.

Chapter 4: Experimental Setup

While an accurate robotic model is critical to accurately predicting the motion and deflection of a manipulator, the experimental verification is equally important. Furthermore, the technique presented in this thesis involves finding unknown stiffness values in the model based on experimental results. Therefore, the final success of the entire method is dependent on the technique to gather data, reduce data and verify resulting data.

The experimental protocol was divided into three different regimes each defining a different volume of the workspace in which testing will occur. The first regime encompasses a selected area of one square foot in front of the robot at the most likely location in the workspace for composite lay-up. The second regime is an eight square foot region, inclusive of the first data set. The third regime is the entire robotic workspace in front of the waist axis, with care taken to avoid the actual boundary of the workspace due to singularities. Furthermore, the first and second regimes will limit the load orientation to true vertical only, while the third regime will involve more varied joint angles and load orientations. The following section provides a technique for point selection since within each regime it is necessary to choose specific points at which to collect data.

4.1 Point Selection

Since each joint is modeled with three stiffnesses, twenty-one unknown stiffnesses exist in the workspace, including stiffnesses associated with a pseudo joint at the zeroeth DH frame. When the joint stiffness is approximated as zeroeth order, each stiffness only contains one unknown term, therefore there are twenty-one unknown coefficients. A zeroeth order stiffness is consistent with a constant stiffness, higher orders of stiffness are assumed to be polynomials. These coefficients will be referred to as stiffnesses for the remainder of the thesis. In order to ensure a well-constrained regression problem, a minimum of twenty-one samples should be used. However, a significantly more dense protocol of twenty-seven points within each workspace was chosen since three points in three directions yields twenty-seven combinations of evenly

spaced data points. When coupled with nine loads applied at each location, the total number of test cases is 243. Data points are distributed within the workspace by subdividing the cubic space into twenty-seven sub-cubes and then assigning a point at the center of each sub-cube. For example, if a total of eight points (2^3) is chosen, the eight locations will be defined as shown in Table 4.1, relative to a local origin. Figure 4.1 shows the eight data points graphically. A simple code to generate these points for a general parallelepiped is included in appendix section 10.4.4 .

Table 4.1 Example Data Points

X	Y	Z
0.25	0.25	0.25
0.25	0.25	0.75
0.25	0.75	0.25
0.25	0.75	0.75
0.75	0.25	0.25
0.75	0.25	0.75
0.75	0.75	0.25
0.75	0.75	0.75

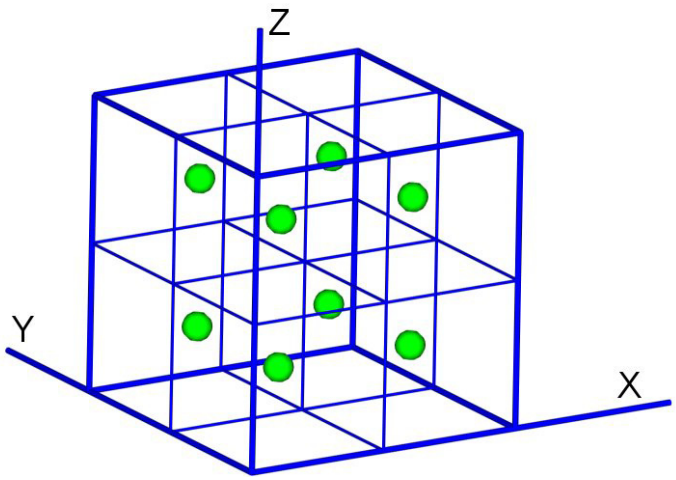


Figure 4.1 Example Data Points

Following the technique outlined above, twenty-seven data points for the first regime were located. Although the second regime would ideally encompass a two-foot cube, due to mechanical limitations of the robot, it is not possible to orient such a volume

inside the workspace. As a result, a two-foot square was centered about the shoulder joint and extruded forward through the workspace. This volume was used to define the data points for the second regime. The third regime of points were selected to lie forward of the waist joint of the robot. All three regimes of data are listed in Table 4.2, and shown graphically in Figure 4.2. The face-plate of the end-effector was oriented facing along the negative z-axis of the zeroeth frame for all tests. The only limitation placed on these points was the mechanical limits of the robot to reach the points. While the first and second regimes have twenty-seven independent locations, the third only has nine locations, since each point is subjected to three different load sets.

Table 4.2 Experimental Data Points

Point	Regime 1			Regime 2			Regime 3		
	X	Y	Z	X	Y	Z	X	Y	Z
1	22	-4	-10	15.5	12	-12	25	-24	4.5
2	22	0	-10	24.4	12	-12	12	-24	4.5
3	22	4	-10	33.3	12	-12	12	-24	-10
4	26	-4	-10	17	12	0	25	-24	-10
5	26	0	-10	25.5	12	0	20	-8.3	0
6	26	4	-10	34	12	0	25	16	4.5
7	30	-4	-10	24	12	12	12	16	4.5
8	30	0	-10	19.5	12	12	12	16	-10
9	30	4	-10	15	12	12	25	16	-10
10	22	-4	-6	15.5	0	12	25	-24	-10
11	22	0	-6	21	0	12	12	-24	-10
12	22	4	-6	27	0	12	25	16	-10
13	26	-4	-6	36	0	0	12	16	-10
14	26	0	-6	28.5	0	0	20	-8.3	**-9.5**
15	26	4	-6	21	0	0	25	-24	**-3**
16	30	-4	-6	19.5	0	-12	12	-24	**-2.3**
17	30	0	-6	27.3	0	-12	12	16	4.5
18	30	4	-6	35	0	-12	25	16	4.5
19	22	-4	-2	33	-12	-12	25	-24	4.5
20	22	0	-2	24.3	-12	-12	12	-24	4.5
21	22	4	-2	15.5	-12	-12	12	16	4.5
22	26	-4	-2	17	-12	0	25	16	4.5
23	26	0	-2	25.5	-12	0	20	-8.3	0
24	26	4	-2	34	-12	0	25	16	-10
25	30	-4	-2	24	-12	12	12	16	-10

26	30	0	-2	17	-12	12	12	-24	-10
27	30	4	-2	10	-12	12	25	-24	-10

** Red Text indicates the 100 lb test failed

** Bold text indicates a point is moved slightly to allow loading

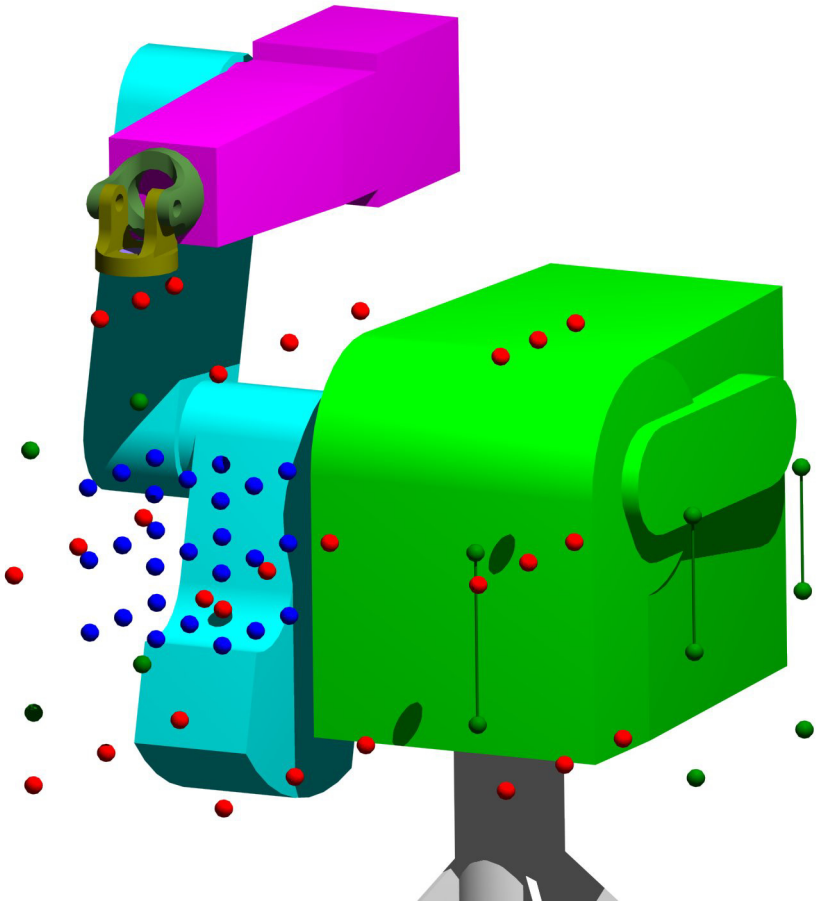


Figure 4.2 Experimental Data Points

In the latter regimes it is difficult in some cases to choose points in the workspace that the robot can mechanically reach. The point selection process for these points was found to be much easier experimentally than analytically since the robot could be placed in straight line mode and moved forwards and backwards to the extremes of the region of interest. The middle point on each of the sides can then be interpolated between the two existing end points. In this method, the resulting points are in Cartesian coordinates and not joint angles. In order to generate joint angles, MATLAB code in appendix section 10.4.2 solves the inverse kinematics and selects the elbow-up solutions given a position and orientation in space. Experimental data points are shown in Figure 4.2 with three

pairs of points connected by lines. These denote points that were moved during experimentation due to mechanical blockages of the loading cable, and the new location is denoted in Table 4.2. In each case, the upper point is the chosen location, while the lower point is the actual location of experimentation. Once the manipulator is positioned in the workspace, a load is applied and the locations of the sensors are measured.

4.2 Deflection Measurement

Experimental position data were gathered using a Northern Digital Inc. OPTOTRAK model 3020; a non-contact tracking device using infra-red emitting tags mounted to the robot. The OPTOTRAK can track the position of up to 256 tags with an accuracy of 0.1 mm at marker rates up to 3500 Hz and angles up to 120 degrees [6]. The OPTOTRAK can compute 3-D position data for each marker at rates comparable to 450 Hz for three sensors [6]. In order to correlate the position of the robot relative to the OPTOTRAK, four markers are attached to the robot base. These markers allow for the relative alignment of the OPTOTRAK and the robot to be determined as well as any flexure or rotation of the robot base to be detected. Figure 4.3 shows the approximate locations of the markers attached to the robot base. All markers were mounted with two-sided foam tape.

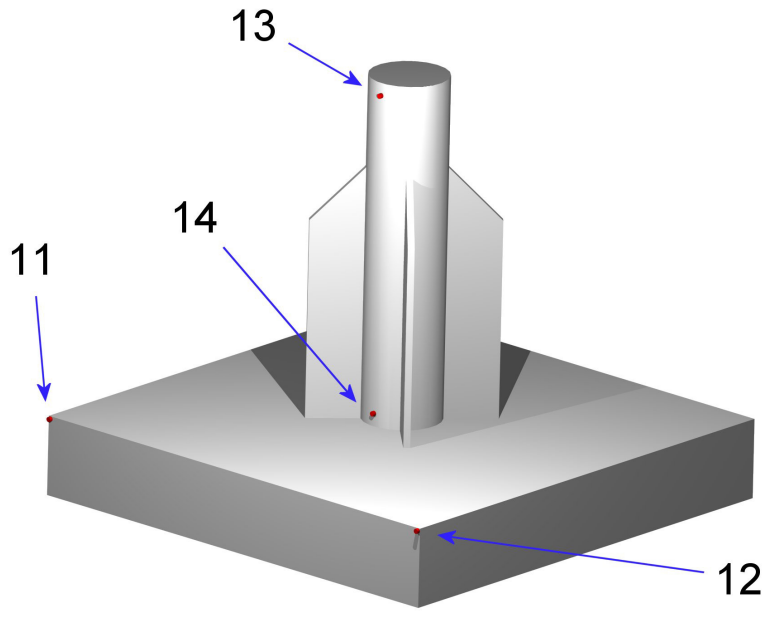


Figure 4.3 Base Markers

In order to measure the deflection of the manipulator, a set of four markers were mounted to a flange on the end-effector, as shown in Figure 4.4.

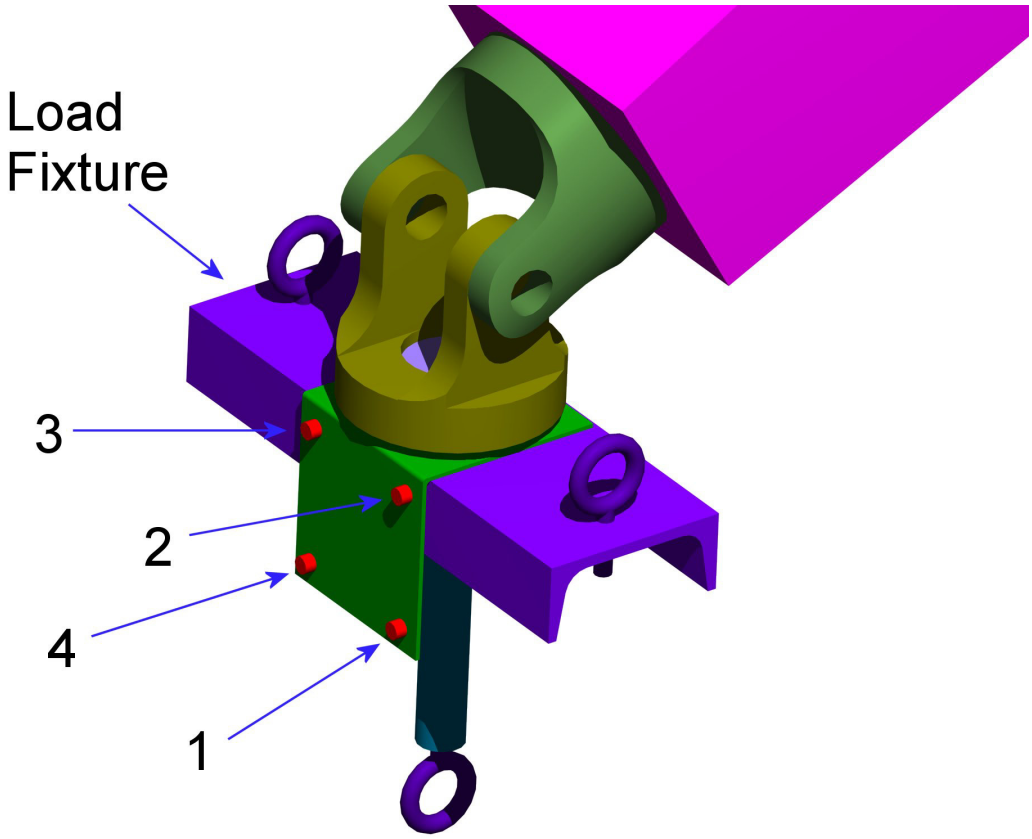


Figure 4.4 End-Effector Markers

Markers were also mounted along the cable applying the load, as shown in Figure 4.5. These markers provided data to compute the actual load vector for each test case. In contrast to the previous markers, the cable markers were mounted with foam tape and a pair of cable ties to keep the foam tape from peeling off the cable, as shown in Figure 4.6. The simple bracket added to the end-effector was designed to both attach markers and apply loads. While end-effector markers were mounted to this flange, as shown in Figure 4.4, the cable sensors were moved to mount on the eye-bolts during horizontal testing of regime 3, as shown in Figure 4.7. A complete discussion of the reporting of the OPTOTRAK results and the correlation between tests and data set is included in Appendix section 10.7 .

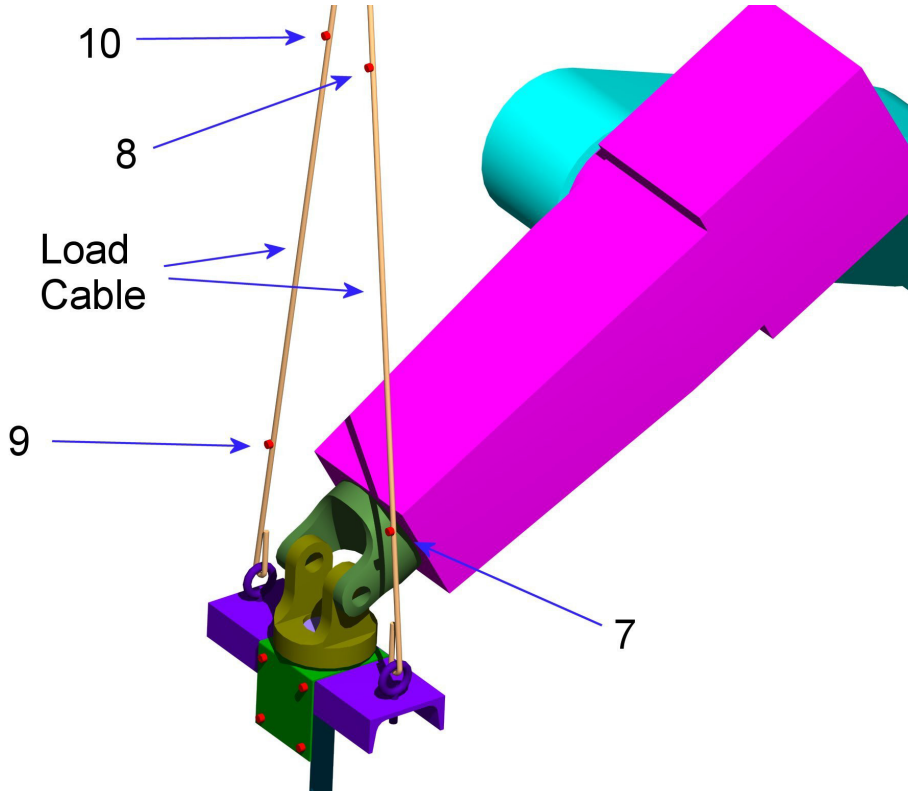


Figure 4.5 Load Cable Sensor Locations

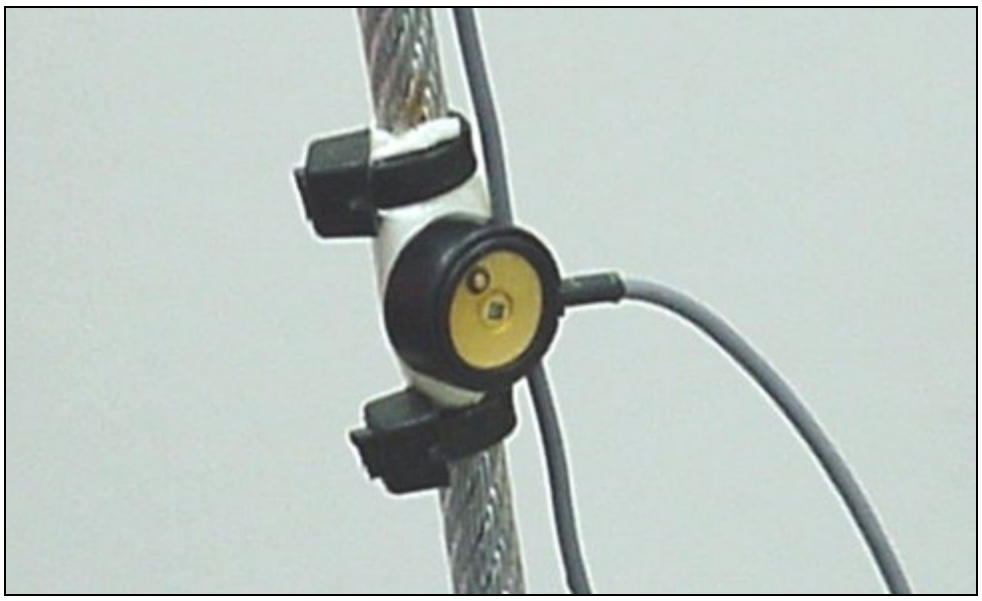


Figure 4.6 Cable Sensor Mounting Technique

Figure 4.7 Regime Three Cable Sensor Mounting

4.3 Overall System

To have sufficient range to see both base markers as well as those attached to the end-effector, the OPTOTRAK was located at a distance of roughly 4 meters (13 feet) from the first robot axis. Figure 4.8 shows the visual range of the OPTOTRAK, with the robot in position. Perpendicular to the page, the OPTOTRAK has sufficient range to capture the extremes of the robot workspace, with a total width of over 2 meters (7 feet).

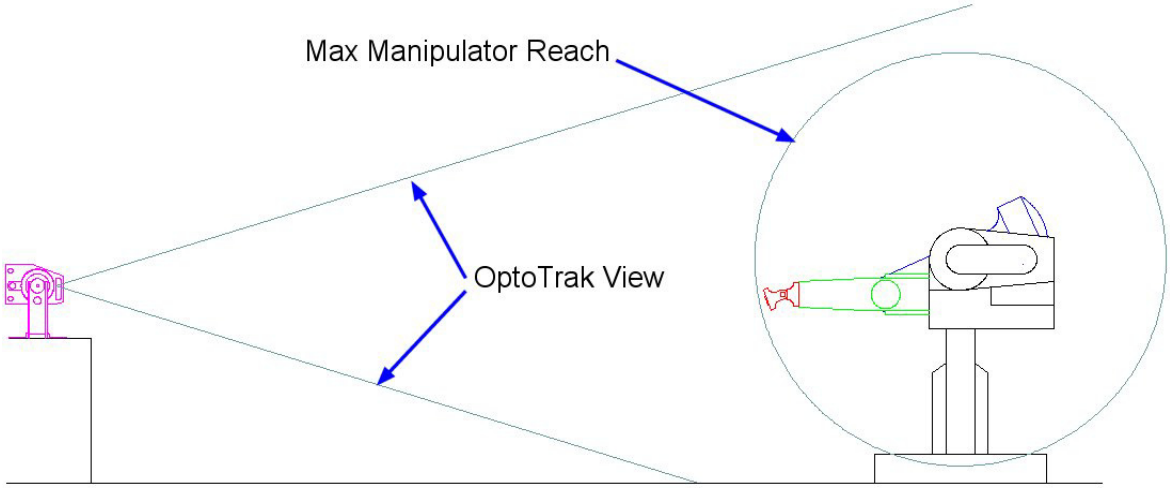


Figure 4.8 OPTOTRAK Range

4.4 Loading Methods

In order to calculate stiffness based on a deflection measurement, it is necessary to apply a known load to the manipulator. For this research, the load consists of a selection of buckets of sand. Each bucket was weighed to within one-quarter of a pound at the beginning of the testing.

Loads were applied in three different directions; normal to the floor, and parallel to the floor both behind and to the left of the robot, in Figure 4.8 this corresponds to down, to the right and out of the page. The robotic manipulator was fitted with a loading bracket, shown in Figure 4.4, on the end-effector to allow direct application of loads via a cable. Figure 4.9 shows the overall system during a vertical load case. A 45 N (10 lb) counter balance was attached to the weight bucket end of the load cable to counter the weight of the loading fixture. In addition to the counter balance, varying weights were added to simulate the reaction force from a composite surface.

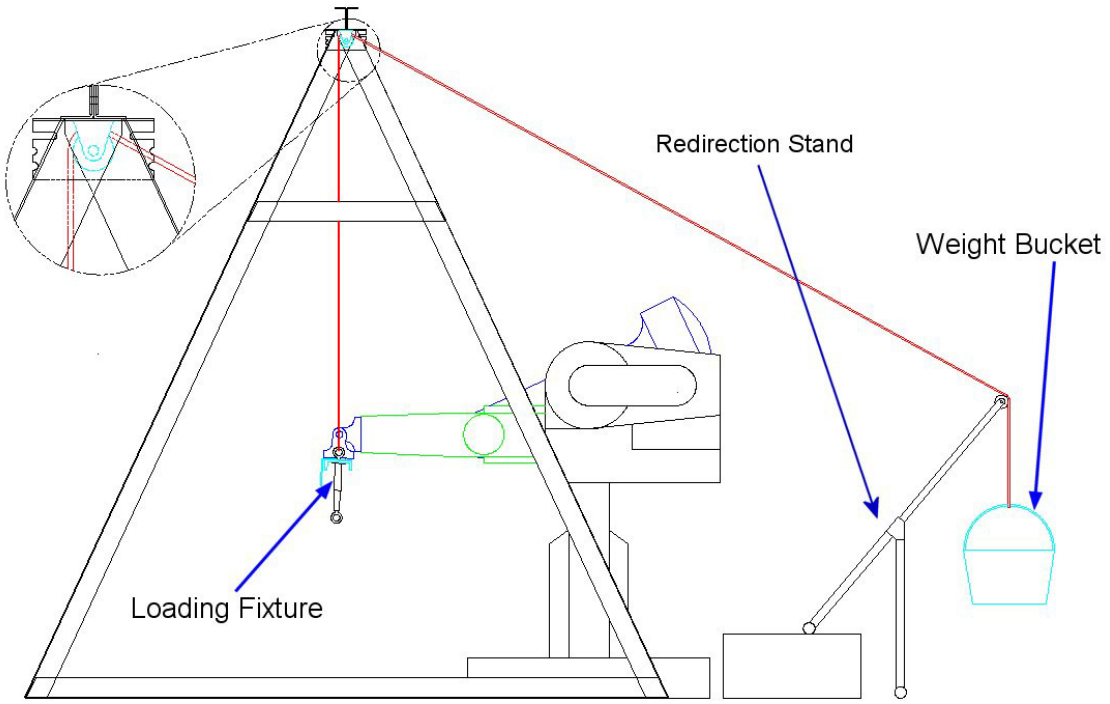


Figure 4.9 Overall System with Vertically Applied Load

A complete list of the loads applied is contained in Table 4.3 in both standard and metric units.

Table 4.3 Applied Loads

Number	Pounds	Variation (lbs)	Newtons
1	0	±0.25	0
2	5.3	±0.25	24.1
3	10.3	±0.25	46.8
4	15.3	±0.25	69.5
5	20.3	±0.25	92.3
6	23.7	±0.25	107.7
7	50.0	±0.25	227.3
8	76.3	±0.25	346.8
9	100.3	±0.25	455.9

Both load cases parallel to the floor were applied in a method similar to the method in Figure 4.9. The redirection stand was replaced with a single bar that ran from

one upright triangle to the other and the roller was mounted on this bar inline with the end-effector position. The weight cable ran from the end-effector to the roller on the bar and then turned towards the floor to the attachment point for the weights as shown in Figure 4.10. The two methods discussed above were used for all experimental tests.

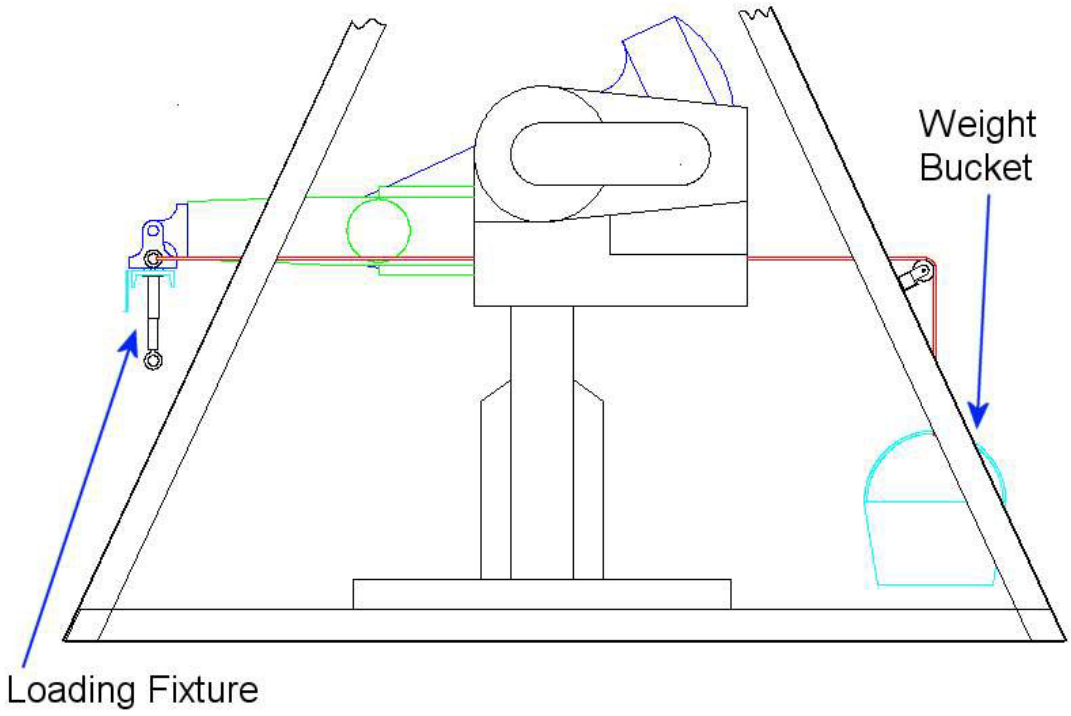


Figure 4.10 Overall System with Horizontally Applied Load

4.5 Coordinate Frame Assignment

As with many measurement systems, it was necessary to relate two otherwise unrelated coordinate frames. In the case of this research, the two frames of interest are the OPTOTRAK, or measurement frame, and the robot, or experimental frame. Relating these two frames enables the exact location of the sensors to be determined relative to the robot coordinate frames. Although the sensors could be located with calipers and dial indicators, it is very difficult to find the distance to an axis of revolution when the exact location of the axis is not easily measured on the manipulator. As such, the OPTOTRAK was used to determine the transforms between the sensors and the DH coordinate frames. While satisfying the immediate need, this technique has the added advantage of

developing the transformations at the same accuracy as the data itself, maintaining a constant level of accuracy in the computations.

There are two separate groups of sensors, each of which must be related to their respective DH coordinate frames. The first group includes the four sensors mounted to the bracket on the end-effector of the robot. These sensors are related to the nearest frame, that is, the sixth DH frame. The second group consists of four sensors mounted to the base and are related to the zeroeth DH frame. Both techniques follow a similar solution, and therefore the solution for the wrist will be presented and only differences or clarifications for the ground frame will be discussed in detail.

It was necessary to relate the OPTOTRAK frame through a transform to the sixth DH frame (or zeroeth DH Frame). Since the goal was only to relate the position of the sensors to the DH coordinate frames, it was not necessary to develop a method to solve for general OPTOTRAK to DH frame coordinate transformations during the solution. Each DH coordinate system was related to the physical robot based on the motion of a joint. By moving select joints through their range of motion and measuring the locations of the sensors it was possible to determine how those sensors were positioned and oriented relative to the axis about which the motion occurred. By performing this technique on at least two intersecting axes, it was then possible to fully relate the OPTOTRAK coordinate frame with the DH coordinates for the tested robot orientation. Static transforms could then be developed between each sensor's data and associated DH coordinate frames. The steps used to develop these transformations are outlined in Figure 4.11 and discussed in detail in the following several paragraphs.

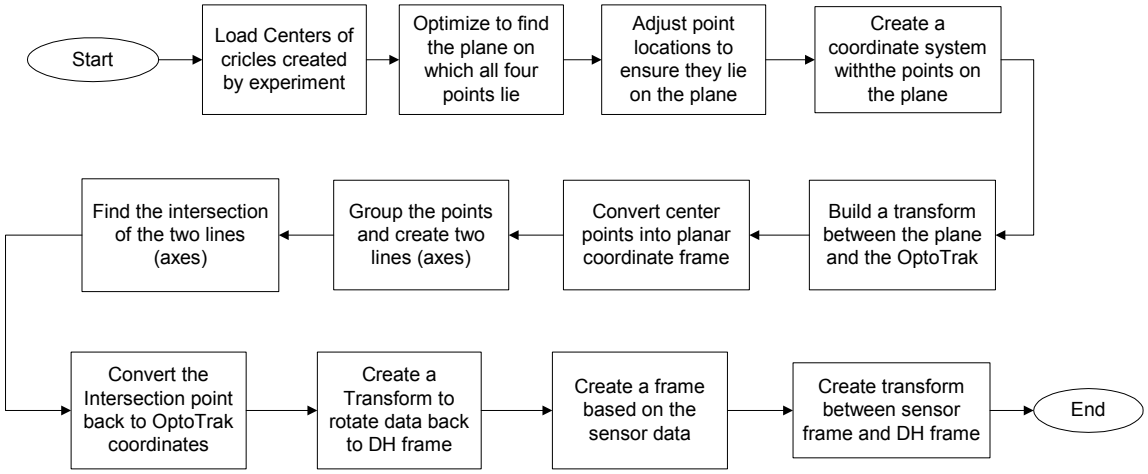


Figure 4.11 Coordinate Frame Assignment Overview

The overall technique involves moving the manipulator through two independent wrist motions while recording data from the wrist sensors. This data is then used to extrapolate where the axes lie and thus where the sensors are in relation to the DH coordinate frames that are associated with those axes.

Initially only joint 5 is rotated, creating a selection of points along a set of circles, resulting in one circle for each sensor. The joint is rotated by small angles with the robot's teach pendant and a sample is taken at each point after the dynamic transients due to the motion minimize. Initial experiments were attempted dynamically, that is, recording data during the motion of the manipulator, but the level of error in the data was much higher when compared to recording data statically. This difference is a result of the gears meshes causing noise in the position during motion. A similar procedure is applied on joint 6, resulting in a total of 4 arcs of data for each rotation. Results of these two rotations are shown in Figure 4.12 and Figure 4.13, and the results of rotations about joints 1 and 2 are shown in Figure 4.14 and Figure 4.15. Data reduction code for each of the four trials is included in appendix sections 10.3.1 , 10.3.2 , 10.3.3 , and 10.3.4 corresponding to test on joints 1, 2 5 and 6, respectively. The MATLAB code used to extract the data traces for plotting purposes is included in appendix section 10.5.8 .

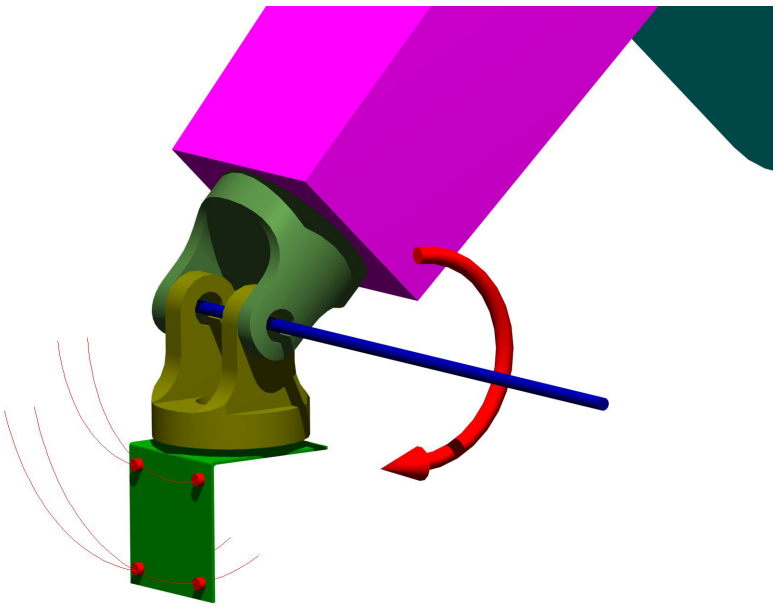


Figure 4.12 Data Set from Joint 5 Motion

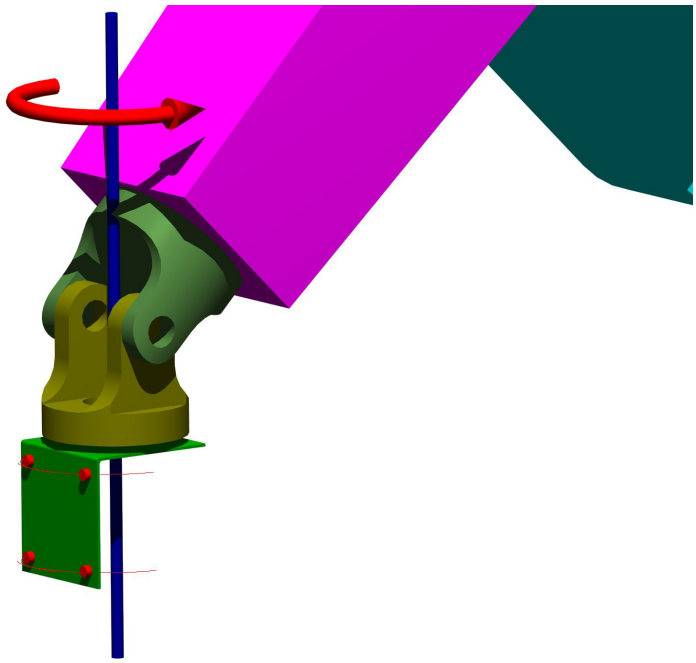


Figure 4.13 Data Set from Joint 6 Motion

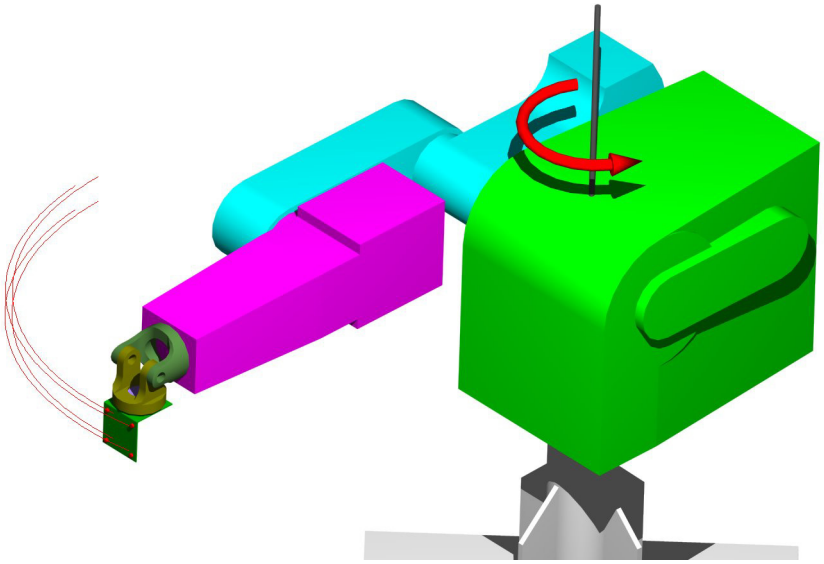


Figure 4.14 Data Set From Joint 1 Motion

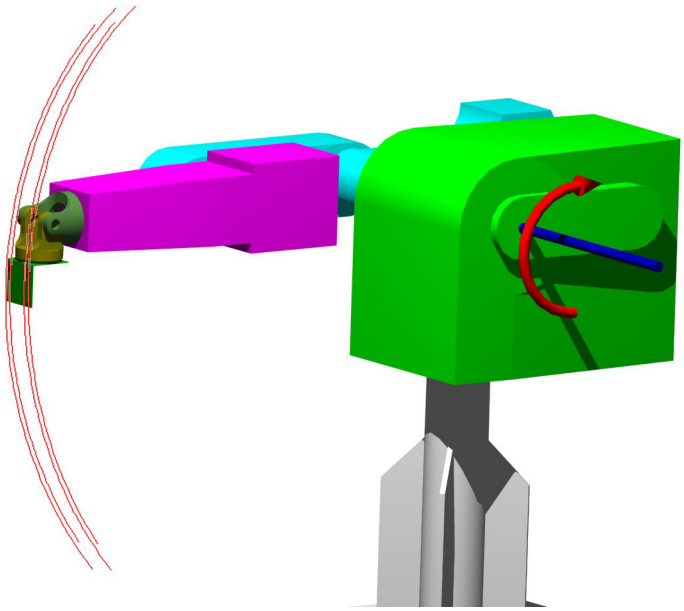


Figure 4.15 Data Set From Joint 2 Motion

In order to locate the center of the circle, the data is divided into three evenly spaced groups along the arc. One sensor from each group is chosen such that the sensors are as evenly spaced as possible. Center points are computed for each set of three sensor results by first establishing two vectors between the three points, as defined in Equation 4.1 and shown in Figure 4.16.

Equation 4.1

$$Vect1a = Pnt2 - Pnt1$$
$$Vect1b = Pnt2 - Pnt3$$

The cross product of the two resulting vectors is a vector normal to the plane of the circle created by the three points, as defined in Equation 4.2.

Equation 4.2

$$Norm1 = Vect1a \times Vect1b$$

Equation 4.5 defines the radius of the circle, where *Mag1*, *Mag2*, and *Mag3* are the lengths of the sides of the triangle, defined in Equation 4.4, and Equation 4.3. The area is defined by Equation 4.6, based on the vectors of two sides of the triangle.

Equation 4.3

$$Side1 = Pnt3 - Pnt2$$
$$Side2 = Pnt1 - Pnt2$$
$$Side3 = Pnt3 - Pnt1$$

Equation 4.4

$$Mag1 = \sqrt{Side1 \bullet Side1'}$$
$$Mag2 = \sqrt{Side2 \bullet Side2'}$$
$$Mag3 = \sqrt{Side3 \bullet Side3'}$$

Equation 4.5

$$R = \frac{Mag1 Mag2 Mag3}{4area}$$

Equation 4.6

$$area = \frac{1}{2}\left(Side1 \times Side2\right)$$

Based on the radius of the circle and the normal vector, the center is located by assembling a series of vectors. Figure 4.16 shows the layout of the vectors defined in the following steps. The vector named *Half1* is computed, as in Equation 4.7, and located at the middle point as shown.

Equation 4.7

$$Half1 = \frac{Side1}{2}$$

This formulation locates the center point of the *Side1* vector. The distance from this point to the center is needed to formulate the second vector. Using the law of cosines, the base angle can be derived, as in Equation 4.8.

Equation 4.8

$$\cos\left(2\alpha\right) = \frac{\left|Side1\right|^2 - 2R^2}{-2R^2}$$

The center distance, denoted *Rscale*, is computed based on the simple trigonometric relationship in Equation 4.9.

Equation 4.9

$$Rscale = R\cos\left(\alpha\right)$$

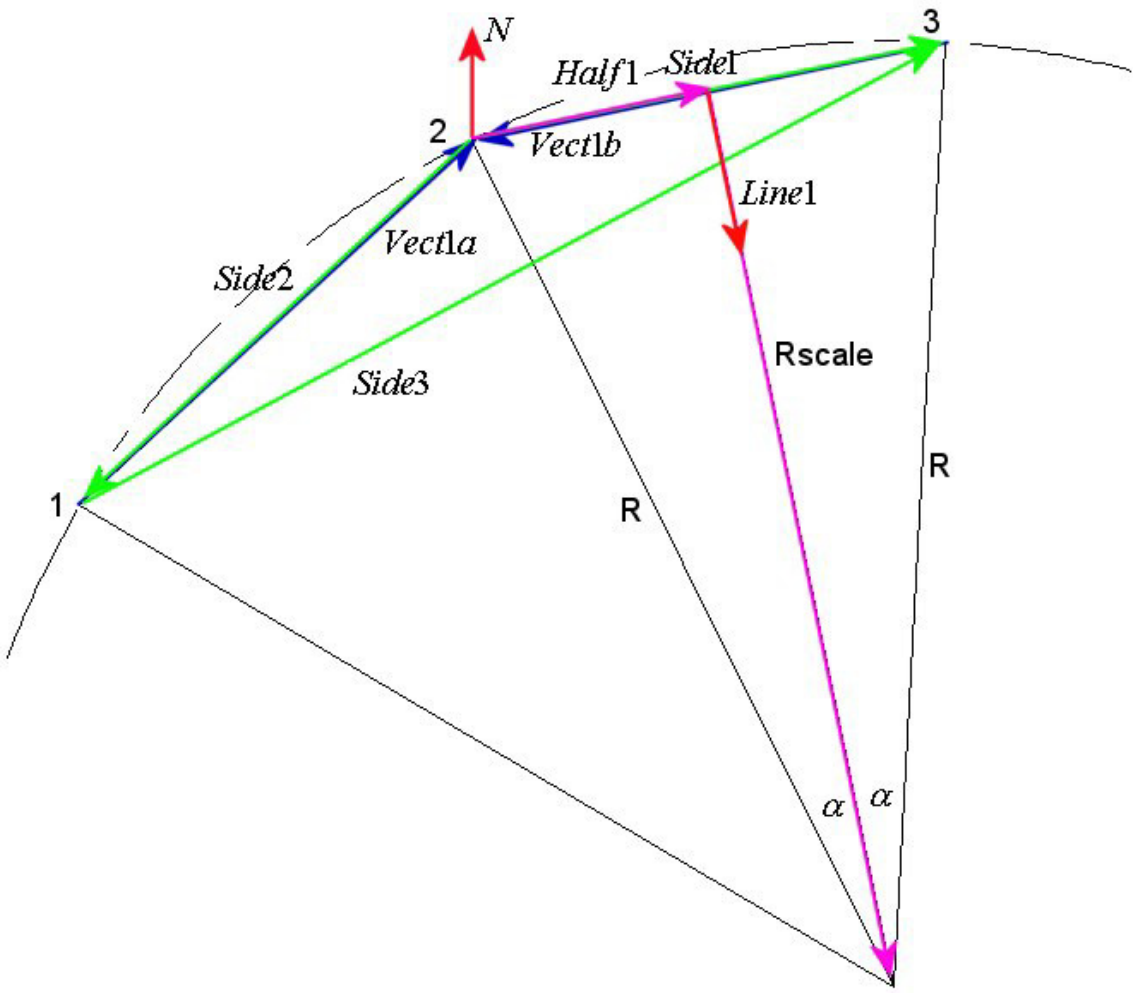


Figure 4.16 Circle Center Point Location

The direction of the *Rscale* is found based on the direction of the normal vector and the *Half1* vector, which are both normal to *Rscale*. Equation 4.10 yields a vector to orient *Rscale*.

Equation 4.10

$$Line1 = -Norm1 \times Half1$$

The final vector locating the center point is the vector addition of the vector from the zeroeth DH frame to the middle point, plus *Half1* and *Rscale* oriented along *Line1*, as described in Equation 4.11. *Line1* is reduced to a unit vector and then scaled by *Rscale* to provide the proper position and orientation.

Equation 4.11

$$P_{center} = PNT2 + Half1 + Rscale * \frac{Line1}{\sqrt{Line1 \bullet Line1'}}$$

The process to compute the center point is repeated until all data samples have been used and the results are averaged to find the final center points.

The center points of the four circles define two axes of revolution, namely the axes of joints 5 and 6. A small optimization routine is used to fit a generic plane to the four center points, included in appendix section 10.5.14 . This optimization routine uses the *fminunc* solver to find the best-fit plane to four points, where the plane is defined in Equation 4.12 [23]. The *fminunc* optimization routine is an unconstrained nonlinear function minimization routine included with the MATLAB optimization toolbox. In Equation 4.12, {A, B, C, D} are variables defining the orientation of the plane in 3-dimensional space.

Equation 4.12

$$Ax + By + Cz + D = 0$$

Once the planar coefficients are defined, the points are projected onto the plane a distance *D*, as defined in Equation 4.13 [23], along the negative normal vector of the plane. If {A,B,C} are not all zero, then the normal vector based on the planar formulation in Equation 4.12 is simply Equation 4.14 [23].

Equation 4.13

$$D_{off} = \frac{\left|Ax_1 + By_1 + Cz_1 - D\right|}{\sqrt{A^2 + B^2 + C^2}}$$

Equation 4.14

$$Normal = \langle A, B, C \rangle$$

The projection accounts for slight errors in the locations of the center points relative to the best-fit plane and is typically less than fifty micrometers in total distance. A coordinate frame is then defined based on the four planar points, as shown in Figure 4.17, and a transform is created between the OPTOTRAK frame and the newly created planar frame. The four planar points are then transformed into the planar frame, with

Equation 4.15, and the intersection between the two axes, formed by the four points, is calculated and shown as point *int* in Figure 4.17.

Equation 4.15

$$^{p\ln}P = T_{opt}^{p\ln\ opt}P$$

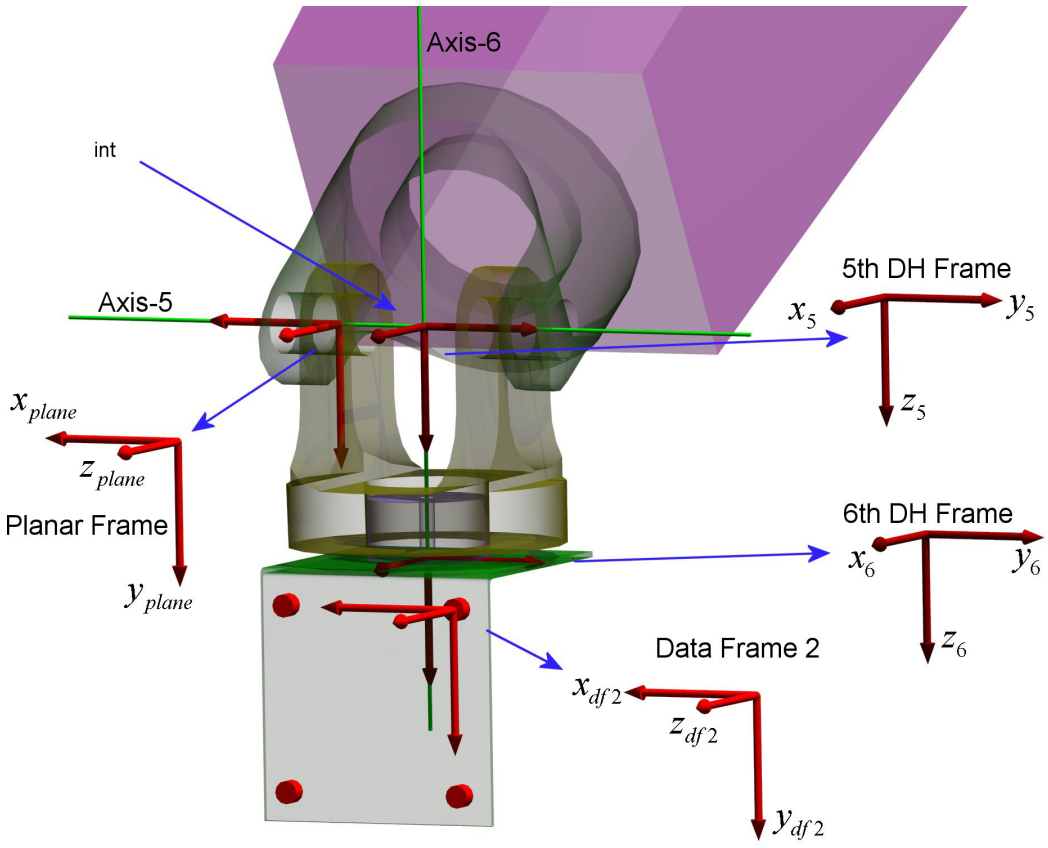


Figure 4.17 Planar Frame Assignment

In order to find the intersection, two lines are created with the four center points, corresponding to the two axes of revolution for the experimental measurements, denoted axis-5 and axis-6. The intersection point is calculated and then converted back into the OPTOTRAK frame, as in Equation 4.16.

Equation 4.16

$$^{opt}P_{\text{int}} = T_{p\ln}^{opt\ p\ln}P_{\text{int}}$$

A transform between the fifth DH frame and the OPTOTRAK frame is then built, based on the knowledge that the intersection point is coincident with the origin of the

fifth frame, and knowing how two of the axes are defined in OPTOTRAK coordinates. Combined with the transform between DH frames 5 and 6 a transform from the sixth DH frame to the OPTOTRAK frame is calculated, as shown in Equation 4.17.

Equation 4.17

$$T_6^{opt} = T_5^{opt} T_6^5$$

The four data sensors are then grouped into four groups of three sensors each and used to define independent coordinate systems. In Figure 4.17 only one of these four coordinate systems is shown, and is denoted DF2 (data frame 2). These coordinate systems are used to generate transformation matrices between the OPTOTRAK frame and each of the sensor frames. Using transformation algebra, the transforms between the sixth frame and each sensor frame are then computed as in Equation 4.18 for the second data frame. The code to create the transformations from the experimental data is included in appendix section 10.3.5 for the waist and 10.3.6 for the wrist.

Equation 4.18

$$T_{df2}^6 = T_{opt}^6 T_{df2}^{opt}$$

4.6 Gathering and Verifying Data

Once the method to relate sensor data to the DH frames is presented, the techniques used to gather data must be addressed. Furthermore, it is necessary to look at the steps that were taken to ensure the data was accurate and relatively error-free. All experimental data points were sampled at a rate of 20 Hz for ten seconds by the OPTOTRAK. Results were stored to sequentially numbered files and then imported into MATLAB. MATLAB reads each file sequentially, as demonstrated in the experimental data reduction code in appendix section 10.4.5 , from a single regime of testing, and performed several operations to ensure the data is consistent.

The first step in verifying the experimental data is to remove points that were blocked during the trial. A point can appear blocked due to several reasons. Sporadically, the OPTOTRAK sensors could get flooded with infrared light from the sun and cause an error that resulted in an apparently blocked sensor. This only occurred for a

few moments under good conditions and therefore did not affect many samples at a time. Certain other sensors were blocked because the arm of the robot obscured them from the view of the OPTOTRAK, or the sensor were turned in such a manner that it was not viewable. Care was taken during the experiment to ensure critical sensors were never obscured, however some optional sensors mounted to the arm were blocked much of the time. The optional sensors were only required during the stiffness trials. A blocked sensor returns a distance greater than 1E10 millimeters, and is easily removed and replaced since a nominal value is not more than 4000 to 5000 millimeters. The algorithm makes a decision based on the percentage of data that is bad due to a blocked sensor. If less than ten percent of the data is obscured, the mean of the remaining data is computed and the erroneous data points are replaced with the mean value. However, if more than ten percent is in error, the entire data set is set to zero allowing easy identification later in the code. Regardless of the decision, all modifications are written to a log file for reference.

The second data test involves looking at the spread of the data. As with most experimental data sets, outlier points may exist. Outlier points are particularly important to this research because the four hundred data samples are averaged to determine a single number for each sample set. An outlier has enough weight to pull this average away from the rest of the data significantly. Figure 4.18 shows an outlier relative to the rest of the data in the set, and Figure 4.19 shows a scaled version.

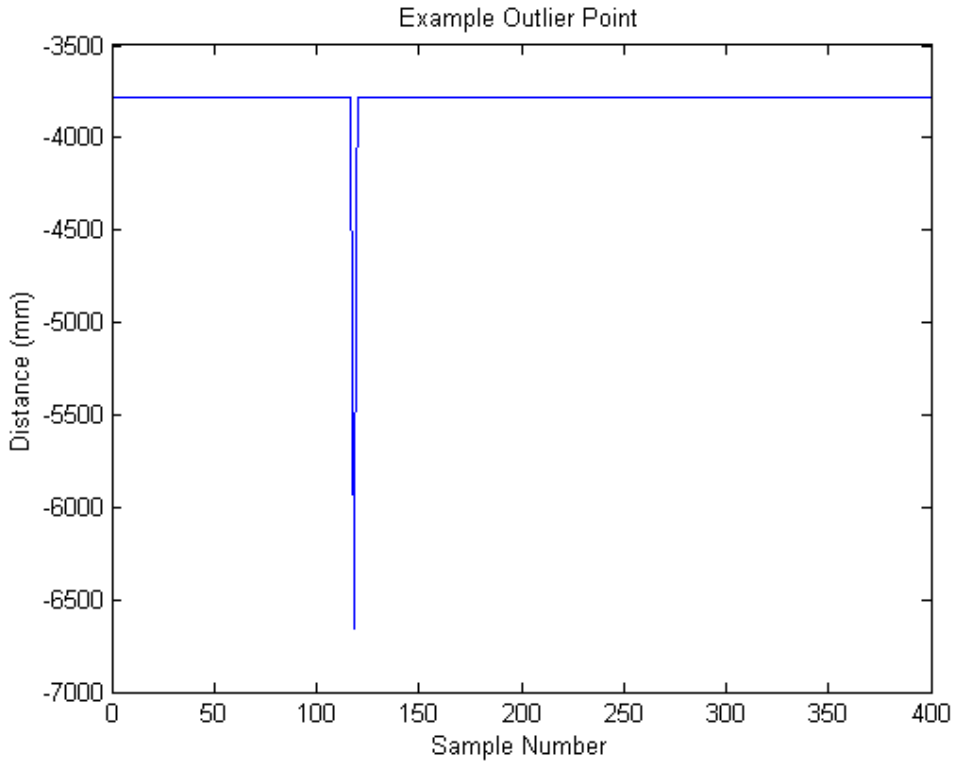


Figure 4.18 Example Outlier Point

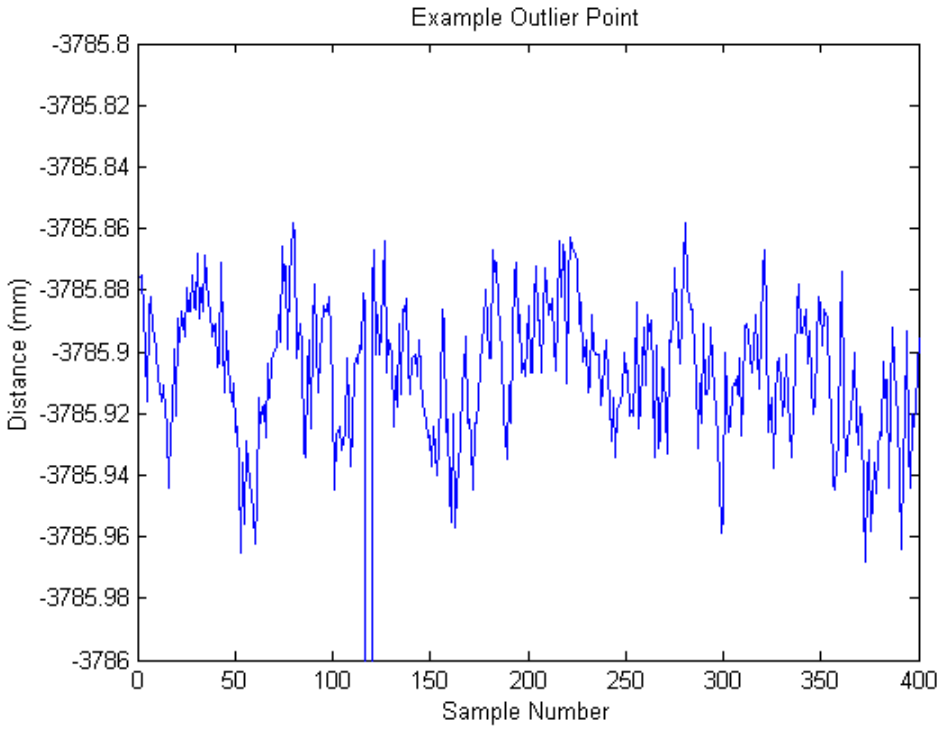


Figure 4.19 Zoomed Outlier Point

Without knowing which points are true outliers ahead of time, an iterative procedure is used to isolate the outliers. Initially, an average of the entire data set is calculated. The difference between each point and the average is computed and compared against a specified range. At the beginning, the range is set to one millimeter on either side of the average. Therefore, if all differences are less than one millimeter then no points are considered outliers. If more than ten percent of the data remains outside the specified range, then the range is increased by one half of a millimeter. If more than ten percent remains outside the new range, the range is increased again and the percentage outside checked until one of two conditions is met. The first condition is that more than ninety percent of the data falls inside the range, and therefore the remaining points are classified as outliers. The second condition occurs if the range passes six millimeters at which point an error flag is tripped and the data must be reviewed. For the data gathered during the experimentation, the second condition did not occur and a review was not required. Figure 4.20 though Figure 4.23 show several plots based on the above erroneous data set. Each progress plot shows the increased band as the algorithm attempts to capture ninety percent of the data. The final plot shows the corrected data set. Figure 4.18 shows a typical example of how far the average value can shift due to an outlier. The MATLAB code used to generate these plots is included in appendix section 10.5.7 .

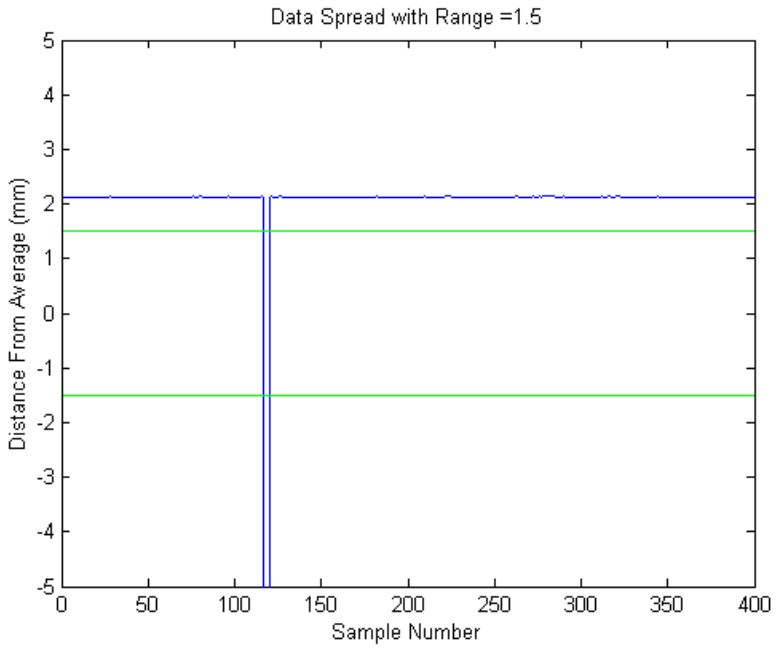


Figure 4.20 Data Spread with Range = 1.5 mm

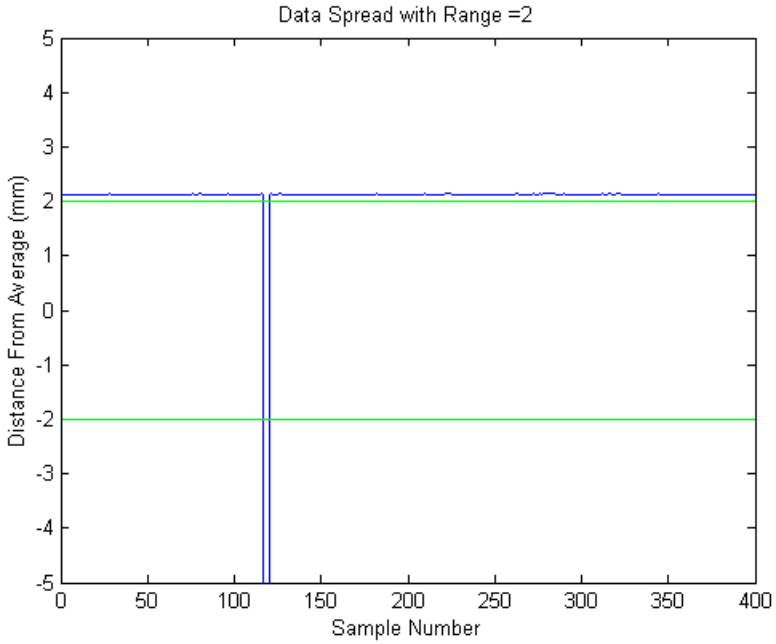


Figure 4.21 Data Spread with Range = 2 mm

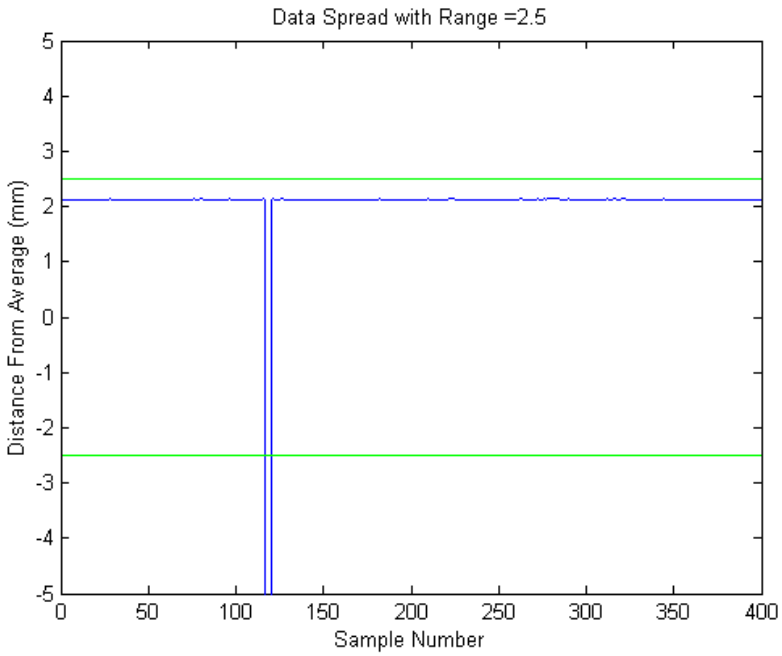


Figure 4.22 Data Spread with Range = 2.5 mm

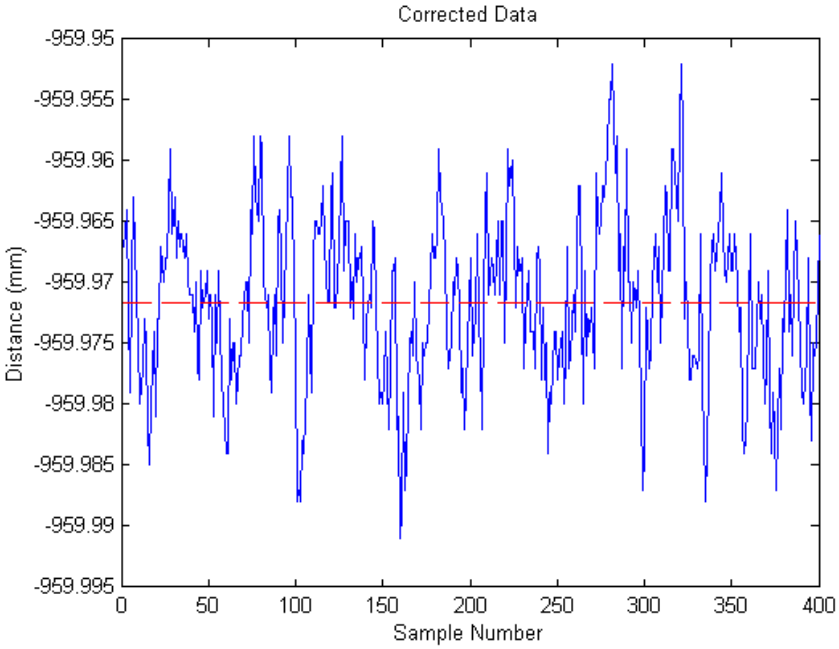


Figure 4.23 Corrected Data

4.7 Data Reduction

Each set of data contains the angle at which the manipulator was oriented, the load applied, and the resulting displacements and rotations. Both the results and the loads

need to be transformed for them to be used by the optimization routine. The following sections discuss how the results are compiled.

4.7.1 Applied Load Data

After the results from the OPTOTRAK have been checked for the errors mentioned above, the data from the sensors mounted to the load cables are used in combination with the applied load to solve for the X, Y and Z direction force components seen by the end-effector. The Cartesian load components in OPTOTRAK coordinates are based on unit vectors oriented along each cable, denoted *Lva* and *Lvb* for Load Vectors A and B, respectively. These two vectors are simply the difference in the positions of the sensors on each cable. Subtracting the bottom point from the top ensures the vectors consistently point downward. Each cable's Cartesian load components are normalized and then multiplied by half of the total load to find the X, Y and Z direction forces on the end-effector, as shown in Equation 4.19. Only half of the load is applied since it is distributed evenly between the two sides of the cable. The use of rollers and pulleys in joining the load cables together ensures the load is evenly balanced.

Equation 4.19

$$Fva = \frac{Load}{2}\frac{Lva}{\sqrt{(Lva \bullet Lva)}} \quad Fvb = \frac{Load}{2}\frac{Lvb}{\sqrt{(Lvb \bullet Lvb)}}$$

The total force is the sum of the loads from each cable. This result is in OPTOTRAK coordinates and is converted into the zero frame of robot coordinates by multiplying the load vector by the required rotation matrix, as in Equation 4.20. Conversion of the load into the zero frame ensures a simple operation to back out the joint torques since the Jacobian between the sixth frame (where the load is applied) and the zero frame is readily available.

Equation 4.20

$$^{0}LoadVect = R^{0}_{opt}\,{}^{opt}LoadVect$$

4.7.2 Position and Rotation Data

The next task for each set of data is to find the change in position and orientation of the end-effector due to the applied load. Initially, the absolute position and orientation for every load case are measured in the OPTOTRAK frame. After converting the results into the zero frame, as in Equation 4.21, the difference between loaded and unloaded cases are computed, as in Equation 4.22.

Equation 4.21

$$^{0}Data = T_{opt}^{0} \; ^{opt}Data$$

Equation 4.22

$$^{0}Data_{optim} = ^{0}Data_{Load} - ^{0}Data_{unloaded}$$

The position and orientation are based on the results from the four sensors attached to the end-effector whose data frames are detailed in Figure 4.24. All four data frames are attached to the same bracket and move through the same rotations and displacements. This redundancy provides higher confidence in the solution.

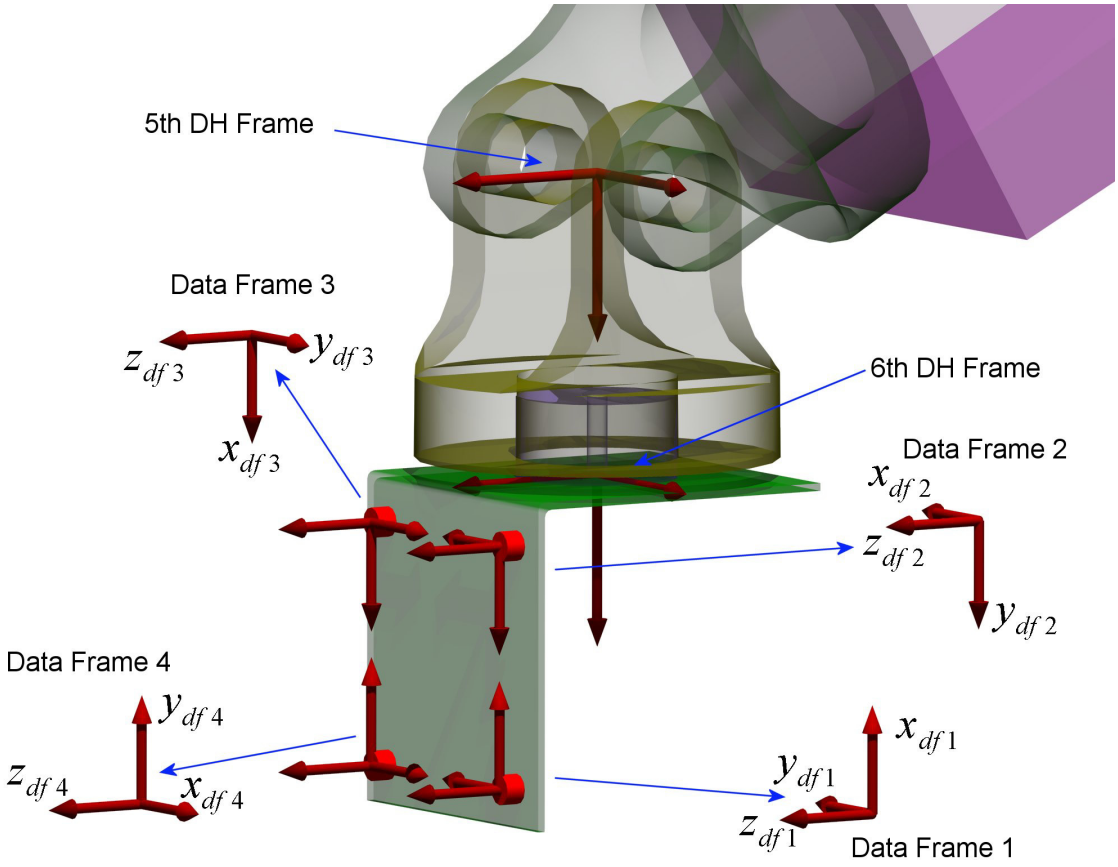


Figure 4.24 Data Frame Coordinate Frames

Each data frame coordinate system is normalized to one in all directions and then a transform from each data frame to the OPTOTRAK frame is built from the unit vectors of the coordinate system. Based on the results from the sensors mounted to the base, a base coordinate frame is built. A transform from the OPTOTRAK to the base frame is built based on unit vectors extracted from the coordinate frame. The base coordinate frame is shown in Figure 4.25 along with the zeroeth frame and the fictitious plane on which the base frame is located.

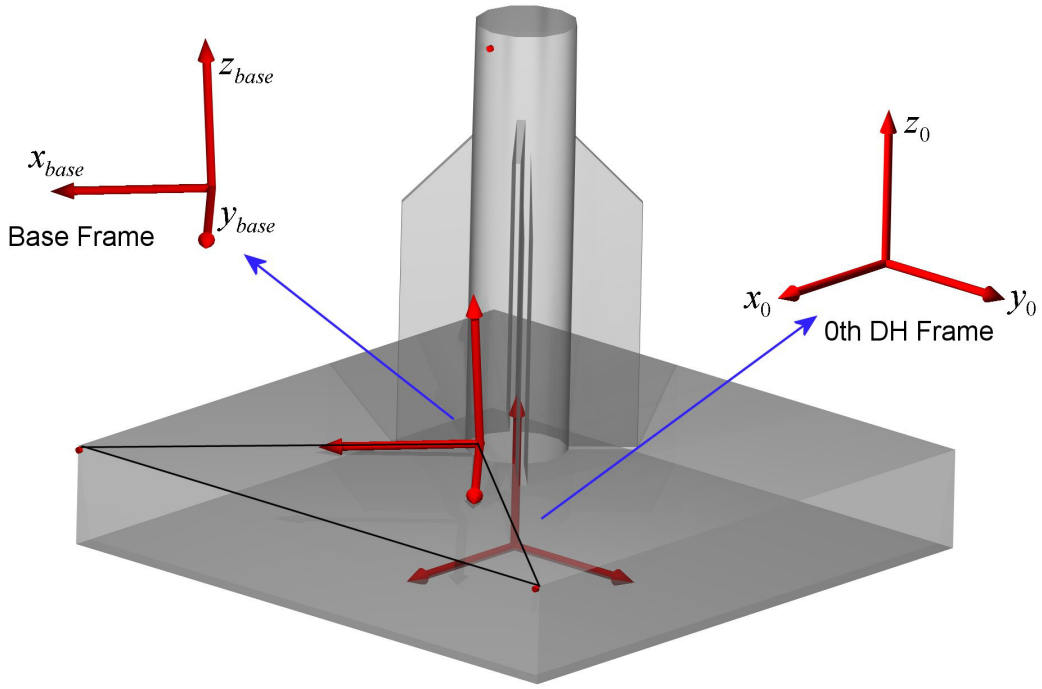


Figure 4.25 Base Coordinate System

To compare resulting deflections and rotations, all data must be in the same frame, which in this research is the zeroeth DH frame. A *6a* frame is created and defined as identical to the sixth frame under no load, but deflects and rotates relative to the sixth frame when loaded, as shown in Figure 4.26. The following three equations show the transform algebra to build the transform between the *6a* frame and the zero frame. Equation 4.23 creates a transform between the zeroeth DH and OPTOTRAK frames based on two previously calculated transforms. A set of transforms between the data frames and the OPTOTRAK are combined with the transforms from the sixth frame to the data frames to find where the sixth frame is in OPTOTRAK coordinates, as in Equation 4.24. Finally, combining the two previous results generates a transform between the undeflected sixth frame and the deflected *6a* frame, as in Equation 4.25. As shown in the equations, a *6a* frame is based on the knowledge of how the zeroeth DH frame and the OPTOTRAK frame relate as well as how the sixth DH frame and the OPTOTRAK frame relate. While this may initially seem inconsistent, the data frame and

original sixth DH frame are rigidly attached to each other. Therefore, even with a deflected manipulator, this relationship is still consistent.

Equation 4.23

$$T_0^{opt} = T_{base}^0 \left(T_{base}^{opt}\right)^{-1}$$

Equation 4.24

$$T_6^{opt} = T_{df}^{opt} \left(T_{df}^6\right)^{-1}$$

Equation 4.25

$$T_{6a}^0 = T_{opt}^0 T_6^{opt}$$

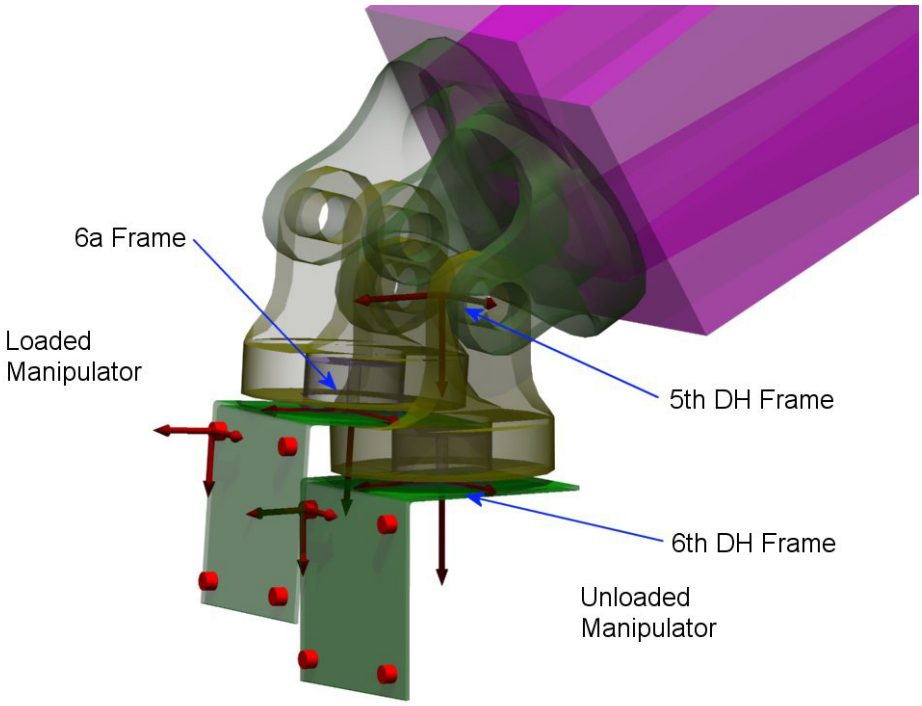


Figure 4.26 6a Frame Definition

Once the *6a* frame is calculated, the X, Y, and Z positions and rotations about each axis in the zero frame are extracted. The experimental tests include nine load cases at each position. In order to look at the change in position and orientation, the zero load results are subtracted from the loaded results from each of these tests. These results, along with the angles of the manipulator joints, and the load applied are then formatted and written to a file for the regression routine presented in Chapter 5.

Chapter 5: Optimization

The method chosen to find unknown joint stiffnesses could have relied on one of many different techniques. However, in order to solve the problem in a general way, without creating system specific solution techniques, it was important to elect a solution technique that was not dependant on a specific manipulator. Optimization provides a generalized, configurable framework to combine a model with experimental data and solve for unknown variables. Furthermore, optimization has the ability to solve a problem multiple times resulting in multiple solutions, and generate a qualitative value reflecting how well a specific solution satisfies the problem. However, optimization is very broad and as such requires selection of a specific technique to solve for the unknown joint stiffnesses.

5.1 Regression of Unknown Joint Stiffnesses

Based on a constant stiffness model for axial twist, lateral deflection and torsional modes, the total number of unknown stiffnesses is twenty-one. If twenty-seven data points are measured, each with nine loads, there will be sufficient data to solve for the unknown stiffnesses. Since each stiffness cannot practically be isolated and measured experimentally, the entire problem is solved with a regression-based solution method. The analysis method used to solve for the unknown stiffnesses given experimental load and deflection data can be based on several different techniques, which can be distinguished by the type of information required for the solution. The following brief discussion will provide justification for the ultimate choice for the optimization algorithm for this research.

The problem can be cast in the form of a regression objective function, where the goal is to minimize an objective function. The simplest method is to choose candidate values for the independent variable, such as the joint stiffnesses, and evaluate the objective function. The smallest value of the objective function is then chosen as the solution. This technique, which only relies on the value of the objective function, is known as a zeroeth order method [7]. There are various methods to select the candidate

independent variables, whether at random or by a Hooke and Jeeves minimization technique, for example.

To further increase the efficiency of an optimization, it is common to include gradient information, and then search in the negative gradient direction [7] for the minimum. This allows a more intelligent point selection when compared to zeroeth order methods. However, in computing the gradient, the first derivative of the objective function is required. Depending on the formulation of the objective function, the first derivative calculation may be difficult or impossible to calculate, and even then only attainable through numerical calculations rather than through closed form derivatives. These gradient based methods are known as first order optimization techniques and while each subsequent step takes a longer time, when compared to zeroeth order methods, they can provide more intelligent steps, ultimately leading to a faster optimization.

Efficiency increases beyond first order optimization are accomplished by including both the gradient and the Hessian matrix. This incorporates both first and second derivative information of the objective function, and is therefore known as second order optimization. As a result, second order methods can provide more accurate steps than either first or zeroeth order methods [7]. However, increased accuracy comes at the cost of increased computation at each step.

Second order optimization was chosen for the solution of joint stiffnesses, providing increased point selection accuracy by including gradient and Hessian information through numerical gradients. Furthermore, the computational increase when compared to zeroeth order methods is small since the individual solutions require tenths of a second to compute on the optimization computer. Based on second order techniques two different optimization algorithms are used; a function minimization and constraints algorithm and a minimization of maximum error algorithm.

Both implementations use built-in MATLAB functions to handle the optimization that numerically compute the gradient and Hessian matrices at each step. MATLAB functions, *fmincon* and *fminimax*, use a Sequential Quadratic Programming (SQP) solution technique to minimize the objective function. SQP solution techniques utilize Kuhn-Tucker equations as the basis and attempt to configure Lagrange multipliers

directly. Implementation of quasi-Newton methods "guarantee superlinear convergence by accumulating second order information" [24]. Furthermore, "SQP methods represent state-of-the-art nonlinear programming methods." Work by Schittowski has shown an SQP version that "outperforms every other tested method in terms of efficiency, accuracy, and percentage of successful solutions, over a large number of test problems" [24]. A complete discussion of the implementation of each algorithm is provided by Mathworks in [24]. At the basis of the optimization technique is the objective function.

5.2 The Objective Function

In order to implement the optimization, it is necessary to develop an objective function. The goal of the objective function is to increase in value with solutions farther from the experimental results or to decrease with solutions closer to the experimental results. The following section will describe the development of the objective function in a step-by-step manner, presenting details on the justification for each step.

Initially, the goal is to find an objective function that will be zero when the optimized solution matches the experimental solution exactly, and then increase in value as the optimized solution diverges. Equation 5.1 shows the initial form for the objective function where each variable is a three-by-one matrix relating to the position (*Pos*) or rotation (*Rot*) of the analytical (*Ana*) or experimental (*Exp*) results. The position and rotation contributions are broken out separately because they will have inherently different scales and this will allow weighting of one over the other if necessary. Each component, either position or rotation, is summed over its three components, that is, positions in the X-, Y- and Z-axes or rotation about said axes.

Equation 5.1

$$\sum_{1}^{3}\left(ExpPos - AnaPos\right) + \sum_{1}^{3}\left(ExpRot - AnaRot\right)$$

The above equation is unreliable since the numerical signs of the two values, experimental position and analytical position for example, may negate a negative and produce a faulty low value of the objective function. To compensate for this issue, the absolute value of each term is computed prior to finding the difference, as shown in Equation 5.2.

Equation 5.2

$$\sum_{1}^{3}\left(\left|ExpPos\right|-\left|AnaPos\right|\right)+\sum_{1}^{3}\left(\left|ExpRot\right|-\left|AnaRot\right|\right)$$

The value of the objective function is linearly dependent on the difference between experimental and analytical solutions at this point. It is prudent to increase this dependence to a higher order based on the error, such that as the analytical solution moves away from the experimental results, the objective function increases in value by a power law, as shown squared in Equation 5.3.

Equation 5.3

$$\sum_{1}^{3}\left(\left|ExpPos\right|-\left|AnaPos\right|\right)^{2}+\sum_{1}^{3}\left(\left|ExpRot\right|-\left|AnaRot\right|\right)^{2}$$

It is possible, and in fact preferable, for the difference between analytical and experimental data to be less than 1 millimeter or radian. In the case of this research the rotations are less than one one-thousandths of a radian typically. In such situations, the square of the result only gets smaller, and not larger as it should when squared. Therefore, each term will include an addition of one to the difference between the experimental and analytical results, as shown in Equation 5.4.

Equation 5.4

$$\sum_{1}^{3}\left(\left|ExpPos\right|-\left|AnaPos\right|+1\right)^{2}+\sum_{1}^{3}\left(\left|ExpRot\right|-\left|AnaRot\right|+1\right)^{2}$$

Similar to an issue earlier, it is possible now that the difference in the analytical and experimental results could be smaller than one and actually reduce the objective function below one again. This is compensated for by requiring the difference between analytical and experimental results to be positive, as in Equation 5.5.

Equation 5.5

$$\sum_{1}^{3}\left(\left\|ExpPos\right|-\left|AnaPos\right\|+1\right)^{2}+\sum_{1}^{3}\left(\left\|ExpRot\right|-\left|AnaRot\right\|+1\right)^{2}$$

In summary, this algorithm sums the errors in position and orientation after computing each one independently. In order to avoid situations where the numerical signs of the experimental and analytical contributions cancel each other out, the absolute

value of each term is computed, and the difference is then computed. Two versions are the objective function were used during optimization, corresponding to the full range of loads and a reduced range of loads. Both codes are included in appendix sections 10.5.4 and 10.5.10 .

5.2.1 Optimization Configuration Options

Besides calling the objective function, the optimization algorithm requires constraints and automatic termination settings. Both linear equalities and inequalities are left out of the solution, since none exist in the problem. All compliances are bounded to exist between 10^2 and 10^{20} $\frac{Nmm}{rad}$. These bounds keep the compliances from running into infinity unnecessarily and keep them away from zero where the manipulator would be extremely compliant and unrealistic. The upper bound may seem too high at first, but slight deflections can be detected with compliances of 10^{10} and the bounds should not limit the scope of the problem unnecessarily. As a result, the upper bound was set significantly higher than required. The optimization algorithm also requires a seed value for the design variables, the joint stiffnesses, to start the iterative solution. Both the maximum number of function evaluations and the maximum number of iterations are set high enough to allow the optimizer to find a minimum rather than quit as a result of a cycle limit. The MATLAB optimization functions accept a variety of options with the *optimset* command. Table 5.1 lists the options specified for the optimization algorithm, and a brief description of each option. The front-end optimization code is included for both full and reduced load range solutions in appendix section 10.5.5 and 10.5.11 , respectively.

Table 5.1 Optimization Settings

Option Name	Option Value	Description
LargeScale	Off	Disables large scale solvers
Tolfun	1E-10	Termination value for the objective function
TolX	1E-10	Termination value for X increment
MaxFunEvals	400000	Maximum number of function evaluations
MaxIter	100000	Maximum number of iterations
Diagnostics	on	Enable diagnostics

With the experimental setup as well as the optimization and theoretical progression covered, the following chapter will present comments on the collected data and then Chapter 7 will present results from the optimization.

Chapter 6: Calibration and Collected Data

Based on the model, experimental setup and solution techniques, it is necessary to describe how data is collected and reduced, and how initial experimental methods were employed to quantify stiffnesses.

6.1 Test and Calibration Experiments

In order to accurately model the manipulator, it was necessary to determine the order of the stiffnesses for each joint. For this research, it was assumed all joint stiffnesses share the same order. Early in the experimental work, two experiments were conducted to determine the order of the joint stiffnesses. Each of these experiments are presented with their results.

6.1.1 Stiffness Test 1

During initial testing, the robot was configured with the shoulder joint at ninety degrees and the elbow oriented at minus ninety degrees, that is, the arm was pointing straight up and the forearm was pointing directly forward, as shown in Figure 6.1. In this orientation, loading the end-effector produces a known moment about the elbow joint. Measuring the deflection under varying loads produces a stiffness that is closely approximated by a second order deflection curve, as shown in Figure 6.2. A second order deflection curve is consistent with a first order stiffness. Therefore, even though the deflection is second order, the stiffness is only first order resulting in two unknown variables per stiffness. A first order deflection curve computed a total cumulative error of 1.1×10^{-3} mm compared to a second order deflection curve error of 3.6×10^{-4} mm. Although improved, the second order deflection curves require a first order stiffness, increasing the computational difficulty of the model. This elbow joint deflection curve was modified to account for the shoulder rotations and a rolling motion of the base resulting in approximately 0.1 millimeters of total vertical motion. The standard deviation of the recorded data is 0.055 mm. This data spread is small enough to be

neglected when compared to the OPTOTRAK's stated accuracy of 0.1 mm. Data reduction code for the first stiffness test is included in appendix section 10.2.7 .

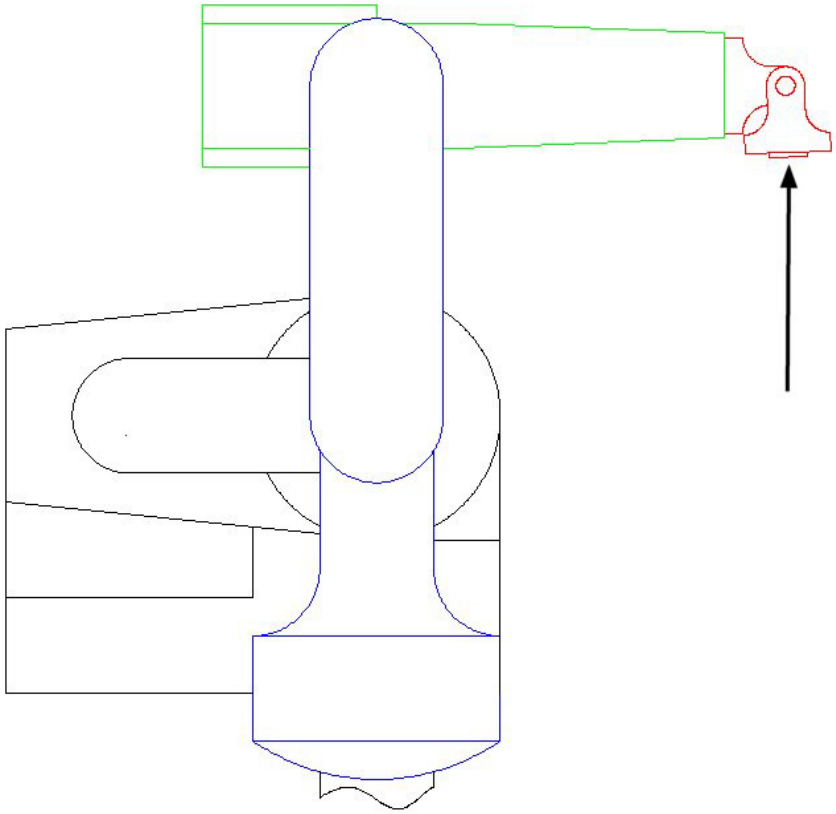


Figure 6.1 Stiffness Experiment #1

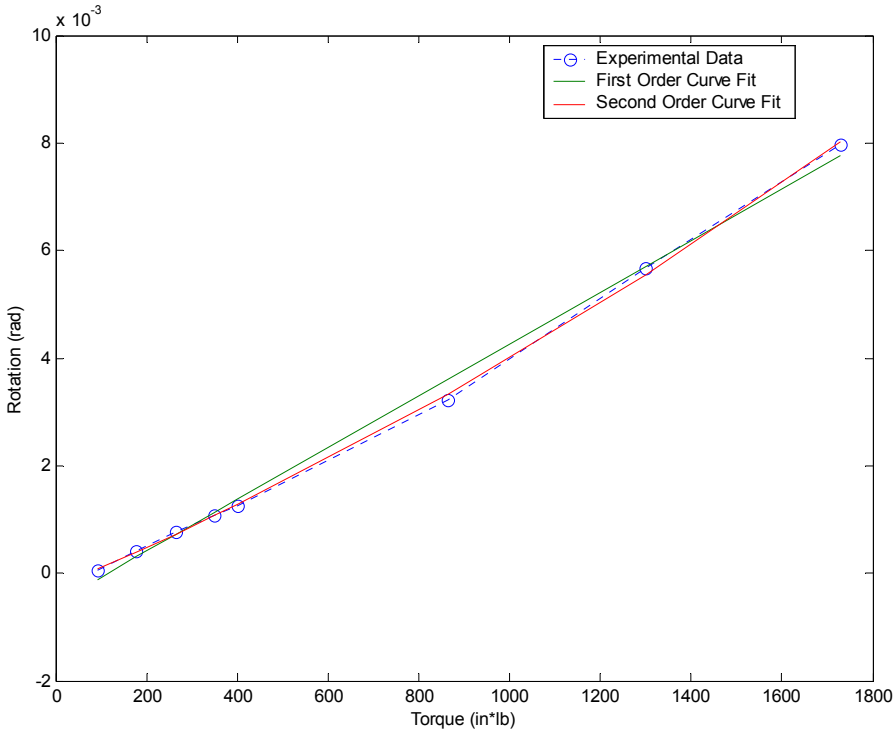


Figure 6.2 Stiffness Experiment #1 Results

6.1.2 Stiffness Test 2

In order to verify the joint order, the experiment was repeated on the shoulder joint. Extending the arm forward, as shown in Figure 6.3, the loads are again applied in the upward direction. The deflection due to the elbow joint and rolling of the base are subtracted from the total deflection. Using the modified deflection, the stiffness data closely fits a second order deflection curve, as shown in Figure 6.4. In this test, the first order deflection curve computed a total cumulative error of 9.4 x 10^{-4} mm compared to the second order deflection curve error of 8.6 x 10^{-4} mm.

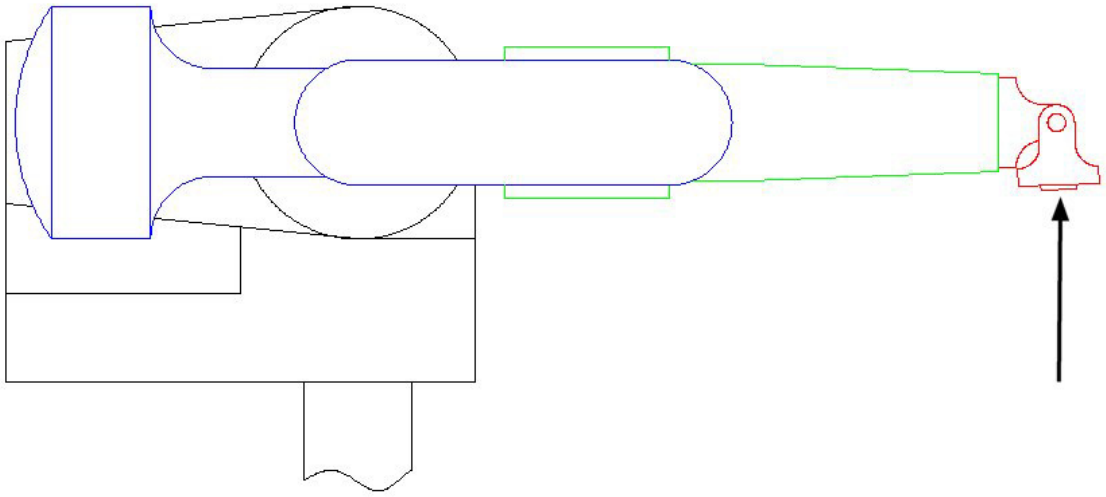


Figure 6.3 Stiffness Experiment #2

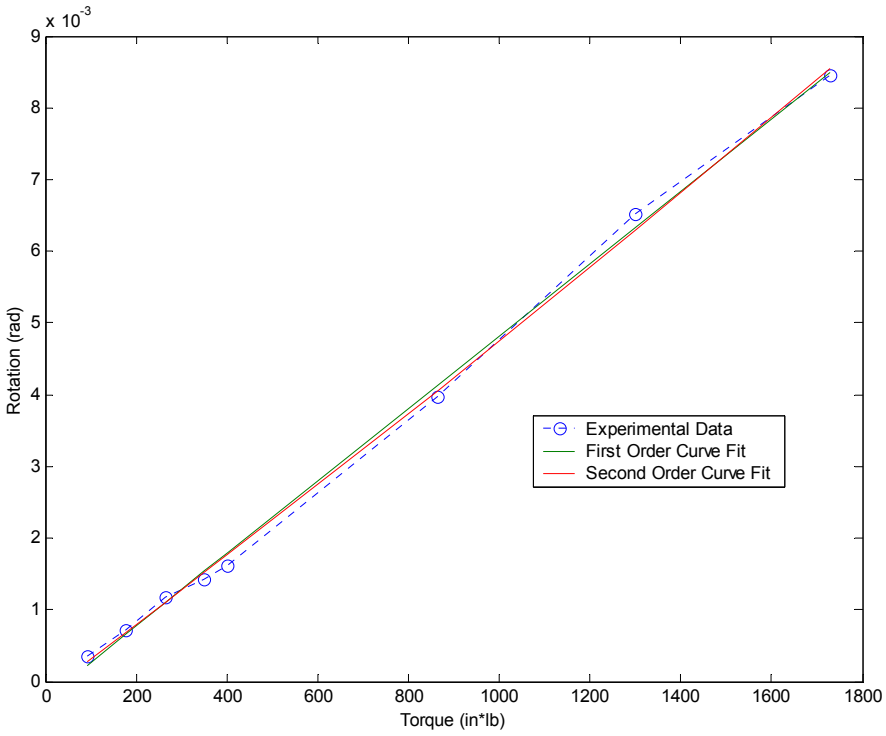


Figure 6.4 Stiffness Experiment #2 Results

The results from the two stiffness order characterization experiments appear similar including approximately the same maximum deflection. This is initially unintuitive since the second test has the arm extended and the shoulder joint feels a much

larger moment than during the initial experiment. However, the mechanical construction of the manipulator is such that the elbow joint is driven by a belt and several additional gear meshes when compared with the shoulder. The belt is susceptible to stretching and the multiple gear engagements allow increased backlash. Therefore, the stiffness of the elbow joint is best fit by a second order curve, and the shoulder joint is assumed to be of similar order, but significantly higher stiffness. Data reduction code for the second stiffness experiment is included in appendix section 10.2.8

Although the second order deflection curve models provide a marginally better fit to the experimental data, the compliant model was designed with the first order deflection curve. The zeroeth order stiffness, corresponding to a first order deflection, allowed for significantly easier computations than a first order stiffness as a result of the method used to compute the deflection under load. As mentioned previously, the deflected manipulator shape is determined by developing a rotation based on the applied torque and the assumed joint stiffness, as shown in Equation 6.1.

Equation 6.1

$$\theta = \frac{\tau}{K}$$

If the stiffness is constant, it is straightforward to solve for the stiffness given a torque and experimental deflection. However, as the order of the stiffness increases, it becomes a function of the angle, and the problem results in multiple roots as in Equation 6.2 for a first order stiffness.

Equation 6.2

$$\theta = \frac{\tau}{K_1\theta + K_2}$$

It is much easier computationally to solve the constant stiffness problem, and this provides for a good initial solution technique. It would be practical to go back into the model once the zeroeth order stiffness method was verified and increase the order of the problem. However, the computational increase is much more than simply selecting the correct root from the solution of Equation 6.2. For each additional stiffness value, the regression must generate gradients, and thus increasing the order of the stiffness from zeroeth to first order could increase the time required to optimize by an order of

magnitude or more. Regardless of the exact model used, there are certain experimental guidelines that aid in collecting error-free and consistent data as detailed below.

6.2 Experimental Techniques

Throughout experimental testing, several issues arose regarding the quality and consistency of the data. The method of loading the manipulator was directly coupled to the measured deflection. Since the desired application is composite lay-up, which requires a constant force, it was critical to eliminate any impact due to loading. With smaller weights, less than 40 lbs, this can be done manually; however, the heavier loads required an alternate to simply lifting and hanging the weight by hand on the end of the cable. The heavier weights are lifted with the aid of an hydraulic floor jack to minimize any impact from loading. This technique increased the repeatability of the data, and resulted in much smoother curves.

Similarly, when using the OPTOTRAK 3020, it was necessary to eliminate all motion of the measuring unit itself. The system should be mounted on a stable, motion free surface and any physical contact with the table should be avoided. The vibration due to air compressors or other machinery in the vicinity can be detected by the OPTOTRAK through the sensors mounted to the load cable. The standard deviation of the data was shown to increase an order of magnitude when a small air compressor was operating in the lab. The hysterisis of the return position after loading can be minimized to less than four percent total deflection by completely unloading the manipulator between tests. Completely unloading the manipulator involves removing both the applied weight and the counter balance from the cable assembly. Once the cable slackens, the counter balance can be reapplied and the next load can then be attached. While the loads applied through the cable may dominate the deflection for the heavier loads, the internal weight of the manipulator's arm will have a significant impact on the deflection for light loads.

6.3 Quality of Collected Data

Each data sample contains two hundred data points based on twenty seconds of measurement at ten hertz. It is necessary to investigate the distribution of the resulting data to ensure the data reduction techniques are appropriate. It was assumed the data

followed a normal distribution, which supports averaging the data and other statistical techniques. Figure 6.5 shows a sample of data taken during the third regime on sensor three in the Z direction on trial number 110.

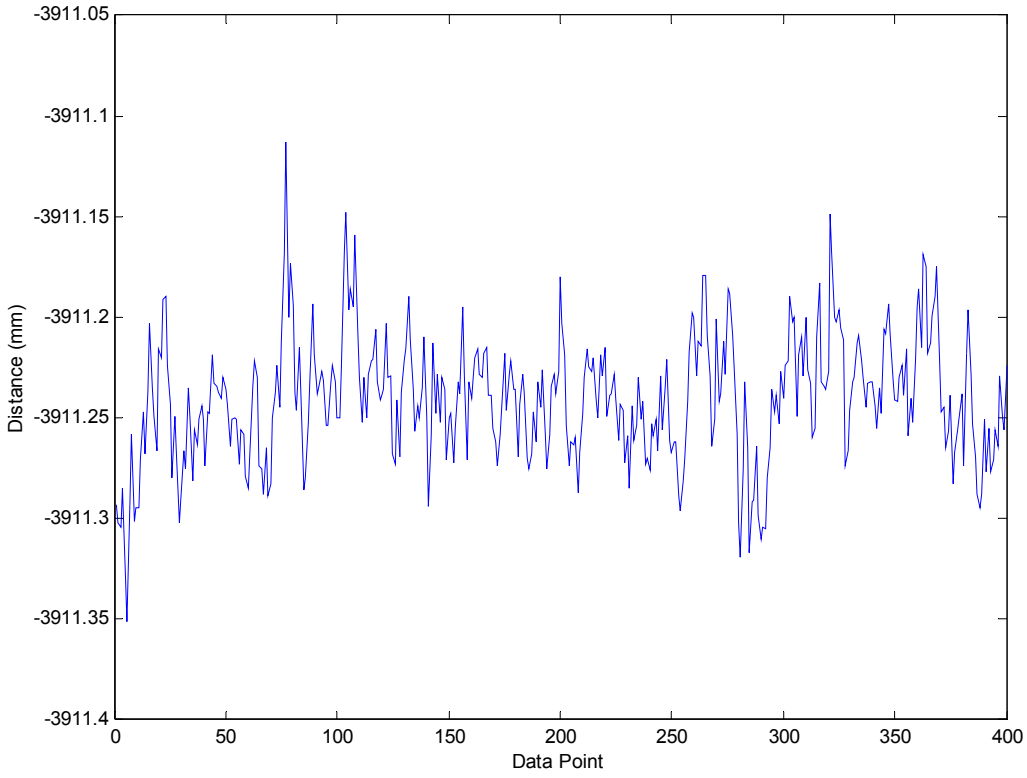


Figure 6.5 Raw Data for Sensor 3, in the Z-direction, during Test #110

It can be seen from the figure that the data is subject to some noise, however the vast majority of the data is between -3911.2 mm and -3911.3 mm, which is a range of 0.1 mm. This is consistent with the stated OPTOTRAK accuracy of 0.1 mm. Figure 6.6 shows the histogram for the above data set assuming 30 bins. Overlayed is the corresponding normal distribution curve. It is clear the data follows a normal distribution, lending confidence to the quality of the results. Histogram plot routines and computations were computed with code included in appendix section 10.4.3 .

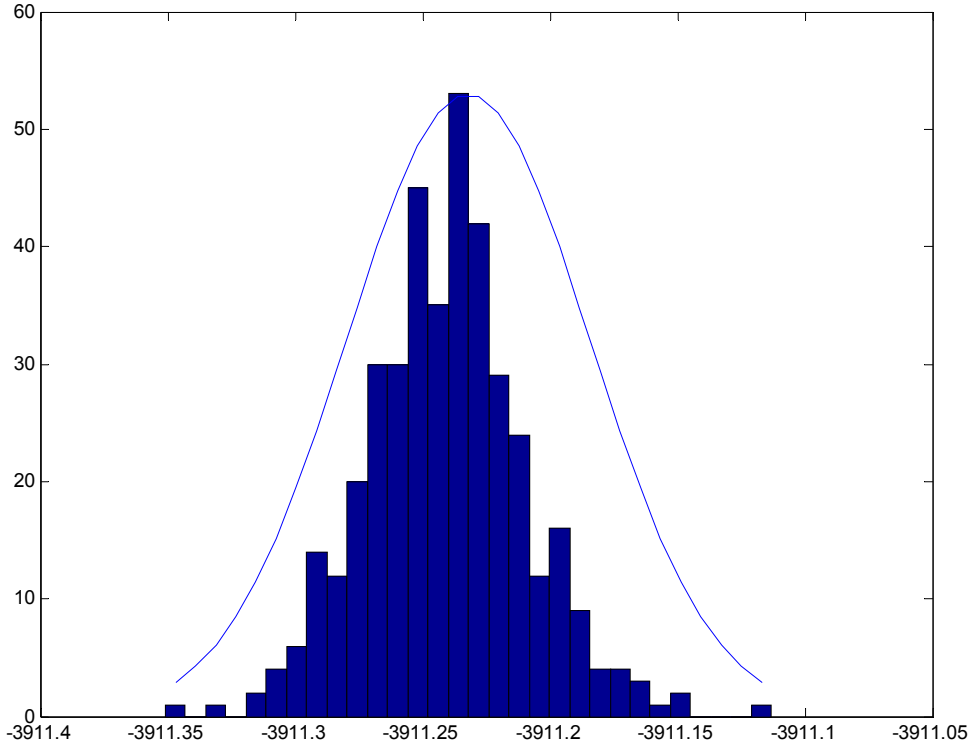


Figure 6.6 Histogram for Sensor 3, in the Z Direction, during Test #110

6.4 Center of Mass Experiment

The compliant deflection of the robot manipulator is based on all sources of loading including the internal loading due to the manipulator's own weight. As a result, it is necessary to investigate the effect of the location of the centers of mass since their locations are not explicitly known. This experiment is necessary to determine if the locations of the centers of mass have a significant effect on the manipulator's deflection. In the event the locations are critical, the variables locating the centers of mass would be included as design variables in the optimization. Although the robot was designed with counter weights to locate the center of mass approximately at the joints, it is possible in multiple configurations for the center of mass to shift away from the joint axis when subjected to end-effector loading. This displacement can change the computed compliance of the manipulator. In order to verify the effect of off-center link weight, the

computer compliance model was updated with the final joint stiffnesses from regime three, as shown in Table 6.1.

Table 6.1 Center of Mass Stiffness Matrix

N*mm/rad			
DH Coordinate Frame	X-Axis	Y-Axis	Z-Axis
0	1E7	1E7	1.3E9
1	1.8E8	7.8E8	1.2E8
2	5.5E7	1E7	7.1E7
3	4.4E8	7E7	8.2E6
4	1E7	1.5E7	4.0E6
5	9E6	4E6	1E7
6	1E7	1E7	1E7

The centers of mass are individually perturbed up to two and one-half inches to either side of the joint axes and the resulting change in end-effector location is calculated. The main body of the robot weighs 1281 N (288 lbs), while the upper arm weighs 1068 N (240 lbs), the forearm weighs 400 N (90 lbs) and the wrist weighs 111 N (25 lbs). The location of the main body mass has no effect on the end-effector position since it rests only on the supporting column which is assumed stiff in the model. Furthermore, the location of the wrist weight is small in comparison to the weight of the arm and therefore is not tested individually. As a result, only the weights of the upper arm and forearm will be checked for location sensitivity. First, the mass of the upper arm is perturbed, and then the mass of the forearm is perturbed. Each mass is tested separately and the maximum deflections relative to the center location over the range are shown in Table 6.2, while the solution data is displayed in Figure 6.7 and Figure 6.8. Since the maximum computed difference in end-effector position is 0.019 mm and the accuracy of the OPTOTRAK is 0.1 mm, the effect of the location of the centers of mass is negligible. Consequently, the problem will not require additional design variables to locate the centers of mass.

Table 6.2 Center of Mass Test Results

All units: mm			
	X Deflection	Y Deflection	Z Deflection
Weight 2	2E-3	5E-5	1.9E-2
Weight 3	2E-3	3.5E-5	1.4E-2

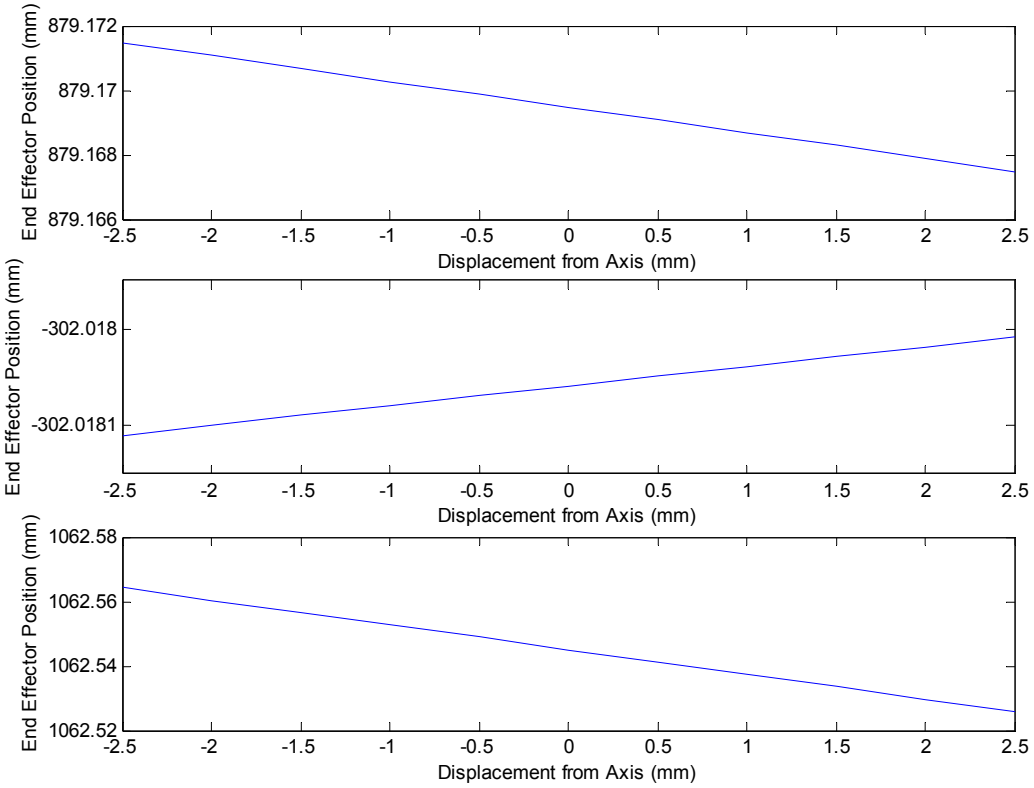


Figure 6.7 Sensitivity to Link 2 Center of Mass Location

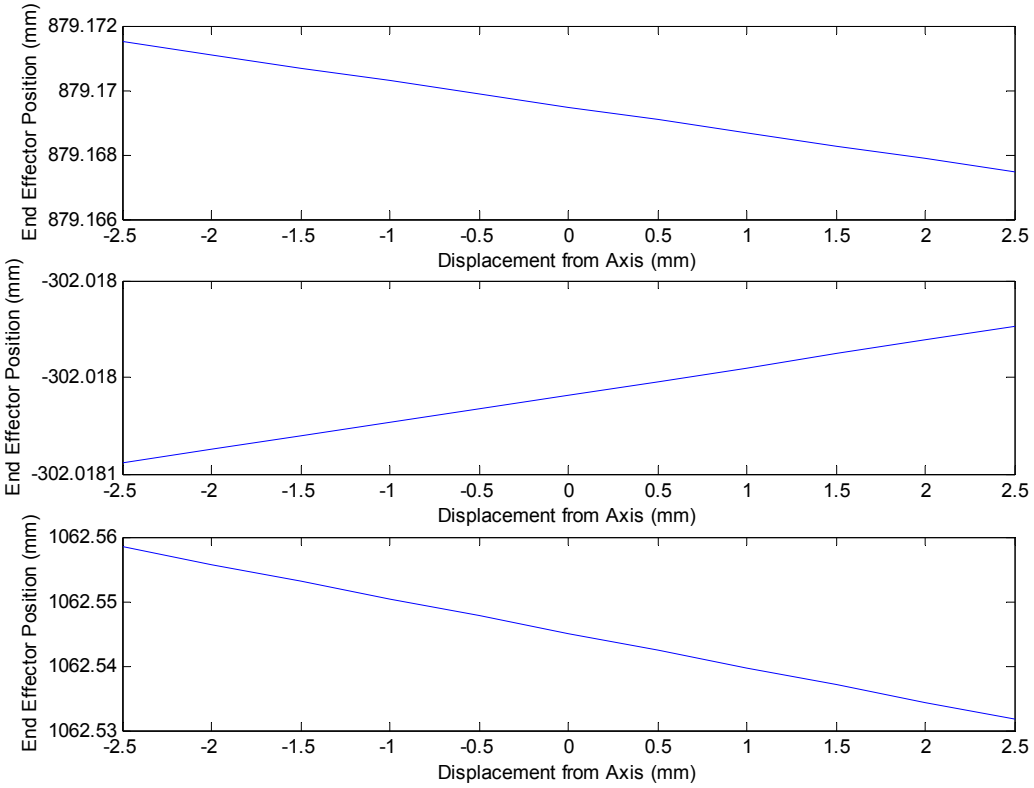


Figure 6.8 Sensitivity to Link 3 Center of Mass Location

6.5 Elbow-Up Versus Elbow-Down Experiment

A separate test was run to investigate the differences between elbow-up and elbow-down configurations of the robot. This experiment verifies the different compliance in an elbow-up and elbow-down configuration. Due to different stiffnesses between the elbow and shoulder joints, the two closures of the arm result in different deflections. The robot was aligned with the arm pointing forward, the forearm rotated down forty-five degrees and the wrist rotated another forty-five degrees so the tool plate was facing the floor. Each manipulator position is shown in Figure 6.10. Using the same technique as the stiffnesses experiments, the robot end-effector was loaded with each of the eight loads and the motion of the end-effector measured at each step. Figure 6.9 shows the deflections in each coordinate direction for the elbow-up (red dashed lines) and elbow-down (solid blue lines) cases respectively.

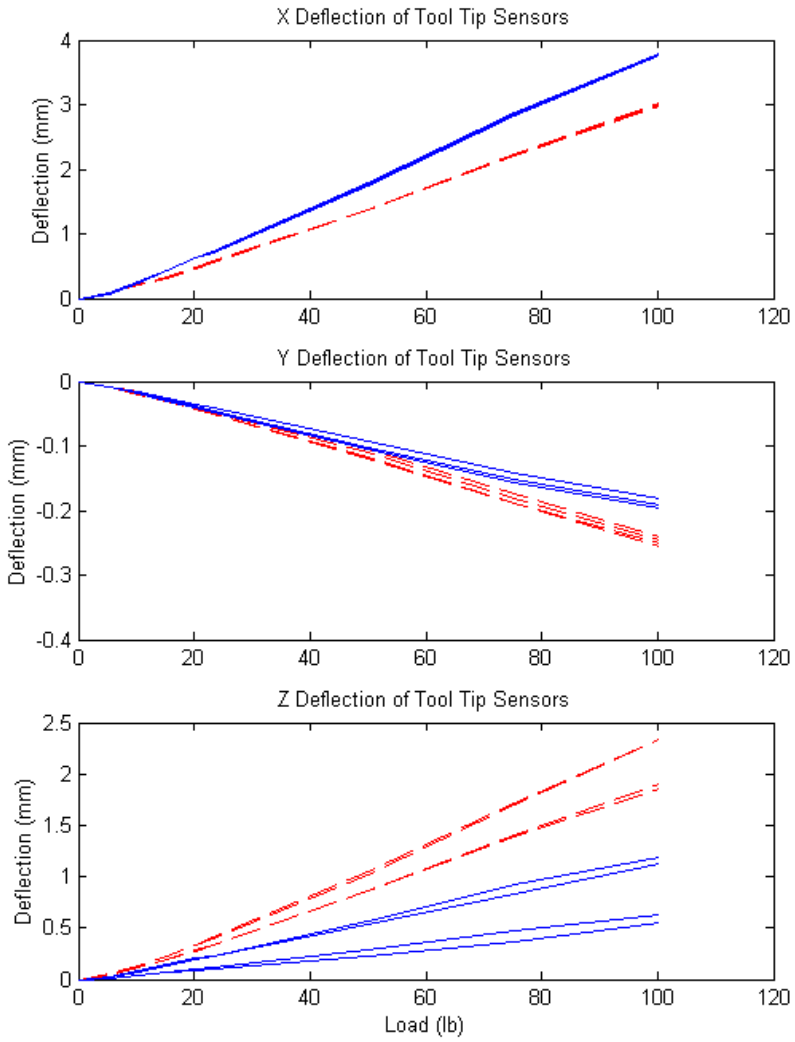


Figure 6.9 Elbow-Up versus Elbow-Down Deflection Results

As shown in Figure 6.9, the differences in elbow orientation cause significantly different deflections. The most significant changes are evident in the Z direction, where the elbow-up has a maximum deflection of nearly 2.4 mm, compared to approximately 1.3 mm for the elbow-down configuration. This difference was the result of the moment arm to the elbow joint orientation with regard to the load. For the elbow-up case, the forearm was aligned at an angle of forty-five degrees below the horizontal, resulting in a rotation about the joint that brought the end-effector away from the robot. In the elbow-down case, the forearm was level with the floor, and when deflecting at joint 3 actually

moved back towards the robot, however a slight deflection in the shoulder joint resulted in a net motion away from the robot. The elbow-up versus elbow-down data reduction codes are included in appendix sections 10.2.5 and 10.2.6 and the modified torque code and plotting routines are included in 10.2.1 and 10.2.2 respectively.

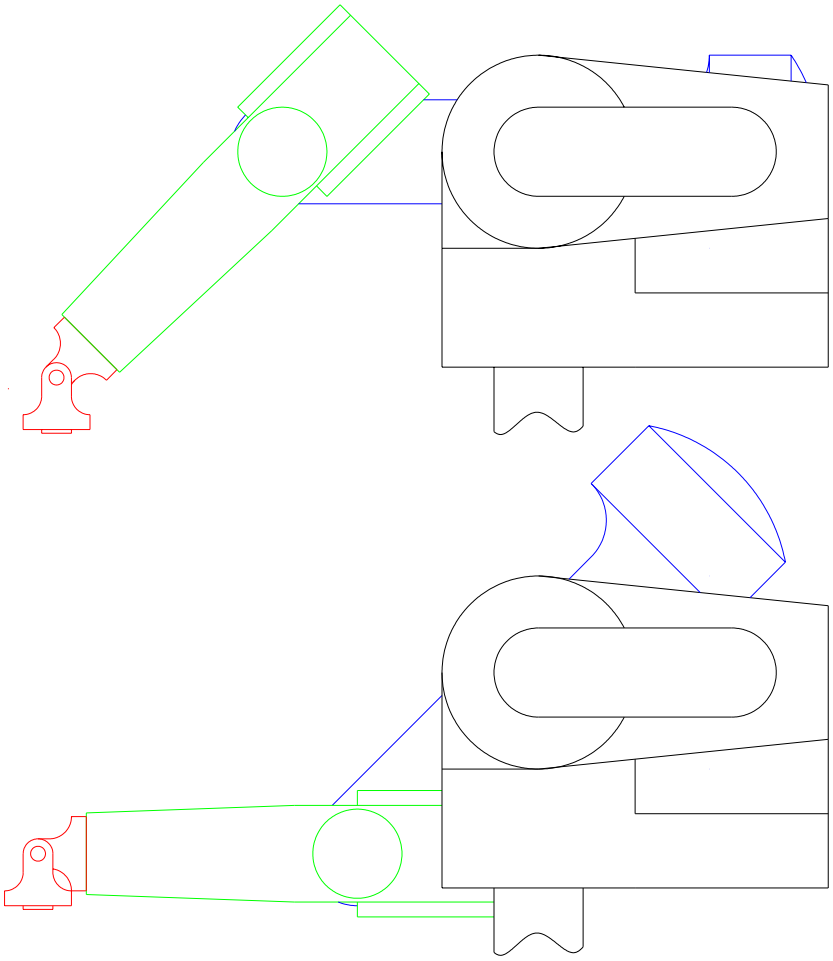


Figure 6.10 Elbow-Up and Elbow-Down Manipulator Positions

6.6 Experimental Results

Experimentally, all results were recorded as position in the OPTOTRAK's workspace. The displacement vectors and rotation angles were then extracted from the raw data and used by the optimization. Figure 6.11 shows an example of the displacement information for the third regime. The first third of the experiments were subjected to a load in the Z-dir, while the middle and last thirds were subjected to loads in the X-dir and Y-dir respectively. This data shows an obvious correlation with the load

that is applied. Figure 6.12 shows the rotations developed for the same data set. While the displacements show good correlation, the rotations fail to show a trend with the applied load. The rotations about the Z-Axis show the closest pattern to that which was expected, but still fail to consistently predict the orientation. Rotation and displacement plots were generated with MATLAB code included in appendix section 0.

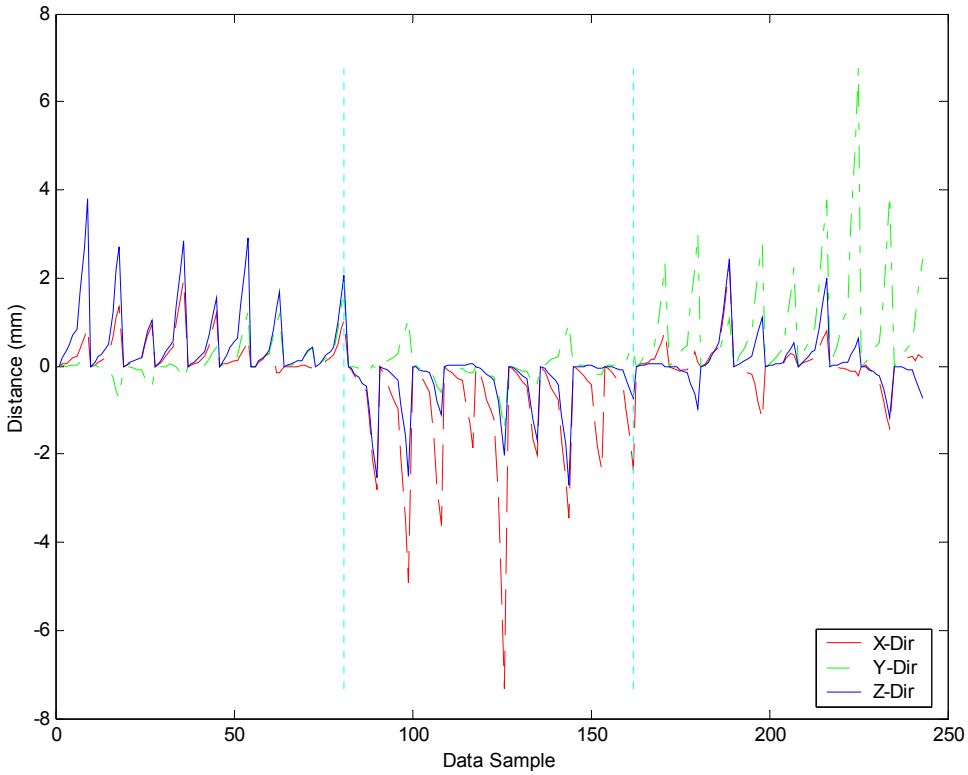


Figure 6.11 Raw Displacement Data

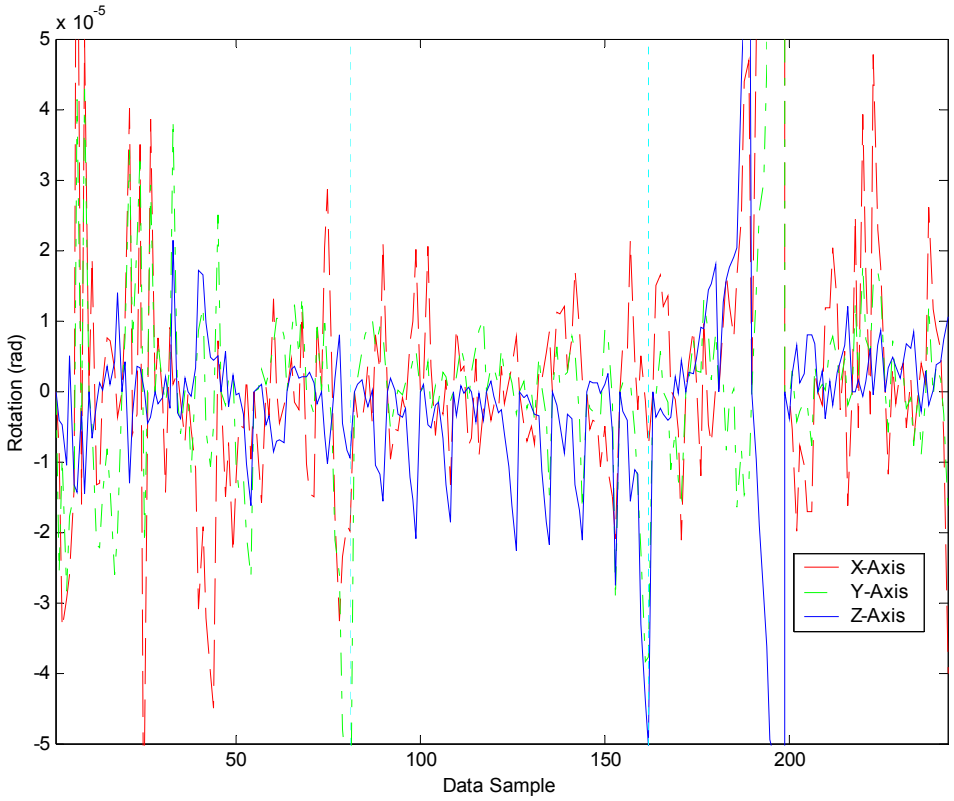


Figure 6.12 Raw Rotation Data

The displacement information shows deflections up to three or four millimeters on average. However, the rotations are only twenty or thirty microradians, or roughly one to two millidegrees. Since the OPTOTRAK is accurate to 0.1 millimeters, the validity of the rotations needs to be verified. The rotations are extracted from the translations of the four sensors on the end-effector. These sensors are nearly eighty millimeters apart and an average of seventy-two millimeters from the center of rotation, the axis of joint 6. Based on a tangential motion of the sensor a distance of 0.1 millimeters, the minimum resolvable rotation is 1.39E-3 radians. This minimum threshold is two orders of magnitude larger than the experimentally derived rotations. As a result, the experimental rotations are nearly zero and will be assumed as such for some optimized solutions.

Chapter 7: Optimization Results

The initial optimization technique assumed the displacement and rotation of the manipulator increased and decreased together as a function of the load. That is, if the displacement matched experimental results very well, the rotations should also be reasonable. Based on this assumption, the following results are driven primarily by the displacement in the optimization. While the rotation is included, it was not magnified to carry an equal weight. Further testing is included at the end of the chapter to highlight the results when the rotations are weighted comparable to the deflections.

7.1 Manual Optimization

As with many optimization problems, selection of initial conditions can be difficult. However, this choice is very important, since a start point relatively far from a minimum takes much longer to converge than one close to a minimum. In order to help minimize the iterations necessary for the optimization to find a minimum, a manual optimization technique was used to help choose intelligent start conditions. The basic method involves assuming a stiffness matrix and computing the resulting manipulator deflection. Rather than looking only at the objective function value, a set of plots are computed for each iteration, showing how well the assumed model performs in comparison to the experimental results. Plotting the experimental data and the assumed solution together allows a very quick visual inspection to quantify any correlation in the data. After changing a single variable in the stiffness matrix, a new set of plots is calculated. A few more steps and a rough gradient is easily developed for the variable of interest. The variable is reset after noting the best value, and this procedure is repeated for each variable allowing a rough gradient for the stiffness matrix to be developed. This technique was applied separately to each regime of data prior to initializing the full optimization solver to try to locate good initial conditions. An alternate benefit of this technique is the ability to determine which stiffness terms are sensitive to the applied load cases. For this discussion, only the manual optimization for the third regime will be

presented. The first and second regimes follow a similar method, but have fewer terms influencing the behavior of the model.

The first step is to find a stiffness high enough to allow no significant deflection in the manipulator. This stiffness is used to hold all joint deflections small while a single term is weakened. For the case of the third regime, a selected stiffness of 1E10 Nmm/rad was chosen. The resulting deflections are shown in Figure 7.1, Figure 7.2, and Figure 7.3, corresponding to the X-, Y-, and Z-direction displacements of the end-effector. Deflections are calculated in each of the three coordinate directions, viewed, and quantified individually. The vertical lines on the plots break the results into three different regions, relating to the areas of the workspace or the loading that was applied. This is a very important step, as it allows a good understanding of how the different stiffness terms relate to the motion of the end-effector. The top plots for each figure show the absolute difference between the experimental motion and the model's predicted motion. Each line is a test point in the workspace, and the magnitude of the data is the absolute distance between the experimental and optimized data sets. The second plot for each figure shows the experimental versus assumed solution as a function of the experimental testing trial. The twenty-seven load sets can easily be detected in the experimental results, whereas the assumed solution is stiff enough, at 1E10 Nmm/rad, to show no noticeable deflection. MATLAB code included in appendix section 10.5.3 was used for full load cases and code in 10.5.9 was used for reduced load set cases for manual optimization.

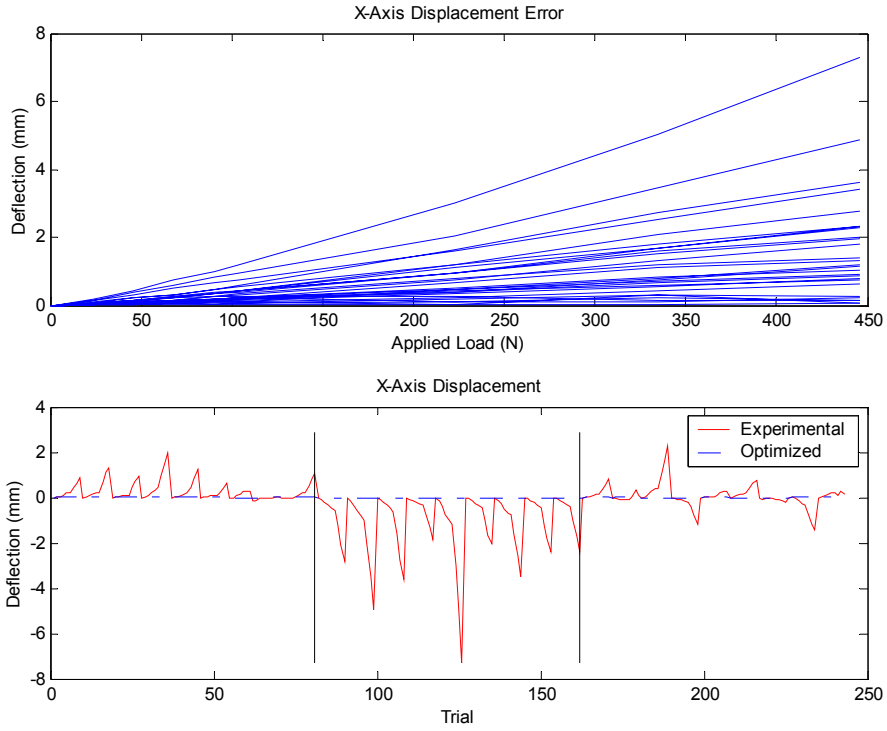


Figure 7.1 X Direction End-Effector Deflection

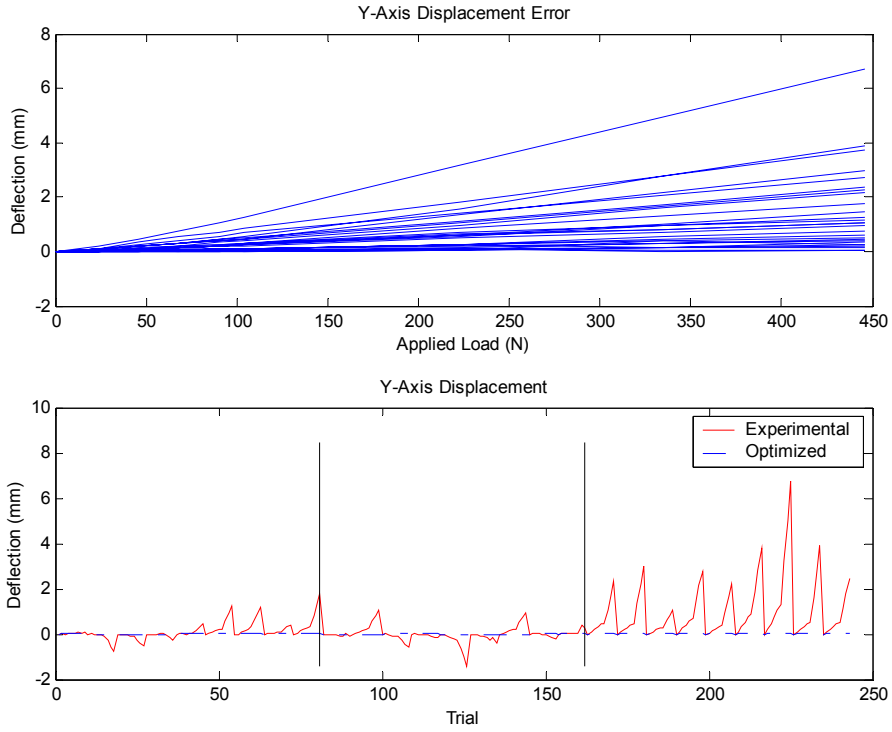


Figure 7.2 Y Direction End-Effector Deflection

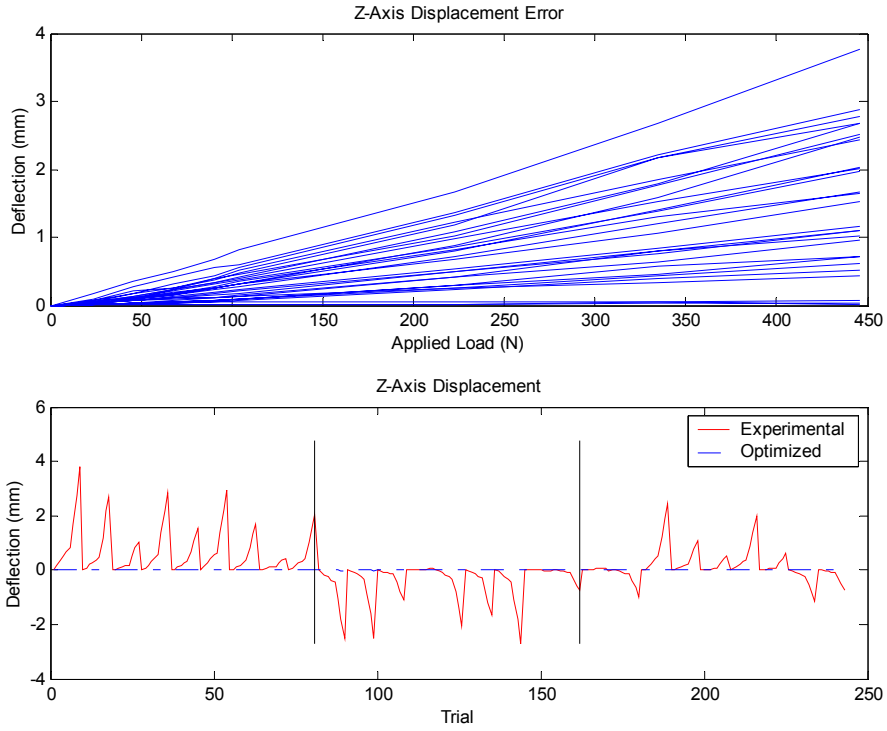


Figure 7.3 Z Direction End-Effector Deflection

The stiffness matrix contains twenty-one terms, corresponding to seven coordinate frames combined with the X-, Y- and Z-axes for each. Four of these terms are not represented in the model and are therefore set to an arbitrarily high stiffness, in this case 1E15 Nmm/rad. These four stiffnesses are the X and Y rotations about the zeroeth frame, the second frame's Y rotation and the sixth frame's Z rotation. These anomalies are fully investigated in Section 3.5 . Stepping through the matrix, one term at a time, the stiffness is lowered by an order of magnitude or two and the resulting plots are quantified as *No Effect*, *Improve*, *Too Soft*, or some other nomenclature to indicate where improvements exist. The results of the test runs for the third regime are presented in Table 7.1. The leftmost columns define the *DH Frame* and the corresponding *Axis* about which the stiffness was reduced. The *Direction* column specifies which axes the results are along, that is, an X in *Direction* corresponds to deflections in the X direction of the specified *Axis* in the specified *DH Frame*.

Table 7.1 Results from Third Regime Manual Optimization

DH Frame	Axis	Direction	Stiffness Nmm/rad			
			1.00E+10	1.00E+08	1.00E+07	1.00E+06
1	X	X	No Effect	No Effect		Too Soft
		Y	No Effect	No Effect		Too Soft
		Z	No Effect	**Slight Improve**		**Too Soft**
2	X	X	No Effect	Flip-Flop	Flip-Flop	Too Soft
		Y	No Effect	Flip-Flop	Flip-Flop	Too Soft
		Z	No Effect	No Effect	No Effect	Flip-Flop
3	X	X	*No Effect*	*Flip-Flop*		*Flip-Flop*
		Y	*No Effect*	*Flip-Flop*		*Flip-Flop*
		Z	*No Effect*	*No Effect*		*No Effect*
4	X	X	*No Effect*	*No Effect*		*Flip-Flop*
		Y	*No Effect*	*No Effect*		*No Effect*
		Z	*No Effect*	*No Effect*		*No Effect*
5	X	X	*No Effect*	*No Effect*		*Flip-Flop*
		Y	*No Effect*	*No Effect*		*No Effect*
		Z	*No Effect*	*No Effect*		*No Effect*
6	X	X	*No Effect*	*No Effect*		*No Effect*
		Y	*No Effect*	*No Effect*		*No Effect*
		Z	*No Effect*	*No Effect*		*No Effect*
1	Y	X	*No Effect*	*Flip-Flop*		
		Y	*No Effect*	*Flip-Flop*		
		Z	*No Effect*	*No Effect*		
3	Y	X	No Effect	**Slight Improve**		**Too Soft**
		Y	No Effect	**Slight Improve**		**Too Soft**
		Z	No Effect	**Slight Improve**		**Too Soft**
4	Y	X	*No Effect*	*No Effect*		*Flip-Flop*
		Y	*No Effect*	*No Effect*		*Flip-Flop*
		Z	*No Effect*	*No Effect*		*No Effect*
5	Y	X	*No Effect*	*No Effect*		*Flip-Flop*
		Y	*No Effect*	*No Effect*		*Flip-Flop*
		Z	*No Effect*	*No Effect*		*No Effect*
6	Y	X	*No Effect*	*No Effect*		*No Effect*
		Y	*No Effect*	*No Effect*		*No Effect*
		Z	*No Effect*	*No Effect*		*No Effect*
0	Z	X	*No Effect*	*Flip-Flop*	*Too Soft*	
		Y	*No Effect*	*Flip-Flop*	*Too Soft*	
		Z	*No Effect*	*No Effect*	*No Effect*	
1	Z	X	No Effect	**Slight Improve**	Flip-Flop	
		Y	No Effect	**Slight Improve**	Flip-Flop	
		Z	No Effect	**Improve**	**Too Soft**	
2	Z	X	No Effect	**Improve**	**Too Soft**	
		Y	No Effect	**Improve**	**Too Soft**	
		Z	No Effect	**Improve**	**Too Soft**	
3	Z	X	*No Effect*	*No Effect*		*Flip-Flop*
		Y	*No Effect*	*No Effect*		*Flip-Flop*
		Z	*No Effect*	*No Effect*		*No Effect*

4	Z	X	No Effect	No Effect	Slight Improve
		Y	No Effect	No Effect	Slight Improve
		Z	No Effect	No Effect	No Effect
5	Z	X	No Effect	No Effect	No Effect
		Y	No Effect	No Effect	No Effect
		Z	No Effect	No Effect	No Effect

An improvement is quantified as a stiffness change that results in an improved correlation between the experimental and assumed solution. Once an improved stiffness is further reduced, the solution typically shows much larger deflections than the experimental results, resulting in a *Too Soft* rating. A *Flip-Flop* is a result where some trials show improvement but other trials show deflection in the wrong direction. This type of result likely shows a stiffness that needs to be less than 1E10 Nmm/rad but tends to cause questionable results when softened independently.

The results in Table 7.1 are highlighted to illuminate certain features. The blue bold text shows an improvement of some kind, while the lavender italicized text shows no real improvement over the range of tests. The remaining black text shows an improvement at some level between the tested ranges. Based on the above observations, only five stiffnesses play dominant roles in the deflected shape. This does not mean the other stiffnesses are insignificant, but rather, simply indicates they do not play a dominant role and will be left to the optimization for modification. After several more tests for the five stiffnesses of interest, a final result was achieved and is shown in Table 7.2, as well as graphically in Figure 7.4 through Figure 7.6.

Table 7.2 Manual Optimization Results

Stiffness	Final Value (Nmm/rad)
1X	1E8
2X	4E7
3Y	1E8
1Z	2E8
2Z	3E7

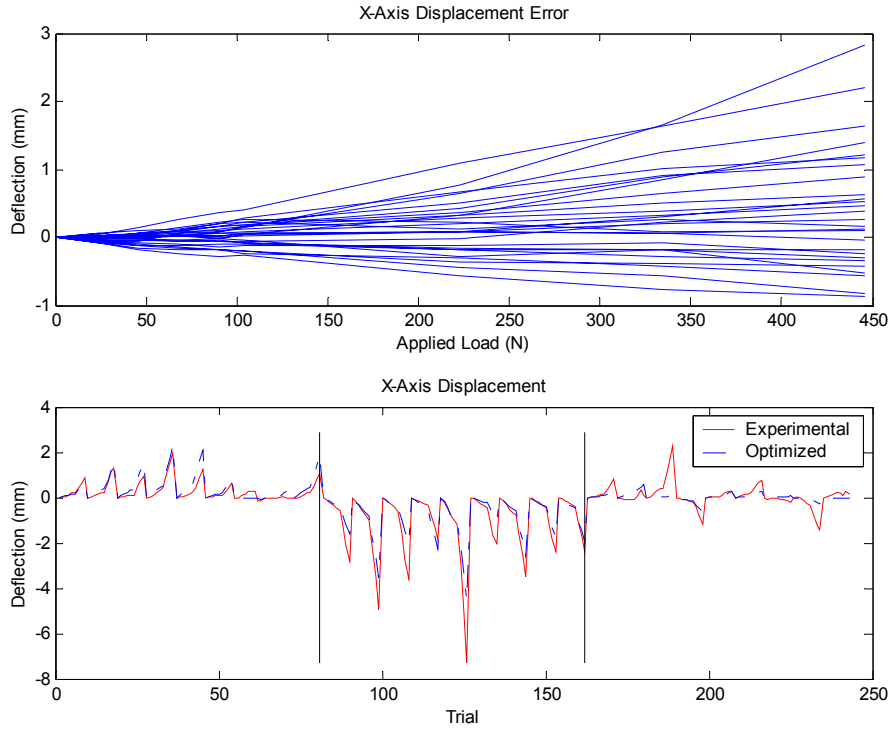


Figure 7.4 Third Regime X Direction Manual Optimization Results

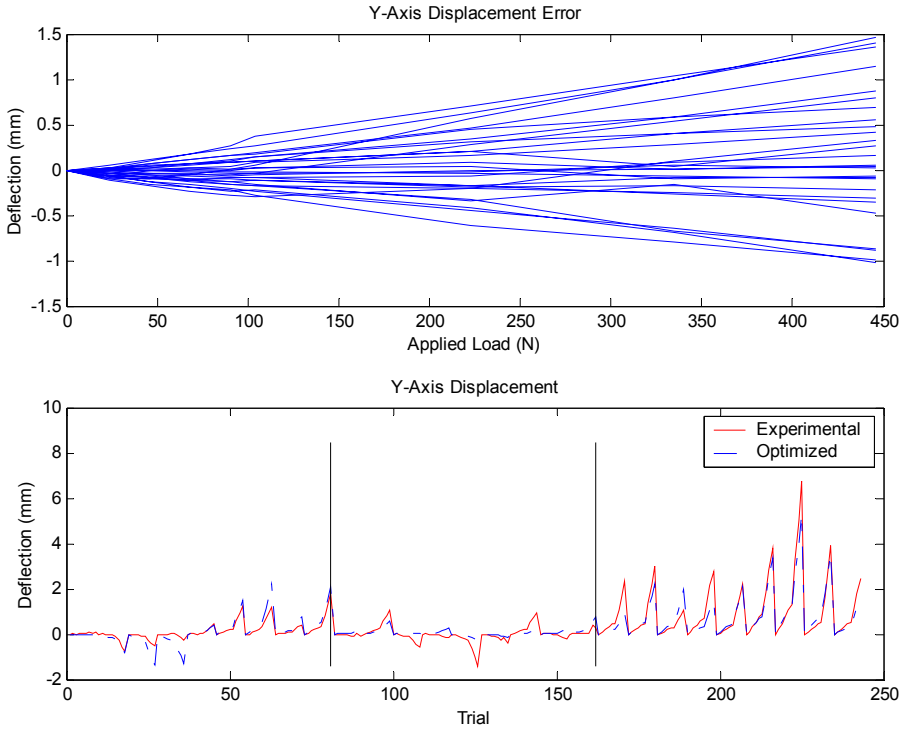


Figure 7.5 Third Regime Y Direction Manual Optimization Results

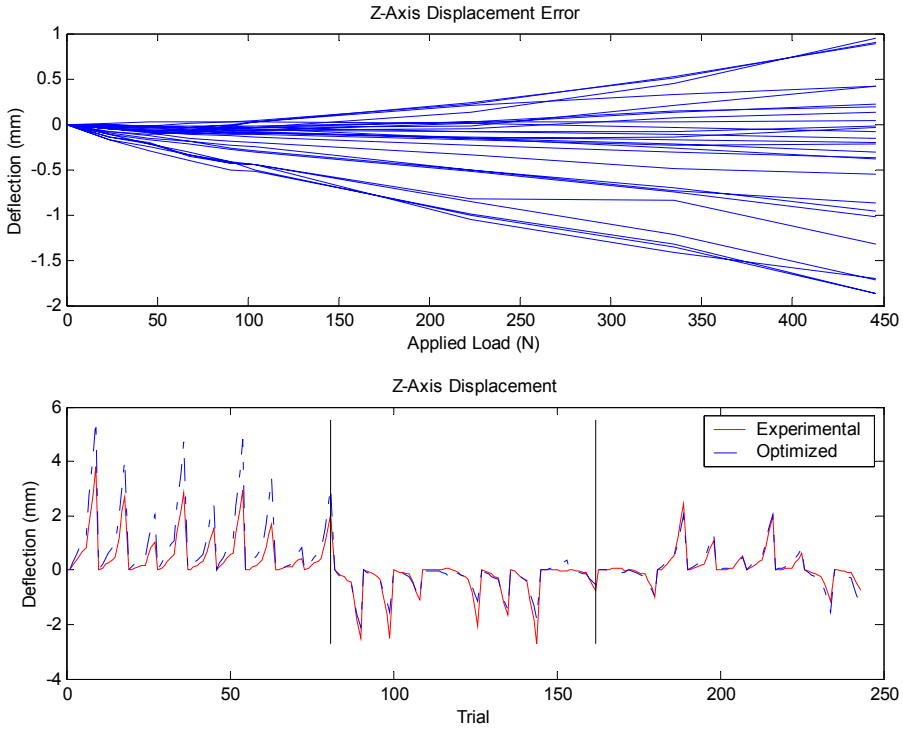


Figure 7.6 Third Regime Z Direction Manual Optimization Results

Documenting results with a succession of six plots is both cumbersome and excessive. While the individual X, Y, and Z direction displacements are important for understanding how each stiffness value impacts the manipulators compliance, that level of detail is not necessary to understand and visualize the results of the optimization. As a result, the X, Y, and Z direction displacements will be combined into an absolute displacement according to Equation 7.1 for many future plots.

Equation 7.1

$$D_{abs} = \sqrt{X^2 + Y^2 + Z^2}$$

Figure 7.4, Figure 7.5, and Figure 7.6 can be reduced with Equation 7.1 to the plots shown in Figure 7.7.

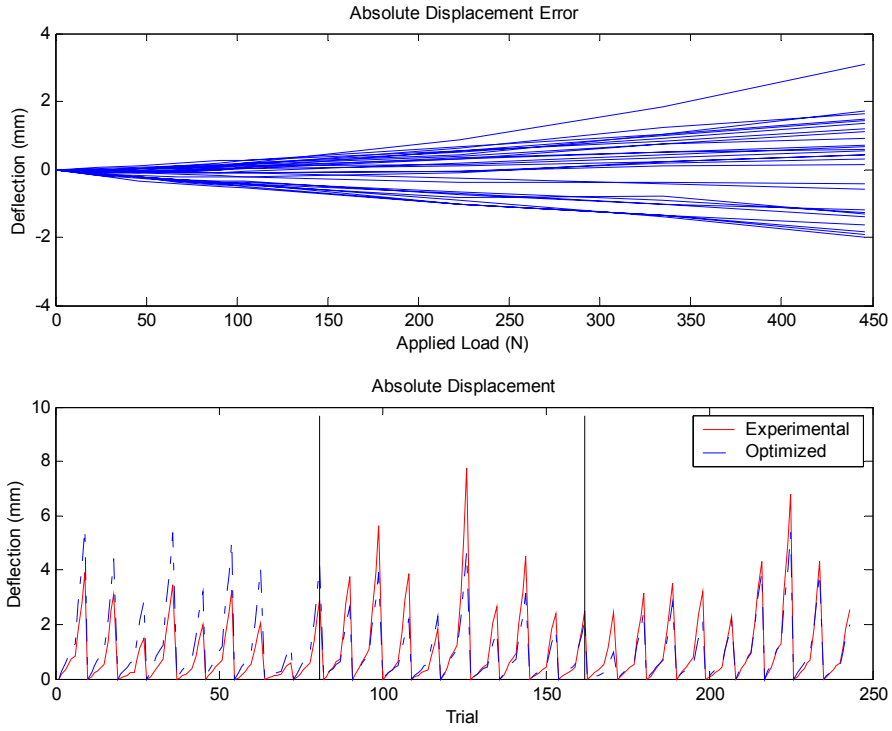


Figure 7.7 Third Regime Manual Optimization Absolute Displacement Error

7.2 Unweighted Optimization Results

The results from the optimization are broken down into twelve separate problems. Each of three regimes has two noteworthy solutions, corresponding to solutions based on the full load set and a reduced load set. Two complete sets of these solutions are presented corresponding to different weightings of the rotational component in the objective function. The reduced load set consists of all loads up to and including the 111 N (25 lb) load. This is important to the research because it presents a more realistic view of the likely loads for the manipulator when manufacturing composite materials.

Overall, the solutions are arrived at through a combination of the *fmincon* and *fminimax* solvers. The *fmincon* was used initially for the research, and therefore was able to find reasonable solutions for most problems; however, application of the *fminimax* solver on the results improved the overall behavior. This improvement was not always reflected in the objective function itself, since the *fminimax* solver attempts to reduce the

worst errors at the cost of lesser errors on other data points, resulting in only small changes in the objective function for some solutions.

Solutions to the optimization problem were difficult to predict based on initial conditions for this work. As a result, multiple initial condition sets are used to attempt to find a global minimum. Initial conditions typically were 1E7, 1E9, 1E10 Nmm/rad for all stiffness terms and the manual optimization solution. The result from the *fmincon* solver was used to start an *fminimax* solution. Furthermore, later regimes also use solutions from previous regimes as initial conditions. Using multiple algorithms helped verify the solutions from each solver.

The regimes of data were established to allow each successive regime to learn from the prior less complicated regime. In the same manner, the results from each regime will be presented and discussed in the order of the regime, covering the full load cases first and then separately discussing the reduced load set solutions. All result plots were generated with MATLAB codes in appendix sections 10.5.6 and 10.5.12 corresponding to full range loads and reduced range loads, respectively.

7.2.1 Results for Regime 1 Full Load Range

The first regime of testing included twenty-seven locations within a one-foot cube located directly in front of the manipulator. Each location was subject to nine loads, including an unloaded case. The manipulator was oriented in the elbow-up configuration for all tests, with no wrist reversals. The final objective function value achieved was 1555.37 based on Equation 5.5. While the absolute magnitude of the objective function has little meaning, it is very useful for comparison with other solutions. Figure 7.8 shows the absolute displacement error for the results of the first regime when considering all loads.

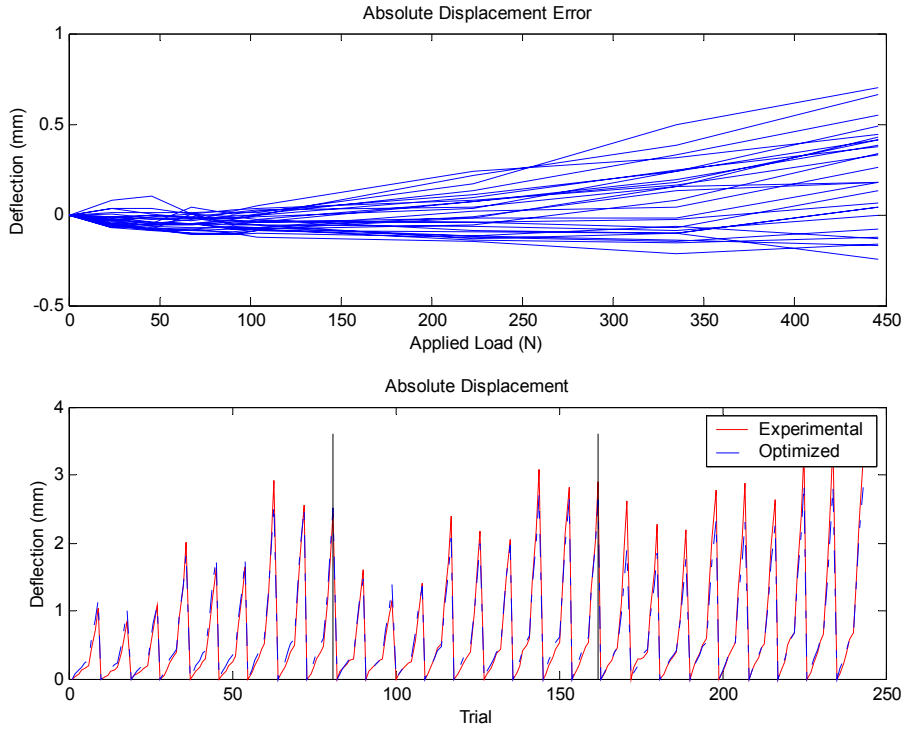


Figure 7.8 First Regime Full Load Absolute Displacement Plot

The overall error is well below one millimeter even at a load of 445 N (100 lbs). The second plot shown in Figure 7.8 displays the tight correlation between the experimental and optimized solution. This regime of data performs very well since the manipulator's arm does not significantly change orientation between the test locations. Figure 7.9 presents the rotational results for the first regime. While the X-axis rotations are small compared with experimental results, both the Y-axis and Z-axis rotations are much larger. However, a rough average of three milliradians is significantly less than one quarter of a degree rotation. Depending on the design of the end-effector, this slight misalignment could be inconsequential, or magnify due to an offset and become a noticeable error. In order to avoid this type of complication, it is recommended that end-effector tools be designed such that the contact point with the surface is located as close to the tool plate as possible. Further rotation plots will be included in the appendix for reference with the results discussed in the text.

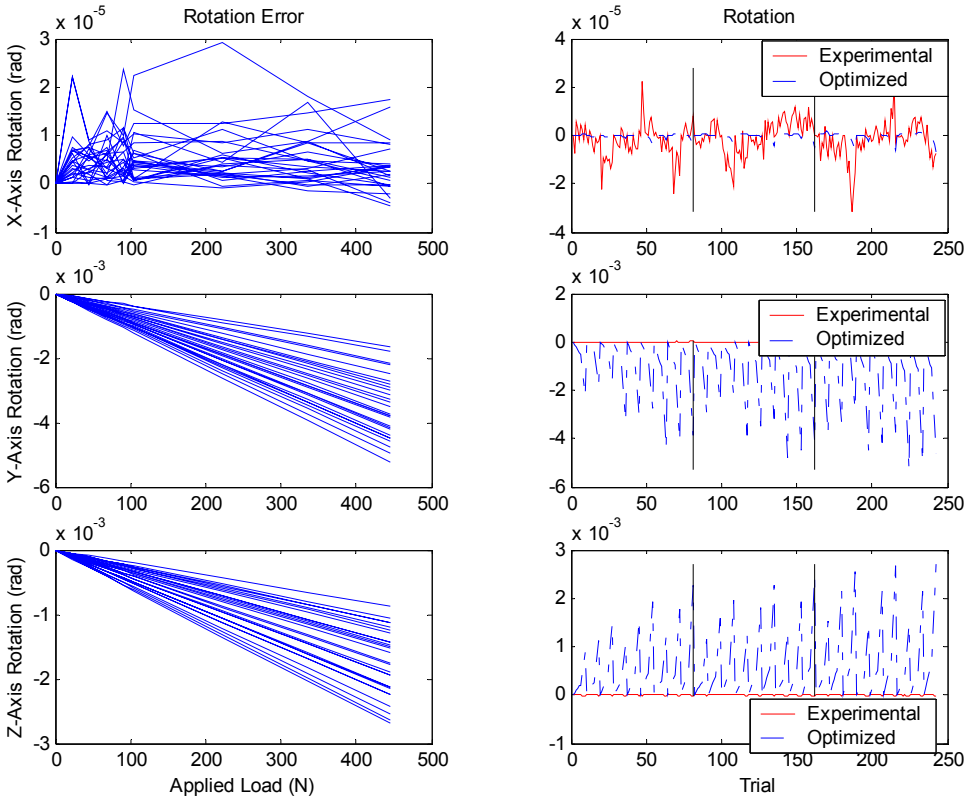


Figure 7.9 First Regime Full Load Set Rotation Plots

Figure 7.10 shows the percentage error between the optimized and experimental solutions relative to the experimental results.

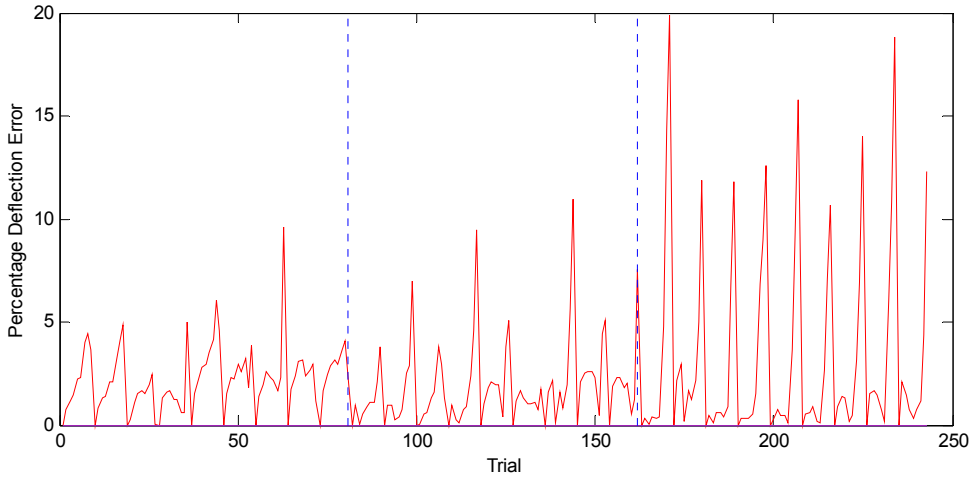


Figure 7.10 First Regime Full Load Set Percentage Error

For the vast majority of points, the compliant model was able to predict the experimental results within five percent. The high load points show significantly higher errors in the latter third of the trials. This range of tests corresponds to the data points that are closest in elevation to the shoulder axis. The trials were arranged such that the initial eighty-one samples were 254 millimeters (10 inches) below the shoulder, the middle eighty-one samples were 152 millimeters (6 inches) below the shoulder and the last eighty-one samples were only 51 millimeters (2 inches) below the shoulder. Therefore, the stiffness model derived, and shown in Table 7.3, works best at elevations more than 51 millimeters (2 inches) below the shoulder axis.

Consistent with the results from the manual optimization for this regime, only two stiffnesses played a significant role in the compliance of the manipulator. These two stiffnesses can be intuitively chosen by looking at the orientation of the manipulator and the loads applied. Both joint 2 and joint 3 see significant moments about their direction of motion when compared with other joint loadings, therefore, it is expected these two terms will be dominant. Based on DH convention, joint 2's motion occurs about the Z-axis of DH frame 1.

Table 7.3 First Regime Full Load Set Stiffness Matrix

DH Frame	X	Y	Z
0	N/A	N/A	1E10
1	1E10	1E10	1.995E8
2	1E10	N/A	4.185E7
3	1E10	1E10	1E10
4	1E10	1E10	1E10
5	1E10	1E10	1E10
6	1E10	N/A	1E10

7.2.2 Results for Regime 2 Full Load Range

The second regime included sample points in a much larger range than the first. All points lie within a roughly 2-foot cube, both above and below the shoulder axis of the manipulator. Once again, the manipulator was constrained to remain in the elbow-up

configuration, and there were no wrist reversals. The final objective function value was 1650.27, less than 100 points above the results from the first regime. However, even 100 points is easily visible in the results, shown in Figure 7.11.

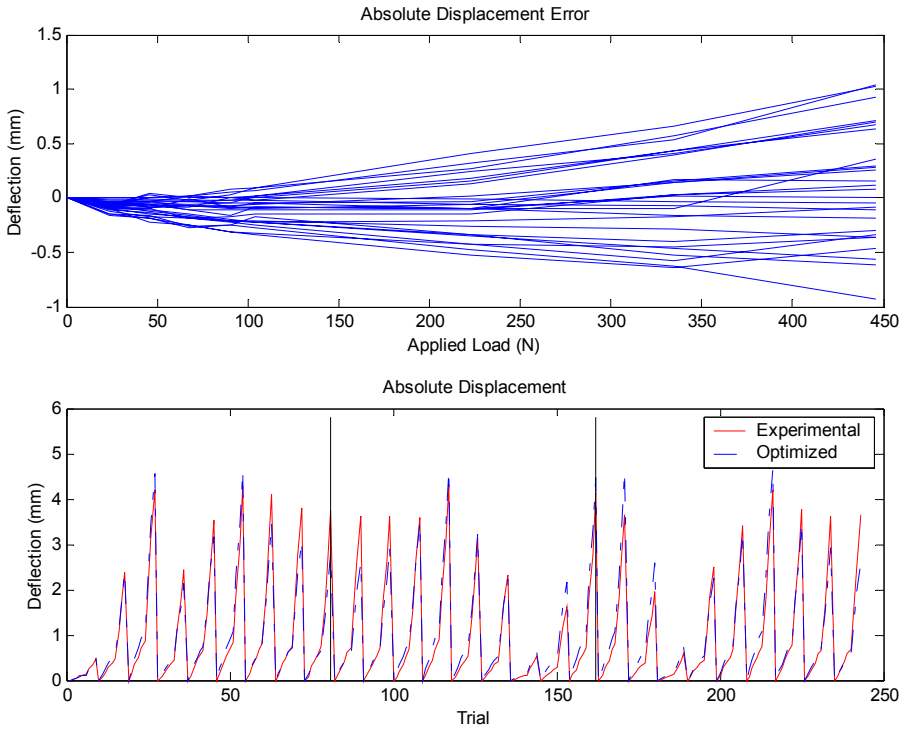


Figure 7.11 Second Regime Full Load Absolute Displacement Plot

It is evident from Figure 7.11 that almost all loads were within one millimeter of the experimental results. Furthermore, results up to 200 N (~45 lbs) stayed within one-half millimeter of the experimental results. The majority of the error is shown to be in the X and Y directions for this regime. Figure 7.12 and Figure 7.13 show the absolute displacement plots for the X and Y directions, and it is easily identified where the discrepancies exist.

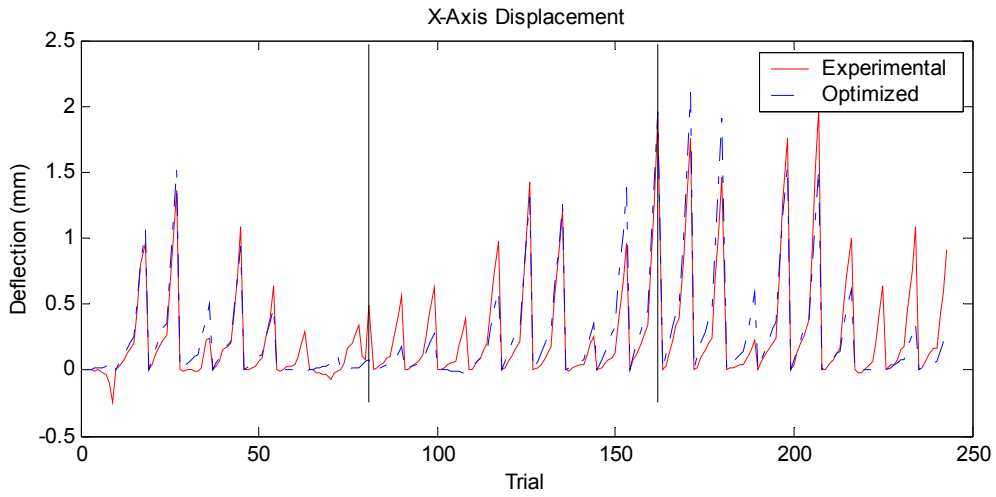


Figure 7.12 Second Regime Full Load Set X Direction Displacement

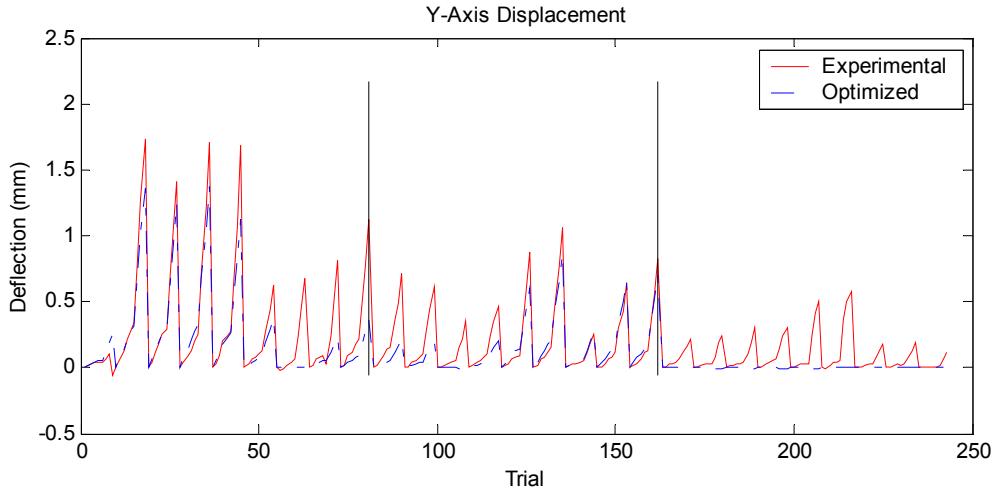


Figure 7.13 Second Regime Full Load Set Y Direction Displacement

These shortcomings are a direct result of the experimental techniques and point selection for the second regime. All loads are oriented in the vertical direction, almost exclusively acting on Z-axis stiffnesses. With no loads to isolate or even partially excite the X-axis and Y-axis stiffnesses, the half to three-quarter millimeter errors are expected. The third regime will include loads to excite the X-axis and Y-axis deflections and provide a more consistent fit.

Appendix section 10.1.1 presents the rotational error components for the second regime. Once again, the X-axis rotations stay well within acceptable limits, however both the Y-axis and Z-axis show additional error when compared to the first regime.

Errors reach roughly one-third of a degree for both the Y-axis and Z-axis rotations. While still small, these errors may accumulate and result in end-effector misalignment, especially when the tool attached to the end-effector has a contact point far away from the tool plate.

Figure 7.14 presents the overall error percentage between the experimental and optimized solutions.

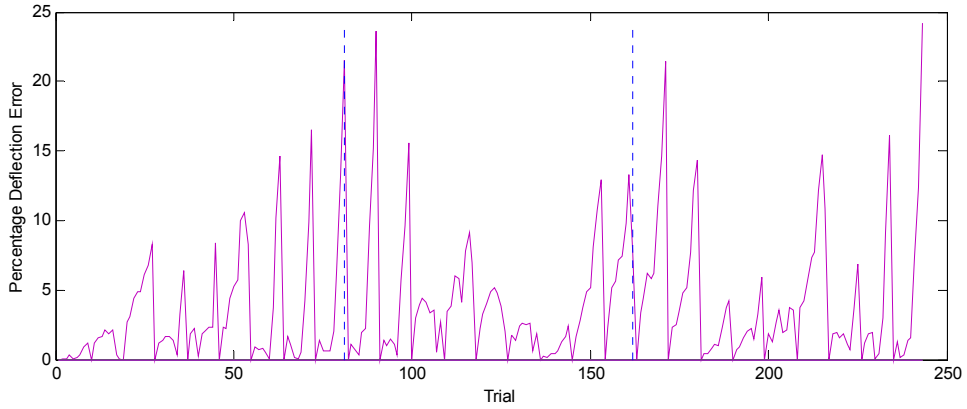


Figure 7.14 Second Regime Full Load Set Percentage Error

As in the first regime, the workspace was divided in three sections, resulting in three clearly defined areas as shown in Figure 7.14. The first third of the trials were located 305 millimeters (12 inches) to the left (away from the arm) of the torso, with the middle third located on the X-Z plane of the zeroeth DH frame, and the last third located 305 millimeters (12 inches) to the right of the robot. The last third was very close to planar with the arm when the robot was in its zero position. Looking at the errors in Figure 7.14, it is evident the largest errors occur at the ends of each group, which corresponds to locations in the workspace that are far from the shoulder and torso of the robot. Specifically, the points 305 millimeters (12 inches) to the right of the torso, and those points at the front edge of the workspace were especially susceptible to poor correlation with experimental data. However, even with these errors the overall percent error is still below six or seven percent for the majority of data points. Applications should therefore test a workspace larger than the proposed area of concern, since the poor results tend to exist towards the boundaries.

Once again, with loads only in the vertical, there are two terms in the stiffness matrix that were expected to change. Table 7.4 shows the final stiffness matrix and fits the predictions.

Table 7.4 Second Regime Full Load Set Stiffness Matrix

DH Frame	X	Y	Z
0	N/A	N/A	1E10
1	1E10	1E10	1.5176E8
2	1E10	N/A	3.5124E7
3	1E10	1E10	1E10
4	1E10	1E10	1E10
5	1E10	1E10	1E10
6	1E10	N/A	1E10

Relative to the first regime, both stiffnesses of interest reduced in magnitude. This softening is a result of the broader range of the data points, requiring a slightly more compliant manipulator to meet the selection of points.

Interpretation of the stiffness matrix is not intuitive relative to how the manipulator will deflect. Figure 7.15 shows a scaled predicted deflection based on the compliant model. All joint rotations were scale by a factor of forty to produce the large deflection shown. The floating coordinate system is scaled to the deflection in each coordinate direction of the zero frame displaying how a load in only the Z direction can result in motions and rotations along all three coordinate axes. The deflected manipulator was created by applying the predicted rotations about joints two and three only, as shown in Table 7.5. The particular results shown are based on results for location fourteen (shown for regime two in Table 4.2) subjected to the 445 N load.

Table 7.5 Applied Rotations for Figure 7.15

Joint	Theta Rotation
2	-2.7427
3	1.9076

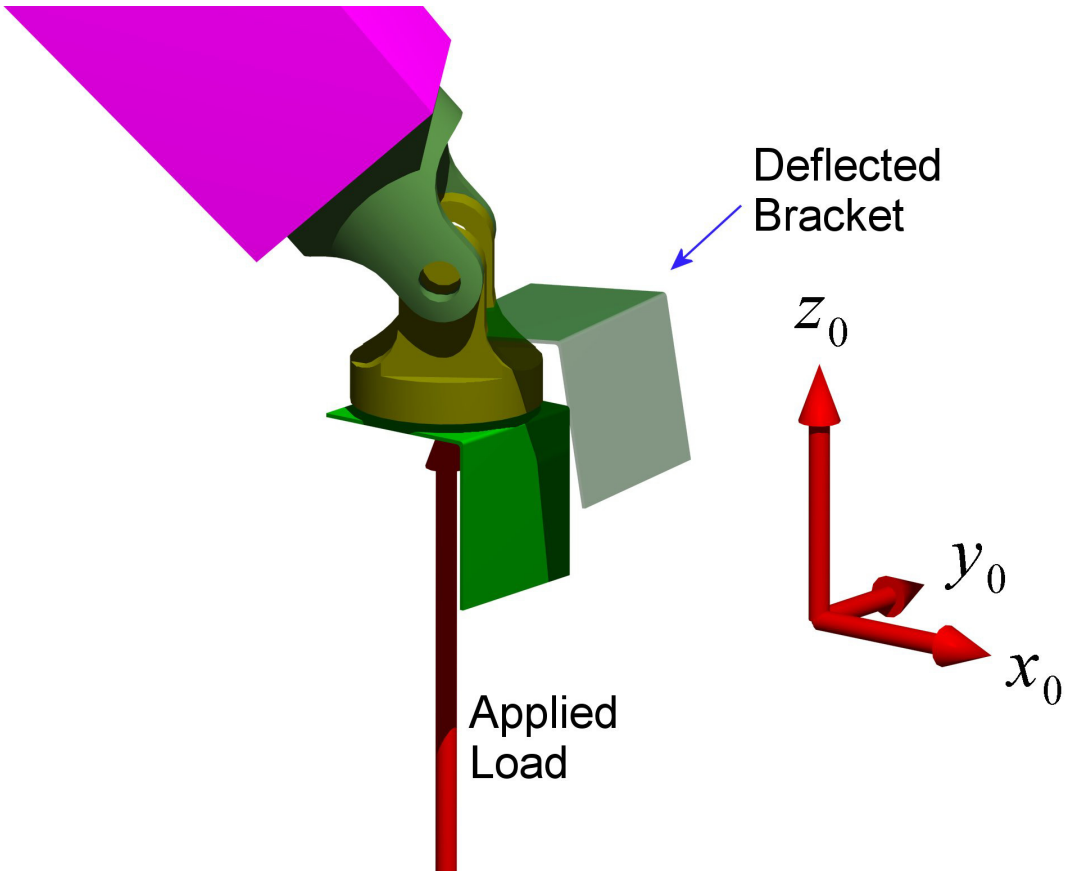


Figure 7.15 Second Regime Full Load Deflection

7.2.3 Results for Regime 3 Full Load Range

The third regime was comprised of more varied load cases and points in the workspace than either of the previous regimes. The loads were applied not only in the Z Direction but also in the X Direction and Y Direction. The actual load points in the workspace were spread throughout the manipulator's reachable workspace in all directions, but remained forward of the shoulder axis. This regime only uses nine locations, but each location sees the entire range of loads from three different directions, resulting in 243 total trials. The final optimized objective function value was 1808.63, roughly 150 higher than the second regime. Figure 7.16 shows the resulting deflection for the data points of the third regime.

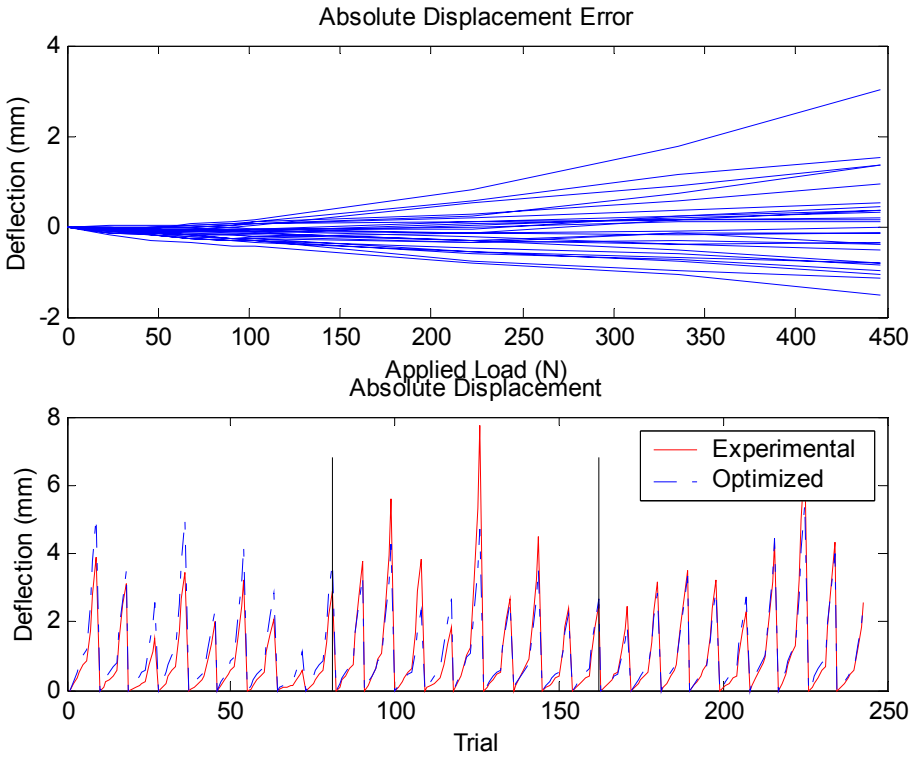


Figure 7.16 Third Regime Full Load Absolute Displacement Plot

The results appear to have a single outlier point at the middle of the data set, and shown on the high side of the first plot in Figure 7.16. Neglecting this load set, the remaining results all fall well within two millimeters of deflection for the range of loads applied. The first third of the results show the optimized solution overestimating the displacements. Investigating the component deflections shows this overestimation is primarily in the Z Direction. While there are no obvious trends in the latter two-thirds of the data, the Z Direction underestimates the deflections for these points. Rotational data is shown in appendix section 10.1.1 . In comparison to prior regimes, all rotational errors increased in magnitude. While X-axis errors increased to the maximum level seen in the second regime, both the Y-axis and Z-axis showed errors in rotation exceeding one degree. Figure 7.17 presents the percent error relative to experimental results.

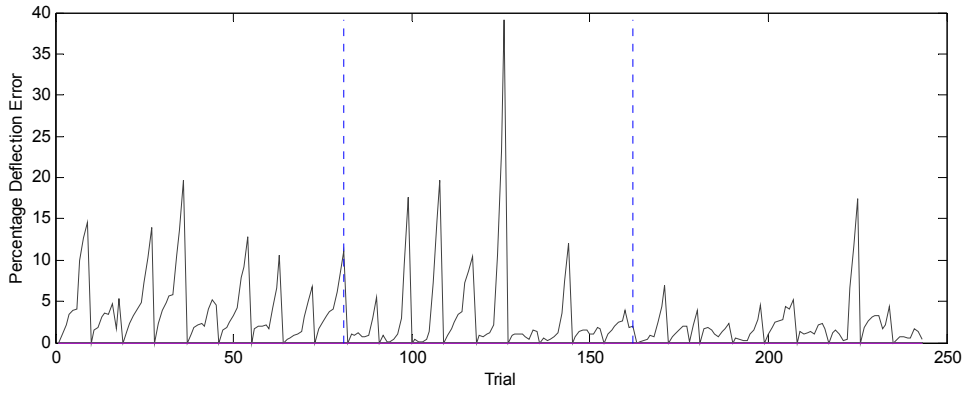


Figure 7.17 Third Regime Full Load Set Percentage Error

Ignoring the single outlier in the middle, the percentage error is within five or six percent overall. The heaviest loads show increases up to fifteen or twenty percent in some cases, but these remain the exception rather than the norm. The third group of results maintains a very small error percentage on average, and this corresponds to the loads applied in the Y Direction. Consistent with this observation is a very good correlation of the Y Direction error over the entire regime. All but three of the loadsets match the Y Direction results within roughly one-half millimeter, as shown in Figure 7.18. Therefore, based on stiffness matrix for this solution, an application would be best oriented to have the primary loads applied in the Y direction.

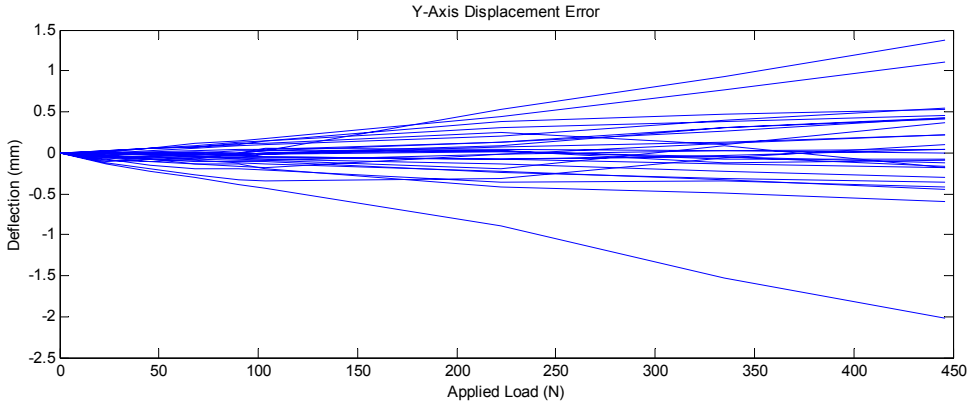


Figure 7.18 Third Regime Y Direction Displacement Error

The final stiffness matrix is dependent on many more stiffnesses than either of the prior regimes, as shown in Table 7.6.

Table 7.6 Third Regime Full Load Set Stiffness Matrix

DH Frame	X	Y	Z
0	N/A	N/A	1.3250E9
1	1.8091E8	7.8416E8	1.2216E8
2	5.4512E7	N/A	7.1204E7
3	4.4006E8	7.0302E7	8.2092E6
4	1.0330E7	1.4885E7	4.0011E6
5	9.0820E6	3.9980E6	1E7
6	9.9565E6	N/A	1E7

When compared with the first and second regimes, the third regime performed better using a stiffness of 1E7 Nmm/rad as a base rather than 1E10 Nmm/rad. This helps support the behavior of the second regime where overall the manipulator appears too stiff in the horizontal directions.

As with the second regime, the stiffnesses do not intuitively show the deflection of the manipulator. Figure 7.19 shows the predicted deflection for the manipulator subjected to a 445 N load in the Y direction at location twenty-three in Table 4.2 for the third regime. Joint rotations were scaled forty times to provide the significant deflection. In comparison to the second regime, this deflection is due to rotations about almost all theta and alpha DH variables resulting in a motion primarily in the Y and Z directions of the zero frame, as shown in Table 7.7.

Table 7.7 Applied Rotations for Figure 7.19

Joint	Alpha Rotation (deg)	Theta Rotation (deg)
1	6.0441	0.3907
2	7.8488	-2.0446
3	1.1451	0.2583
4	6.3237	7.3611
5	9.7980	4.2417
6	-3.8994	0

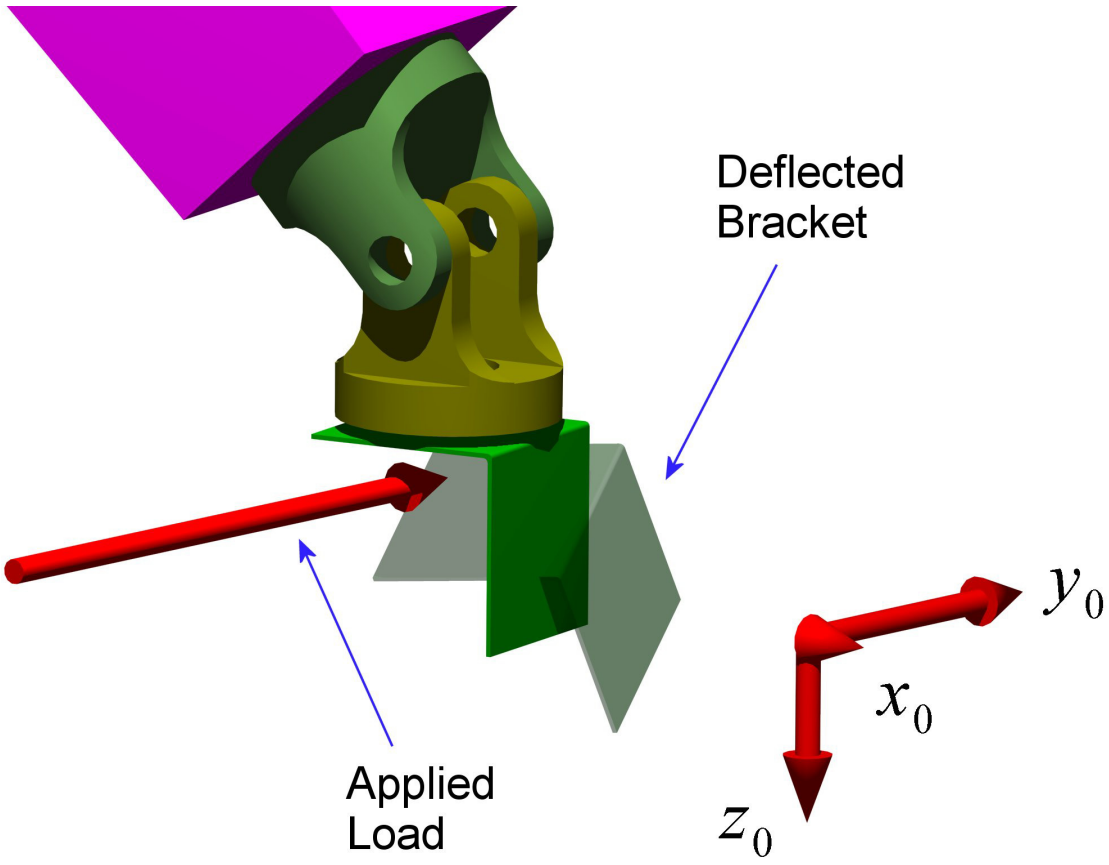


Figure 7.19 Third Regime Full Y Direction Load Deflection

Despite the increasing error in the rotations, the initial assumption that the rotational error was tightly coupled to the displacement error was verified. As the displacement error increased in magnitude, so did the rotational errors.

7.2.4 Results for Regime 1 Reduced Load Range

The purpose of also converging the optimization for a reduced load range is to better match the projected application for the manipulator of interest. The Merlin robot used for the experimentation has a stated maximum capacity of 89 N (20 lbs). As a result, only loads less than or equal to 111 N (25 lbs) were used for the reduced load range solutions. The experimental data was the same data used for the prior three optimization solutions, the only difference being that the reduced load range dropped the 222, 334, and 445 N (50, 75 and 100 lbs) loads from the data set. The final objective function value for the reduced regime one optimization was 997.439, and the results are

shown in Figure 7.20. The objective function value is significantly smaller than the corresponding full range value; however, this is expected since the number of data points has been reduced.

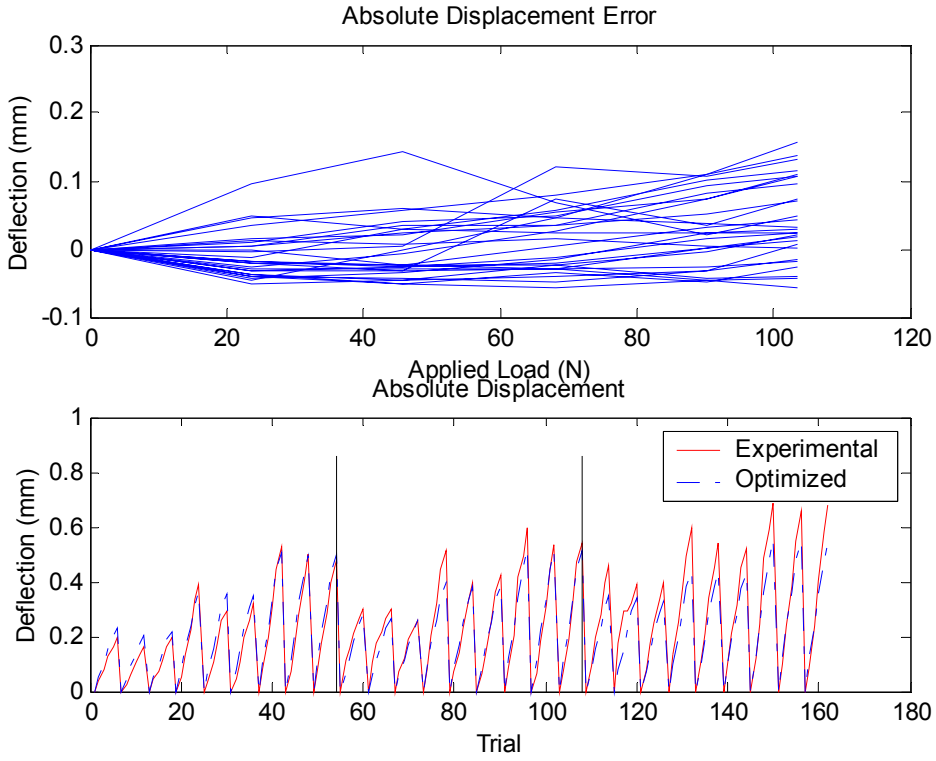


Figure 7.20 First Regime Reduced Load Absolute Displacement Plot

Considering only loads up to 111 N (25 lbs), the optimized solution maintains a maximum error less than 0.2 mm when compared to the experimental results. This is only two times the accuracy of the measurement system and represents an exceptional result. Overall, the results show negligible trends and approximate experimental data consistently across the set of trials. Investigating the rotational results, presented in appendix section 10.1.2 , show similar patterns to those seen in the full load set results. The X-axis rotations are negligible, while both the Y-axis and Z-axis developed larger rotations from the model than experimental results. As expected, based on the reduced load set, the error stays well below one-tenth of a degree for the entire test. Figure 7.21 shows the percentage error relative to the experimental results.

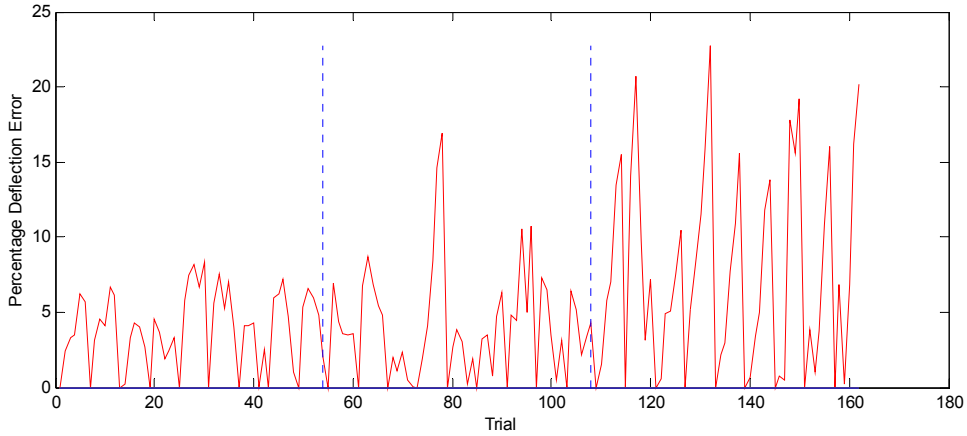


Figure 7.21 First Regime Reduced Load Set Percentage Error

While the majority of the data points are within six or seven percent of the experimental results, the latter third show a significantly higher error. This error, as with the full data set results, corresponds to the data points that are closest to the same elevation as the shoulder axis. The remaining points are all well below the shoulder axis for this regime. The percentage errors are roughly the same as those gained from the full range optimization. While reducing the range of loads does reduce the absolute error between the optimized solution and the experimental results, the percentage difference remains consistent. This indicates a limit in the optimization routine to improve the overall behavior significantly. It is likely to be dependent on the accuracy of the results and the cumulative error from so many experimental data points.

Even with the slight errors near the shoulder axis, the stiffness matrix in Table 7.8, provides a very accurate model of the actual robot for the reduced load set.

Table 7.8 First Regime Reduced Load Set Stiffness Matrix

DH Frame	X	Y	Z
0	N/A	N/A	1E10
1	1E10	1E10	2.0111E8
2	1E10	N/A	5.7126E7
3	1E10	1E10	1E10
4	1E10	1E10	1E10
5	1E10	1E10	1E10
6	1E10	N/A	1E10

In comparison to the stiffness matrix resulting from the full load optimization, the two stiffness values of interest changed by 0.81 percent, for joint 1, and 26.7 percent, for joint 2, both about the Z-axis. Even a change of 26 percent still maintains the same order of magnitude for the stiffness. Both stiffnesses soften relative to the full data set results.

7.2.5 Results for Regime 2 Reduced Load Range

Based on the reduced load cases, the second regime resulted in a large improvement over the full load case optimization. The final objective function was 1023.11, only 25 points higher than the first regime solution based on the reduced load set. A majority of the load sets predicted within 0.1 mm of the experimental data, however some samples predicted defections up to 0.3 mm. These samples correspond to Z Direction locations of –305 and 0 millimeters (-12 and 0 inches) for the most part. Therefore, the results shown in Figure 7.22 are best suited to points above the shoulder axis in the workspace.

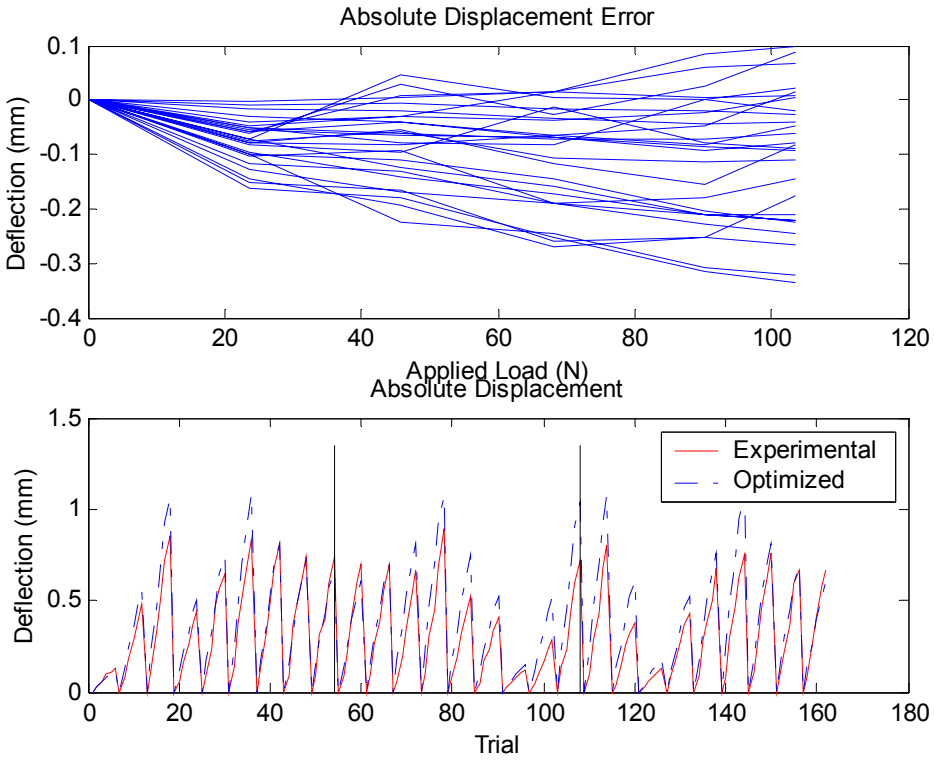


Figure 7.22 Second Regime Reduced Load Absolute Displacement Plot

Appendix section 10.1.2 shows the rotational results for this regime of testing. As expected, the rotational errors in this regime are smaller than the errors from the full load solution for the second regime, but also larger than the reduced load set for the first regime. However, the maximum error, on the Y-axis rotation, has a maximum of about one-tenth of a degree.

It is also noted from the component plots, that the majority of the error causing certain load sets to perform more softly is a result of deflections in the Z Direction. Looking at the percentage error between the two solutions, the current solution is shown in Figure 7.23, there does not appear to be a definite trend relating to which points in the workspace perform better. This result is likely due to a randomness in the experimental results.

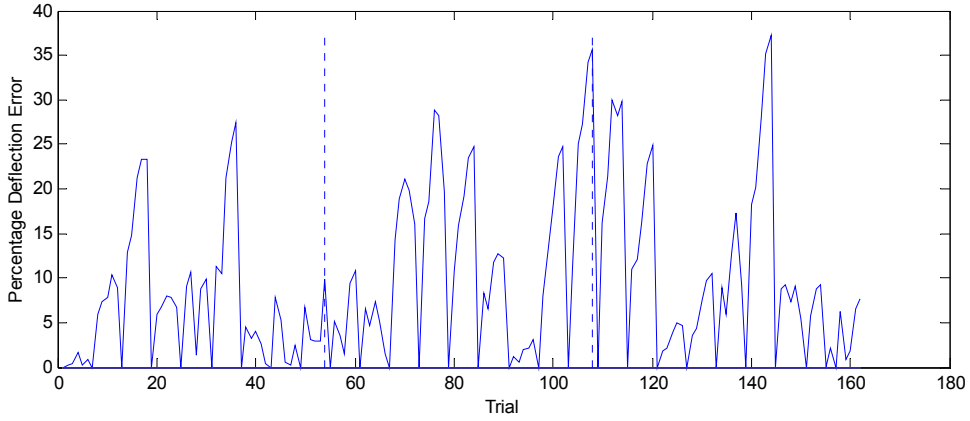


Figure 7.23 Second Regime Reduced Load Set Percentage Error

It is also evident from the component plots, that the X Direction and Y Direction errors are significant because the model does not have the required compliances in those directions. This is a direct result, as in the first regime, of the lack of load cases to isolate the compliances in the X and Y directions. The final stiffness matrix is shown Table 7.9.

Table 7.9 Second Regime Reduced Load Set Stiffness Matrix

DH Frame	X	Y	Z
0	N/A	N/A	1E10
1	1E10	1E10	1.5176E8
2	1E10	N/A	3.5124E7
3	1E10	1E10	1E10
4	1E10	1E10	1E10
5	1E10	1E10	1E10
6	1E10	N/A	1E10

This stiffness matrix is the same as the matrix resulting from the second regime of data when the full load set was applied. This is reasonable because one of the initial conditions for the reduced data set was the result from the full data set optimization. The other start conditions failed to improve the result. However, when compared with the results from the full load solution for the second regime, the percentage error is slightly higher on average for the reduced load set. This is attributed to the fact that the same stiffness matrix was found to be acceptable, while the higher loads were not included in the solution.

7.2.6 Results for Regime 3 Reduced Load Range

The reduced range of loads for the third regime was only those loads up to and including the 111 N (25 pound) load. The results predict a total error between the model and the experimental data to remain below roughly 0.4 mm. Figure 7.24 shows the absolute displacement error for this optimization. The final objective function was 1044.2, only twenty-one points above the second regime.

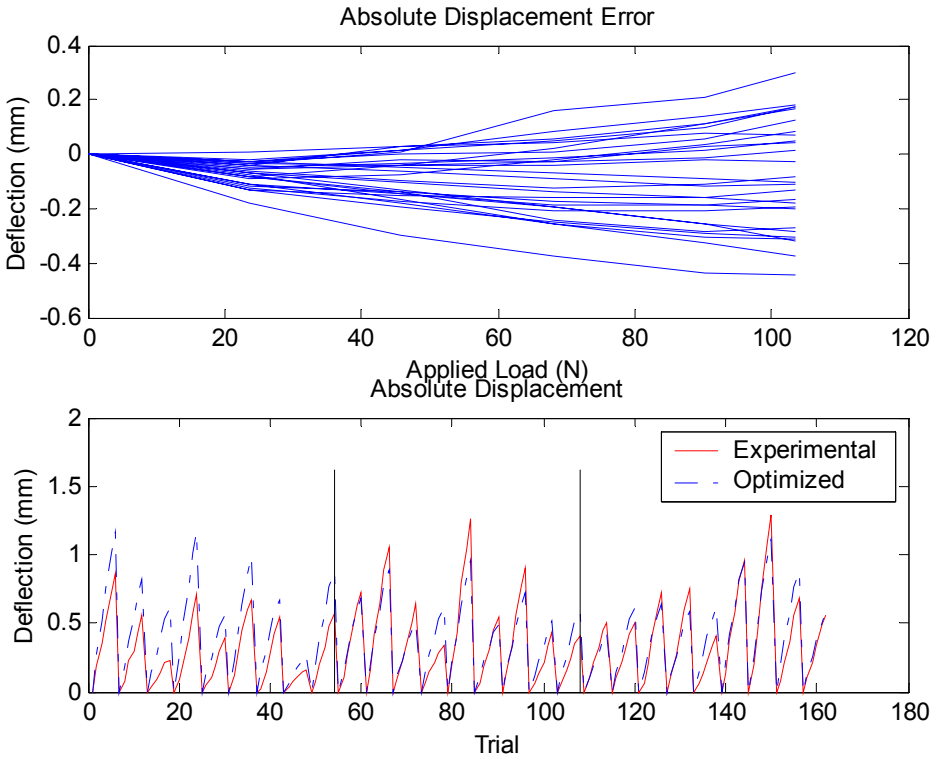


Figure 7.24 Third Regime Reduced Load Absolute Displacement Plot

It is evident from the results that the first third of the tests have a significantly higher error than the remaining two thirds. The first third corresponds to the loads in the Z Direction. When the loading was oriented to act in only the X Direction or Y Direction, the model typically underestimated the deflection, however, the Z Direction showed an overestimation. The rotations, as shown in appendix section 10.1.2 , were at worst one-quarter of a degree. This does not show the large increase in error between the second and third regime as seen with the full load set. Consistent with previous results, the X-axis rotational error is negligible, while the Y-axis and Z-axis rotations play the dominant roles. However, this shows one-fifth the rotational error of the third regime under the full load set. Figure 7.25 shows the percentage error for the optimization.

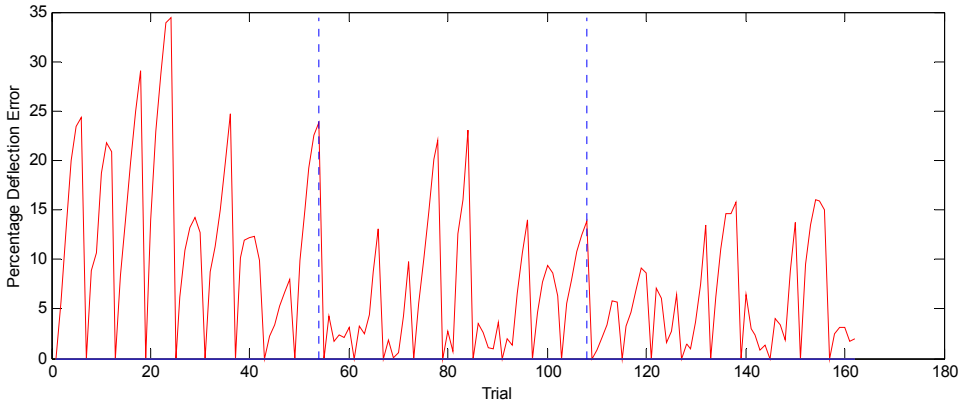


Figure 7.25 Third Regime Reduced Load Set Percentage Error

As expected, the highest error is in the first third of the experiment, although the average error is still between eight and ten percent for the remaining tests. The main reason the model has a difficult task predicting the experimental results for this regime is a result of how the manipulator deflected for the X Direction and Y Direction loads. As shown in Figure 7.26 and Figure 7.27, the X Direction and Y Direction performed poorly when the load was in the other direction.

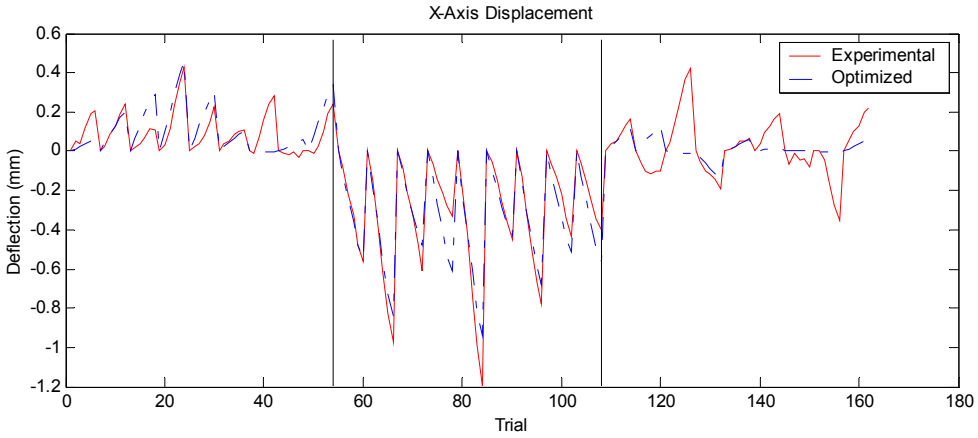


Figure 7.26 Third Regime Reduced Load Set X Direction Displacement

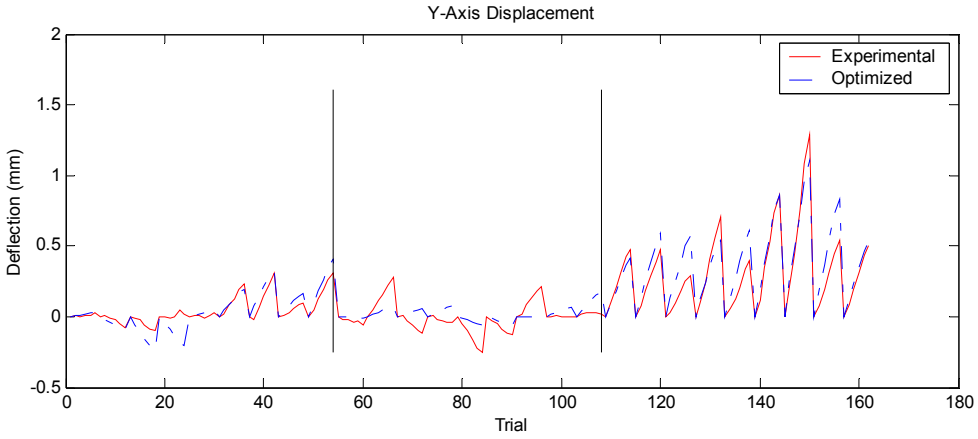


Figure 7.27 Third Regime Reduced Load Set Y Direction Displacement

For the first third of the experiments, the load was in the Z Direction, during the second third the load was moved to act in the X Direction, and the last third was subjected to loads in the Y Direction. The final stiffness matrix is shown in Table 7.10.

Table 7.10 Third Regime Reduced Load Set Stiffness Matrix

DH Frame	X	Y	Z
0	N/A	N/A	1.3250E9
1	1.1809E8	7.8416E8	1.2225E8
2	5.4611E7	N/A	7.1294E7
3	4.4006E8	7.0392E7	8.2601E6
4	1.0392E7	1.4870E7	4.5012E6
5	9.2040E6	1.1419E7	1E10
6	1E10	N/A	1E10

7.3 Weighted Optimization Results

The previous results all used unweighted rotations and displacements in the objective function. However, this results in an objective function that is not sensitive to rotational errors. Therefore, the optimization was modified to weigh the rotational components comparable to the displacements. Since rotational information is computed in radians, compared to millimeters for displacements, the average difference is three orders of magnitude. Scaling the rotation error by one thousand raises its importance in

the optimization to that of the displacement. All three rotational components were used in the objective function, although only rotations about the Z-axis showed visible correlation with the applied loads. While all six cases are presented, only significant results will be reported extensively.

7.3.1 Results for Regime 1 Full Load Case

As with the unweighted solution, the optimized solution tracked the experimental results very well, maintaining a maximum error below 0.6 millimeters at 445 N (100 lb). The final objective function value was 1557, which is within two points of the corresponding objective function for the unweighted objective function. This accounts for the similar absolute displacement plot when compared with the results from the weighted objective function. The current plot is shown in Figure 7.28 and shows slight improvements at the 445 N (100 lb) loads.

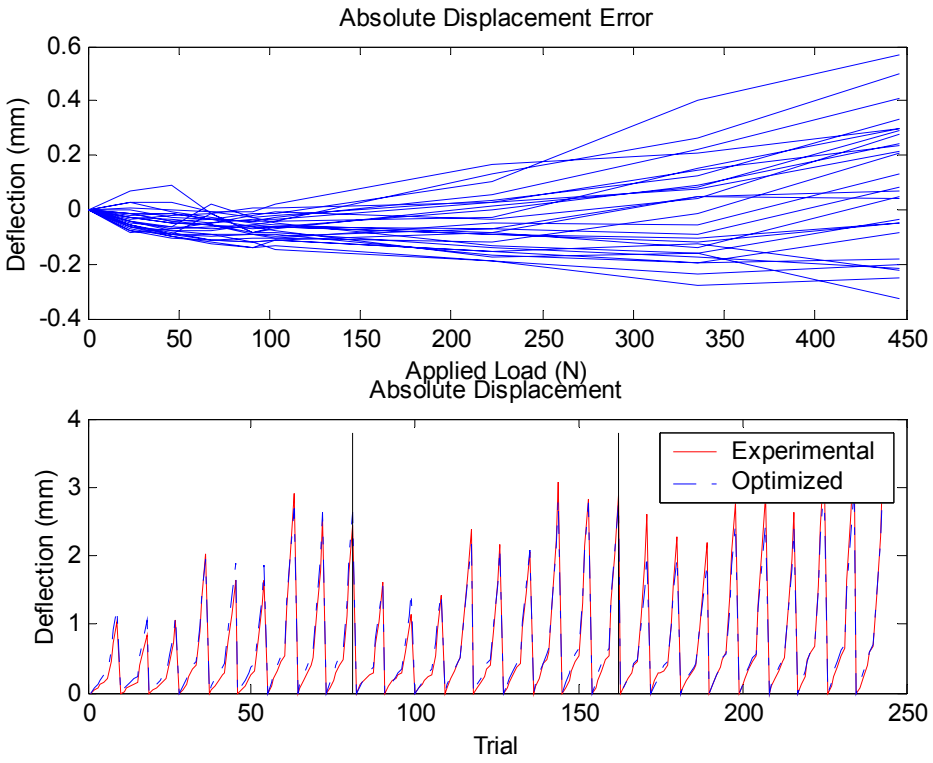


Figure 7.28 First Regime Full Load Absolute Displacement Plot

The peak errors are reduced between two and four percent relative to the weighted solution. Figure 7.29 presents the percentage error between experimental and optimized

results. Recalling the corresponding plot (Figure 7.10) from the weighted results, the basic shape is captured in the error, but the peaks are reduced.

The rotations, shown in appendix section 10.1.3 , show a slight improvement about the Y-axis, but remain virtually unchanged about either X-axis or the Z-axis. The final stiffness matrix is shown in Table 7.11 and is very similar results to unweighted solutions.

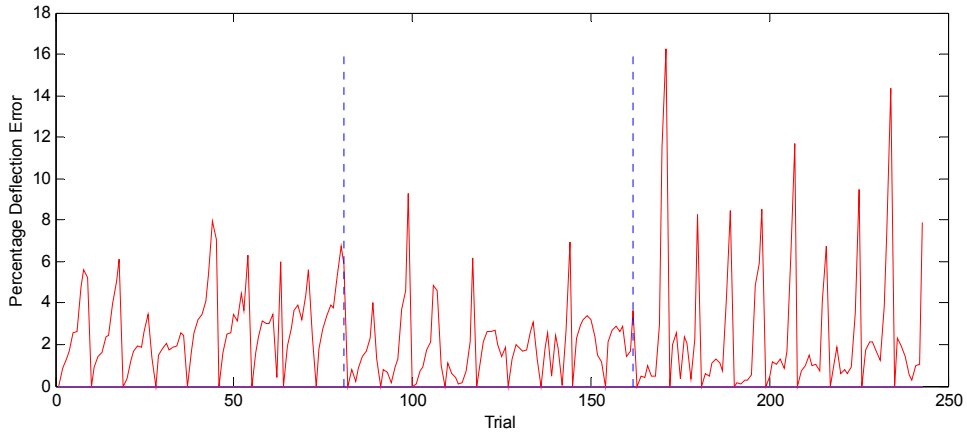


Figure 7.29 First Regime Full Load Set Percentage Error

Table 7.11 First Regime Full Load Stiffness Matrix

DH Frame	X	Y	Z
0	N/A	N/A	1E10
1	1E10	1E10	1.9941E8
2	1E10	N/A	3.8557E7
3	1E10	1E10	1E10
4	1E10	1E10	1E10
5	1E10	1E10	1E10
6	1E10	N/A	1E10

7.3.2 Results for Regime 2 Full Load Case

The second regime with the weighted objective function resulted in an objective function value of 1615, thirty-five points below the corresponding weighted solutions. This different is reflected in the absolute displacement in Figure 7.30, where the

maximum errors are reduced slightly. As with the first regime, the maximum percent error was reduced, while the lower errors remained unchanged. The percent error plot is shown in Figure 7.31. The rotations, shown in appendix section 10.1.3 , show an interesting trend, where despite reductions in displacement error, rotations saw an increase in error overall. The X-axis rotations increased an order of magnitude, while Y-axis and Z-axis show no appreciable change. However, the increase about the X-axis is still an order of magnitude below that of either other axes. The rotations show no significant changes relative to the unweighted solution.

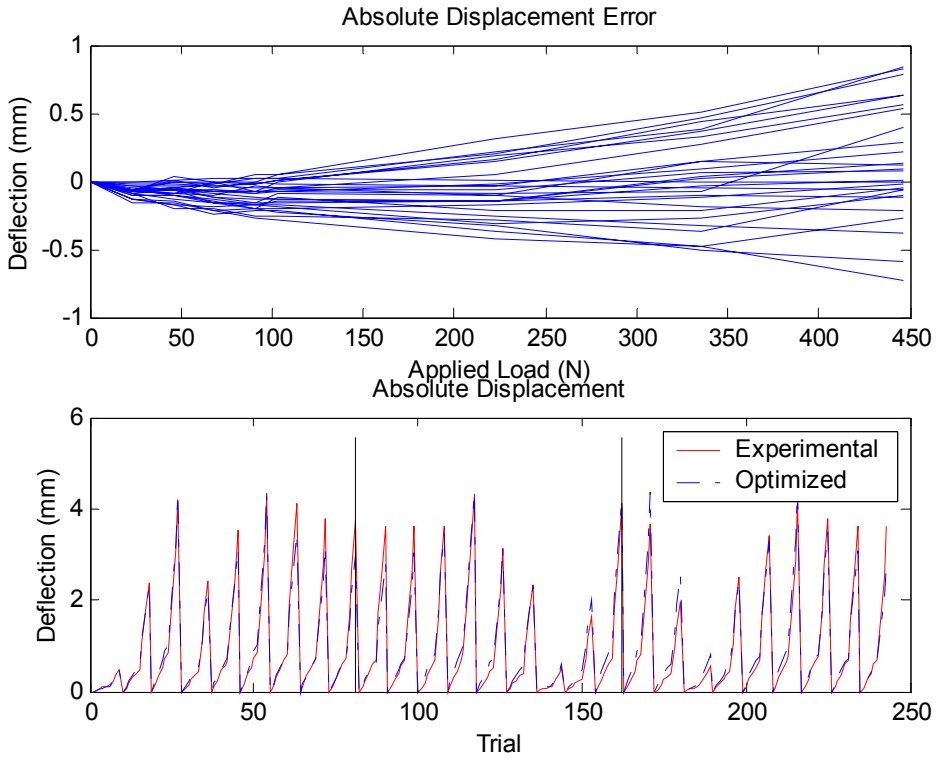


Figure 7.30 Second Regime Full Load Absolute Displacement Plot

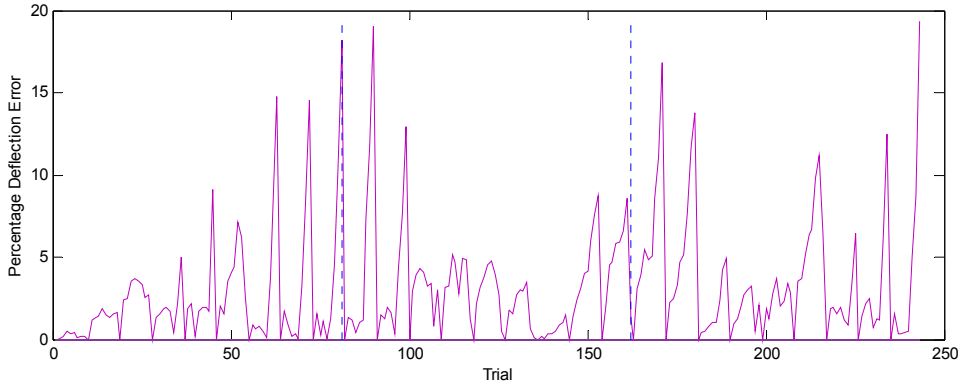


Figure 7.31 Second Regime Full Load Set Percentage Error

Table 7.12 Second Regime Full Load Stiffness Matrix

DH Frame	X	Y	Z
0	N/A	N/A	5.2794E7
1	5.3219E8	6.2323E7	1.9578E8
2	3.4822E7	N/A	4.4810E7
3	4.0423E7	2.2118E8	7.8954E6
4	1.6282E7	1.6891E7	1.3744E7
5	1.4595E7	1.3744E7	1E10
6	1E10	N/A	1E10

As seen in the stiffness matrix, all influential terms moved from their initial conditions. This is in stark contrast to the second regime with the unweighted objective function, where only two terms showed a causal relationship with the objective function value. This change is a direct result of the weighting of the rotational components in the objective function. Furthermore, this change improved the overall objective function by slightly over two percent.

7.3.3 Results for Regime 3 Full Load Case

In contrast to the second regime, the third regime minimized to a final objective function value of 1827, nearly twenty points higher than the corresponding unweighted solution. It is evident in Figure 7.32 that the error at 445 N (100 lb) is higher on average than the unweighted solution. The percentage error in Figure 7.33 also confirms the

increased error, with larger errors in both the second and third sections of the experiment. It appears the weighted solution optimized to the first third of the experimental trials rather than the last third like the unweighted solution. This change is most likely due to the objective function weighting, but could also be a result of the initial conditions. Rotational information showed no significant trends and maximum results were comparable to those of the unweighted solution.

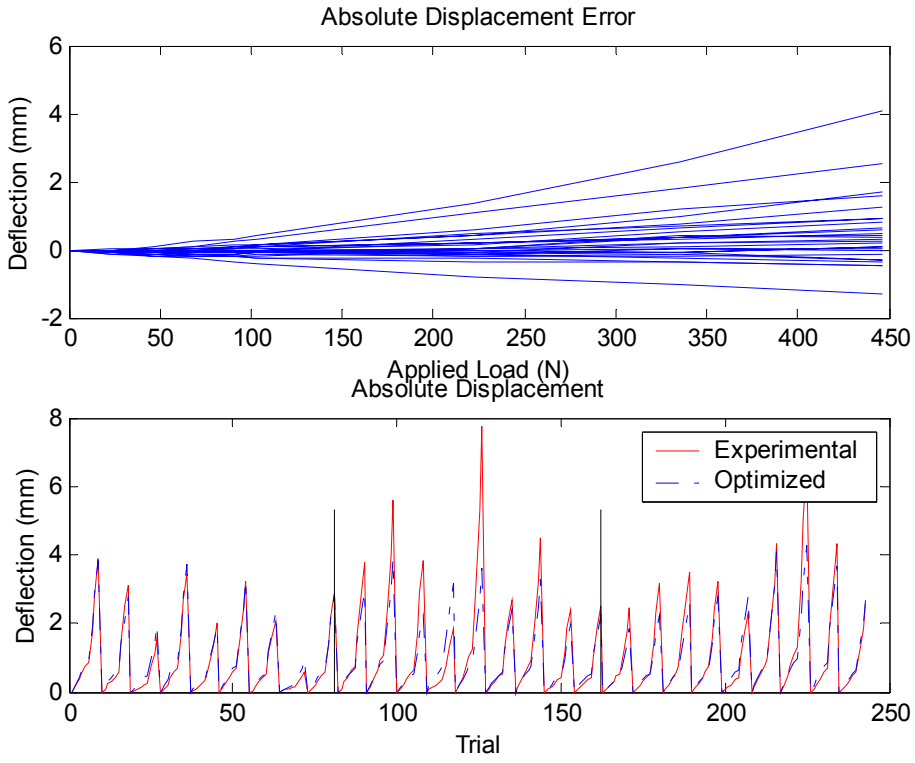


Figure 7.32 Third Regime Full Load Absolute Displacement Plot

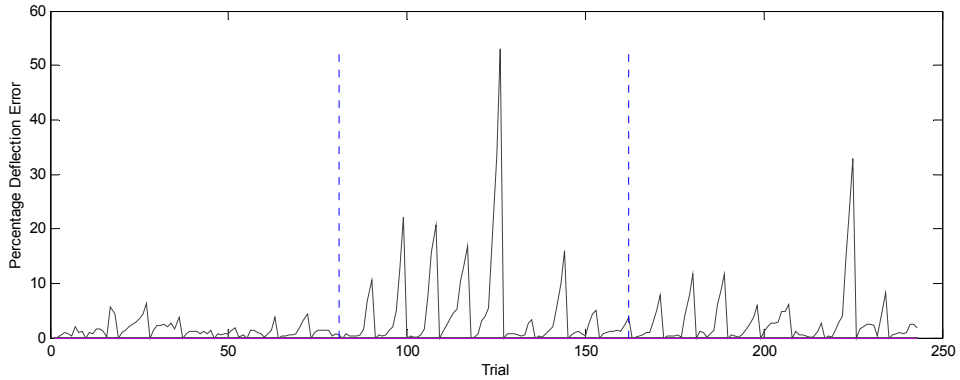


Figure 7.33 Second Regime Full Load Set Percentage Error

The rotations for this regime, shown in appendix section 10.1.3 , were worse about the X-axis when compared with the unweighted solution. However, both the Y-axis and Z-axis rotations showed slight improvements. As shown in the final stiffness matrix, Table 7.13, the model is a function of virtually all stiffness terms.

Table 7.13 Third Regime Full Load Stiffness Matrix

DH Frame	X	Y	Z
0	N/A	N/A	3.2658E8
1	1.2940E8	3.8934E8	1.8579E8
2	6.2831E7	N/A	7.6581E7
3	1.3374E8	1.0594E5	1.4518E7
4	1.1126E7	2.9816E6	5.8924E6
5	1.6199E7	5.8914E6	1E10
6	1.0084E7	N/A	1E10

7.3.4 Results for Regime 1 Reduced Load Range

As with the unweighted solutions, reducing the load set improved the performance over the desired range. The final objective function was 999.94, again within two points of the unweighted solution. The absolute displacement in Figure 7.34 performs essentially the same as the results seen in the unweighted solution. The percentage error was reduced in the last third of the tests, but increased in both other thirds relative to the unweighted solutions, as presented in Figure 7.35.

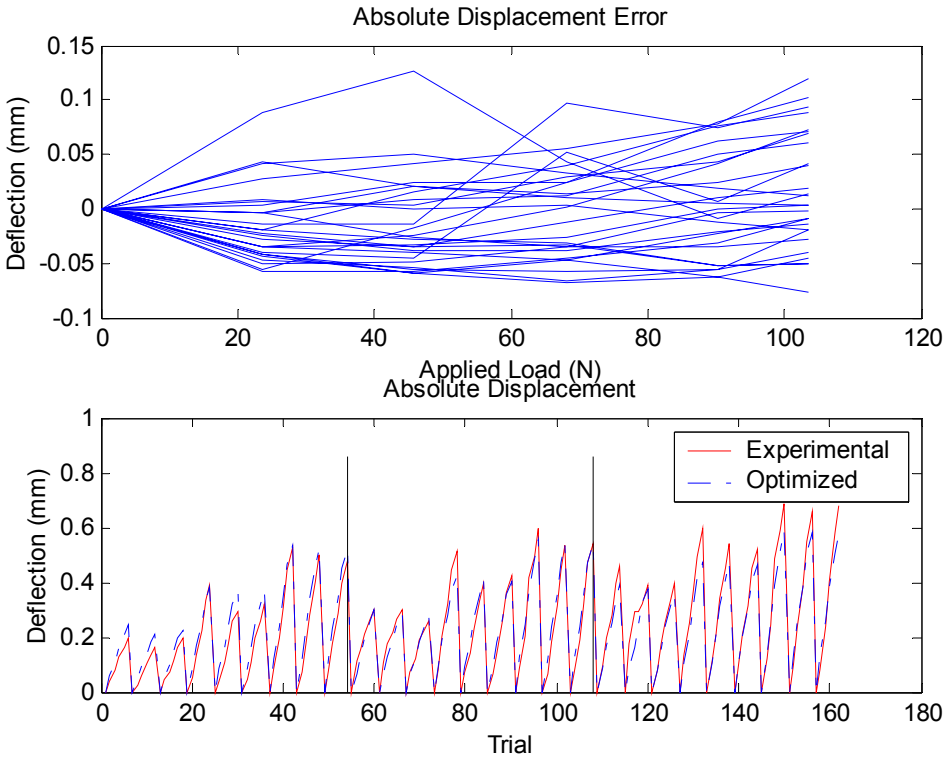


Figure 7.34 First Regime Reduced Load Absolute Displacement Plot

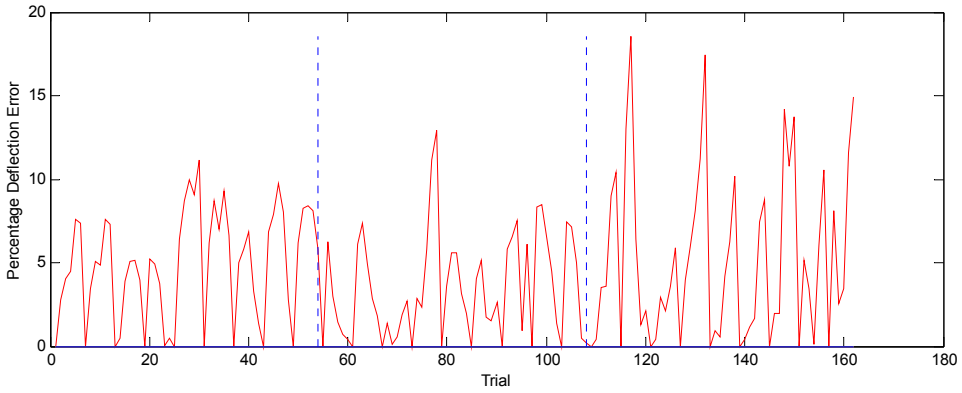


Figure 7.35 First Regime Reduced Load Set Percentage Error

The rotations, shown in appendix section 10.1.4 , show no significant changes relative to the unweighted regime. The final stiffness matrix is shown in Table 7.14, and as expected is only dependent on two of the stiffness terms.

Table 7.14 First Regime Reduced Load Stiffness Matrix

DH Frame	X	Y	Z
0	N/A	N/A	1E10
1	1E10	1E10	2.1128E8
2	1E10	N/A	4.9384E7
3	1E10	1E10	1E10
4	1E10	1E10	1E10
5	1E10	1E10	1E10
6	1E10	N/A	1E10

7.3.5 Results for Regime 2 Reduced Load Range

The second regime again shows a significant improvement over the unweighted solution. The final objective function value was 1007, compared to 1023 for the unweighted result. This difference is immediately noticeable in the absolute displacement plot, Figure 7.36. The 111 N (25 lb) loads are predicted by the weighted solution up to one-tenth of a millimeter better than the unweighted solution. Furthermore, the results are centered closer about zero. That is, in the unweighted solution, the deflection error was negative for the majority of the points, however, for this case, the results have shifted closer to a zero deflection error. The percentage error, in Figure 7.37, shows the same improvements over the unweighted solution. All three sections of the plot have noticeable reductions over the unweighted solutions, resulting in average errors not exceeding ten percent for any section, with maximum errors inside twenty-five percent.

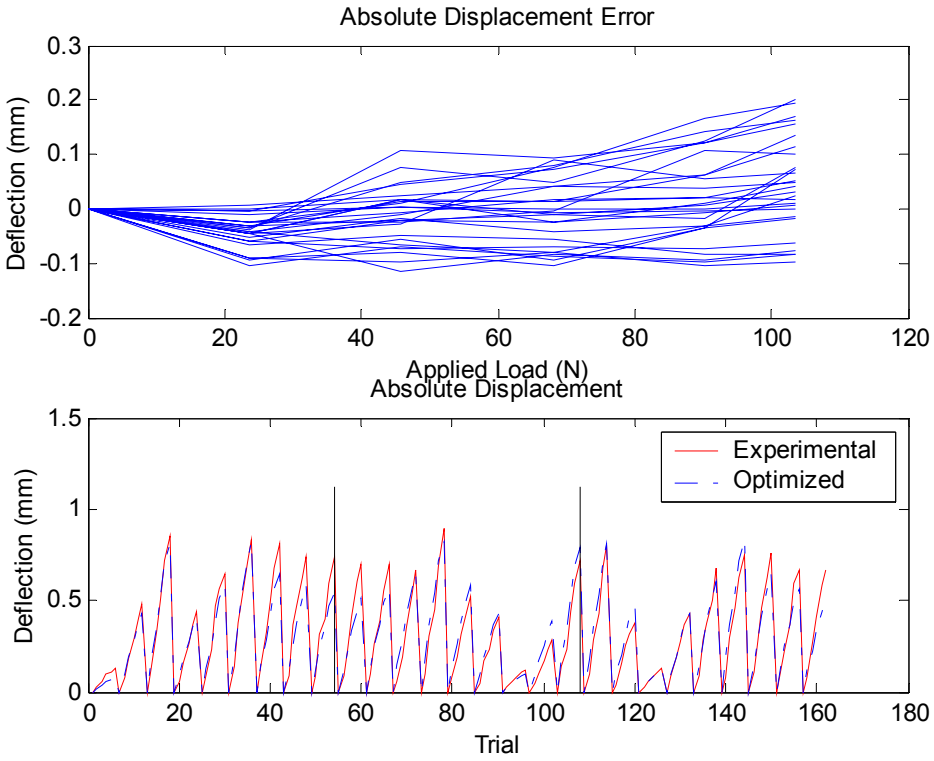


Figure 7.36 Second Regime Reduced Load Absolute Displacement Plot

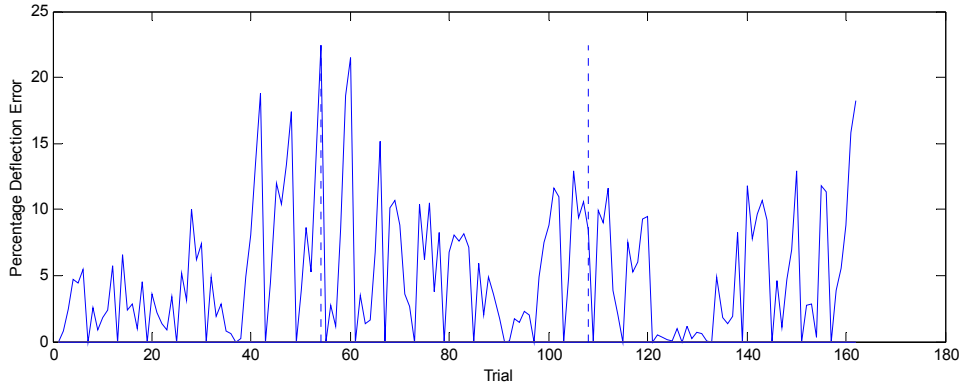


Figure 7.37 Second Regime Reduced Load Set Percentage Error

The X-axis rotations, see appendix section 10.1.4 , are comparable to the unweighted solution, and the Z-axis show slight improvements. The Y-axis rotations improve a full twenty to twenty-five percent when compared with the unweighted solution. The corresponding stiffness matrix is shown in Table 7.15.

Table 7.15 Second Regime Reduced Load Set Stiffness Matrix

DH Frame	X	Y	Z
0	N/A	N/A	1E10
1	1E10	1E10	2.2786E8
2	1E10	N/A	4.1177E7
3	1E10	1E10	1E10
4	1E10	1E10	1E10
5	1E10	1E10	1E10
6	1E10	N/A	1E10

7.3.6 Results for Regime 3 Reduced Load Range

The final objective function value for the third regime with the weighted objective function is 1034, ten points below the corresponding unweighted solution. As with the second regime, the results centered themselves better about zero, that is, the optimized solutions were distributed more evenly above and below the experimental results. This had the effect of shifting the absolute displacement plot higher relative to zero, and reducing the objective function slightly. The percentage error in the first section of Figure 7.39 shows a significant reduction over the unweighted solution, with the average error reducing upwards of fifteen percent, resulting in well less than ten percent error overall. The second and third sections show a minor increase in percentage error.

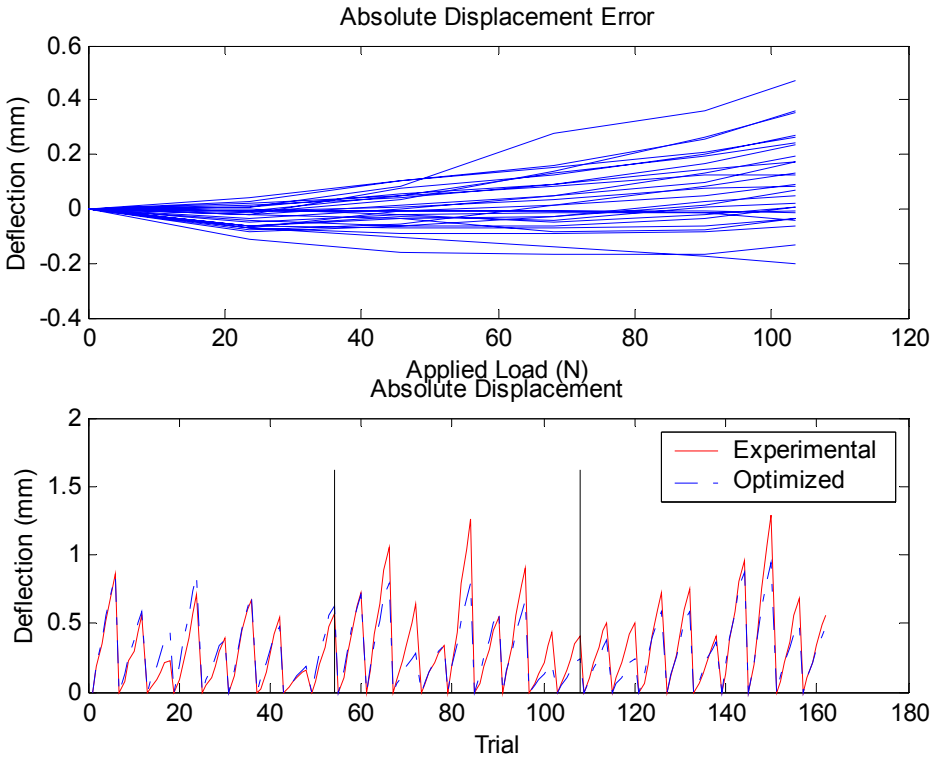


Figure 7.38 Third Regime Reduced Load Absolute Displacement Plot

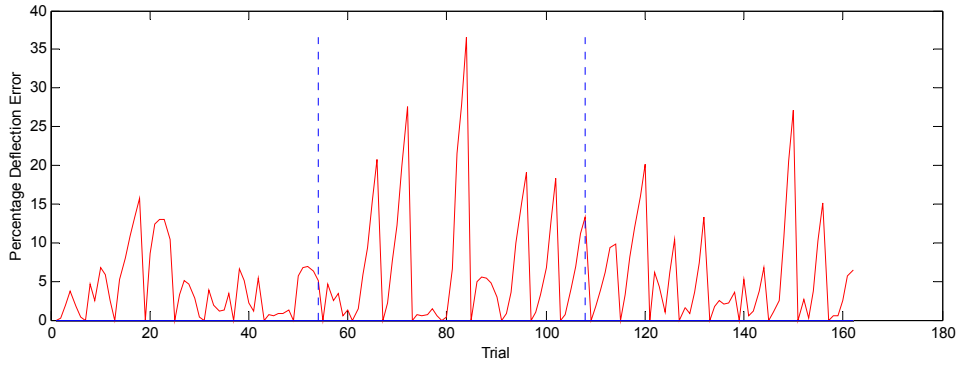


Figure 7.39 Third Regime Reduced Load Set Percentage Error

As shown in appendix section 10.1.4 , the Z-axis rotations show little change from the corresponding unweighted solution and the Y-axis rotations show a slight improvement. However, X-axis rotations are worse by a factor of two compared to the unweighted results. The stiffness matrix for this third regime solution is presented in Table 7.16.

Table 7.16 Third Regime Reduced Load Set Stiffness Matrix

DH Frame	X	Y	Z
0	N/A	N/A	1.2927E8
1	5.3067E8	2.3899E9	1.6036E8
2	5.7538E8	N/A	8.0215E7
3	4.0225E8	1.2486E8	2.1249E7
4	1.5912E7	3.6171E6	6.4615E6
5	2.2368E7	6.4616E6	1E10
6	1E10	N/A	1E10

7.4 Summary of Results

The final values of the objective functions for the various optimization solutions are summarized in Table 7.17. Comparing the effectiveness of the weighted and unweighted objective functions each produced a lower objective function value three of six times. However, the differences between the two objective function values tend to be larger when the weighted solution produced the smaller value. The inconsistency in the trends is almost certainly due to choices of initial conditions and simply the unpredictability of the optimization routine. Interestingly, regime one produced a smaller objective function value for the unweighted case for both the full and reduced load. Regime two produced smaller objective function values for the weighted case under both load cases. However, regime three split the difference producing the smallest value once for the weighted and once for the unweighted solution.

Table 7.17 Summary of Objective Function Values

	Weighted Objective Function	Unweighted Objective Function
Full Load		
Regime 1	1557.00	1555.37
Regime 2	1615.00	1650.27
Regime 3	1827.00	1808.63
Reduced Load		
Regime 1	999.94	997.44
Regime 2	1007.00	1023.11
Regime 3	1034	1044.20

Chapter 8: Conclusions

The goal of this research was to develop a model for a 6-DOF robotic manipulator capable of predicting the compliant behaviors of the end-effector under varied loads. Based on three regimes of testing and numerical optimization, results were generated and quantified with regard to overall effectiveness. Common to optimization methods, the final solution is dictated by time and necessity rather than finding the global minimum, or the ideal solution.

The results show that given a 6-DOF robotic manipulator with spherical joints, the technique presented in this thesis can reduce the positioning error by greater than eighty-five percent on average when compared with an infinitely stiff ideal model. The compliant model is able to reduce errors from five millimeters to below one millimeter in end-effector position. Rotational errors can be minimized to less than one degree over a wide range of loads, and less than one-quarter of a degree for typical load sets. Furthermore, the solution has been developed in such a way to allow implementation into a robot controller once the compliance of the manipulator has been characterized.

The overall technique has been developed in as general a framework as possible. While the forward and inverse kinematics are manipulator specific, the technique has been documented with the intent of easy transition to other manipulators. Furthermore, the use of a six-degree of freedom manipulator for this research provides a complex solution, when compared with many other manipulators. The development of torques due to loading and the compliant model are generalized to enable application to other manipulators. The optimization technique is a general technique without customization beyond the objective function. Throughout the research, several important conclusions regarding the flexibility of the method arose. Depending on the design of the end-effector, a slight misalignment could be inconsequential, or magnify due to an offset and become a noticeable error. In order to avoid this type of complication, it is recommended that end-effector tools be designed such that the contact point with the surface is located as close to the tool plate as possible.

The design provides reasonable rotation tracking, however, as the rotational errors increase, it becomes increasingly critical to have the contact point of any tool near the

faceplate of the end-effector. Furthermore, when choosing the workspace in which to perform tests, it is important to test outside the region of interest because in many cases, the worst points existed on the outer edge of the tested workspace. This research verified the assumption that rotational and displacement errors track together. Interestingly, almost all solutions had an error of five to ten percent between the experimental and optimized solution. This error is independent of the maximum deflection or other experimentally specific terms. Therefore, this resultant error is likely a function of the cumulative error from the wide distribution of the experimental data points.

Chapter 9: References

[1] Dowling, Norman, E. *Mechanical Behavior of Materials.* Prentice Hall. Englewood Cliffs, New Jersey. 1993. pp. 85 – 91.

[2] Craig, John J. *Introduction to Robotics: Mechanicals and Control. 2nd ed.* Addison-Wesley Publishing Company, Inc. 1989.

[3] Spong, Mark W. and Vidyasagar, M. *Robot Dynamics and Control*. John Wiley and Sons. New York. 1989.

[4] "Products: Viper FPS-3000." http://www.cinmach.com/products/adva_set.htm (15 November 2001).

[5] Loos, Alfred C. and Shih, Po-Jen. *On-Line Consolidation of Thermoplastic Composites*. Center for Composite Materials and Structures, Virginia Tech. Blacksburg, VA. 1997.

[6] "Products: OPTOTRAK" http://www.ndigital.com/OPTOTRAK.html (15 November 2001).

[7] Vanderplaats, Garret N. *Numerical Optimization Techniques for Engineering Design. 3rd Ed.* Vanderplaats Research and Development Inc. 1999. pp. 84 – 120.

[8] Pieper, D. and Roth, B. "The Kinematics of Manipulators Under Computer Control." *Proceedings of the Second International Congress on Theory of Machines and Mechanisms.* Vol. 2. Zakopane, Poland, 1969. pp 159 – 169.

[9] Lozinski, Chrisopher A., Rourke, Jonathan M, and Whitney, Daniel E. "Industrial Robot Forward Calibration Method and Results" 1986 *Transaction of the ASME, Journal of Dynamic Systems, measurement and Control.* Vol 108. pp. 1 - 8.

[10] Derby, Stephen. *The Deflection and Compensation of General Purpose Robot Arms*. Mechanism and Machine Theory Vol 18, No 6, Pages 445-450. 1983.

[11] Bodur, I.N., and Derby, S.J. *Analysis and Modeling of the Positioning Inaccuracy of Industrial Manipulators in Off-Line Programming Part 1: Due to Static Forces and Joint Clearances*. Proceedings of the 20th Biennial Mechanisms Conference. DE-Vol 15, NO 3. pp. 383-391. 1969.

[12] Chiang, Hsiang-Dih and Cui, Xui Xun, and Lin, Paul P. *An Improved Method for On-Line Calculation and Compensation of the Static Deflection at a Robot End – Effector*. Journal of Robotic Systems, Vol 8, No 2. pp. 267 – 288.

[13] Mooring, B. W. *The Effect of Joint Axis Misalignment on Robot Positioning Accuracy*. In the Proceedings of the International Computers in Engineering Conference, Pages 151-155. ASME, New York, 1983.

[14] Mooring B. W. and Tang, G. R. *An Improved Method for Identifying the Kinematic Parameters in a Six Axis Robot*. In the Proceedings of the International Computers in Engineering Conference, Pages 79-85. ASME, New York, 1984.

[15] Demeester F. and Brussel, H. Van. *Experimental Compliance Breakdown of Industrial Robots.* Journal of Mechanical Design. Vol 116 Num 4 Dec 1994. pp 1065-1072.

[16] Stone, Henry. *Kinematic Modeling, Identification, and Control of Robotic Manipulators.* Kluwer Academic Publishers, 1987.

[17] Gong, Chunhe, Yuan, Jingxia, and Ni, Jun. *Nongeometric error identification and compensation for robotic system by inverse calibration.* International Journal of Machine Tools & Manufacture. Vol 40 Num 14 Sep 2000. pp 2119-2137.

[18] Shetty, Bharath Ram. Ang, Marcelo H Jr., *Active compliance control of a PUMA 560 robot*. Proceedings of the IEEE International Conference on Robotics and Automation. Vol 4 1996. IEEE, Piscataway, New Jersey. pp 3720-3725.

[19] Caenen, J L. Angue, J C. *Identification of geometric and non geometric parameters of robots*. Proc 1990 IEEE International Conference on Robotics and Automation. Published by IEEE, Computer Society, Los Alamitos, CA, USA. p 1032-1037.

[20] Whitney, E. Daniel, Lozinski, Christopher M, Rourke, Johnathan M. Industrial Robot Calibration Method and Results. Journal of Dynamic Systems Measurement. Vol 108. Num 1. March 1982. pp 1 – 8.

[21] Calkins, Joseph M. Canfield, Stephen L. Salerno, Robert J and Reinholtz, Charlse F. *An Optimization-Based, Matrix Operator Approach to Real-Time Compensation of Static Deflection in Robotic Manipulators.* Applied Mechanisms Conference. 1995.

[22] Suh. C. and Radcliffe, C. W., *Kinematics and Mechanism Design*, John Wiley & Sons, 1978.

[23] Stewart, James. *Calculus.* Brooks/Cole Publishing Company 1991 pp 665 - 670.

[24] Optimization Toolbox: For Use with MATLAB. http://www.mathworks.com/access/helpdesk/help/pdf_doc/optim/optim_tb.pdf (07 November 2001)

Chapter 10: Appendix

Detailed Appendix Table of Contents

10.1 Rotational Plots

10.1.1 Full Load Case, Unweighted Objective Function

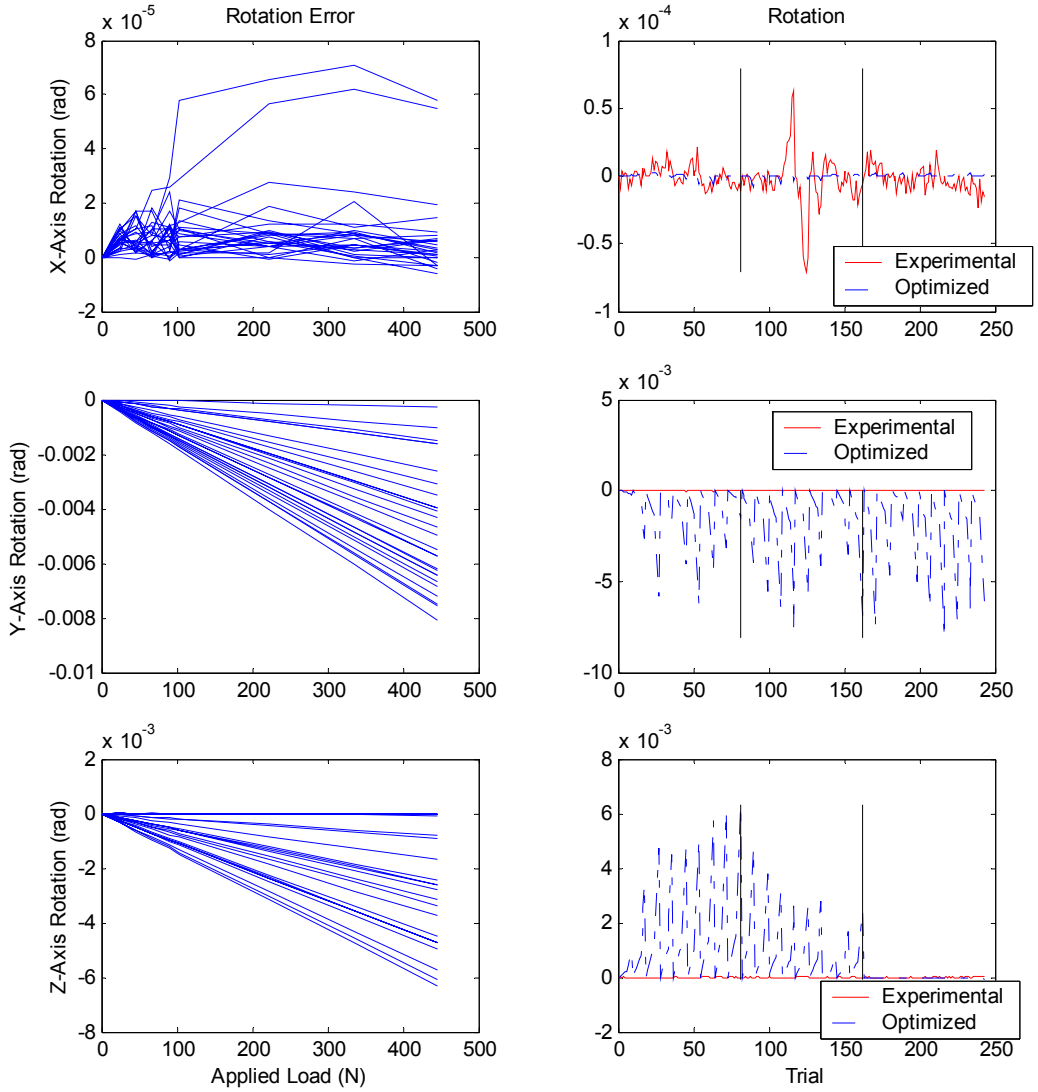


Figure 10.1 Rotational Error for Second Regime, Full Load Case, Unweighted Objective Function

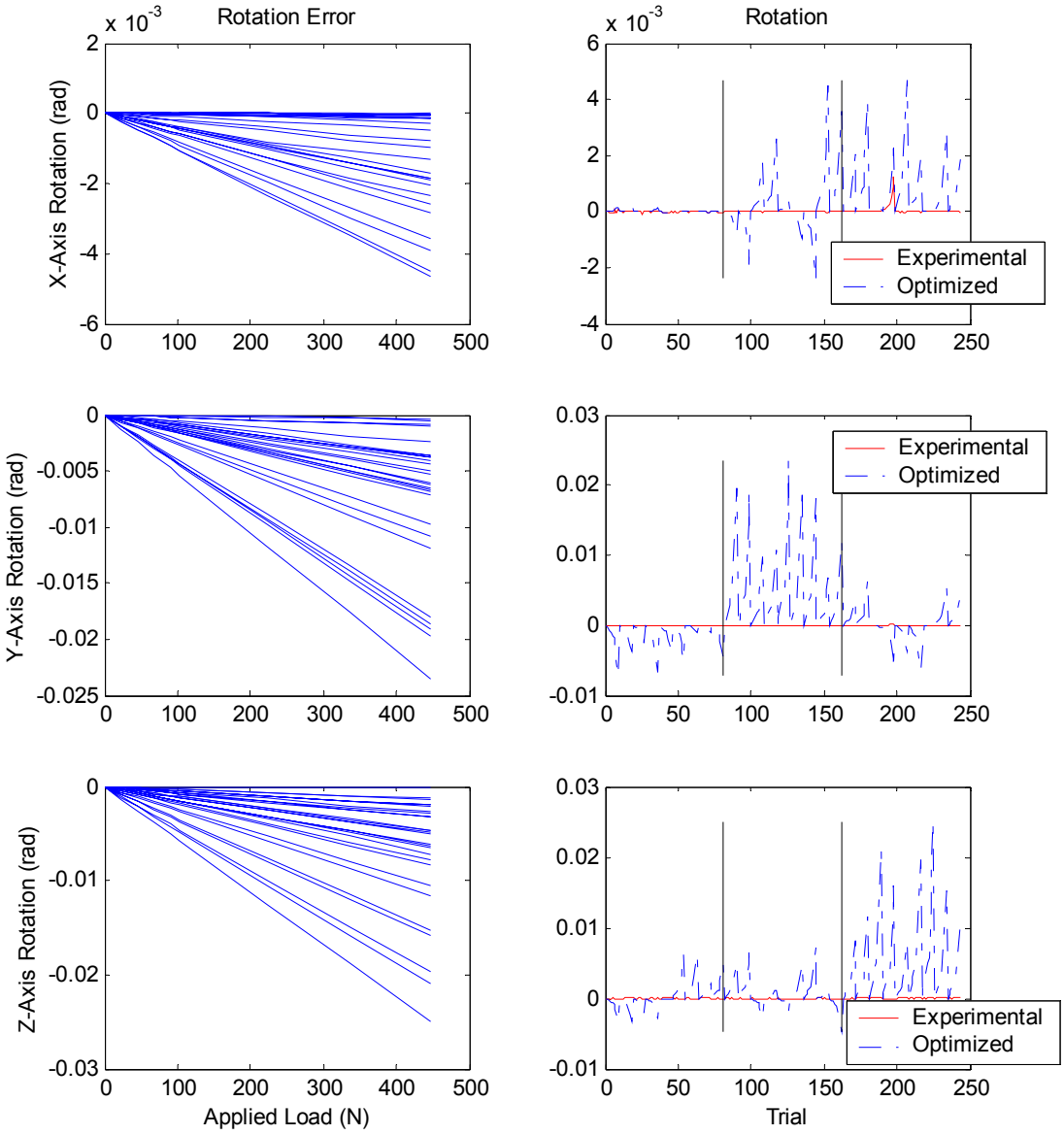


Figure 10.2 Rotational Error for Third Regime, Full Load Case, Unweighted Objective Function

10.1.2 Reduced Load Case, Unweighted Objective Function

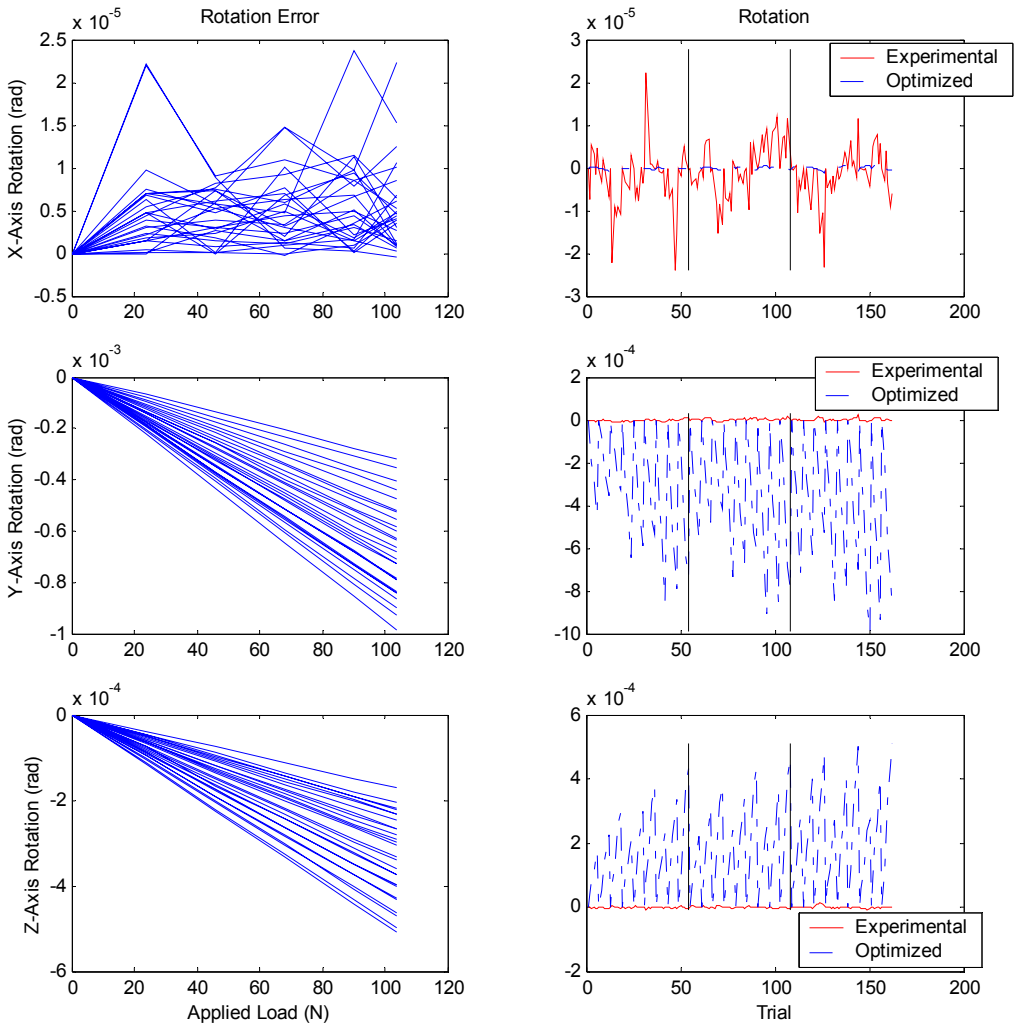


Figure 10.3 Rotational Error for First Regime, Reduced Load Case, Unweighted Objective Function

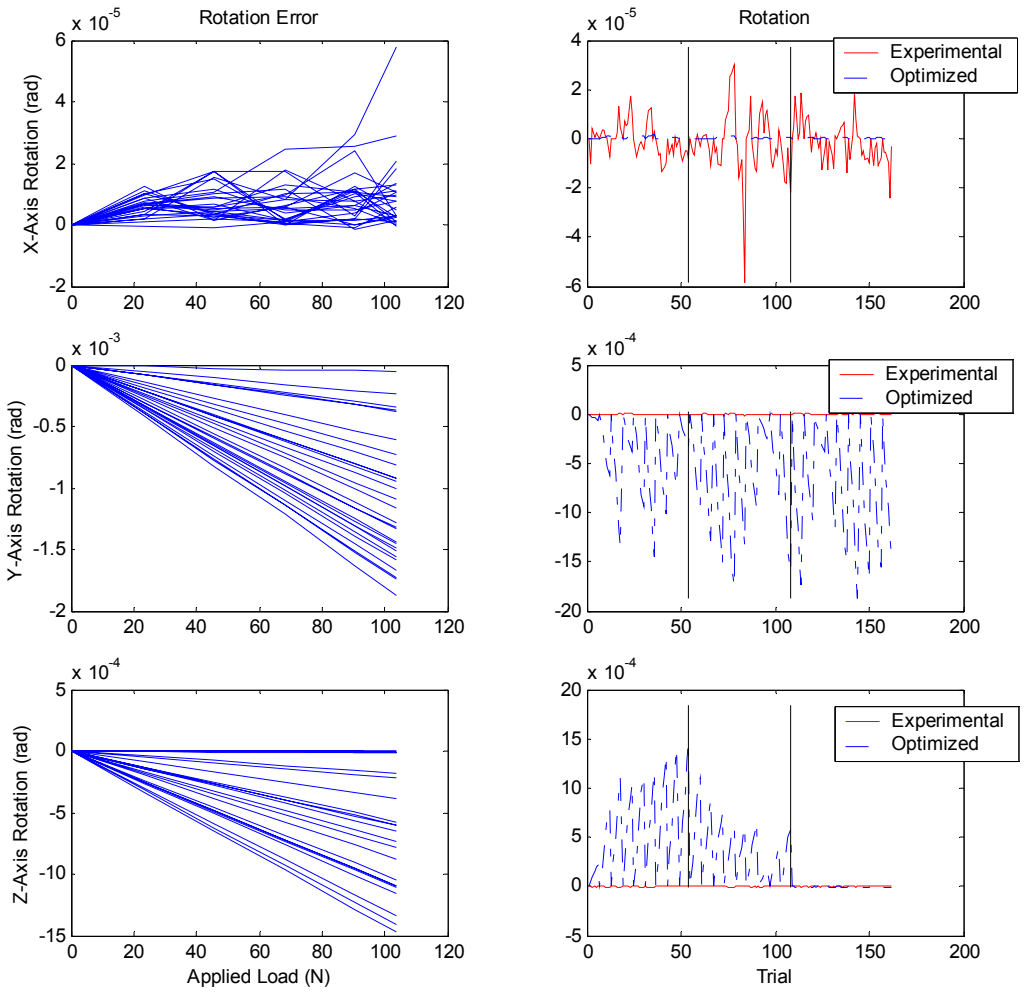


Figure 10.4 Rotational Error for Second Regime, Reduced Load Case, Unweighted Objective Function

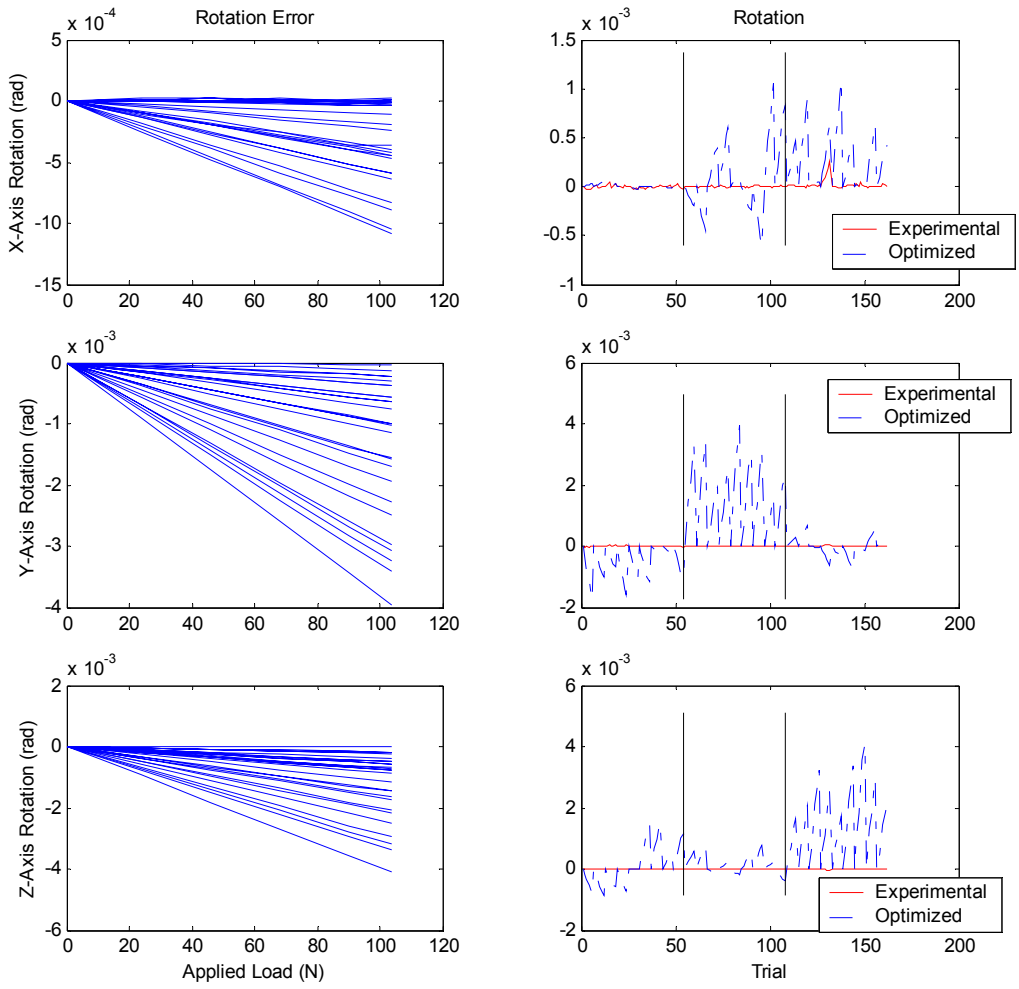


Figure 10.5 Rotational Error for Third Regime, Reduced Load Case, Unweighted Objective Function

10.1.3 Full Load Case, Weighted Objective Function

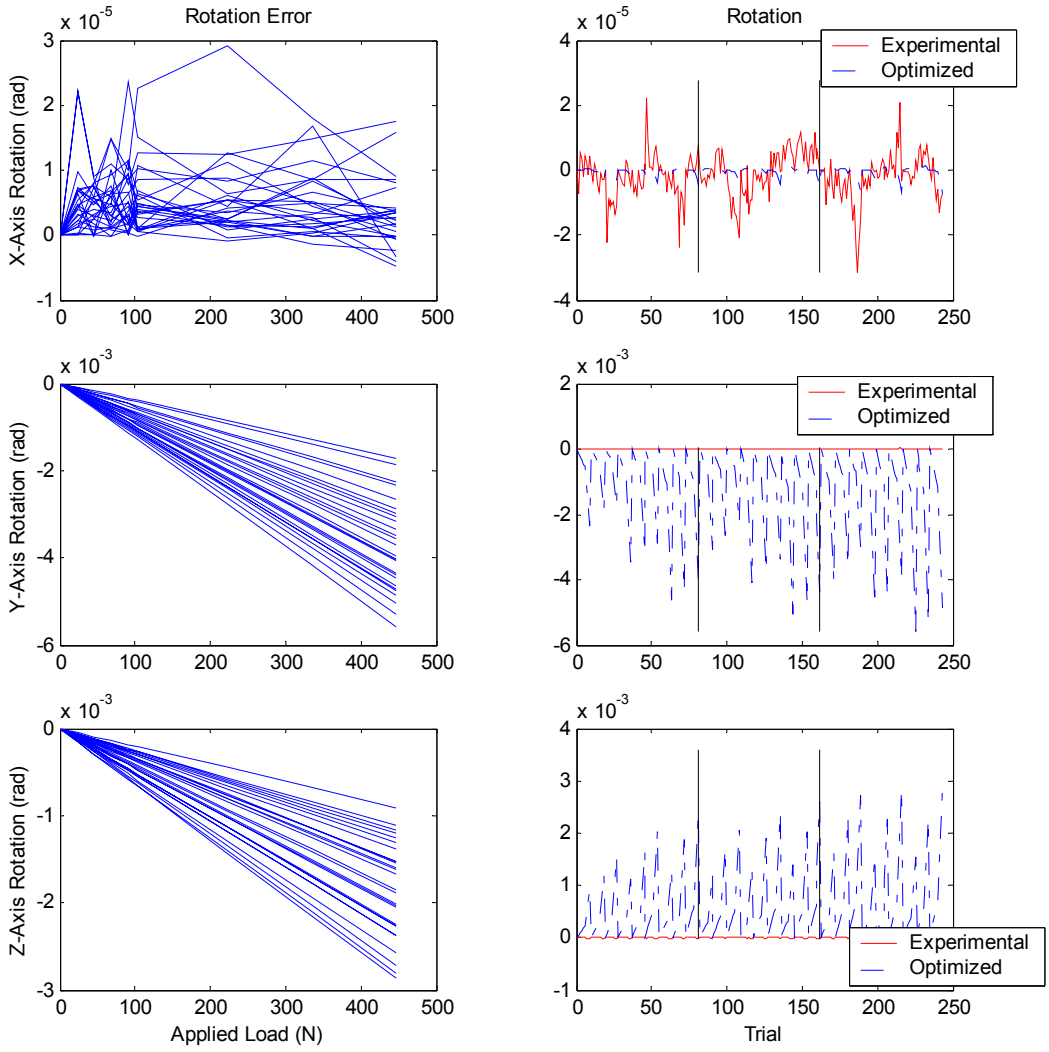


Figure 10.6 Rotational Error for First Regime, Full Load Case, Weighted Objective Function

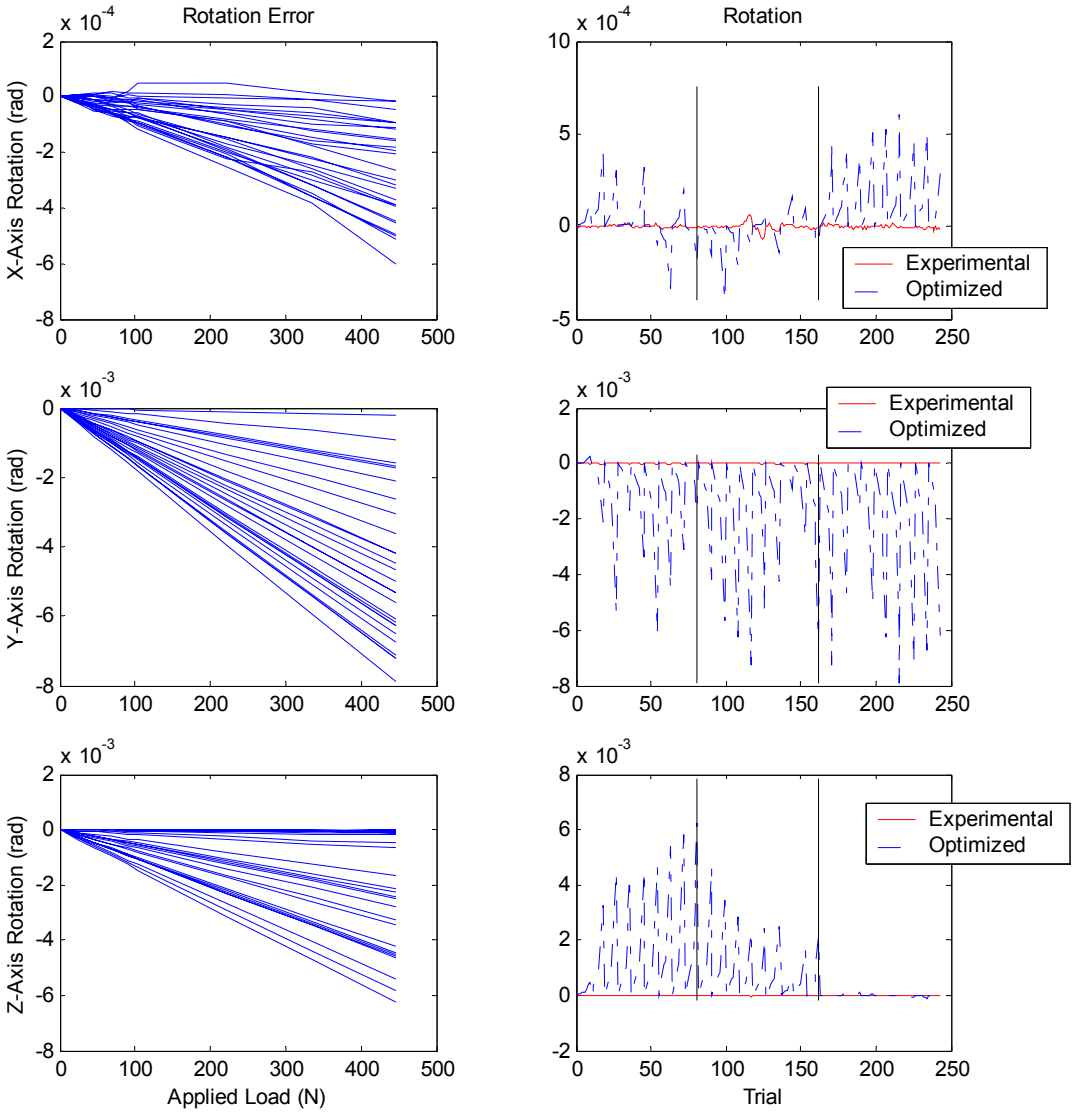


Figure 10.7 Rotational Error for Second Regime, Full Load Case, Weighted Objective Function

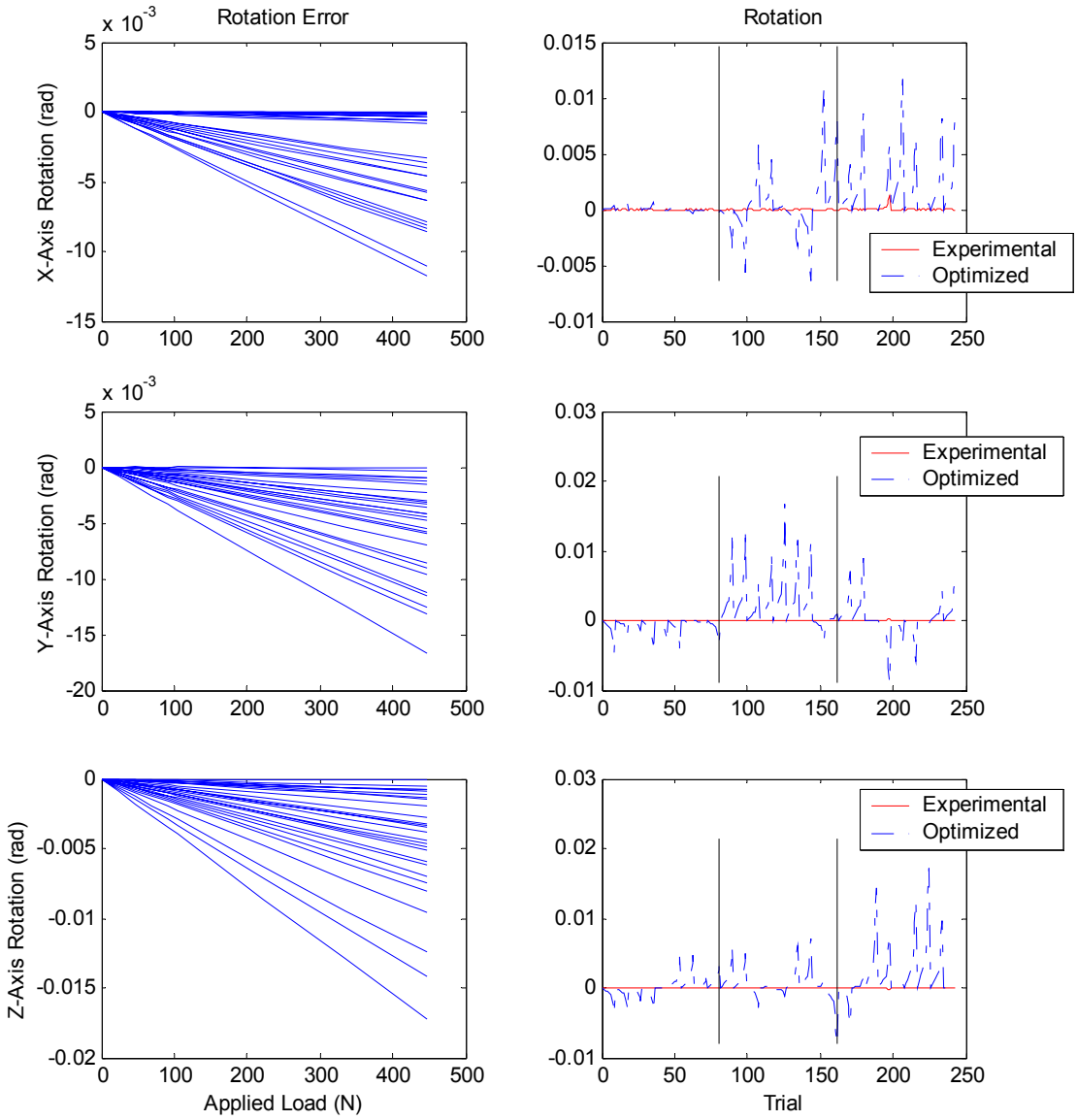


Figure 10.8 Rotational Error for Third Regime, Full Load Case, Weighted Objective Function

10.1.4 Reduced Load Case, Weighted Objective Function

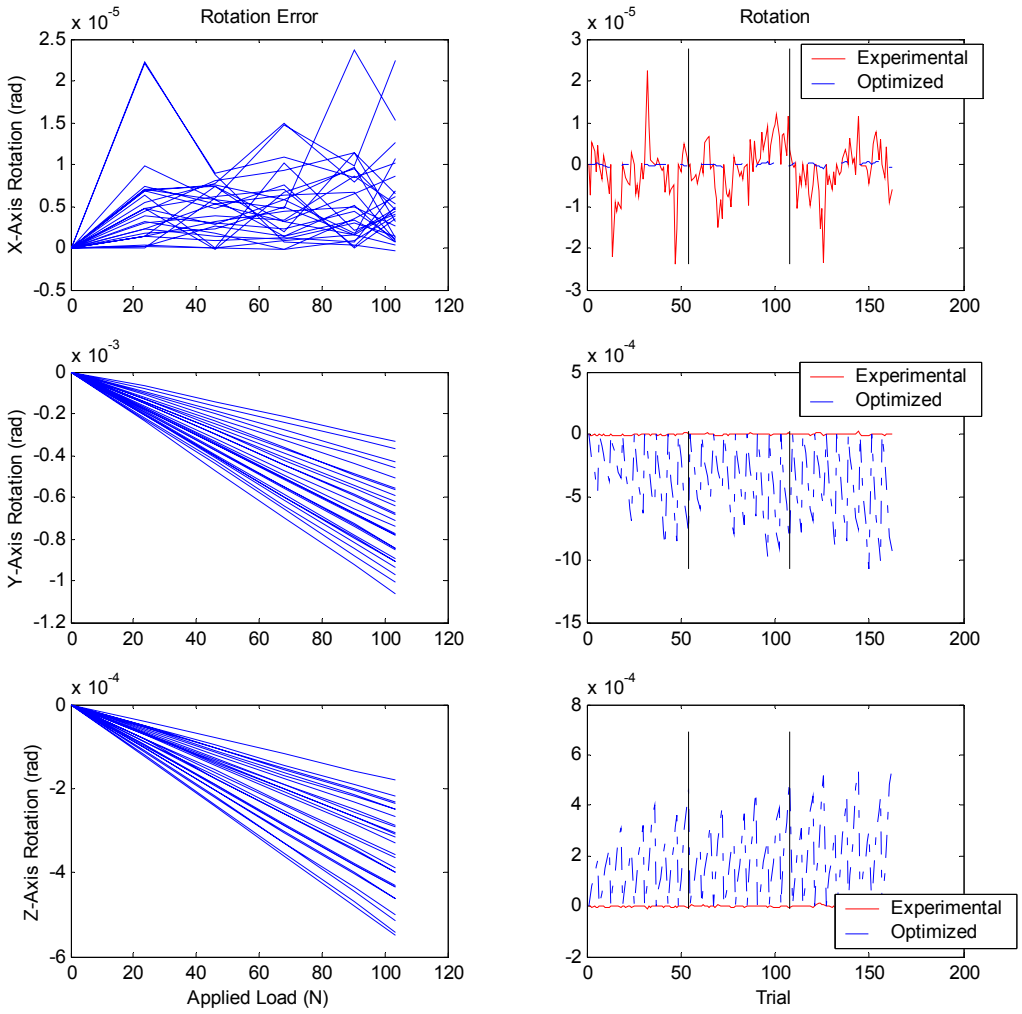


Figure 10.9 Rotational Error for First Regime, Reduced Load Case, Weighted Objective Function

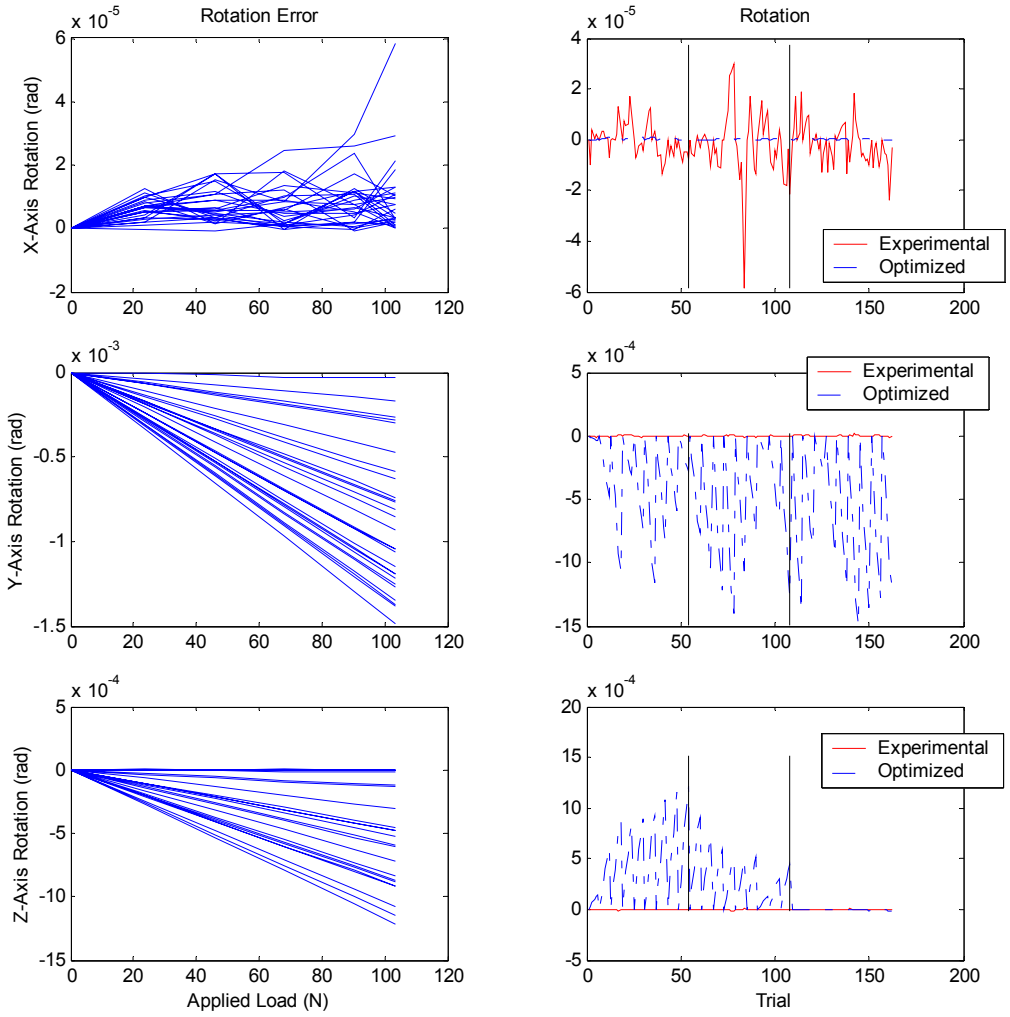


Figure 10.10 Rotational Error for Second Regime, Reduced Load Case, Weighted Objective Function

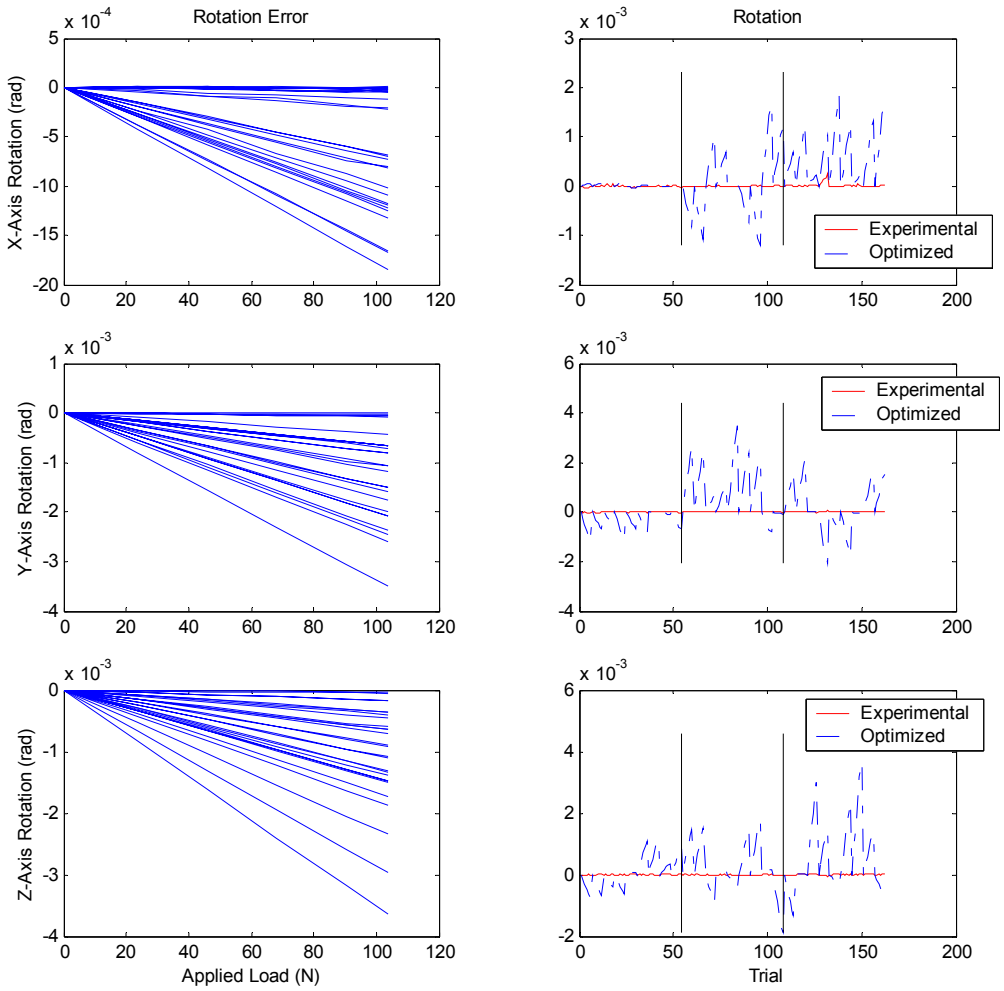


Figure 10.11 Rotational Error for Third Regime, Reduced Load Case, Weighted Objective Function

10.2 Preliminary Test Codes

10.2.1 Elbow-Up Elbow-Down Torque Code

- Code Filename: eup_edn_torque_v031002.m
- Code Description: Modified Torque code to compute the applied torques based on a variable arm weight location.
- Inputs:
  i. Theta joint angles
  ii. Cartesian load applied to end-effector
  iii. Torques (about Cartesian axes) applied to end-effector
  iv. Location of weight frame 2
  v. Location of weight frame 3
- Outputs:
  i. Torque matrix
- Application: Used for sensitivity study on locations of weight frames relative to end-effector position.
- Notes: This code is a modified version of the basic torque code. The changes allow a user passed location for weight frames 2 and 3 to be passed around the loading of Constant.mat.

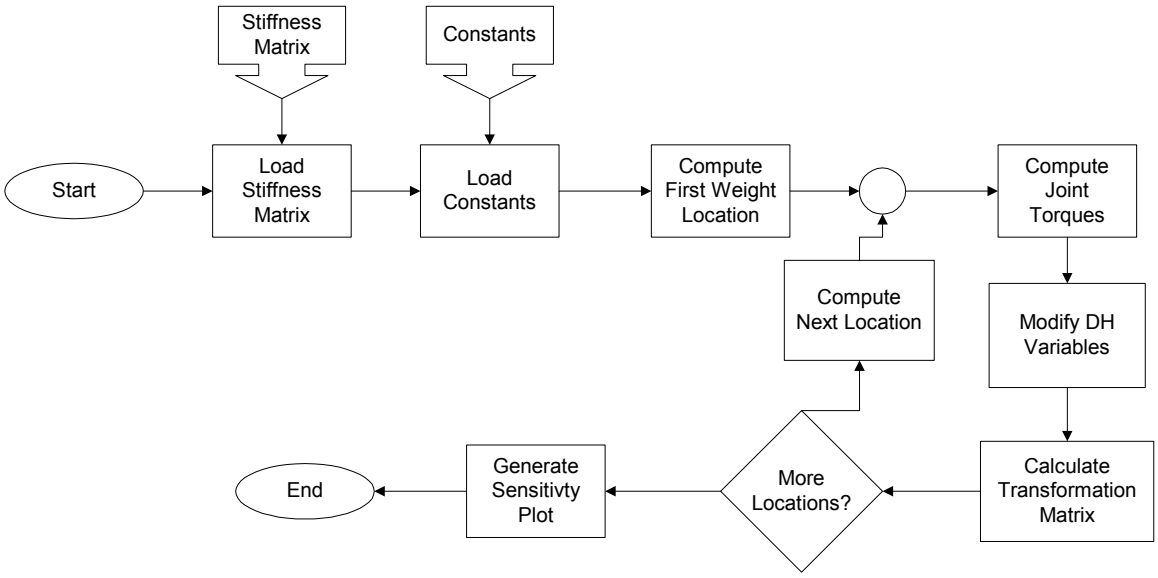
Only modifications from the torque.m code in 10.5.13 are shown here.

```
function [Torque] =
eupedn_torque_v031002(t,Loadend,FLAG,apptor,dx2,dz3);

% t = theta values for computation (radians)
% disp = Locations of weight frames relative to 'primary' link
frame
% Loadend = Applied Load at end-effector

% ------------------------------------------------------------------
% User definable Settings

load c:\data\Constant.mat

dx(2) = dx2;
dz(3) = dz3;
% ------------------------------------------------------------------
```

10.2.2 Elbow-Up Elbow-Down Plot Routine

- Code Filename: eup_edn_v031002.m
- Code Description: Front-end code to generate plots
- Inputs: N/A
- Outputs: Plots

- Application: Generate plots to view the effect of a misplaced weight on the arm of the manipulator
- Notes: N/A




```
% Mark W. Abbott
% 3-10-2002

% Code to predict the sensitivity to center of mass locations

clear all;
close all;
clc;

% ------------------------------------------------------------------

%K1 = 2E8*ones(4,1);
%K1(2) = 2E7;
%K2 = 9E6*ones(3,1);
%K3 = 1E8*ones(7,1);
%K4 = 2E8*ones(7,1);

%K = [[K1;K2],K3,K4];

load c:\data\r3\finalresults\r3_full_final.mat
K = x;

Theta = [0,0,0,0,-90,0]*pi/180;
apptor = zeros(1,3);
Endload = zeros(6,1);

load c:\data\Constant.mat

di = [d1;d2;0;d4;0;d6];
ai = [0;a2;0;0;0;0];
```

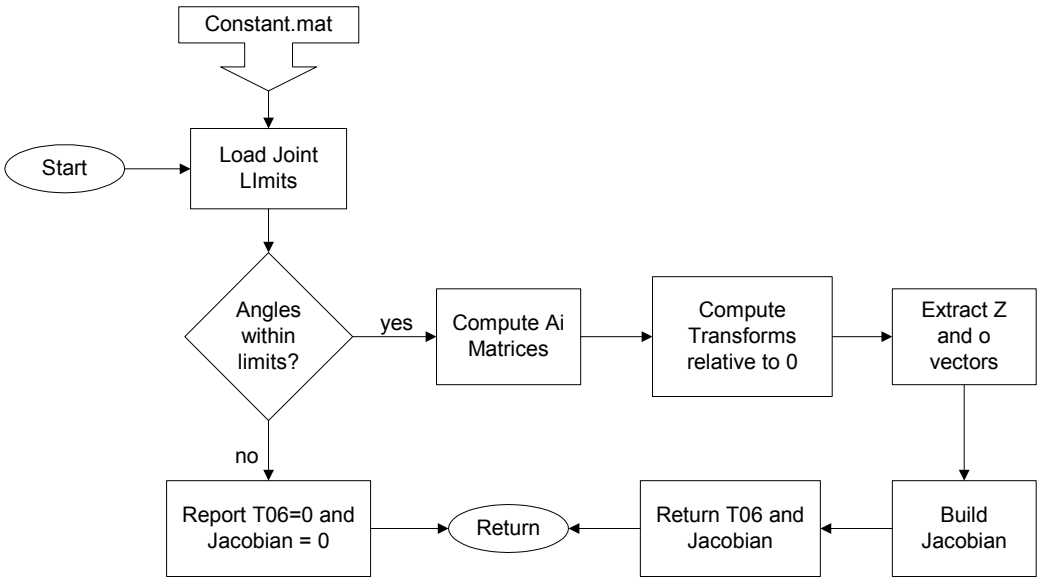
```
alphai = pi/2*[1;0;1;-1;1;0];

%-----------------------------------------------------------------

OPTIONS = [];

dx2 = dx(2);
dz3 = dz(3);
x = zeros(11,1);
y = zeros(11,1);
z = zeros(11,1);
dx2axis = zeros(11,1);
dz3axis = zeros(11,1);

for k = 1:11

    dx(2) = dx2 - 2.5 + (k - 1)*0.5;
    dx2axis(k) = dx(2);

%    dz(3) = dz3 - 2.5 + (k - 1)*0.5;
%    dz3axis(k) = dz(3);

    Jacobian = zeros(6,6);
    Jtorque = zeros(3,6);
    ModTheta = zeros(6,1);
    ModPos = zeros(3,1);
    Kx = K(:,1);
    Ky = K(:,2);
    Kz = K(:,3);

    % Calculate the Joint Torques

    Jtorque =
eup_edn_torque_v031002(Theta,Endload,OPTIONS,apptor,dx(2),dz(3));

    % Calculate the Changes in Angles

    ThetaDelta = squeeze(Jtorque(:,3))./Kz;
    AlphaDelta = squeeze(Jtorque(:,1))./Kx;
    ContribDelta = squeeze(Jtorque(:,2))./Ky;
    ContribDelta = ContribDelta(2:7);

    % Set to Zero because the current model is unable to accomodate
the loads

    ContribDelta(2) = 0;
    ContribDelta(6) = 0;

    % Apply Deltas to existing D-H variables

    ModTheta = Theta' + ThetaDelta(1:6) + ContribDelta;
    ModAlpha = alphai + AlphaDelta(2:7);
    Moda = ai;
    Modd = di;

    % Calculate the new End-effector Position
```

```
    [J1,T1] =
forwardk_v022502(ModTheta,ModAlpha,OPTIONS,Moda,Modd);

    % Extract the Position Vector for Analysis

    x(k) = T1(1,4);
    y(k) = T1(2,4);
    z(k) = T1(3,4);

end

if abs(dx2axis) > 0

    dx2axis_1 = dx2axis(6);
    for i = 1:11
        dx2axis(i) = dx2axis(i) - dx2axis_1;
    end

    figure
    suptitle('Sensitivity to Link 2 Center of Mass Location');
    subplot(3,1,1); plot(dx2axis,x);
    xlabel('Displacement from Axis (mm)');
    ylabel('End-effector Position (mm)');
    subplot(3,1,2); plot(dx2axis,y);
    xlabel('Displacement from Axis (mm)');
    ylabel('End-effector Position (mm)');
    subplot(3,1,3); plot(dx2axis,z);
    xlabel('Displacement from Axis (mm)');
    ylabel('End-effector Position (mm)');

else
    dz3axis_1 = dz3axis(6);
    for i = 1:11
        dz3axis(i) = dz3axis(i) - dz3axis_1;
    end

    figure
    suptitle('Sensitivity to Link 3 Center of Mass Location');
    subplot(3,1,1); plot(dz3axis,x);
    xlabel('Displacement from Axis (mm)');
    ylabel('End-effector Position (mm)');
    subplot(3,1,2); plot(dz3axis,y);
    xlabel('Displacement from Axis (mm)');
    ylabel('End-effector Position (mm)');
    subplot(3,1,3); plot(dz3axis,z);
    xlabel('Displacement from Axis (mm)');
    ylabel('End-effector Position (mm)');

end
```

10.2.3 Basic Forward Kinematics

- Code Filename: Merlin_FK.m
- Code Description: Standard forward kinmatics solver for a 6-DOF revolute Merlin manipulator.

- Inputs:
    i. Theta joint angles
    ii. Manipulator distances and offsets
- Outputs:
    i. Jacobian matrix
    ii. Transformation Matrix between 0 and 6^{th} frame
- Application: Used to compute the end-effector position and pose assuming ideal robot assumptions.
- Notes: N/A




```
% Merlin Forward Kinematic Solver
% Mark W. Abbott
% October 5, 2001

% MATLAB code to compute the forward kinematics and Jacobian for 6-
DOF Merlin Robots
%
% Returns the Jacobian and the Transformation Matrix between frames
0 and 6
% If desired joint angles exceed mechanical limits, both output are
set to 0.
%

function [J,T06] = Merlin_FK(t,d1,FLAG,d2,a2,d4,d6);

% ------------------------------------------------------------------
% User definable Settings

t1up = 175;      % Upper Limit for Joint 1
t1lwr = -115;    % Lower Limit for Joint 1
t2up = 230;      % Upper Limit for Joint 2
t2lwr = -55;     % Lower Limit for Joint 2
t3up = 135;      % Upper Limit for Joint 3
```

```
t3lwr = -135;    % Lower Limit for Joint 3
t5up = 90;       % Upper Limit for Joint 5
t5lwr = -90;     % Lower Limit for Joint 5

% ------------------------------------------------------------------


% Test Angles to ensure they are within the mechanical limits

ta = t*180/pi;
Error = 0;

if ta(1) > t1up | ta(1) < t1lwr
    fprintf('\nTheta 1 (%5.5g) is outside the work
envelope\n',ta(1));
    Error = 1;
end

if ta(2) < t2lwr | ta(2) > t2up
    fprintf('\nTheta 2 (%5.5g) is outside the work
envelope\n',ta(2));
    Error = 1;
end

if ta(3) > t3up | ta(3) < t3lwr
    fprintf('\nTheta 3 (%5.5g) is outside the work
envelope\n',ta(3));
    Error = 1;
end

if ta(5) > t5up | ta(5) < t5lwr
    fprintf('\nTheta 5 (%5.5g) is outside the work
envelope\n',ta(5));
    Error = 1;
end

if Error == 0

    % Setting up Transformation Matrices from one frame to the next
A1 is from frame 0 to 1

    A01=[cos(t(1)),0,sin(t(1)),0;sin(t(1)),0,-
cos(t(1)),0;0,1,0,d1;0,0,0,1];
    A12=[cos(t(2)),-
sin(t(2)),0,a2*cos(t(2));sin(t(2)),cos(t(2)),0,a2*sin(t(2));0,0,1,d2
;0,0,0,1];
    A23=[-
sin(t(3)),0,cos(t(3)),0;cos(t(3)),0,sin(t(3)),0;0,1,0,0;0,0,0,1];
    A34=[cos(t(4)),0,-sin(t(4)),0;sin(t(4)),0,cos(t(4)),0;0,-
1,0,d4;0,0,0,1];
    A45=[cos(t(5)),0,sin(t(5)),0;sin(t(5)),0,-
cos(t(5)),0;0,1,0,0;0,0,0,1];
    A56=[cos(t(6)),-
sin(t(6)),0,0;sin(t(6)),cos(t(6)),0,0;0,0,1,d6;0,0,0,1];

    % Establishing Transformations from frame 0 to each frame
```

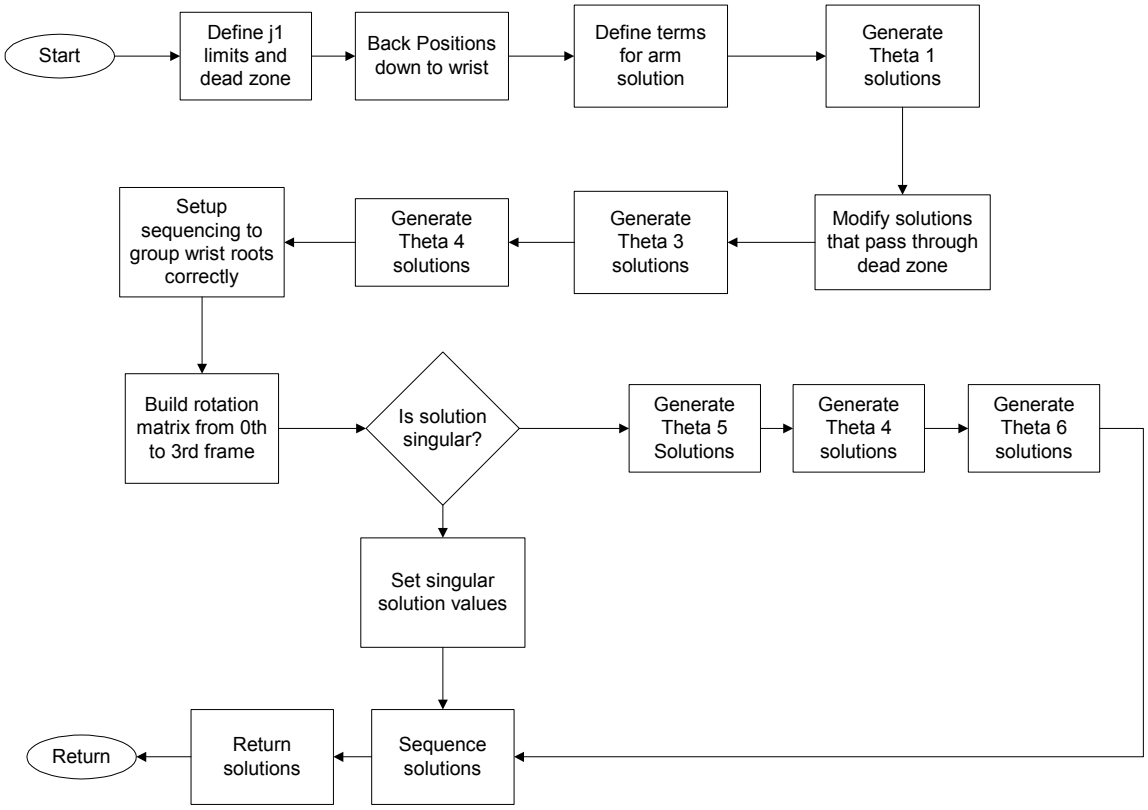
```
    T01=A01;
    T02=T01*A12;
    T03=T02*A23;
    T04=T03*A34;
    T05=T04*A45;
    T06=T05*A56;

    % Extracting z and o vectors for Jacobian Development

    z0=[0;0;1];
    z1=T01(1:3,3);
    z2=T02(1:3,3);
    z3=T03(1:3,3);
    z4=T04(1:3,3);
    z5=T05(1:3,3);
    z6=T06(1:3,3);
    o0=[0;0;0];
    o1=T01(1:3,4);
    o2=T02(1:3,4);
    o3=T03(1:3,4);
    o4=T04(1:3,4);
    o5=T05(1:3,4);
    o6=T06(1:3,4);

    % Crossing vectors to create Jacobian

    J1=[cross(z0,o6 - o0);z0];
    J2=[cross(z1,o6 - o1);z1];
    J3=[cross(z2,o6 - o2);z2];
    J4=[cross(z3,o6 - o3);z3];
    J5=[cross(z4,o6 - o4);z4];
    J6=[cross(z5,o6 - o5);z5];

    J = [J1, J2, J3, J4, J5, J6];

else
%    fprintf('Due to the Errors above this point,\n');
%    fprintf('Calculation of the Forward Kinmatics has been
terminated\n\n');
    J = 0;  T06 = 0;
end
```

10.2.4 Basic Inverse Kinematics

- Code Filename: Merlin_IK.m
- Code Description: Kinematic solver for Inverse Kinematics of Merlin 6-DOF robot.
- Inputs:
    i. Rotation Matrix for end-effector orientation
    ii. Position of end-effector in 0^{th} frame
    iii. Manipulator offsets and distances
- Outputs:

i. Up to eight sets of theta solutions

- Application: Used to solve for all mechanically possible orientations of the manipulator to reach the specified position and orientation.
- Notes: This code will return only valid solutions to the inverse kinematics problem, errors are reported to the screen for non valid solutions.




```
% Inverse Kinematic Solution Generator
% Mark W. Abbott
% October 5, 2001

% MATLAB code to compute the inverse kinematics for 6-DOF Merlin
Robots
%
% Based on a desired rotation and position vector, up to 8 solutions
are computed
%   to orient the end-effector properly.
% Singular wrist configs have been included.
%

function output = Merlin_IK(R,d,FLAG,d1,d2,a2,d4,d6);

% ------------------------------------------------------------------
% User definable Settings

t1up = 175;      % Upper Limit for Joint 1
t1lwr = -115;    % Lower Limit for Joint 1
```

```
t1dead = 70;    % Positive angle which Joint 1 cannot rotate through

% -------------------------------------------------------------------
% Clear theta variables

t1 = [0;0];
t2 = [0;0;0;0];
t3 = [0;0];
t4 = [0;0];
t5 = [0;0];
t6 = [0;0];

% Back off to wrist

px = d(1) - d6*R(1,3);
py = d(2) - d6*R(2,3);
pz = d(3) - d6*R(3,3);

% Define Terms for Arm Solution

r = pz - d1;
s = sqrt(px^2 + py^2 - d2^2);

% Generate Both solutions for Theta 1

alpha = atan2(py,px);
beta = atan2(d2,s);

t1(1) = alpha + beta;
t1(2) = alpha - beta + pi;

% Compensates for solutions which pass through the dead zone (back
right side of theta 1 motion)

if t1(1) > (t1up + t1dead)*pi/180
    t1(1) = t1(1) - 2*pi;
elseif t1(1) < (t1lwr - t1dead)*pi/180
    t1(1) = t1(1) + 2*pi;
end

if t1(2) > (t1up + t1dead)*pi/180
    t1(2) = t1(2) - 2*pi;
elseif t1(2) < (t1lwr - t1dead)*pi/180
    t1(2) = t1(2) + 2*pi;
end

% Generate Both Solutions for Theta 3

D3 = (s^2 + r^2 - a2^2 - d4^2)/(2*a2*d4);
t3(1) = atan2(sqrt(1-D3^2),D3);
t3(2) = atan2(-sqrt(1-D3^2),D3);

% Generate 4 Solutions for Theta 2

D2 = (d4^2 - s^2 - r^2 - a2^2)/(-2*sqrt(s^2 + r^2)*a2);
t2(1) = atan2(r,s) - atan2(sqrt(1-D2^2),D2);
t2(2) = atan2(r,s) - atan2(-sqrt(1-D2^2),D2);
```

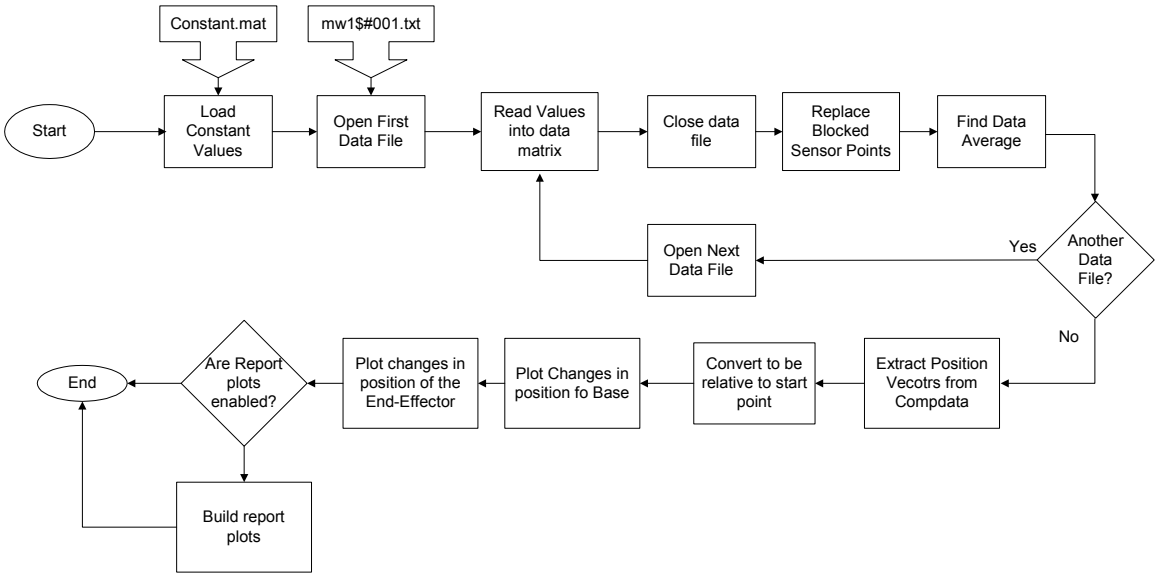
```
t2(3) = atan2(r,-s) - atan2(sqrt(1-D2^2),D2);
t2(4) = atan2(r,-s) - atan2(-sqrt(1-D2^2),D2);

% Define Terms for Wrist Solution

wrist = zeros(3,8);

% Establishing the sequencing to group roots correctly on output

for i = 1:4
    if i == 1
        a = 1; b = 1; c = 1; w1 = 1; w2 = 2;
    elseif i == 2
        a = 1; b = 2; c = 2; w1 = 3; w2 = 4;
    elseif i == 3
        a = 2; b = 3; c = 1; w1 = 5; w2 = 6;
    elseif i == 4
        a = 2; b = 4; c = 2; w1 = 7; w2 = 8;
    end

    % Establish Rotation Matrix from 0 to 3.

    A1=[cos(t1(a)),0,sin(t1(a)),0;sin(t1(a)),0,-
cos(t1(a)),0;0,1,0,d1;0,0,0,1];
    A2=[cos(t2(b)),-
sin(t2(b)),0,a2*cos(t2(b));sin(t2(b)),cos(t2(b)),0,a2*sin(t2(b));0,0,
1,d2;0,0,0,1];
    A3=[-
sin(t3(c)),0,cos(t3(c)),0;cos(t3(c)),0,sin(t3(c)),0;0,1,0,0;0,0,0,1];

    Hold = A1*A2*A3;
    R031 = Hold(1:3,1:3);
    U1 = R031'*R;

    % Check for and solve singular solution

    if abs(U1(1,3)) < 10^-10 & abs(U1(2,3)) < 10^-10
        t5(1) = 0;
        t5(2) = 0;
        t4(1) = atan2(U1(2,1),U1(1,1));
        t4(2) = 0;
        t6(1) = 0;
        t6(2) = t4(1);

  %      fprintf('Merlin_IK.m Output\n\n');
  %      fprintf('Theta 1 = %d  or %d\n',t1(1)*180/pi,t1(2)*180/pi);
  %      fprintf('Theta 2 = %d  or %d  or %d  or
%d\n',t2(1)*180/pi,t2(2)*180/pi,t2(3)*180/pi,t2(4)*180/pi);
  %      fprintf('Theta 3 = %d  or %d\n',t3(1)*180/pi,t3(2)*180/pi);
  %      fprintf('\nWARNING:  Wrist is in singular position\n\n');
  %      fprintf('Theta 5 = %d\n', t5(1)*180/pi);
  %      fprintf('Theta 4 + 6 = %d\n\n', t4(1)*180/pi);
  %      fprintf('Alternate:\n');
  %      fprintf('Theta 5 = %d\n', t5(2)*180/pi);
  %      fprintf('Theta 4 + 6 = %d\n', t4(2)*180/pi);
    else
        % Generate Both Solutions for Theta 5
```

```
        t5(1) = atan2(sqrt(1-U1(3,3)^2),U1(3,3));
        t5(2) = -t5(1);

        % Generate Both Solutions for Theta 4

        t4(1) = atan2(U1(2,3),U1(1,3));
        t4(2) = t4(1) + pi;

        % Generate Both Solutions for Theta 6

        t6(1) = atan2(U1(3,2),-U1(3,1));
        t6(2) = t6(1) + pi;

 %      fprintf('Merlin_IK.m Output\n\n');
 %     fprintf('Theta 1 = %f  or %f\n',t1(1)*180/pi,t1(2)*180/pi);
 %      fprintf('Theta 2 = %f  or %f  or %f  or
%f\n',t2(1)*180/pi,t2(2)*180/pi,t2(3)*180/pi,t2(4)*180/pi);
 %      fprintf('Theta 3 = %f  or %f\n',t3(1)*180/pi,t3(2)*180/pi);
 %      fprintf('Theta 4 = %f  or %f\n',t4(1)*180/pi,t4(2)*180/pi);
 %      fprintf('Theta 5 = %f  or %f\n',t5(1)*180/pi,t5(2)*180/pi);
 %      fprintf('Theta 6 = %f  or %f\n',t6(1)*180/pi,t6(2)*180/pi);
    end

    % Sequence solutions for transfer to main program

    wrist(:,w1) = [t4(1);t5(1);t6(1)];
    wrist(:,w2) = [t4(2);t5(2);t6(2)];
end

% Begin Solution Grouping

ans_1 = [t1(1);t2(1);t3(1);wrist(:,1)];
ans_2 = [t1(1);t2(1);t3(1);wrist(:,2)];
ans_3 = [t1(1);t2(2);t3(2);wrist(:,3)];
ans_4 = [t1(1);t2(2);t3(2);wrist(:,4)];
ans_5 = [t1(2);t2(3);t3(1);wrist(:,5)];
ans_6 = [t1(2);t2(3);t3(1);wrist(:,6)];
ans_7 = [t1(2);t2(4);t3(2);wrist(:,7)];
ans_8 = [t1(2);t2(4);t3(2);wrist(:,8)];

ans1 = [ans_1,ans_2,ans_3,ans_4,ans_5,ans_6,ans_7,ans_8];

% Specify Output

output = [ans1];
```

10.2.5 Data Reduction for Elbow-Down Trial

- Code Filename: Datareduce_elbowdn_v2.m
- Code Description: Reduces Optotrak results to meaningful data for Elbow-Down trial.
- Inputs: N/A

- Outputs:
    i. Generates plots to view results
    ii. Saves results to backup.mat file in active directory
- Application: Used to build the backup.mat from the results of the Optotrak session.
- Notes: Standard unit loads are only used for plotting purposes and do not effect the solution. This code is very similar to the corresponding Elbow-Up data reduction code.




```
% Program to reduce the data from the OptoTrak

clear all;
close all;
clc;

% User Specified Settings

numofloads = 9;
localpath = 'c:\data\elbowdn2\';

diagnosticplots = 0;            % 1 = yes, 0 = no
reportplots = 1;           % 1 = yes, 0 = no

d1 = 38.125*25.4;               % Distance from zero frame to first
frame (from floor to shoulder)
d2 = 11.9*25.4;                 % Offset from center of robot trunk
to outside of shoulder
a2 = 17.375*25.4;               % Length of Link from shoulder to
elbow
d4 = 17.25*25.4;                % Length of Link from elbow to
wrist
d6 = 3.5*25.4;                  % Distance from center of wrist to
tool faceplate
```

```
loads = [5.3;10.3;15.3;20.3;23.3;50;75.3;100.3];

% Begin Code

path1 = strcat(localpath,'backup.mat');
testval = fopen(path1);
if testval == -1

    compdata = zeros(14,3,numofloads);
    compstd = zeros(14,3,numofloads);

    for k = 1:numofloads

        % Build Filename and path into a string

        filenumber = k;
        numstr= num2str(filenumber);

        if filenumber < 100
            numstr = strcat('0',numstr);
        end
        if filenumber < 10
            numstr = strcat('0',numstr);
        end

        filename = strcat(localpath,'mw1$#',numstr,'.mwa');

        % Establish data matrix and open file

        data = zeros(14,3,200);
        fid = fopen(filename);
        fprintf('\nOpening %s', filename);
        titletrash = fgetl(fid);

        % Fill data matrix

        for i = 1:200
            for j = 1:14
                tline = fgetl(fid);
                tlinenum = str2num(tline);
                data(j,:,i) = tlinenum(length(tlinenum) -
2:length(tlinenum));
            end
        end

        fclose(fid);

        % Extract the mean value from the data matrix

        meandata = zeros(14,3);
        stddata = zeros(14,3);
        for i = 1:14
            for j = 1:3
                meandata(i,j) = mean(data(i,j,:));
                stddata(i,j) = std(data(i,j,:));
            end
```

```
        end

        compdata(:,:,k) = meandata;
        compstd(:,:,k) = stddata;

        % Remove Obstructed data Points from the compdata samples

        for i = 1:14
            for j = 1:3
                for k = 1:numofloads
                    if abs(compdata(i,j,k)) > 10^10
                        compdata(i,j,k) = 0;
                    end
                    if abs(compstd(i,j,k)) > 10^10
                        compstd(i,j,k) = 0;
                    end
                end
            end
        end

    end

    save c:\data\elbowdn2\backup.mat compdata compstd;

else
    load c:\data\elbowdn2\backup.mat
end

fclose('all');

% Building Position vectors from compdata to ease manipulation
% Initializing vectors

xpos_ef = zeros(length(loads) + 1,4);
ypos_ef = zeros(length(loads) + 1,4);
zpos_ef = zeros(length(loads) + 1,4);

xpos_b = zeros(length(loads) + 1,4);
ypos_b = zeros(length(loads) + 1,4);
zpos_b = zeros(length(loads) + 1,4);

displef = zeros(length(loads) + 1,4);

% Populating Vectors

for i = 1:length(loads) + 1
    xpos_ef(i,1:4) = compdata(1:4,1,i)';
    ypos_ef(i,1:4) = compdata(1:4,2,i)';
    zpos_ef(i,1:4) = compdata(1:4,3,i)';

    xpos_b(i,1:4) = compdata(11:14,1,i)';
    ypos_b(i,1:4) = compdata(11:14,2,i)';
    zpos_b(i,1:4) = compdata(11:14,3,i)';
end

% Changing positions to be relative to start location
```

```
for i = length(loads) + 1:-1:1
    xpos_ef(i,:) = xpos_ef(i,:) - xpos_ef(1,:);
    ypos_ef(i,:) = ypos_ef(i,:) - ypos_ef(1,:);
    zpos_ef(i,:) = zpos_ef(i,:) - zpos_ef(1,:);

    xpos_b(i,:) = xpos_b(i,:) - xpos_b(1,:);
    ypos_b(i,:) = ypos_b(i,:) - ypos_b(1,:);
    zpos_b(i,:) = zpos_b(i,:) - zpos_b(1,:);
end

% Determine the total displacement per load step

displ_ef = sqrt(xpos_ef.^2+ypos_ef.^2+zpos_ef.^2);

% Diagnostic plots of the Changes in positions of base and end-
effector sensors

if diagnosticplots == 1
    figure
    subplot(3,2,1);
plot([0;loads],xpos_ef(:,1),':',[0;loads],xpos_ef(:,2),'.-
',[0;loads],xpos_ef(:,3),[0;loads],xpos_ef(:,4));
    xlabel('Load (lb)');
    ylabel('Deflection (mm)');
    title('X Deflection of Tool Tip Sensors');
    legend('Sensor 1','Sensor 2','Sensor 3','Sensor 4',0);

    subplot(3,2,3);
plot([0;loads],ypos_ef(:,1),':',[0;loads],ypos_ef(:,2),'.-
',[0;loads],ypos_ef(:,3),[0;loads],ypos_ef(:,4));
    xlabel('Load (lb)');
    ylabel('Deflection (mm)');
    title('Y Deflection of Tool Tip Sensors');
    legend('Sensor 1','Sensor 2','Sensor 3','Sensor 4',0);

    subplot(3,2,5);
plot([0;loads],zpos_ef(:,1),':',[0;loads],zpos_ef(:,2),'.-
',[0;loads],zpos_ef(:,3),[0;loads],zpos_ef(:,4));
    xlabel('Load (lb)');
    ylabel('Deflection (mm)');
    title('Z Deflection of Tool Tip Sensors');
    legend('Sensor 1','Sensor 2','Sensor 3','Sensor 4',0);

    subplot(3,2,2);
plot([0;loads],xpos_b(:,1),':',[0;loads],xpos_b(:,2),'.-
',[0;loads],xpos_b(:,3),[0;loads],xpos_b(:,4));
    xlabel('Load (lb)');
    ylabel('Deflection (mm)');
    title('X Deflection of Base Sensors');
    legend('Sensor 11','Sensor 12','Sensor 13','Sensor 14',0);

    subplot(3,2,4);
plot([0;loads],ypos_b(:,1),':',[0;loads],ypos_b(:,2),'.-
',[0;loads],ypos_b(:,3),[0;loads],ypos_b(:,4));
    xlabel('Load (lb)');
    ylabel('Deflection (mm)');
    title('Y Deflection of Base Sensors');
```

```
    legend('Sensor 11','Sensor 12','Sensor 13','Sensor 14',0);

    subplot(3,2,6);
plot([0;loads],zpos_b(:,1),':',[0;loads],zpos_b(:,2),'.-
',[0;loads],zpos_b(:,3),[0;loads],zpos_b(:,4));
    xlabel('Load (lb)');
    ylabel('Deflection (mm)');
    title('Z Deflection of Base Sensors');
    legend('Sensor 11','Sensor 12','Sensor 13','Sensor 14',0);

    figure
    plot([0;loads],displ_ef(:,1),':',[0;loads],displ_ef(:,2),'.-
',[0;loads],displ_ef(:,4),'--',[0;loads],displ_ef(:,4));
    xlabel('Load (lb)');
    ylabel('Deflection (mm)');
    title('Total Deflection of End-effector Sensors');
    legend('Sensor 1','Sensor 2','Sensor 3','Sensor 4',0);

end


if reportplots == 1


    figure


    subplot(3,1,1);
plot([0;loads],xpos_ef(:,1),':',[0;loads],xpos_ef(:,2),'.-
',[0;loads],xpos_ef(:,3),'--',[0;loads],xpos_ef(:,4));
    ylabel('Deflection (mm)');
    title('X Deflection of Tool Tip Sensors');
    legend('Sensor 1','Sensor 2','Sensor 3','Sensor 4',0);

    subplot(3,1,2);
plot([0;loads],ypos_ef(:,1),':',[0;loads],ypos_ef(:,2),'.-
',[0;loads],ypos_ef(:,3),'--',[0;loads],ypos_ef(:,4));
    ylabel('Deflection (mm)');
    title('Y Deflection of Tool Tip Sensors');
    legend('Sensor 1','Sensor 2','Sensor 3','Sensor 4',0);

    subplot(3,1,3);
plot([0;loads],zpos_ef(:,1),':',[0;loads],zpos_ef(:,2),'.-
',[0;loads],zpos_ef(:,3),'--',[0;loads],zpos_ef(:,4));
    xlabel('Load (lb)');
    ylabel('Deflection (mm)');
    title('Z Deflection of Tool Tip Sensors');
    legend('Sensor 1','Sensor 2','Sensor 3','Sensor 4',0);
end
```

10.2.6 Data Reduction for Elbow-Up Trial

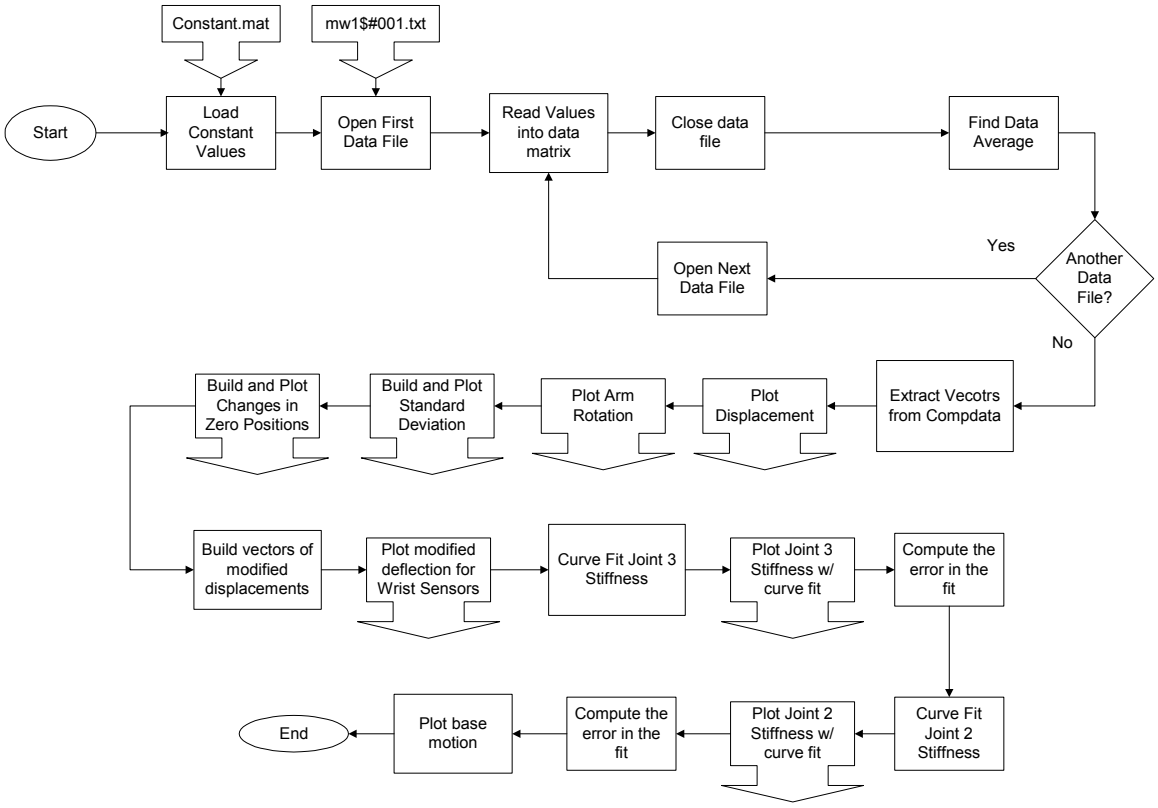
- Code Filename: Datareduce_elbowup_v2.m
- Code Description: Reduces Optotrak results to meaningful data for Elbow-Up trial.
- Inputs: N/A
- Outputs:
    i. Generates plots to view results
    ii. Saves results to backup.mat file in active directory
- Application: Used to build the backup.mat from the results of the Optotrak session.
- Notes: Standard unit loads are only used for plotting purposes and do not effect the solution. This code is very similar to the corresponding Elbow-Down data reduction code.

```
Only the differences between the Elbow-Up and Elbow-Down are shown.
Elbow-Down code is included in 10.2.5
```

```
% User Specified Settings

numofloads = 9;
localpath = 'c:\data\elbowup3\';

diagnosticplots = 0;            % 1 = yes, 0 = no
reportplots = 1;                % 1 = yes, 0 = no
```

10.2.7 Data Reduction for Stiffness Test #1

- Code Filename: Datareduce_stifft1.m
- Code Description: Code to reduce data from Optotrak for Stiffness Test 1.
- Inputs: N/A
- Outputs:
    i. Default backup.mat file
    ii. Plots for visual pleasure
- Application: Used to reduce and verify the data from Stiffness Trial #1.
- Notes: There is an error in the length of d1, however, it does not propagate to other solutions, and is not significant to the final solutions. This code is very similar to the Data Reduction Code for Stiffness Test #1.




```
% Program to reduce the data from the OptoTrak

clear all;
close all;
clc;

% User Specified Settings

numofloads = 17;
localpath = 'c:\data\stifft1d\';

d1 = 38.125*25.4;                % Distance from zero frame to first
frame (from floor to shoulder)
d2 = 11.9*25.4;                  % Offset from center of robot trunk to
outside of shoulder
a2 = 17.375*25.4;                % Length of Link from shoulder to
elbow
d4 = 17.25*25.4;                 % Length of Link from elbow to wrist
d6 = 3.5*25.4;                   % Distance from center of wrist to
tool faceplate
loads = [5.3;10.3;15.3;20.3;23.3;50;75.3;100.3];
loads2 = d4*[5.3;10.3;15.3;20.3;23.3;50;75.3;100.3]./25.4;
loads3 = d4*[5.3;10.3;15.3;20.3;23.3;50;75.3;100.3]./25.4;

% Begin Code

path1 = strcat(localpath,'backup.mat');
testval = fopen(path1);
```

```
if testval == -1

    compdata = zeros(14,3,numofloads);
    compstd = zeros(14,3,numofloads);

    for k = 1:numofloads

        % Build Filename and path into a string

        filenumber = k;
        numstr= num2str(filenumber);

        if filenumber < 100
            numstr = strcat('0',numstr);
        end
        if filenumber < 10
            numstr = strcat('0',numstr);
        end

        filename = strcat(localpath,'mw1$#',numstr,'.mwa');

        % Establish data matrix and open file

        data = zeros(14,3,200);
        fid = fopen(filename);
        fprintf('\nOpening %s', filename);
        titletrash = fgetl(fid);

        % Fill data matrix

        for i = 1:200
            for j = 1:14
                tline = fgetl(fid);
                tlinenum = str2num(tline);
                data(j,:,i) = tlinenum(length(tlinenum) -
2:length(tlinenum));
            end
        end

        fclose(fid);

        % Extract the mean value from the data matrix

        meandata = zeros(14,3);
        stddata = zeros(14,3);
        for i = 1:14
            for j = 1:3
                meandata(i,j) = mean(data(i,j,:));
                stddata(i,j) = std(data(i,j,:));
            end
        end

        compdata(:,:,k) = meandata;
        compstd(:,:,k) = stddata;

    end
```

```
    save c:\data\stifft1d\backup.mat compdata compstd;

else
    load c:\data\stifft1d\backup.mat
end

fclose('all');

% Begin Plotting Routine

vert_1 = zeros(length(loads),1);
vert_2 = zeros(length(loads),1);
vert_3 = zeros(length(loads),1);
vert_4 = zeros(length(loads),1);
zchng_5 = zeros(length(loads),1);
zchng_6 = zeros(length(loads),1);

for m = 1:length(loads)
    vert_1(m) = compdata(1,1,2*m) - compdata(1,1,1);
    vert_2(m) = compdata(2,1,2*m) - compdata(2,1,1);
    vert_3(m) = compdata(3,1,2*m) - compdata(3,1,1);
    vert_4(m) = compdata(4,1,2*m) - compdata(4,1,1);
    zchng_5(m) = compdata(5,3,2*m) - compdata(5,3,1);
    zchng_6(m) = compdata(6,3,2*m) - compdata(6,3,1);
end

figure
plot(loads,vert_1,'-.',loads,vert_2,':',loads,vert_3,'--
',loads,vert_4);
title('Unmodified Vertical Displacement for Sensors 1, 2, 3, and 4');
xlabel('Weight (lb)');
ylabel('Displacement (mm)');
legend('Sensor 1','Sensor 2','Sensor 3','Sensor 4',0);

figure
plot(loads,zchng_5,loads,zchng_6);
title('Change in Position of arm during initial stiffness trial');
xlabel('Weight (lb)');
ylabel('Displacement (mm)');
legend('Sensor 5','Sensor 6',0);

std_sq = zeros(14,1);

for i = 1:14
    for j = 1:3
        plotstd(i,j) = mean(compstd(i,j,:));
    end
    std_sq(i) = sqrt(sum(plotstd(i,:).^2));
end

figure
plot(std_sq,'o')
title('Standard Deviation');
xlabel('Sensor Number');
ylabel('Standard Deviation');
%legend('X-Dir','Y-Dir','Z-Dir');
```

```
for m = 1:length(loads) + 1
    zero_1(m) = sqrt(sum((compdata(1,1,2*m - 1) -
compdata(1,1,1)).^2));
    zero_2(m) = sqrt(sum((compdata(2,1,2*m - 1) -
compdata(2,1,1)).^2));
    zero_3(m) = sqrt(sum((compdata(3,1,2*m - 1) -
compdata(3,1,1)).^2));
    zero_4(m) = sqrt(sum((compdata(4,1,2*m - 1) -
compdata(4,1,1)).^2));
    zero_5(m) = sqrt(sum((compdata(5,1,2*m - 1) -
compdata(5,1,1)).^2));
    zero_6(m) = sqrt(sum((compdata(6,1,2*m - 1) -
compdata(6,1,1)).^2));
end

figure
plot([0;loads],zero_1,[0;loads],zero_2,[0;loads],zero_3,[0;loads],zero_
4,[0;loads],zero_5,[0;loads],zero_6);
%plot([0;loads],zero_1,[0;loads],zero_2);
title('3-D Variation in Zero Positions for Sensors 1 - 6');
xlabel('Load');
ylabel('Magnitude of Position (mm)');
legend('Sensor 1','Sensor 2','Sensor 3','Sensor 4','Sensor 5','Sensor
6',0);

% Generate Plots for report specifically

% Back off motion of joint 2 from joint 3 deflection data

compdata_mod = compdata;
theta = zeros(4,2*length(loads)+1);

for i = 1:2*length(loads)+1
    for j = 1:4
        theta(j,i) = atan2(compdata(6,3,i)-compdata(5,3,i),a2) -
atan2(compdata(6,3,1)-compdata(5,3,1),a2);
        %  Theta is for joint 2 in this case
        compdata_mod(j,1,i) = compdata(j,1,i) - a2*(1-cos(theta(j,i)))
- d4*sin(theta(j,i));   % Vertical only
    end
end

vert_1a = zeros(length(loads),1);
vert_2a = zeros(length(loads),1);
vert_3a = zeros(length(loads),1);
vert_4a = zeros(length(loads),1);
base_11 = zeros(length(loads),3);
base_12 = zeros(length(loads),3);
base_13 = zeros(length(loads),3);
base_14 = zeros(length(loads),3);

for m = 1:length(loads)
    vert_1a(m) = compdata_mod(1,1,2*m) - compdata_mod(1,1,1);
    vert_2a(m) = compdata_mod(2,1,2*m) - compdata_mod(2,1,1);
    vert_3a(m) = compdata_mod(3,1,2*m) - compdata_mod(3,1,1);
    vert_4a(m) = compdata_mod(4,1,2*m) - compdata_mod(4,1,1);
```

```
    base_11(m,1) = compdata(11,1,2*m) - compdata(11,1,1);
    base_12(m,1) = compdata(12,1,2*m) - compdata(12,1,1);
    base_13(m,1) = compdata(13,1,2*m) - compdata(13,1,1);
    base_14(m,1) = compdata(14,1,2*m) - compdata(14,1,1);
    base_11(m,2) = compdata(11,2,2*m) - compdata(11,2,1);
    base_12(m,2) = compdata(12,2,2*m) - compdata(12,2,1);
    base_13(m,2) = compdata(13,2,2*m) - compdata(13,2,1);
    base_14(m,2) = compdata(14,2,2*m) - compdata(14,2,1);
    base_11(m,3) = compdata(11,3,2*m) - compdata(11,3,1);
    base_12(m,3) = compdata(12,3,2*m) - compdata(12,3,1);
    base_13(m,3) = compdata(13,3,2*m) - compdata(13,3,1);
    base_14(m,3) = compdata(14,3,2*m) - compdata(14,3,1);
end

vert_1b = vert_1a - base_11(:,1)*d4/(18*254.4);
vert_2b = vert_2a - base_12(:,1)*d4/(18*254.4);
vert_3b = vert_3a - base_13(:,1)*d4/(18*254.4);
vert_4b = vert_4a - base_14(:,1)*d4/(18*254.4);


figure
plot(loads,vert_1b,'-.',loads,vert_2b,':',loads,vert_3b,'--
',loads,vert_4b);
title('Modified Vertical Displacement for Sensors 1, 2, 3, and 4');
xlabel('Weight (lb)');
ylabel('Displacement (mm)');
legend('Sensor 1','Sensor 2','Sensor 3','Sensor 4',0);

% Curve Fit for Joint 3

x = loads3;          % Torques
y = atan2(vert_2b,d4);  % Angles

[p,s] = polyfit(x,y,2)
[f,delta] = polyval(p,x,s)

figure
plot(x,y,'o:',x,f)
%title('Figure 5:  First Stiffness Order Characterization (Joint 3)');
xlabel('Torque (in*lb)');
ylabel('Rotation (rad)');
legend('Experimental Data','Approximate Curve',0);

%  compute the error for different curve fits
error3_2 = sum(abs(y-f));
[p1,s] = polyfit(x,y,1);
[f1,delta] = polyval(p1,x,s);
error3_1 = sum(abs(y-f1));
fprintf('\nJoint 3 Curve Fit Error:  First Order: %g     Second Order:
%g',error3_1,error3_2);

figure
plot(loads,vert_1b-vert_1,'-.',loads,vert_2b-vert_2,':',loads,vert_3b-
vert_3,'--',loads,vert_4b-vert_4);
title('Modifications to Vertical Displacement for Sensors 1, 2, 3, and
4');
```

```
xlabel('Weight (lb)');
ylabel('Displacement (mm)');
legend('Sensor 1','Sensor 2','Sensor 3','Sensor 4',0);

theta_p =
[theta(:,2),theta(:,4),theta(:,6),theta(:,8),theta(:,10),theta(:,12),th
eta(:,14),theta(:,16)];

x = loads2;
y = squeeze(theta_p(2,:))';

[p,s] = polyfit(x,y,2)
[f,delta] = polyval(p,x,s)

figure
plot(x,y,'o:',x,f)
title('Figure 5a:  First Stiffness Order Characterization (Joint 2)');
xlabel('Torque (in*lb)');
ylabel('Rotation (rad)');

%  compute the error for different curve fits
error2_2 = sum(abs(y-f));
[p1,s] = polyfit(x,y,1);
[f1,delta] = polyval(p1,x,s);
error2_1 = sum(abs(y-f1));
fprintf('\n\nJoint 2 Curve Fit Error:  First Order: %g     Second
Order: %g',error2_1,error2_2);

% plot to see motion of base points during testing

figure
plot(loads,base_11(:,1),':',loads,base_12(:,1),':',loads,base_13(:,1),'
:',loads,base_14(:,1),':',loads,base_11(:,2),'--',loads,base_12(:,2),'-
-',loads,base_13(:,2),'--',loads,base_14(:,2),'-
',loads,base_11(:,3),loads,base_12(:,3),loads,base_13(:,3),loads,base_1
4(:,3));
title('Change in Vertical Base Location for Sensors 11, 12, 13, and
14');
xlabel('Weight (lb)');
ylabel('Displacement (mm)');
legend('X - Sensor 11','X - Sensor 12','X - Sensor 13','X - Sensor
14','Y - Sensor 11','Y - Sensor 12','Y - Sensor 13','Y - Sensor 14','Z
- Sensor 11','Z - Sensor 12','Z - Sensor 13','Z - Sensor 14',0);
```

10.2.8 Data Reduction for Stiffness Test #2

- Code Filename: Datareduce_stifft2.m
- Code Description: Code to reduce data from Optotrak for Stiffness Test 2.
- Inputs: N/A
- Outputs:
    i. Default backup.mat file
    ii. Plots for visual pleasure
- Application: Used to reduce and verify the data from Stiffness Trial #2.

- Notes: There is an error in the length of d1, however, it does not propagate to other solutions, and is not significant to the final solutions. This code is very similar to the Data Reduction Code for Stiffness Test #1.

```
% Program to reduce the data from the OptoTrak

clear all;
close all;
clc;

% User Specified Settings

numofloads = 17;
localpath = 'c:\data\stifft2e\';

d1 = 38.125*25.4;                % Distance from zero frame to first
frame (from floor to shoulder)
d2 = 11.9*25.4;                  % Offset from center of robot trunk
to outside of shoulder
a2 = 17.375*25.4;                % Length of Link from shoulder to
elbow
d4 = 17.25*25.4;                 % Length of Link from elbow to wrist
d6 = 3.5*25.4;                   % Distance from center of wrist to
tool faceplate

loads = [5.3;10.3;15.3;20.3;23.3;50;75.3;100.3];
loads3 = d4*[5.3;10.3;15.3;20.3;23.3;50;75.3;100.3]./25.4;
loads2 = (a2+d4)*[5.3;10.3;15.3;20.3;23.3;50;75.3;100.3]./25.4;

% Begin Code

path1 = strcat(localpath,'backup.mat');
testval = fopen(path1);
if testval == -1

    compdata = zeros(14,3,numofloads);
    compstd = zeros(14,3,numofloads);

    for k = 1:numofloads

        % Build Filename and path into a string

        filenumber = k;
        numstr= num2str(filenumber);

        if filenumber < 100
            numstr = strcat('0',numstr);
        end
        if filenumber < 10
            numstr = strcat('0',numstr);
        end

        filename = strcat(localpath,'mw1$#',numstr,'.mwa');

        % Establish data matrix and open file
```

```
        data = zeros(14,3,200);
        fid = fopen(filename);
        fprintf('\nOpening %s', filename);
        titletrash = fgetl(fid);

        % Fill data matrix

        for i = 1:200
            for j = 1:14
                tline = fgetl(fid);
                tlinenum = str2num(tline);
                data(j,:,i) = tlinenum(length(tlinenum) -
2:length(tlinenum));
            end
        end

        fclose(fid);

        % Extract the mean value from the data matrix

        meandata = zeros(14,3);
        stddata = zeros(14,3);
        for i = 1:14
            for j = 1:3
                meandata(i,j) = mean(data(i,j,:));
                stddata(i,j) = std(data(i,j,:));
            end
        end

        compdata(:,:,k) = meandata;
        compstd(:,:,k) = stddata;

        % Remove Obstructed data Points from the compdata samples

        for i = 1:14
            for j = 1:3
                for k = 1:numofloads
                    if abs(compdata(i,j,k)) > 10^10
                        compdata(i,j,k) = 0;
                    end
                    if abs(compstd(i,j,k)) > 10^10
                        compstd(i,j,k) = 0;
                    end
                end
            end
        end

    end


    save c:\data\stifft2e\backup.mat compdata compstd;

else
    load c:\data\stifft2e\backup.mat
end

fclose('all');
```

```
% Begin Plotting Routine

vert_1 = zeros(length(loads),1);
vert_2 = zeros(length(loads),1);
vert_3 = zeros(length(loads),1);
vert_4 = zeros(length(loads),1);
zchng_5 = zeros(length(loads),1);
base_11 = zeros(length(loads),3);
base_12 = zeros(length(loads),3);
base_13 = zeros(length(loads),3);
base_14 = zeros(length(loads),3);

for m = 1:length(loads)
    vert_1(m) = compdata(1,1,2*m) - compdata(1,1,1);
    vert_2(m) = compdata(2,1,2*m) - compdata(2,1,1);
    vert_3(m) = compdata(3,1,2*m) - compdata(3,1,1);
    vert_4(m) = compdata(4,1,2*m) - compdata(4,1,1);
    zchng_5(m) = compdata(5,3,2*m) - compdata(5,3,1);
    base_11(m,1) = compdata(11,1,2*m) - compdata(11,1,1);
    base_12(m,1) = compdata(12,1,2*m) - compdata(12,1,1);
    base_13(m,1) = compdata(13,1,2*m) - compdata(13,1,1);
    base_14(m,1) = compdata(14,1,2*m) - compdata(14,1,1);
    base_11(m,2) = compdata(11,2,2*m) - compdata(11,2,1);
    base_12(m,2) = compdata(12,2,2*m) - compdata(12,2,1);
    base_13(m,2) = compdata(13,2,2*m) - compdata(13,2,1);
    base_14(m,2) = compdata(14,2,2*m) - compdata(14,2,1);
    base_11(m,3) = compdata(11,3,2*m) - compdata(11,3,1);
    base_12(m,3) = compdata(12,3,2*m) - compdata(12,3,1);
    base_13(m,3) = compdata(13,3,2*m) - compdata(13,3,1);
    base_14(m,3) = compdata(14,3,2*m) - compdata(14,3,1);
end

vert_1b = vert_1 - base_11(:,1)*(a2+d4)/457 - zchng_5;
vert_2b = vert_2 - base_12(:,1)*(a2+d4)/457 - zchng_5;
vert_3b = vert_3 - base_13(:,1)*(a2+d4)/457 - zchng_5;
vert_4b = vert_4 - base_14(:,1)*(a2+d4)/457 - zchng_5;

figure
plot(loads,vert_1,loads,vert_2,loads,vert_3,loads,vert_4);
title('Vertical Displacement for Sensors 1, 2, 3, and 4');
xlabel('Weight (lb)');
ylabel('Displacement (mm)');
legend('Sensor 1','Sensor 2','Sensor 3','Sensor 4',0);

figure
plot(loads,vert_1b,loads,vert_2b,loads,vert_3b,loads,vert_4b);
title('Modified Vertical Displacement for Sensors 1, 2, 3, and 4');
xlabel('Weight (lb)');
ylabel('Displacement (mm)');
legend('Sensor 1','Sensor 2','Sensor 3','Sensor 4',0);

for i = 1:14
    for j = 1:3
        plotstd(i,j) = mean(compstd(i,j,:));
    end
```

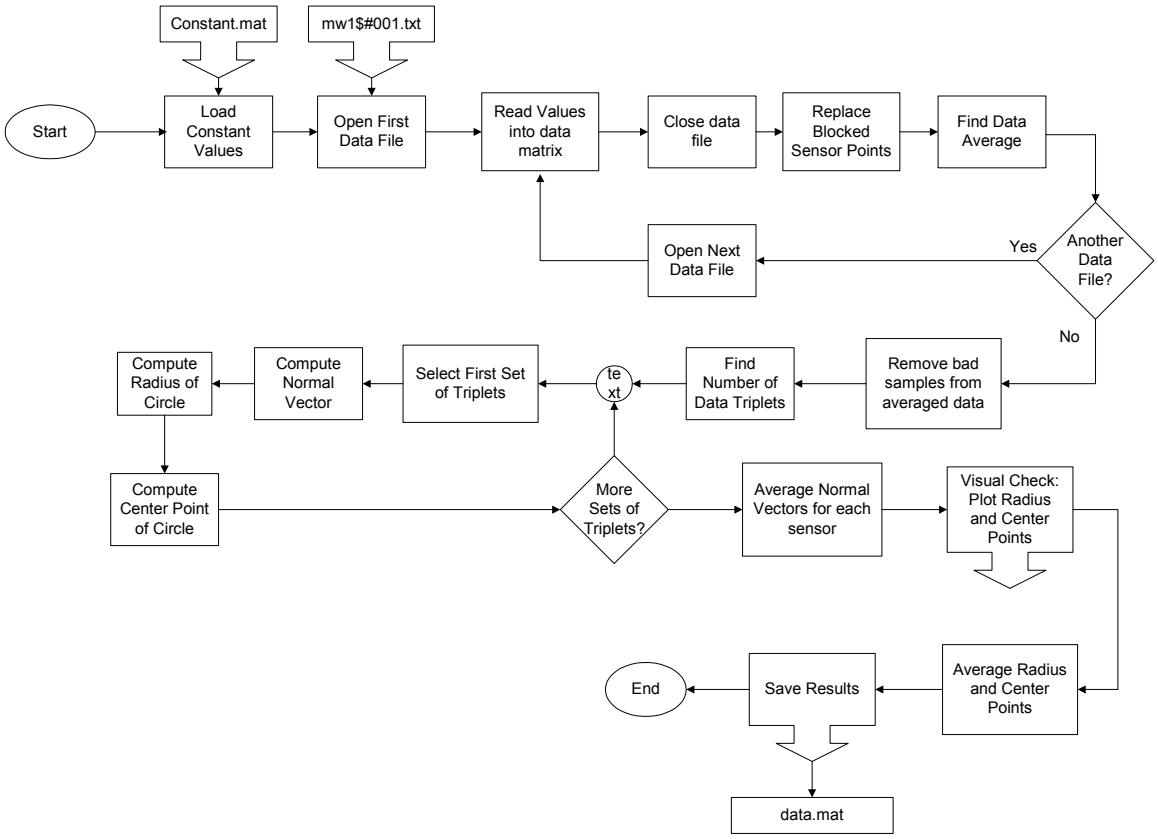
```
end

figure
plot(plotstd,'o')
title('Standard Deviation');
xlabel('Sensor Number');
ylabel('Stndard Deviation');
legend('X-Dir','Y-Dir','Z-Dir');

for m = 1:length(loads) + 1
    zero_1(m) = sqrt(sum((compdata(1,1,2*m - 1) -
compdata(1,1,1)).^2));
    zero_2(m) = sqrt(sum((compdata(2,1,2*m - 1) -
compdata(2,1,1)).^2));
    zero_3(m) = sqrt(sum((compdata(3,1,2*m - 1) -
compdata(3,1,1)).^2));
    zero_4(m) = sqrt(sum((compdata(4,1,2*m - 1) -
compdata(4,1,1)).^2));
    zero_5(m) = sqrt(sum((compdata(5,1,2*m - 1) -
compdata(5,1,1)).^2));
    zero_6(m) = sqrt(sum((compdata(6,1,2*m - 1) -
compdata(6,1,1)).^2));
end

figure
plot([0;loads],zero_1,[0;loads],zero_2,[0;loads],zero_3,[0;loads],zer
o_4);
title('Variation in Zero Positions for Sensors 1 - 6');
xlabel('Load');
ylabel('Magnitude of Position (mm)');
legend('Sensor 1','Sensor 2','Sensor 3','Sensor 4',0);


figure
plot(loads2,zchng_5)
title('Vertical Displacement for Sensor 5');
xlabel('Torque (in*lb)');
ylabel('Displacement (mm)');
legend('Sensor 5',0);

theta_2 = atan2(zchng_5,a2);

figure
plot(loads2,theta_2)
title('Rotation of Joint 2');
xlabel('Torque (in*lb)');
ylabel('Rotation (rad)');


% Generate Plots for report specifically

x = loads3;
y = atan2(vert_2b-zchng_5,d4);

[p,s] = polyfit(x,y,2);
[f,delta] = polyval(p,x,s);
```

```
%  compute the error for different curve fits
error3_2 = sum(abs(y-f));
[p1,s] = polyfit(x,y,1);
[f1,delta] = polyval(p1,x,s);
error3_1 = sum(abs(y-f1));
fprintf('\nJoint 3 Curve Fit Error:  First Order: %g     Second
Order: %g',error3_1,error3_2);

figure
plot(x,y,'o:',x,f)
%title('Figure 7:  Second Stiffness Order Characterization (Joint
3)');
xlabel('Torque (in*lb)');
ylabel('Rotation (rad)');
legend('Experimental Data','Approximate Curve',0);

x = loads2;
y = theta_2;

[p,s] = polyfit(x,y,2);
[f,delta] = polyval(p,x,s);

%  compute the error for different curve fits
error2_2 = sum(abs(y-f));
[p1,s] = polyfit(x,y,1);
[f1,delta] = polyval(p1,x,s);
error2_1 = sum(abs(y-f1));
fprintf('\n\nJoint 2 Curve Fit Error:  First Order: %g     Second
Order: %g',error2_1,error2_2);

figure
plot(x,y,'o:',x,f)
title('Figure 7a:  Second Stiffness Order Characterization (Joint
2)');
xlabel('Torque (in*lb)');
ylabel('Rotation (rad)');
legend('Experimental Data','Approximate Curve',0);

% plot to see motion of base points during testing

figure
plot(loads,base_11(:,1),':',loads,base_12(:,1),':',loads,base_13(:,1)
,':',loads,base_14(:,1),':',loads,base_11(:,2),'--
',loads,base_12(:,2),'--',loads,base_13(:,2),'--
',loads,base_14(:,2),'—
',loads,base_11(:,3),loads,base_12(:,3),loads,base_13(:,3),loads,base
_14(:,3));
title('Change in Vertical Base Location for Sensors 11, 12, 13, and
14');
xlabel('Weight (lb)');
ylabel('Displacement (mm)');
legend('X - Sensor 11','X - Sensor 12','X - Sensor 13','X - Sensor
14','Y - Sensor 11','Y - Sensor 12','Y - Sensor 13','Y - Sensor
14','Z - Sensor 11','Z - Sensor 12','Z - Sensor 13','Z - Sensor
14',0);
```

10.3 Coordinate System Transform Codes

10.3.1 Rotation Test Algorithm for Joint 1 Jog Test

- Code Filename: j1axisjog.m
- Code Description: Code reduces the OptoTrak results from the Coordinate Frame Transform tests.
- Inputs:
    i. OptoTrak data
- Outputs:
    i. Data.mat file including center and radius information
- Application: Used to prepare data for the j1j2axis.m file.
- Notes:




```
% Program to reduce the data from the OptoTrak

clear all;
close all;
clc;

% User Specified Settings
```

```
numofloads = 90;
localpath = 'c:\data\j1axisjog\';

load c:\data\constant.mat

% Begin Code

path1 = strcat(localpath,'backup.mat');
testval = fopen(path1);
if testval == -1

    compdata = zeros(14,3,numofloads);
    compstd = zeros(14,3,numofloads);

    for k = 1:numofloads

        % Build Filename and path into a string

        filenumber = k;
        numstr= num2str(filenumber);

        if filenumber < 100
            numstr = strcat('0',numstr);
        end
        if filenumber < 10
            numstr = strcat('0',numstr);
        end

        filename = strcat(localpath,'mw1$#',numstr,'.mwa');

        % Establish data matrix and open file

        data = zeros(14,3,40);
        fid = fopen(filename);
        fprintf('\nOpening %s', filename);
        titletrash = fgetl(fid);
        % Fill data matrix

        for i = 1:40
            for j = 1:14
                tline = fgetl(fid);
                tlinenum = str2num(tline);
                data(j,:,i) = tlinenum(length(tlinenum) -
2:length(tlinenum));
            end
        end

        fclose(fid);

       % Extract the mean value from the data matrix

        meandata = zeros(14,3);
        stddata = zeros(14,3);
        for i = 1:14
            for j = 1:3
                meandata(i,j) = mean(data(i,j,:));
                stddata(i,j) = std(data(i,j,:));
```

```
            end
        end

        compdata(:,:,k) = meandata;
        compstd(:,:,k) = stddata;

        % Remove Obstructed data Points from the compdata samples

        for i = 1:14
            for j = 1:3
                for k = 1:numofloads
                    if abs(compstd(i,j,k)) > 10^10
                        compstd(i,j,k) = 0;
                    end
                end
            end
        end

    end

   save c:\data\j1axisjog\backup.mat compdata compstd data;

else
   load c:\data\j1axisjog\backup.mat
end

fclose('all');

data1 = compdata(1:4,:,:);
data2 = compdata(11:14,:,:);
data = [data1;data2];
[a,b,c] = size(data);

length = c;
removed = 0;

% This removes Bad samples from the averaged data

for i = 1:4
    for j = 1:3
        k = 1;
        while k < length - removed + 1
            if abs(data(i,j,k)) > 10^5
                data(:,:,k) = [];
                removed = removed + 1;
            end
            k = k + 1;
        end
    end
end

% Find Maximum Number of triplets

[a1,b1,c1] = size(data);
go = 1;
num = 0;
while go == 1
```

```
    if num <= c1 - 3
        num = num + 3;
    else
        go = 0;
    end
end

Vect1a = zeros(1,3);
Vect1b = zeros(1,3);
Pnt1 = zeros(1,3);
Pnt2 = zeros(1,3);
Pnt3 = zeros(1,3);
Norm1 = zeros(1,3);
Normal = zeros(4,num/3,3);
Radius = zeros(4,num/3);
Center = zeros(4,num/3,3);
Half = zeros(4,num/3);
Line = zeros(4,num/3);
Point1 = zeros(4,num/3,3);
Point2 = zeros(4,num/3,3);
Point3 = zeros(4,num/3,3);

for j = 1:4
    for i = 1:num/3


        % Find Normal Vectors
        Pnt1 = data(j,1:3,i);
        Pnt2 = data(j,1:3,i + num/3);
        Pnt3 = data(j,1:3,i + 2*num/3);
        Vect1a = Pnt2 - Pnt1;
        Vect1b = Pnt2 - Pnt3;
        Norm1 = cross(Vect1a,Vect1b);
        Norm1 = Norm1/sqrt(dot(Norm1,Norm1'));
        Normal(j,i,:) = Norm1;
        Point1(j,i,:) = Pnt1;
        Point2(j,i,:) = Pnt2;
        Point3(j,i,:) = Pnt3;

        % Find Radius
        Side1 = Pnt3 - Pnt2;
        Side2 = Pnt1 - Pnt2;
        Side3 = Pnt3 - Pnt1;
        Mag1 = sqrt(dot(Side1,Side1'));
        Mag2 = sqrt(dot(Side2,Side2'));
        Mag3 = sqrt(dot(Side3,Side3'));
        areaV = 1/2*cross(Side2,Side1);
        area = sqrt(dot(areaV,areaV'));
        R = Mag1*Mag2*Mag3/(4*area);
        Radius(j,i) = R;

        % Find Center Points
        Half1 = Side1/2;
        Line1 = cross(-Norm1,Half1);
        alpha = acos((Mag1^2-2*R^2)/(-2*R^2))/2;
        Rscale = R*cos(alpha);
        Center(j,i,:) = Pnt2 + Half1 +
```

```
Rscale*Line1/sqrt(dot(Line1,Line1'));
    end
end

Normal1 = mean(squeeze(Normal(1,:,:)));
Normal2 = mean(squeeze(Normal(2,:,:)));
Normal3 = mean(squeeze(Normal(3,:,:)));
Normal4 = mean(squeeze(Normal(4,:,:)));

fprintf('All Results are in OptoTrak Coordinates\n');
fprintf('\nPlanar Normal for Point 1 is %4.4g %4.4g
%4.4g',Normal1(1),Normal1(2),Normal1(3));
fprintf('\nPlanar Normal for Point 2 is %4.4g %4.4g
%4.4g',Normal2(1),Normal2(2),Normal2(3));
fprintf('\nPlanar Normal for Point 3 is %4.4g %4.4g
%4.4g',Normal3(1),Normal3(2),Normal3(3));
fprintf('\nPlanar Normal for Point 4 is %4.4g %4.4g
%4.4g',Normal4(1),Normal4(2),Normal4(3));

p1x = [Point1(1,:,1),Point2(1,:,1),Point3(1,:,1)];
p1y = [Point1(1,:,2),Point2(1,:,2),Point3(1,:,2)];
p1z = [Point1(1,:,3),Point2(1,:,3),Point3(1,:,3)];

p2x = [Point1(2,:,1),Point2(2,:,1),Point3(2,:,1)];
p2y = [Point1(2,:,2),Point2(2,:,2),Point3(2,:,2)];
p2z = [Point1(2,:,3),Point2(2,:,3),Point3(2,:,3)];

p3x = [Point1(3,:,1),Point2(3,:,1),Point3(3,:,1)];
p3y = [Point1(3,:,2),Point2(3,:,2),Point3(3,:,2)];
p3z = [Point1(3,:,3),Point2(3,:,3),Point3(3,:,3)];

p4x = [Point1(4,:,1),Point2(4,:,1),Point3(4,:,1)];
p4y = [Point1(4,:,2),Point2(4,:,2),Point3(4,:,2)];
p4z = [Point1(4,:,3),Point2(4,:,3),Point3(4,:,3)];

c1x = [Center(1,:,1),Center(1,:,1),Center(1,:,1)];
c1y = [Center(1,:,2),Center(1,:,2),Center(1,:,2)];
c1z = [Center(1,:,3),Center(1,:,3),Center(1,:,3)];

c2x = [Center(2,:,1),Center(2,:,1),Center(2,:,1)];
c2y = [Center(2,:,2),Center(2,:,2),Center(2,:,2)];
c2z = [Center(2,:,3),Center(2,:,3),Center(2,:,3)];

c3x = [Center(3,:,1),Center(3,:,1),Center(3,:,1)];
c3y = [Center(3,:,2),Center(3,:,2),Center(3,:,2)];
c3z = [Center(3,:,3),Center(3,:,3),Center(3,:,3)];

c4x = [Center(4,:,1),Center(4,:,1),Center(4,:,1)];
c4y = [Center(4,:,2),Center(4,:,2),Center(4,:,2)];
c4z = [Center(4,:,3),Center(4,:,3),Center(4,:,3)];

figure
plot3(p1x,p1y,p1z,'r');
hold on
plot3(p2x,p2y,p2z,'r');
plot3(p3x,p3y,p3z,'r');
plot3(p4x,p4y,p4z,'r');
```

```
plot3(c1x,c2y,c1z,'og');
plot3(c2x,c2y,c2z,'og');
plot3(c3x,c3y,c3z,'og');
plot3(c4x,c4y,c4z,'og');
view(squeeze(Normal(1,1,:)))
axis('square')
hold off

figure
plot(Radius(:,:)')
title('Radius Variations');

figure
subplot(3,1,1); plot(squeeze(Center(:,:,1))')
title('X Center Point Variance');
legend('1','2','3','4');
subplot(3,1,2); plot(squeeze(Center(:,:,2))')
title('Y Center Point Variance');
legend('1','2','3','4');
subplot(3,1,3); plot(squeeze(Center(:,:,3))')
title('Z Center Point Variance');
legend('1','2','3','4');

% Average Radius and Center Points
Center1 = zeros(4,3);
Radius1 = zeros(1,4);

Center1(1,:) =
[mean(squeeze(Center(1,:,1))),mean(squeeze(Center(1,:,2))),mean(squee
ze
(Center(1,:,3)))];
Center1(2,:) =
[mean(squeeze(Center(2,:,1))),mean(squeeze(Center(2,:,2))),mean(squee
ze
(Center(2,:,3)))];
Center1(3,:) =
[mean(squeeze(Center(3,:,1))),mean(squeeze(Center(3,:,2))),mean(squee
ze
(Center(3,:,3)))];
Center1(4,:) =
[mean(squeeze(Center(4,:,1))),mean(squeeze(Center(4,:,2))),mean(squee
ze
(Center(4,:,3)))];

Radius1(1) = mean(Squeeze(Radius(1,:)));
Radius1(2) = mean(Squeeze(Radius(2,:)));
Radius1(3) = mean(Squeeze(Radius(3,:)));
Radius1(4) = mean(Squeeze(Radius(4,:)));

fprintf('\n\nCenter 1:   %5.5g %5.5g %5.5g\n',
Center1(1,1),Center1(1,2),Center1(1,3));
fprintf('Center 2:   %5.5g %5.5g %5.5g\n',
Center1(2,1),Center1(2,2),Center1(2,3));
fprintf('Center 3:   %5.5g %5.5g %5.5g\n',
Center1(3,1),Center1(3,2),Center1(3,3));
fprintf('Center 4:   %5.5g %5.5g %5.5g\n',
Center1(4,1),Center1(4,2),Center1(4,3));
```

```
fprintf('\nRadius 1:   %5.5g     (Max/Min: %5.5g /
%5.5g)\n',Radius1(1),max(Squeeze(Radius1(1))),min(Squeeze(Radius1(1))
));
fprintf('Radius 2:   %5.5g     (Max/Min: %5.5g /
%5.5g)\n',Radius1(2),max(Squeeze(Radius1(2))),min(Squeeze(Radius1(2))
));
fprintf('Radius 3:   %5.5g     (Max/Min: %5.5g /
%5.5g)\n',Radius1(3),max(Squeeze(Radius1(3))),min(Squeeze(Radius1(3))
));
fprintf('Radius 4:   %5.5g     (Max/Min: %5.5g /
%5.5g)\n',Radius1(4),max(Squeeze(Radius1(4))),min(Squeeze(Radius1(4))
));

save c:\data\j1axisjog\data.mat Center1 Radius1 data;
```

10.3.2 Rotation Test Algorithm for Joint 2 Jog Test

- Code Filename: j2axisjog.m
- Code Description: Code reduces the OptoTrak results from the Coordinate Frame Transform tests.
- Inputs:
    i. OptoTrak data
- Outputs:
    i. Data.mat file including center and radius information
- Application: Used to prepare data for the j1j2axis.m file.
- Notes:

This code is very similar to j1axisjog.m detailed in 10.3.1 and therefore only the differences are mentioned here.

```
% User Specified Settings

numofloads = 124;
localpath = 'c:\data\j2axisjog\';

.
.
.


   save c:\data\j2axisjog\backup.mat compdata compstd data;

else
   load c:\data\j2axisjog\backup.mat

end

fclose('all');

.
.
```

```
.



% Average Radius and Center Points
Center2 = zeros(4,3);
Radius2 = zeros(1,4);

Center2(1,:) =
[mean(squeeze(Center(1,:,1))),mean(squeeze(Center(1,:,2))),mean(squee
ze
(Center(1,:,3)))];
Center2(2,:) =
[mean(squeeze(Center(2,:,1))),mean(squeeze(Center(2,:,2))),mean(squee
ze
(Center(2,:,3)))];
Center2(3,:) =
[mean(squeeze(Center(3,:,1))),mean(squeeze(Center(3,:,2))),mean(squee
ze
(Center(3,:,3)))];
Center2(4,:) =
[mean(squeeze(Center(4,:,1))),mean(squeeze(Center(4,:,2))),mean(squee
ze
(Center(4,:,3)))];

Radius2(1) = mean(Squeeze(Radius(1,:)));
Radius2(2) = mean(Squeeze(Radius(2,:)));
Radius2(3) = mean(Squeeze(Radius(3,:)));
Radius2(4) = mean(Squeeze(Radius(4,:)));

fprintf('\n\nCenter 1:   %5.5g %5.5g %5.5g\n',
Center2(1,1),Center2(1,2),Center2(1,3));
fprintf('Center 2:   %5.5g %5.5g %5.5g\n',
Center2(2,1),Center2(2,2),Center2(2,3));
fprintf('Center 3:   %5.5g %5.5g %5.5g\n',
Center2(3,1),Center2(3,2),Center2(3,3));
fprintf('Center 4:   %5.5g %5.5g %5.5g\n',
Center2(4,1),Center2(4,2),Center2(4,3));

fprintf('\nRadius 1:   %5.5g     (Max/Min: %5.5g /
%5.5g)\n',Radius2(1),max(Squeeze(Radius(1))),min(Squeeze(Radius(1))))
;
fprintf('Radius 2:   %5.5g     (Max/Min: %5.5g /
%5.5g)\n',Radius2(2),max(Squeeze(Radius(2))),min(Squeeze(Radius(2))))
;
fprintf('Radius 3:   %5.5g     (Max/Min: %5.5g /
%5.5g)\n',Radius2(3),max(Squeeze(Radius(3))),min(Squeeze(Radius(3))))
;
fprintf('Radius 4:   %5.5g     (Max/Min: %5.5g /
%5.5g)\n',Radius2(4),max(Squeeze(Radius(4))),min(Squeeze(Radius(4))))
;

save c:\data\j2axisjog\data.mat Center2 Radius2 data;
```

10.3.3 Rotation Test Algorithm for Joint 5 Jog Test

- Code Filename: j5axisjog.m
- Code Description: Code reduces the OptoTrak results from the Coordinate Frame Transform tests.
- Inputs:
    i. OptoTrak data
- Outputs:
    i. Data.mat file including center and radius information
- Application: Used to prepare data for the j1j2axis.m file.
- Notes: This code is similar to j1axisjog.m detailed in 10.3.1

```
% Program to reduce the data from the OptoTrak

clear all;
close all;
clc;

% User Specified Settings

numofloads = 54;
localpath = 'c:\data\j5axisjog\';

load c:\data\constant.mat

% Begin Code

path1 = strcat(localpath,'backup.mat');
testval = fopen(path1);
if testval == -1

    compdata = zeros(14,3,numofloads);
    compstd = zeros(14,3,numofloads);

    for k = 1:numofloads

        % Build Filename and path into a string

        filenumber = k;
        numstr= num2str(filenumber);

        if filenumber < 100
            numstr = strcat('0',numstr);
        end
        if filenumber < 10
            numstr = strcat('0',numstr);
        end

        filename = strcat(localpath,'mw1$#',numstr,'.mwa');

        % Establish data matrix and open file

        data = zeros(14,3,40);
        fid = fopen(filename);
```

```
        fprintf('\nOpening %s', filename);
        titletrash = fgetl(fid);

        % Fill data matrix

        for i = 1:40
            for j = 1:14
                tline = fgetl(fid);
                tlinenum = str2num(tline);
                data(j,:,i) = tlinenum(length(tlinenum) -
2:length(tlinenum));
            end
        end

        fclose(fid);

       % Extract the mean value from the data matrix

        meandata = zeros(14,3);
        stddata = zeros(14,3);
        for i = 1:14
            for j = 1:3
                meandata(i,j) = mean(data(i,j,:));
                stddata(i,j) = std(data(i,j,:));
            end
        end

        compdata(:,:,k) = meandata;
        compstd(:,:,k) = stddata;

        % Remove Obstructed data Points from the compdata samples

        for i = 1:14
            for j = 1:3
                for k = 1:numofloads
                    if abs(compstd(i,j,k)) > 10^10
                        compstd(i,j,k) = 0;
                    end
                end
            end
        end

    end

   save c:\data\j5axisjog\backup.mat compdata compstd data;

else
   load c:\data\j5axisjog\backup.mat
end

fclose('all');

data = compdata(1:4,:,:);
[a,b,c] = size(data);

length = c;
removed = 0;
```

```
% This removes Bad samples from the averaged data

for i = 1:4
    for j = 1:3
        k = 1;
        while k < length - removed + 1
            if abs(data(i,j,k)) > 10^5
                data(:,:,k) = [];
                removed = removed + 1;
            end
            k = k + 1;
        end
    end
end

% Find Maximum Number of triplets

[a1,b1,c1] = size(data);
go = 1;
num = 0;
while go == 1
    if num <= c1 - 3
        num = num + 3;
    else
        go = 0;
    end
end

Vect1a = zeros(1,3);
Vect1b = zeros(1,3);
Pnt1 = zeros(1,3);
Pnt2 = zeros(1,3);
Pnt3 = zeros(1,3);
Norm1 = zeros(1,3);
Normal = zeros(4,num/3,3);
Radius = zeros(4,num/3);
Center = zeros(4,num/3,3);
Half = zeros(4,num/3);
Line = zeros(4,num/3);
Point1 = zeros(4,num/3,3);
Point2 = zeros(4,num/3,3);
Point3 = zeros(4,num/3,3);

for j = 1:4
    for i = 1:num/3

        % Find Normal Vectors
        Pnt1 = data(j,1:3,i);
        Pnt2 = data(j,1:3,i + num/3);
        Pnt3 = data(j,1:3,i + 2*num/3);
        Vect1a = Pnt2 - Pnt1;
        Vect1b = Pnt2 - Pnt3;
        Norm1 = cross(Vect1a,Vect1b);
        Norm1 = Norm1/sqrt(dot(Norm1,Norm1'));
        Normal(j,i,:) = Norm1;
        Point1(j,i,:) = Pnt1;
```

```
        Point2(j,i,:) = Pnt2;
        Point3(j,i,:) = Pnt3;

        % Find Radius
        Side1 = Pnt3 - Pnt2;
        Side2 = Pnt1 - Pnt2;
        Side3 = Pnt3 - Pnt1;
        Mag1 = sqrt(dot(Side1,Side1'));
        Mag2 = sqrt(dot(Side2,Side2'));
        Mag3 = sqrt(dot(Side3,Side3'));
        areaV = 1/2*cross(Side2,Side1);
        area = sqrt(dot(areaV,areaV'));
        R = Mag1*Mag2*Mag3/(4*area);
        Radius(j,i) = R;

        % Find Center Points
        Half1 = Side1/2;
        Line1 = cross(-Norm1,Half1);
        alpha = acos((Mag1^2-2*R^2)/(-2*R^2))/2;
        Rscale = R*cos(alpha);
        Center(j,i,:) = Pnt2 + Half1 +
Rscale*Line1/sqrt(dot(Line1,Line1'));
    end
end

Normal1 = mean(squeeze(Normal(1,:,:)));
Normal2 = mean(squeeze(Normal(2,:,:)));
Normal3 = mean(squeeze(Normal(3,:,:)));
Normal4 = mean(squeeze(Normal(4,:,:)));

fprintf('All Results are in OptoTrak Coordinates\n');
fprintf('\nPlanar Normal for Point 1 is %4.4g %4.4g
%4.4g',Normal1(1),Normal1(2),Normal1(3));
fprintf('\nPlanar Normal for Point 2 is %4.4g %4.4g
%4.4g',Normal2(1),Normal2(2),Normal2(3));
fprintf('\nPlanar Normal for Point 3 is %4.4g %4.4g
%4.4g',Normal3(1),Normal3(2),Normal3(3));
fprintf('\nPlanar Normal for Point 4 is %4.4g %4.4g
%4.4g',Normal4(1),Normal4(2),Normal4(3));

p1x = [Point1(1,:,1),Point2(1,:,1),Point3(1,:,1)];
p1y = [Point1(1,:,2),Point2(1,:,2),Point3(1,:,2)];
p1z = [Point1(1,:,3),Point2(1,:,3),Point3(1,:,3)];

p2x = [Point1(2,:,1),Point2(2,:,1),Point3(2,:,1)];
p2y = [Point1(2,:,2),Point2(2,:,2),Point3(2,:,2)];
p2z = [Point1(2,:,3),Point2(2,:,3),Point3(2,:,3)];

p3x = [Point1(3,:,1),Point2(3,:,1),Point3(3,:,1)];
p3y = [Point1(3,:,2),Point2(3,:,2),Point3(3,:,2)];
p3z = [Point1(3,:,3),Point2(3,:,3),Point3(3,:,3)];

p4x = [Point1(4,:,1),Point2(4,:,1),Point3(4,:,1)];
p4y = [Point1(4,:,2),Point2(4,:,2),Point3(4,:,2)];
p4z = [Point1(4,:,3),Point2(4,:,3),Point3(4,:,3)];

c1x = [Center(1,:,1),Center(1,:,1),Center(1,:,1)];
```

```
c1y = [Center(1,:,2),Center(1,:,2),Center(1,:,2)];
c1z = [Center(1,:,3),Center(1,:,3),Center(1,:,3)];

c2x = [Center(2,:,1),Center(2,:,1),Center(2,:,1)];
c2y = [Center(2,:,2),Center(2,:,2),Center(2,:,2)];
c2z = [Center(2,:,3),Center(2,:,3),Center(2,:,3)];

c3x = [Center(3,:,1),Center(3,:,1),Center(3,:,1)];
c3y = [Center(3,:,2),Center(3,:,2),Center(3,:,2)];
c3z = [Center(3,:,3),Center(3,:,3),Center(3,:,3)];

c4x = [Center(4,:,1),Center(4,:,1),Center(4,:,1)];
c4y = [Center(4,:,2),Center(4,:,2),Center(4,:,2)];
c4z = [Center(4,:,3),Center(4,:,3),Center(4,:,3)];

figure
plot3(p1x,p1y,p1z,'r');
hold on
plot3(p2x,p2y,p2z,'r');
plot3(p3x,p3y,p3z,'r');
plot3(p4x,p4y,p4z,'r');
plot3(c1x,c2y,c1z,'og');
plot3(c2x,c2y,c2z,'og');
plot3(c3x,c3y,c3z,'og');
plot3(c4x,c4y,c4z,'og');
view(squeeze(Normal(1,1,:)))
axis('square')
hold off

figure
plot(Radius(:,:)')
title('Radius Variations');

figure
subplot(3,1,1); plot(squeeze(Center(:,:,1))')
title('X Center Point Variance');
legend('1','2','3','4');
subplot(3,1,2); plot(squeeze(Center(:,:,2))')
title('Y Center Point Variance');
legend('1','2','3','4');
subplot(3,1,3); plot(squeeze(Center(:,:,3))')
title('Z Center Point Variance');
legend('1','2','3','4');

% Average Radius and Center Points

Center5 = zeros(4,3);
Radius5 = zeros(1,4);

Center5(1,:) =
[mean(squeeze(Center(1,:,1))),mean(squeeze(Center(1,:,2))),mean(squeeze
(Center(1,:,3)))];
Center5(2,:) =
[mean(squeeze(Center(2,:,1))),mean(squeeze(Center(2,:,2))),mean(squeeze
(Center(2,:,3)))];
Center5(3,:) =
```

```
[mean(squeeze(Center(3,:,1))),mean(squeeze(Center(3,:,2))),mean(squeeze
(Center(3,:,3)))];
Center5(4,:) =
[mean(squeeze(Center(4,:,1))),mean(squeeze(Center(4,:,2))),mean(squeeze
(Center(4,:,3)))];

Radius5(1) = mean(Squeeze(Radius(1,:)));
Radius5(2) = mean(Squeeze(Radius(2,:)));
Radius5(3) = mean(Squeeze(Radius(3,:)));
Radius5(4) = mean(Squeeze(Radius(4,:)));

fprintf('\n\nCenter 1:   %5.5g %5.5g %5.5g\n',
Center5(1,1),Center5(1,2),Center5(1,3));
fprintf('Center 2:   %5.5g %5.5g %5.5g\n',
Center5(2,1),Center5(2,2),Center5(2,3));
fprintf('Center 3:   %5.5g %5.5g %5.5g\n',
Center5(3,1),Center5(3,2),Center5(3,3));
fprintf('Center 4:   %5.5g %5.5g %5.5g\n',
Center5(4,1),Center5(4,2),Center5(4,3));

fprintf('\nRadius 1:   %5.5g     (Max/Min: %5.5g /
%5.5g)\n',Radius5(1),max(Squeeze(Radius(1))),min(Squeeze(Radius(1))));
fprintf('Radius 2:   %5.5g     (Max/Min: %5.5g /
%5.5g)\n',Radius5(2),max(Squeeze(Radius(2))),min(Squeeze(Radius(2))));
fprintf('Radius 3:   %5.5g     (Max/Min: %5.5g /
%5.5g)\n',Radius5(3),max(Squeeze(Radius(3))),min(Squeeze(Radius(3))));
fprintf('Radius 4:   %5.5g     (Max/Min: %5.5g /
%5.5g)\n',Radius5(4),max(Squeeze(Radius(4))),min(Squeeze(Radius(4))));

save c:\data\j5axisjog\data.mat Center5 Radius5 data;
```

10.3.4 Rotation Test Algorithm for Joint 6 Jog Test

- Code Filename: j6axisjog.m
- Code Description: Code reduces the OptoTrak results from the Coordinate Frame Transform tests.
- Inputs:
    i. OptoTrak data
- Outputs:
    i. Data.mat file including center and radius information
- Application: Used to prepare data for the j1j2axis.m file.
- Notes:

This code is very similar to j5axisjog.m detailed in 0. Therefore, only changes between the to will be shown here.

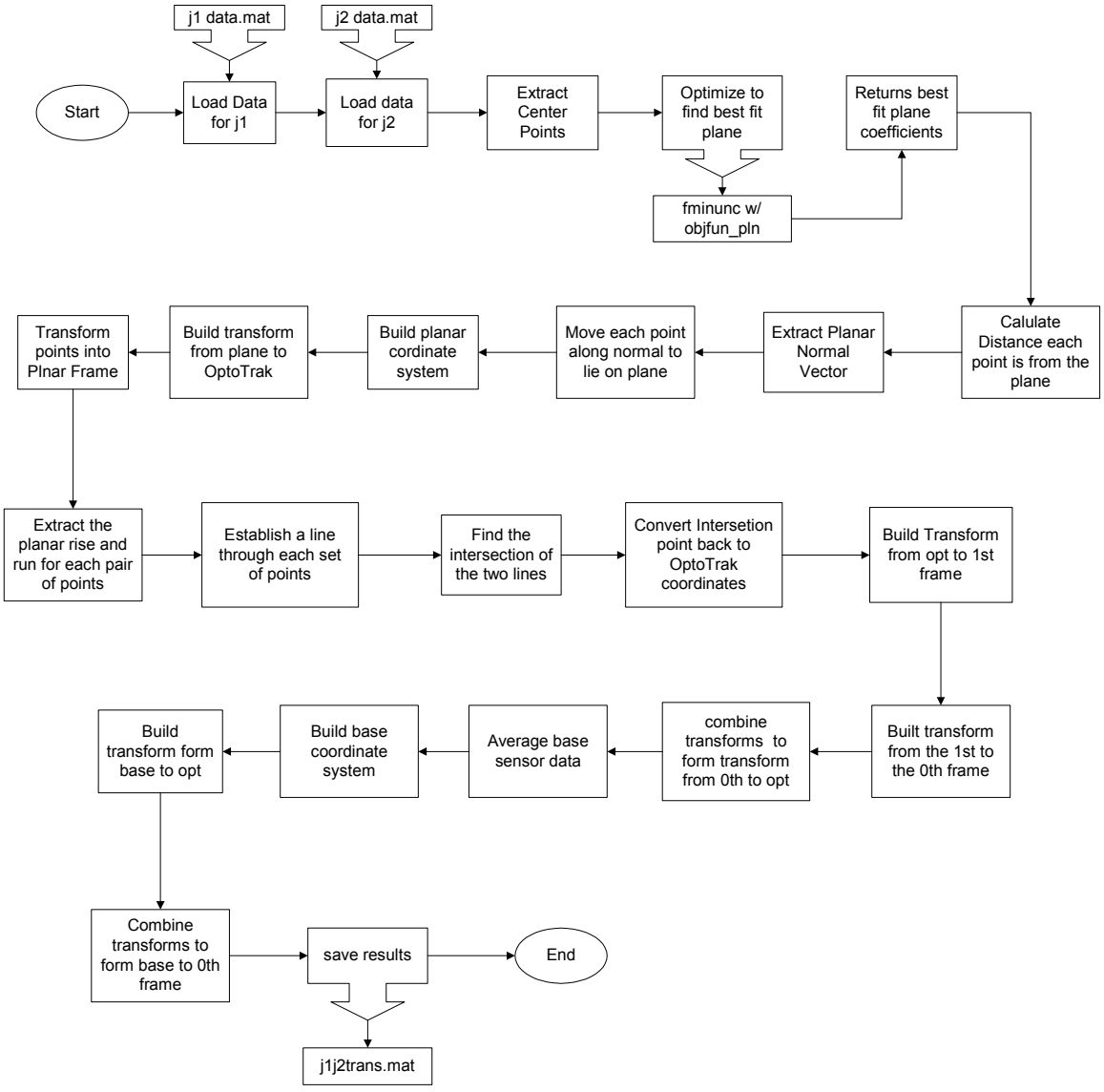
```
% User Specified Settings

numofloads = 69;
localpath = 'c:\data\j6axisjog\';

load c:\data\constant.mat
```

```
.
.
.


   save c:\data\j6axisjog\backup.mat compdata compstd data;

else
   load c:\data\j6axisjog\backup.mat
end

fclose('all');

.
.
.

% Average Radius and Center Points

Center6 = zeros(4,3);
Radius6 = zeros(1,4);

Center6(1,:) =
[mean(squeeze(Center(1,:,1))),mean(squeeze(Center(1,:,2))),mean(squeeze
(Center(1,:,3)))];
Center6(2,:) =
[mean(squeeze(Center(2,:,1))),mean(squeeze(Center(2,:,2))),mean(squeeze
(Center(2,:,3)))];
Center6(3,:) =
[mean(squeeze(Center(3,:,1))),mean(squeeze(Center(3,:,2))),mean(squeeze
(Center(3,:,3)))];
Center6(4,:) =
[mean(squeeze(Center(4,:,1))),mean(squeeze(Center(4,:,2))),mean(squeeze
(Center(4,:,3)))];

Radius6(1) = mean(Squeeze(Radius(1,:)));
Radius6(2) = mean(Squeeze(Radius(2,:)));
Radius6(3) = mean(Squeeze(Radius(3,:)));
Radius6(4) = mean(Squeeze(Radius(4,:)));

fprintf('\n\nCenter 1:   %5.5g %5.5g %5.5g\n',
Center6(1,1),Center6(1,2),Center6(1,3));
fprintf('Center 2:   %5.5g %5.5g %5.5g\n',
Center6(2,1),Center6(2,2),Center6(2,3));
fprintf('Center 3:   %5.5g %5.5g %5.5g\n',
Center6(3,1),Center6(3,2),Center6(3,3));
fprintf('Center 4:   %5.5g %5.5g %5.5g\n',
Center6(4,1),Center6(4,2),Center6(4,3));

fprintf('\nRadius 1:   %5.5g     (Max/Min: %5.5g /
%5.5g)\n',Radius6(1),max(Squeeze(Radius(1))),min(Squeeze(Radius(1))));
fprintf('Radius 2:   %5.5g     (Max/Min: %5.5g /
%5.5g)\n',Radius6(2),max(Squeeze(Radius(2))),min(Squeeze(Radius(2))));
fprintf('Radius 3:   %5.5g     (Max/Min: %5.5g /
%5.5g)\n',Radius6(3),max(Squeeze(Radius(3))),min(Squeeze(Radius(3))));
```

```
fprintf('Radius 4:   %5.5g     (Max/Min: %5.5g /
%5.5g)\n',Radius6(4),max(Squeeze(Radius(4))),min(Squeeze(Radius(4))));

save c:\data\j6axisjog\data.mat Center6 Radius6 data;
```

10.3.5 Transformation Builder for Joints 1 and 2

- Code Filename: j1j2trans.m
- Code Description: Code to find transformation matrices between unrelated coordinate frames
- Inputs:
    i. j1axisjog data
    ii. j2axisjog data
- Outputs:
    i. Transform between Base and 0 frame
    ii. Transform between Base and Opt frame
    iii. Transform between Opt and 0 frame
- Application: This code creates the transforms between the OptoTrak and related robot frames, allowing the relationship between the sensors and the robot to ultimately be derived.
- Notes: N/A




```
% Mark W. Abbott
% December 7, 2001

% Code to find Transformation Matrices between Coordinate Systems

clear all;
close all;
clc;

load c:\data\j1axisjog\data.mat;
load c:\data\j2axisjog\data.mat;

Pnts = [Center1(1,:);Center1(2,:);Center2(3,:);Center2(4,:)];

% Must be ordered as follows:
%   1 - Axis 1, Positive End
%   2 - Axis 1, Negative End
```

```
%   3 - Axis 0, Negative End
%   4 - Axis 0, Positive End

Pnts_d  = zeros(4,1);
Pnts_m = zeros(4,3);
Pnts_pln = zeros(4,4);

% Begin Optimization for Best Fit Plane

x0 = 0.1*[-10 -10 10 10];     %Ax + By + Cz = D

ObjLimit = 10^-10;
Xincr = 10^-10;
maxeval = [20000];
iter = [8000];
options =
optimset('LargeScale','off','Tolfun',ObjLimit,'TolX',Xincr,'MaxFunEvals
',maxeval,'MaxIter',iter);             % Specifies a medium-scale
solver
[x,fval,exitflag,output] = fminunc(@objfun_pln,x0,options,Pnts)

% Project Points onto Plane

% Ref Calc Book, Page 670

Pnts_d(1) = (abs(x(1)*Pnts(1,1) + x(2)*Pnts(1,2) + x(3)*Pnts(1,3) -
x(4))/sqrt(x(1)^2 + x(2)^2 + x(3)^2));
Pnts_d(2) = (abs(x(1)*Pnts(2,1) + x(2)*Pnts(2,2) + x(3)*Pnts(2,3) -
x(4))/sqrt(x(1)^2 + x(2)^2 + x(3)^2));
Pnts_d(3) = (abs(x(1)*Pnts(3,1) + x(2)*Pnts(3,2) + x(3)*Pnts(3,3) -
x(4))/sqrt(x(1)^2 + x(2)^2 + x(3)^2));
Pnts_d(4) = (abs(x(1)*Pnts(4,1) + x(2)*Pnts(4,2) + x(3)*Pnts(4,3) -
x(4))/sqrt(x(1)^2 + x(2)^2 + x(3)^2));

% Build Normal Vector, Orthogonal to Plane

norm = [x(1), x(2), x(3)];
UnitNormal = norm/sqrt(dot(norm,norm));

% Modify each Point Position to sit on plane

for i = 1:length(Pnts_d)
    Val1 = Pnts(i,:) + Pnts_d(i)*UnitNormal;
    CMP1 = x(1)*Val1(1) + x(2)*Val1(2) + x(3)*Val1(3) - x(4);
    Val2 = Pnts(i,:) - Pnts_d(i)*UnitNormal;
    CMP2 = x(1)*Val2(1) + x(2)*Val2(2) + x(3)*Val2(3) - x(4);
    if CMP1 == 0
        Pnts_m(i,:) = Val1;
    elseif CMP2 == 0
        Pnts_m(i,:) = Val2;
    elseif CMP1 < CMP2
        Pnts_m(i,:) = Val1;
        fprintf('\nWarning:  Point not exactly Planar!  Error:  %g\n',
CMP1);
    else
        Pnts_m(i,:) = Val2;
```

```
        fprintf('\nWarning:  Point not exactly Planar!  Error:  %g\n',
CMP2);
    end
end

% Build Coordinate System on Plane

%z_plane = UnitNormal;
%x_plane = Pnts_m(2,:) - Pnts_m(1,:);
%x_plane = x_plane/sqrt(dot(x_plane,x_plane'));
%y_plane = cross(z_plane,x_plane);

x_plane = Pnts_m(2,:) - Pnts_m(1,:);
x_plane = x_plane/sqrt(dot(x_plane,x_plane));
y_plane = Pnts_m(4,:) - Pnts_m(3,:);
y_plane = y_plane/sqrt(dot(y_plane,y_plane));
z_plane = cross(x_plane,y_plane);
z_plane = z_plane/sqrt(dot(z_plane,z_plane));
y_plane = cross(z_plane,x_plane);
y_plane = y_plane/sqrt(dot(y_plane,y_plane));

% Build Transformation Matrix from OptoTrak coordinates to 'plane
coordinates
% This should create a planar problem

col1 = [x_plane';0];
col2 = [y_plane';0];
col3 = [z_plane';0];

R_tmp = [col1,col2,col3];
R = R_tmp(1:3,:);

%Pnts_m(1,:)'
%R*Pnts_m(1,:)'

col4 = [Pnts_m(2,:)';1];

Tpln_opt1 = [col1,col2,col3,col4];
Topt_pln1 = inv(Tpln_opt1);

%Pnts_m(1,:)'
%T*[Pnts_m(1,:)';1]

Pnts_pln(1,:) = [Topt_pln1*[Pnts_m(1,:)';1]]';
Pnts_pln(2,:) = [Topt_pln1*[Pnts_m(2,:)';1]]';
Pnts_pln(3,:) = [Topt_pln1*[Pnts_m(3,:)';1]]';
Pnts_pln(4,:) = [Topt_pln1*[Pnts_m(4,:)';1]]';

Pnts_pln = Pnts_pln(:,1:3);

run_1 = Pnts_pln(2,1) - Pnts_pln(1,1);
rise_1 = Pnts_pln(2,2) - Pnts_pln(1,2);
run_2 = Pnts_pln(3,1) - Pnts_pln(4,1);
rise_2 = Pnts_pln(3,2) - Pnts_pln(4,2);

% Find the intersection of the 2 lines in the plane
```

```
if run_1 == 0
    m1 = 1;
    b1 = 0;
elseif rise_1 == 0
    m1 = 0;
    b1 = Pnts_pln(1,2) - m1*Pnts_pln(1,1);
else
    m1 = rise_1/run_1;
    b1 = Pnts_pln(1,2) - m1*Pnts_pln(1,1);
end

if run_2 == 0
    m2 = 1;
    b2 = 0;
elseif rise_2 == 0
    m2 = 0;
    b2 = Pnts_pln(3,2) - m2*Pnts_pln(3,1);
else
    m2 = rise_2/run_2;
    b2 = Pnts_pln(3,2) - m2*Pnts_pln(3,1);
end

x_int = (b2 - b1)/(m1 - m2);
y_int = m1*x_int + b1;

int_pnt = [x_int,y_int,0];

% Convert the Intersection Point, back into Optotrak Coordinates

tmp = Tpln_opt1*[int_pnt';1];
int_pnt_o = tmp(1:3)';

% Create a transformation matrix to rotate the data back
%   to the 1st frame

z1 = Pnts_m(2,:) - Pnts_m(1,:);
z1 = z1/sqrt(dot(z1,z1));

z0 = Pnts_m(4,:) - Pnts_m(3,:);
z0 = z0/sqrt(dot(z0,z0));

x1 = cross(z1,z0);
x1 = x1/sqrt(dot(x1,x1));
y1 = cross(z1,x1);
y1 = y1/sqrt(dot(y1,y1));

col1b = [x1';0];
col2b = [y1';0];
col3b = [z1';0];
col4b = [int_pnt_o';1];

T1_opt = [col1b,col2b,col3b,col4b];
Topt_1 = inv(T1_opt);

% Must also apply the transform from frame 1 to frame 0, this is where
```

```
the FK's return information

T01 = [1,0,0,0;0,1,0,0;0,0,1,46.45*25.4;0,0,0,1];

Topt_0 = T01*Topt_1;

% Now I Must generate a transform from the 3 base sensors to the zero
coordinate frame.

% Build a Coordinate frame that is centered on the 14th sensor, and the
z-axis
%    is orthognal to the plane created by sensors 14, 11 and 12

% Find the average location of the three sensors of interest

sensor_14 = [mean(data(8,1,:)),mean(data(8,2,:)),mean(data(8,3,:))];
sensor_11 = [mean(data(5,1,:)),mean(data(5,2,:)),mean(data(5,3,:))];
sensor_12 = [mean(data(6,1,:)),mean(data(6,2,:)),mean(data(6,3,:))];

x_base = sensor_12 - sensor_14;
x_base = x_base/sqrt(dot(x_base,x_base));

y_base1 = sensor_11 - sensor_14;
y_base1 = y_base1/sqrt(dot(y_base1,y_base1));

z_base = cross(x_base,y_base1);
z_base = z_base/sqrt(dot(z_base,z_base));
y_base = cross(z_base,x_base);
y_base = y_base/sqrt(dot(y_base,y_base));

col1_base = [x_base';0];
col2_base = [y_base';0];
col3_base = [z_base';0];
col4_base = [sensor_14';1];

Tbase_opt = [col1_base,col2_base,col3_base,col4_base];

Tbase_0 = Topt_0*Tbase_opt;

save c:\data\j1j2trans.mat Tbase_0 Tbase_opt Topt_0
```

10.3.6 Transformation Builder for Joints 5 and 6

- Code Filename: j5j6trans.m
- Code Description: Code to find transformation matrices between unrelated coordinate frames
- Inputs:
    i. j5axisjog data
    ii. j6axisjog data
- Outputs:
    i. Transforms between Data Frames and 6th frame
    ii. Transform between Data Frames and Opt frame

- Application: This code creates the transforms between the OptoTrak and related robot frames, allowing the relationship between the sensors and the robot to ultimately be derived.
- Notes: N/A

```
% Mark W. Abbott
% December 7, 2001

% Code to find Transformation Matrices between Coordinate Systems

clear all;
close all;
clc;

load c:\data\j5axisjog\data.mat;
load c:\data\j6axisjog\data.mat;

Pnts = [Center5(1,:);Center5(2,:);Center6(3,:);Center6(4,:)];

% Must be ordered as follows:
%   1 - Axis 4, Positive End
%   2 - Axis 4, Negative End
%   3 - Axis 6, Negative End
%   4 - Axis 6, Positive End

Pnts_d  = zeros(4,1);
Pnts_m = zeros(4,3);
Pnts_pln = zeros(4,4);

% Begin Optimization for Best Fit Plane

x0 = [-10 -10 10 10];     %Ax + By + Cz = D

ObjLimit = 10^-10;
Xincr = 10^-10;
maxeval = [10000];
iter = [8000];

options =
optimset('LargeScale','off','Tolfun',ObjLimit,'TolX',Xincr,'MaxFunEvals
',maxeval,'MaxIter',iter);             % Specifies a medium-scale
solver
[x,fval,exitflag,output] = fminunc(@objfun_pln,x0,options,Pnts)

% Project Points onto Plane

% Ref Calc Book, Page 670

Pnts_d(1) = (abs(x(1)*Pnts(1,1) + x(2)*Pnts(1,2) + x(3)*Pnts(1,3) -
x(4))/sqrt(x(1)^2 + x(2)^2 + x(3)^2));
Pnts_d(2) = (abs(x(1)*Pnts(2,1) + x(2)*Pnts(2,2) + x(3)*Pnts(2,3) -
x(4))/sqrt(x(1)^2 + x(2)^2 + x(3)^2));
Pnts_d(3) = (abs(x(1)*Pnts(3,1) + x(2)*Pnts(3,2) + x(3)*Pnts(3,3) -
x(4))/sqrt(x(1)^2 + x(2)^2 + x(3)^2));
Pnts_d(4) = (abs(x(1)*Pnts(4,1) + x(2)*Pnts(4,2) + x(3)*Pnts(4,3) -
```

```
x(4))/sqrt(x(1)^2 + x(2)^2 + x(3)^2));

% Build Normal Vector, Orthoganol to Plane

norm = [x(1), x(2), x(3)];
UnitNormal = norm/sqrt(dot(norm,norm));

% Modify each Point Position to sit on plane

for i = 1:length(Pnts_d)
    Val1 = Pnts(i,:) + Pnts_d(i)*UnitNormal;
    CMP1 = x(1)*Val1(1) + x(2)*Val1(2) + x(3)*Val1(3) - x(4);
    Val2 = Pnts(i,:) - Pnts_d(i)*UnitNormal;
    CMP2 = x(1)*Val2(1) + x(2)*Val2(2) + x(3)*Val2(3) - x(4);
    if CMP1 == 0
        Pnts_m(i,:) = Val1;
    elseif CMP2 == 0
        Pnts_m(i,:) = Val2;
    elseif CMP1 < CMP2
        Pnts_m(i,:) = Val1;
        fprintf('\nWarning:  Point not exactly Planar!  Error:  %g\n',
CMP1);
    else
        Pnts_m(i,:) = Val2;
        fprintf('\nWarning:  Point not exactly Planar!  Error:  %g\n',
CMP2);
    end
end

% Build Coordinate System on Plane

%z_plane = UnitNormal;
%x_plane = Pnts_m(2,:) - Pnts_m(1,:);
%x_plane = x_plane/sqrt(dot(x_plane,x_plane'));
%y_plane = cross(z_plane,x_plane);

x_plane = Pnts_m(2,:) - Pnts_m(1,:);
x_plane = x_plane/sqrt(dot(x_plane,x_plane));
y_plane = Pnts_m(4,:) - Pnts_m(3,:);
y_plane = y_plane/sqrt(dot(y_plane,y_plane));
z_plane = cross(x_plane,y_plane);
z_plane = z_plane/sqrt(dot(z_plane,z_plane));
y_plane = cross(z_plane,x_plane);
y_plane = y_plane/sqrt(dot(y_plane,y_plane));

% Build Transformation Matrix from OptoTrak coordinates to 'plane
coordinates
% This should create a planar problem

col1 = [x_plane';0];
col2 = [y_plane';0];
col3 = [z_plane';0];
col4 = [Pnts_m(2,:)';1];

Tpln_opt5 = [col1,col2,col3,col4];
Topt_pln5 = inv(Tpln_opt5);
```

```
Pnts_pln(1,:) = [Topt_pln5*[Pnts_m(1,:)';1]]';
Pnts_pln(2,:) = [Topt_pln5*[Pnts_m(2,:)';1]]';
Pnts_pln(3,:) = [Topt_pln5*[Pnts_m(3,:)';1]]';
Pnts_pln(4,:) = [Topt_pln5*[Pnts_m(4,:)';1]]';

Pnts_pln = Pnts_pln(:,1:3);

run_1 = Pnts_pln(2,1) - Pnts_pln(1,1);
rise_1 = Pnts_pln(2,2) - Pnts_pln(1,2);
run_2 = Pnts_pln(3,1) - Pnts_pln(4,1);
rise_2 = Pnts_pln(3,2) - Pnts_pln(4,2);

% Find the intersection of the 2 lines in the plane

if run_1 == 0
    m1 = 1;
    b1 = 0;
elseif rise_1 == 0
    m1 = 0;
    b1 = Pnts_pln(1,2) - m1*Pnts_pln(1,1);
else
    m1 = rise_1/run_1;
    b1 = Pnts_pln(1,2) - m1*Pnts_pln(1,1);
end

if run_2 == 0
    m2 = 1;
    b2 = 0;
elseif rise_2 == 0
    m2 = 0;
    b2 = Pnts_pln(3,2) - m2*Pnts_pln(3,1);
else
    m2 = rise_2/run_2;
    b2 = Pnts_pln(3,2) - m2*Pnts_pln(3,1);
end

x_int = (b2 - b1)/(m1 - m2);
y_int = m1*x_int + b1;

int_pnt = [x_int,y_int,0];

% Convert the Intersection Point, back into Optotrak Coordinates

tmp = Tpln_opt5*[int_pnt';1];
int_pnt_o = tmp(1:3)';

% Create a transformation matrix to rotate the data back
%   to the 5th frame

z4 = Pnts_m(2,:) - Pnts_m(1,:);
z4 = z4/sqrt(dot(z4,z4'));

z5 = Pnts_m(4,:) - Pnts_m(3,:);
z5 = z5/sqrt(dot(z5,z5'));

x5 = cross(z4,z5);
x5 = x5/sqrt(dot(x5,x5));
```

```
y5 = cross(z5,x5);
y5 = y5/sqrt(dot(y5,y5));

col1b = [x5';0];
col2b = [y5';0];
col3b = [z5';0];
col4b = [int_pnt_o';1];

T5_opt = [col1b,col2b,col3b,col4b];
Topt_5 = inv(T5_opt);

% Must also apply the transform from frame 5 to frame 6, this is where
the FK's return information
% WE Can reflect through franme 5 because this experiment was unloaded
% in the actual load testing,m we will only use the transformation that
represents the
%   physical flange to move the data.

% Theta 6 was zero during this trial..  therefore it is neglected

T56 = [1,0,0,0;0,1,0,0;0,0,1,-88.9;0,0,0,1];

Topt_6 = T56*Topt_5;
T6_opt = inv(Topt_6);

% Build frame using points as follows
%
% Frame Number comes from point at the intersection of all three axis
% Z-Axis is orthoganol to plane of points
% X-Axis goes from the base point to the next point numerically
(CounterClockwise)

frame = 43;

% Data Frame 1

df1_x = data(2,:,frame) - data(1,:,frame);
df1_x = df1_x/sqrt(dot(df1_x,df1_x));
df1_y1 = data(4,:,frame) - data(1,:,frame);
df1_y1 = df1_y1/sqrt(dot(df1_y1,df1_y1));
df1_z = cross(df1_x,df1_y1);
df1_z = df1_z/sqrt(dot(df1_z,df1_z));
df1_y = cross(df1_z,df1_x);
df1_y = df1_y/sqrt(dot(df1_y,df1_y));

col1_df1 = [df1_x';0];
col2_df1 = [df1_y';0];
col3_df1 = [df1_z';0];
col4_df1 = [data(1,:,frame)';1];

Tdf1_opt = [col1_df1,col2_df1,col3_df1,col4_df1];

Tdf1_6 = Topt_6*Tdf1_opt
T6_df1 = inv(Tdf1_6);

% Data Frame 2
```

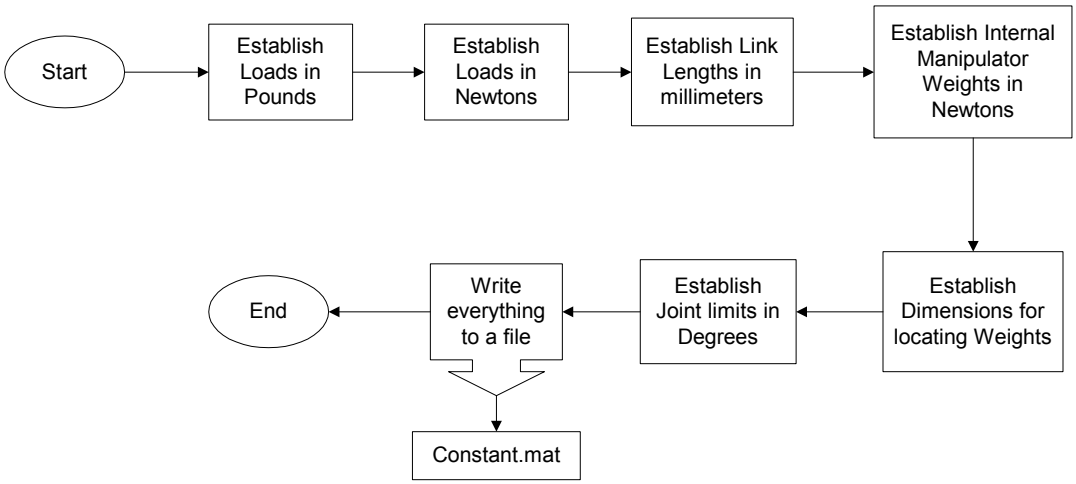
```
df2_x = data(3,:,frame) - data(2,:,frame);
df2_x = df2_x/sqrt(dot(df2_x,df2_x'));
df2_y1 = data(1,:,frame) - data(2,:,frame);
df2_y1 = df2_y1/sqrt(dot(df2_y1,df2_y1'));
df2_z = cross(df2_x,df2_y1);
df2_z = df2_z/sqrt(dot(df2_z,df2_z));
df2_y = cross(df2_z,df2_x);
df2_y = df2_y/sqrt(dot(df2_y,df2_y));

col1_df2 = [df2_x';0];
col2_df2 = [df2_y';0];
col3_df2 = [df2_z';0];
col4_df2 = [data(2,:,frame)';1];

Tdf2_opt = [col1_df2,col2_df2,col3_df2,col4_df2];

Tdf2_6 = Topt_6*Tdf2_opt
T6_df2 = inv(Tdf2_6);

% Data Frame 3

df3_x = data(4,:,frame) - data(3,:,frame);
df3_x = df3_x/sqrt(dot(df3_x,df3_x'));
df3_y1 = data(2,:,frame) - data(3,:,frame);
df3_y1 = df3_y1/sqrt(dot(df3_y1,df3_y1'));
df3_z = cross(df3_x,df3_y1);
df3_z = df3_z/sqrt(dot(df3_z,df3_z));
df3_y = cross(df3_z,df3_x);
df3_y = df3_y/sqrt(dot(df3_y,df3_y));

col1_df3 = [df3_x';0];
col2_df3 = [df3_y';0];
col3_df3 = [df3_z';0];
col4_df3 = [data(3,:,frame)';1];

Tdf3_opt = [col1_df3,col2_df3,col3_df3,col4_df3];

Tdf3_6 = Topt_6*Tdf3_opt
T6_df3 = inv(Tdf3_6);

% Data Frame 4

df4_x = data(1,:,frame) - data(4,:,frame);
df4_x = df4_x/sqrt(dot(df4_x,df4_x'));
df4_y1 = data(3,:,frame) - data(4,:,frame);
df4_y1 = df4_y1/sqrt(dot(df4_y1,df4_y1'));
df4_z = cross(df4_x,df4_y1);
df4_z = df4_z/sqrt(dot(df4_z,df4_z));
df4_y = cross(df4_z,df4_x);
df4_y = df4_y/sqrt(dot(df4_y,df4_y));

col1_df4 = [df4_x';0];
col2_df4 = [df4_y';0];
col3_df4 = [df4_z';0];
col4_df4 = [data(4,:,frame)';1];

Tdf4_opt = [col1_df4,col2_df4,col3_df4,col4_df4];
```

```
Tdf4_6 = Topt_6*Tdf4_opt
T6_df4 = inv(Tdf4_6);

save c:\data\j5j6trans.mat Tdf1_6 Tdf2_6 Tdf3_6 Tdf4_6 Tdf1_opt
Tdf2_opt Tdf3_opt Tdf4_opt
```

10.4 Miscellaneous Data Codes

10.4.1 Set Constant Routine

- Code Filename: SetConst_v022502.m
- Code Description: Code to write problem specific constants to a central location foe referencing.
- Inputs: N/A
- Outputs:
    i. Writes out file Constant.mat
- Application: Used to make sure all codes in the Optimization are using the same constants for the problem.
- Notes: This file contains the correct distances and weights and related constants for the research.

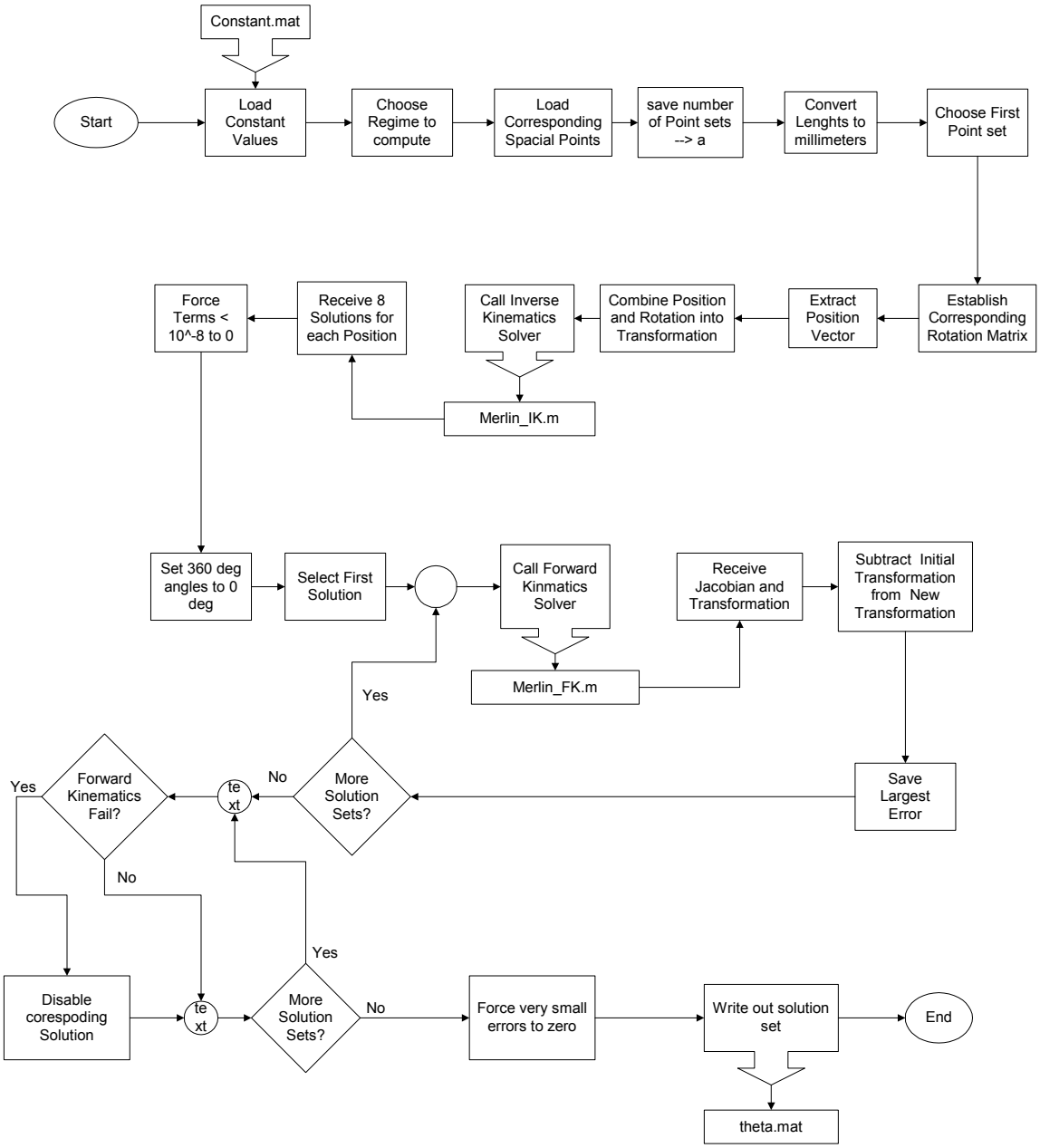



```
% Mark W. Abbott
% February 7, 2002

% SetConst.m

clear all;
close all;
clc;


% Code to write out Constant.mat
% This file contains all mechanical constants, limits and such

Loads_lbs = [5.3;10.3;15.3;20.3;23.3;50;75.3;100.3];
Loads_N = Loads_lbs/0.2248089;
loads = Loads_N;

d1 = 45.46*25.4;                % Distance from zero frame to first
frame (from floor to shoulder)
```

```
d2 = 11.9*25.4;                  % Offset from center of robot trunk
to outside of shoulder
a2 = 17.375*25.4;                % Length of Link from shoulder to
elbow
d4 = 17.25*25.4;                 % Length of Link from elbow to
wrist
d6 = 3.5*25.4;                   % Distance from center of wrist to
tool faceplate

weight = [288,240,90,0,25,0]/0.2249089;    % Weights of robotic
links (N)

dx = [-5;-17.375;0;0;0;0]*25.4;   % Displacements along the x-axis
for each weight frame rel to parent frame
dy = [0;0;0;0;0;0]*25.4;          % Displacements along the y-axis
for each weight frame rel to parent frame
dz = [0;0;-1;0;0;0]*25.4;          % Displacements along the z-axis
for each weight frame rel to parent frame
offset = [dx,dy,dz];


t1up = 175;      % Upper Limit for Joint 1
t1lwr = -115;    % Lower Limit for Joint 1
t2up = 230;      % Upper Limit for Joint 2
t2lwr = -55;     % Lower Limit for Joint 2
t3up = 135;      % Upper Limit for Joint 3
t3lwr = -135;    % Lower Limit for Joint 3
t5up = 90.25;      % Upper Limit for Joint 5
t5lwr = -90.25;    % Lower Limit for Joint 5

save c:\data\Constant.mat
fprintf('Constant.mat was saved\n');
```

10.4.2 Experimental Inverse Kinematics Routine

- Code Filename: IK_Exper.m
- Code Description: Code to generate the theoretical manipulator joint angles to meet the experimental points in the workspace.
- Inputs:
    i. Must have the workspace points defined at the top of the file
- Outputs:
    i. Writes a theta.mat file containing the selected solution
- Application: Used to convert the experimentally implemented data points into joint angles.
- Notes: There is a setting at the bottom that selects which of the 8 solutions from the IK's this code will report, it is set to the elbow-up no wrist reversal selection now. Also, the distance d1 is set to an old value, this is compensated later in the code.




```
% Mark W. Abbott
% October 5, 2001

% MATLAB code to test and verify both the forward and inverse
kinematic modules
%   as well as provide a general basis for theiur use.
%
% Based on a specified set of input thetas (theta_test), the
resulting transformation
%   is constructed and then use to verify the inverse kinematics
```

```
ability to solve the problem.
% A total of 8 solutions are possible, however due to mechanical
limits, not all will
%   be functional at every position.
%

clear all;
close all;
clc;

% ------------------------------------------------------------------
% User Definable settings


d1 = 46.45*25.4;        % Distance from zero frame to first frame
(from floor to shoulder)
d2 = 11.9*25.4;          % Offset from center of robot trunk to
outside of shoulder
a2 = 17.375*25.4;        % Length of Link from shoulder to elbow
d4 = 17.25*25.4;         % Length of Link from elbow to wrist
d6 = 3.5*25.4;           % Distance from center of wrist to tool
faceplate
regime = 3;

Error_Limit = 10^-10;
if regime == 1
pnts = [22    -4   -10   0   90   0;
    22     0   -10   0   90   0;
    22     4   -10   0   90   0;
    26    -4   -10   0   90   0;
    26     0   -10   0   90   0;
    26     4   -10   0   90   0;
    30    -4   -10   0   90   0;
    30     0   -10   0   90   0;
    30     4   -10   0   90   0;
    22    -4    -6   0   90   0;
    22     0    -6   0   90   0;
    22     4    -6   0   90   0;
    26    -4    -6   0   90   0;
    26     0    -6   0   90   0;
    26     4    -6   0   90   0;
    30    -4    -6   0   90   0;
    30     0    -6   0   90   0;
    30     4    -6   0   90   0;
    22    -4    -2   0   90   0;
    22     0    -2   0   90   0;
    22     4    -2   0   90   0;
    26    -4    -2   0   90   0;
    26     0    -2   0   90   0;
    26     4    -2   0   90   0;
    30    -4    -2   0   90   0;
    30     0    -2   0   90   0;
    30     4    -2   0   90   0];
elseif regime == 2
    pnts = [15.5    12  -12     0   90  0;
    24.4    12  -12     0   90  0;
    33.3    12  -12     0   90  0;
```

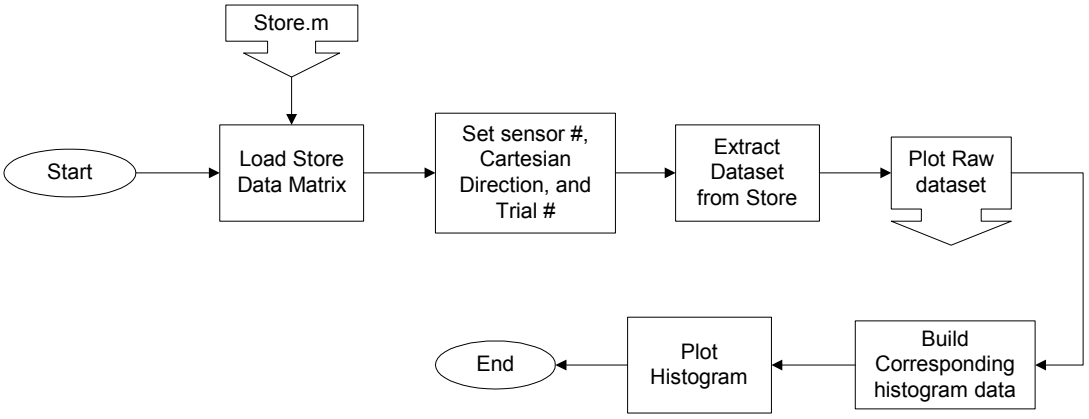
```
    17      12  0       0   90  0;
    25.5    12  0       0   90  -15;
    34      12  0       0   90  -30;
    24      12  12      0   90  -30;
    19.5    12  12      0   90  -35;
    15      12  12      0   90  -50;
    15.5    0   12      0   90  -25;
    21      0   12      0   90  0;
    27      0   12      0   90  0;
    36      0   0       0   90  0;
    28.5    0   0       0   90  0;
    21      0   0       0   90  0;
    19.5    0   -12     0   90  0;
    27.25   0   -12     0   90  0;
    35      0   -12     0   90  0;
    33      -12 -12     0   90  0;
    24.25   -12 -12     0   90  0;
    15.5    -12 -12     0   90  0;
    17      -12 0       0   90  0;
    25.5    -12 0       0   90  0;
    34      -12 0       0   90  0;
    24      -12 12      0   90  0;
    17      -12 12      0   90  0;
    10      -12 12      0   90  0];
elseif regime == 3
    pnts = [25  -24 4.5  0   90  0;
            12  -24 4.5 0   90  0;
            25  16  4.5 0   90  0;
            12  16  4.5 0   90  0;
            20  -8.3    0   0   90  0;
            25  -24 -10 0   90  0;
            12  -24 -10 0   90  0;
            25  16  -10 0   90  0;
            12  16  -10 0   90  0
            20    -8.3    -9.512  0   90  0;
            25  -24 -3.038  0   90  0;
            12  -24 -2.337  0   90  0];
end

% ---------------------------------------------------------------------

[a,b] = size(pnts);


% Convert Robotic Data into Metric Units

for i = 1:a
    pnts(i,1:3) = pnts(i,1:3)*25.4;
end

Solutions = zeros(6,a);

for q = 1:a

    R = [1,0,0;0,-1,0;0,0,-1];

    d = [pnts(q,1);pnts(q,2);pnts(q,3) + d1*25.4];
```

```
T06 = [[R,d];0,0,0,1];

OPTIONS = [];
Verify = Merlin_IK(R,d,OPTIONS,d1,d2,a2,d4,d6);

Output = Verify*180/pi;

% Force very small terms to zero. and 360 -> 0

for x = 1:6
    for y = 1:8
        if abs(Output(x,y)) < 10^-8
            Output(x,y) = 0;
        end
        if Output(x,y) == 360
            Output(x,y) = 0;
        end
    end
end

% Begin Verifying Solutions with Forward Kinematics

CK = zeros(8,1);

[J_1,T06_1] = Merlin_FK2(Verify(:,1),d1,OPTIONS,d2,a2,d4,d6);
Check1 = T06 - T06_1;
CK(1) = max(max(abs(Check1)));

[J_2,T06_2] = Merlin_FK2(Verify(:,2),d1,OPTIONS,d2,a2,d4,d6);
Check2 = T06 - T06_2;
CK(2) = max(max(abs(Check2)));

[J_3,T06_3] = Merlin_FK2(Verify(:,3),d1,OPTIONS,d2,a2,d4,d6);
Check3 = T06 - T06_3;
CK(3) = max(max(abs(Check3)));

[J_4,T06_4] = Merlin_FK2(Verify(:,4),d1,OPTIONS,d2,a2,d4,d6);
Check4 = T06 - T06_4;
CK(4) = max(max(abs(Check4)));

[J_5,T06_5] = Merlin_FK2(Verify(:,5),d1,OPTIONS,d2,a2,d4,d6);
Check5 = T06 - T06_5;
CK(5) = max(max(abs(Check5)));

[J_6,T06_6] = Merlin_FK2(Verify(:,6),d1,OPTIONS,d2,a2,d4,d6);
Check6 = T06 - T06_6;
CK(6) = max(max(abs(Check6)));

[J_7,T06_7] = Merlin_FK2(Verify(:,7),d1,OPTIONS,d2,a2,d4,d6);
Check7 = T06 - T06_7;
CK(7) = max(max(abs(Check7)));

[J_8,T06_8] = Merlin_FK2(Verify(:,8),d1,OPTIONS,d2,a2,d4,d6);
Check8 = T06 - T06_8;
CK(8) = max(max(abs(Check8)));
```

```
    % Establish FKE vector to regulate output to valid solutions
only.

    FKE = zeros(8,1);

    if J_1 == 0 & T06_1 == 0
        FKE(1) = 1;
    end
    if J_2 == 0 & T06_2 == 0
        FKE(2) = 1;
    end
    if J_3 == 0 & T06_3 == 0
        FKE(3) = 1;
    end
    if J_4 == 0 & T06_4 == 0
        FKE(4) = 1;
    end
    if J_5 == 0 & T06_5 == 0
        FKE(5) = 1;
    end
    if J_6 == 0 & T06_6 == 0
        FKE(6) = 1;
    end
    if J_7 == 0 & T06_7 == 0
        FKE(7) = 1;
    end
    if J_8 == 0 & T06_8 == 0
        FKE(8) = 1;
    end

    % Force very small errors to zero for output

    for i = 1:8
        if CK(i) < Error_Limit
            CK(i) = 0;
        end
    end

    if FKE(4) == 1
        Solutions(:,q) = [0;0;0;0;0;0];
    else
        Solutions(:,q) = Output(:,4);
    end

end

if regime == 1
    save c:\data\r1\theta.mat Solutions
elseif regime== 2
    save c:\data\r2\theta.mat Solutions
elseif regime == 3
    save c:\data\r3\theta.mat Solutions
end
```

10.4.3 Data Histogram Plot Routine

- Code Filename: histplots.m
- Code Description: Code to build histogram from results.
- Inputs:
  i. Must Specify the data stream to compute
  ii. Must specify the Store.m file to open
- Outputs:
  i. Plots only
- Application: Used to generate plots to inspect how the results from the OptoTrak are distributed.
- Notes: N/A

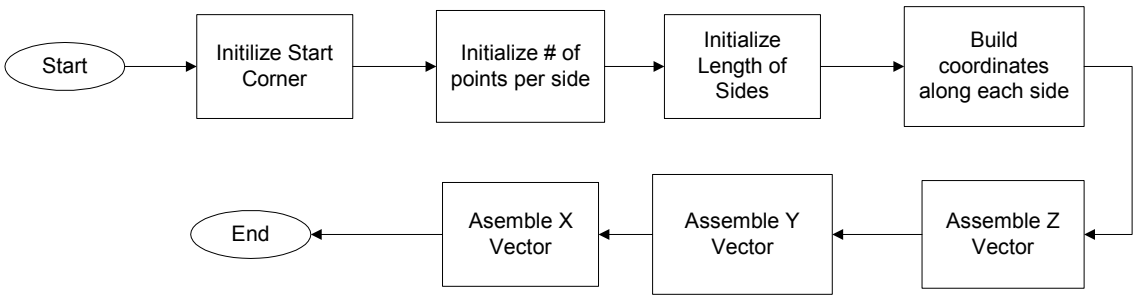



```
% Mark W. Abbott
% March 12, 2002

% Code to Generate Normal plots and histograms

clear all;
close all;
clc;

load c:\data\r3\Store.mat;

a = 3;       % Choose Sensor Number
b = 3;       % Choose X, Y or Z
d = 110;      % Choose Trial Number

dataset = squeeze(Store(a,b,:,d));

figure
plot(dataset)
xlabel('Data Point');
ylabel('Distance (mm)');

var = ['X','Y','Z'];

title1 = sprintf('Raw Data for Sensor %g, in the %s Direction, during
```

```
Test #%g',a,var(b),d);
title(title1)

[N,X] = hist(dataset,30);

figure
hist(dataset,30)
hold on
normalize = max(N);
N_1 = N/normalize;
plot(X,gausswin(30)*max(N));
title2 = sprintf('Histogram for Sensor %g, in the %s Direction,
during Test #%g',a,var(b),d);
title(title2)
```

10.4.4 Points Algorithm

- Code Filename: points.m
- Code Description: Code to generate equally spaced points in a cubic volume.
- Inputs:
  i. Start Corner of cube
  ii. Number of Points per Edge
  iii. Length of Cube Side
- Outputs:
  i. Builds pnts matrix in memory
- Application: Used to generate the point spread for the 1st regime of testing.
- Notes: N/A



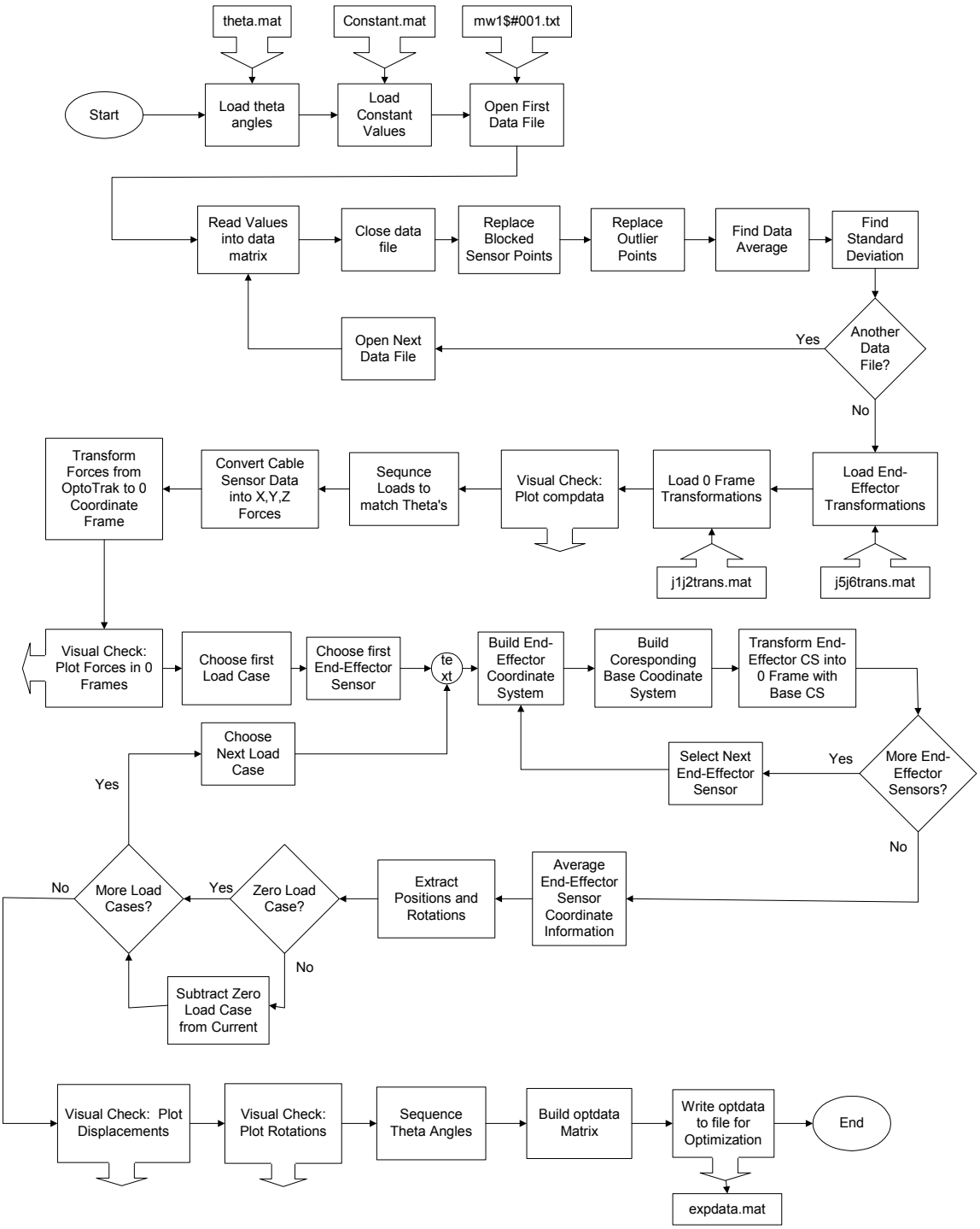

```
% Mark W. Abbott
% Thesis Research

clear all;
close all;
clc;


% Generate points in a square workspace to send the robot to for
compliance measurements

start = [20 -6 -12];
```

```
side = 3;        % NUmbe of Points per edge
size = 12;        % Length of sides of cube

num = side^3;    % Total Number of Points in Workspace

sc_length = size/side;            % length of each subcube

sc_start = sc_length/2;           % Start of subcube

coordinate = zeros(side,1);

for i = 1:side
    coordinate(i) = sc_start + sc_length*(i - 1);
end

pnts = zeros(num,3);         % X, Y, Z

for j = 0:num/side - 1       % Build the Z Vector First
    for i = 1:side
        pnts(i + side*j,3) = coordinate(i);
    end
end

for k = 0:side - 1           % Build the Y Vector
    for m = 1:side
        pnts(m*side - side + 1 + side^2*k:m*side + side^2*k,2) =
coordinate(m);
    end
end

for n = 1:side               % Build the Z Vector
    pnts(n*side^2 - side^2 + 1:n*side^2,1) = coordinate(n);
end

for i = 1:num
    pnts(i,:) = start + pnts(i,:);
end
```

10.4.5 Experimental Data Compiler

- Code Filename: Expdata.m
- Code Description: Code to reduce experimental results into required format for optimization routine.
- Inputs:
    i. Must specify the regime
    ii. Must specify the number of loads
    iii. Correct theta.mat
    iv. j1j2trans.mat
    v. j5j6trans.mat
- Outputs:
    i. Store.mat a file with all of the data read from the files in a matrix
    ii. Expdata.mat file

- Application: This code reduces the OptoTrak data for an entire regime into the correctly formatted and transformed results to pass to the optimization routines. Results are stored in the expdata.mat files.
- Notes: This code assumes a file name convention, must be updated if it is not adhered to.



```
% Mark W. Abbott
% December 4, 2001

% Code to generate expdata.mat file for Optimization to read
% Writes a Matrix OptData of the follow format:
%
%  Fx  Fy  Fz  T1  T2  T3  T4  T5  T6  dx  dy  dz  rx  ry  rz
%
% F is the the applied force,
% T is the Theta angles
% d is the measured change in displacement (at Frame 6)
% r is the measured change in rotation
%

clear all;
close all;
clc;

% User Specified Settings

regime = 2;

numofloads = 243;

diagnosticplots = 1;              % 1 = yes, 0 = no
reportplots = 0;                  % 1 = yes, 0 = no
buildStore = 1;                   % 1 = yes, 0 = no

load c:\data\Constant.mat

% Begin Code

if regime == 3
   load('c:\data\r3\theta.mat');   % Inserts Theta into Workspace
   localpath = 'c:\data\r3\';
elseif regime == 2
   load('c:\data\r2\theta.mat');   % Inserts Theta into Workspace
   localpath = 'c:\data\r2\';
elseif regime == 1
   load('c:\data\r1\theta.mat');   % Inserts Theta into Workspace
   localpath = 'c:\data\r1\';
else
   fprintf('Warning:  Invalid Regime!\n');
end


tic

loadset = zeros(numofloads,1);

Store = zeros(14,3,400,243);

path1 = strcat(localpath,'backup.mat');
testval = fopen(path1);
if testval == -1
```

```
   compdata = zeros(14,3,numofloads);
   compstd = zeros(14,3,numofloads);

   for k = 1:numofloads

      % Build Filename and path into a string

      filenumber = k;
      numstr= num2str(filenumber);

      if filenumber < 100
         numstr = strcat('0',numstr);
      end
      if filenumber < 10
         numstr = strcat('0',numstr);
      end

      filename = strcat(localpath,'mw1$#',numstr,'.mwa');

      % Establish data matrix and open file

      data = zeros(14,3,400);
      fid = fopen(filename);
      fid2 = fopen('c:\data\r3\expdata_log.txt','a');


      fprintf('\nOpening %s', filename);
      fseek(fid2,0,'eof');
      junk = fprintf(fid2,'\nOpening %s', filename);

      titletrash = fgetl(fid);

      % Fill data matrix

      for i = 1:400
         for j = 1:14
            tline = fgetl(fid);
            tlinenum = str2num(tline);
            data(j,:,i) = tlinenum(length(tlinenum) -
2:length(tlinenum));
         end
      end

      fprintf('\Closing %s', filename);
      fseek(fid2,0,'eof');
      junk = fprintf(fid2,'\Closing %s', filename);

      fclose(fid);
      data1 = data;

      % Cleanse Data of missed points etc

      for i = 1:14
         for j = 1:3
            good = find(abs(data(i,j,:)) < 10^10);
            bad = find(abs(data(i,j,:)) > 10^10);
```

```
            if length(bad) < 0.1*400
               ave_tmp = 0;
               for m = 1:length(good)
                  ave_tmp = ave_tmp + data(i,j,good(m));
               end

               ave_tmp = ave_tmp/length(good);

               for m = 1:length(bad)
                  data(i,j,bad(m)) = ave_tmp;
                  fprintf('\nReplacing data(%g,%g,%g) due to blocked
sensor',i,j,bad(m));
                  junk = fprintf(fid2,'\nReplacing data(%g,%g,%g) due
to blocked sensor',i,j,bad(m));
               end
            else
               fprintf('\nWarning:  data(%g,%g,:) has too many
blocked....',i,j);
               junk = fprintf(fid2,'\nWarning:  data(%g,%g,:) has too
many blocked....',i,j);

               %if length(bad) > 100
               data(i,j,:) = 0;
               fprintf('\n%g points in error:  Data set to
zero.',length(bad));
               junk = fprintf(fid2,'\n%g points in error:  Data set
to zero.',length(bad));
               %else
               %%    fprintf('\nCritical Error:  %g points blocked...
require solution.',length(bad));
               %    junk = fprintf(fid2,'\nCritical Error:  %g points
blocked...  require solution.',length(bad));
               %end
            end
         end
      end

      % Cleanse Data of erroneus Measurements etc

      range = 1;

      for i = 1:14
         for j = 1:3
            range = 1;
            while range <= 5
               ave_orig = mean(data(i,j,:));
               data_tmp = data(i,j,:) - ave_orig;
               good1 = find(abs(data_tmp) <= range);
               bad1 = find(abs(data_tmp) > range);
               if length(bad1) <= 0.1*400 & length(bad1) ~= 0
                  sum_tmp = 0;
                  for m = 1:length(good1)
                     sum_tmp = sum_tmp + data(i,j,good1(m));
                  end

                  ave_tmp = sum_tmp/length(good1);
```

```
                  for m = 1:length(bad1)
                     data(i,j,bad1(m)) = ave_tmp;
                     fprintf('\nReplacing data(%g,%g,%g) due to
erroneus measurement   %g   %g',i,j,bad1(m),ave_orig,ave_tmp);
                     junk = fprintf(fid2,'\nReplacing data(%g,%g,%g)
due to erroneus measurement',i,j,bad1(m)');

                  end
                  range = 6;
               elseif length(bad1) > 0.1*400
                  range = range + 0.5;
                  if range > 5
                     fprintf('\Critical Error:  data(%g,%g,:) has too
points (%g) many outside range....',i,j,length(bad1));
                     junk = fprintf(fid2,'\Critical Error:
data(%g,%g,:) has too points (%g) many outside
range....',i,j,length(bad1));

                  end
                  fprintf('\nModifying Range Variable: %g', range);
                  junk = fprintf(fid2,'\nModifying Range Variable:
%g', range);

               elseif length(bad1) == 0
                  range = 6;
               end
            end
         end
      end


      % Build Storage Matrix to Analyze Data

      if buildStore == 1
         Store(:,:,:,k) = data;
      end

      % Extract the mean value from the data matrix

      meandata = zeros(14,3);
      stddata = zeros(14,3);

      for i = 1:14
         for j = 1:3
            meandata(i,j) = mean(data(i,j,:));
            stddata(i,j) = std(data(i,j,:));
         end
      end

      compdata(:,:,k) = meandata;
      compstd(:,:,k) = stddata;

      fclose(fid2);
   end

   if regime == 1
      save c:\data\r1\backup.mat compdata compstd;
```

```
      if buildStore == 1
         save c:\data\r1\Store.mat Store
      end
   elseif regime == 2
      save c:\data\r2\backup.mat compdata compstd;
      if buildStore == 1
         save c:\data\r2\Store.mat Store
      end
   elseif regime == 3
      save c:\data\r3\backup.mat compdata compstd;
      if buildStore == 1
         save c:\data\r3\Store.mat Store
      end
   end
else
   load c:\data\r3\backup.mat
end

fclose('all');
toc

load c:\data\j1j2trans.mat
load c:\data\j5j6trans.mat

% Code to replace 6 bad data points due to overloading of robotic arm
% Uses a 2nd order curve fit to approximate the bad point (assumes it
is the 100 lb case)

compdata_mod = compdata;
if localpath == 'c:\data\r3\'
   badpnts = [99;144;171;198;216;243];
   loadstrim = [0;loads(1:7)];

   for i = 1:length(badpnts)
      for j = 1:14
         for k = 1:3
            [p,s] = polyfit(loadstrim,squeeze(compdata(j,k,badpnts(i)
- 8:badpnts(i) - 1)),2);
            compdata_mod(j,k,badpnts(i)) = polyval(p,loads(8));
         end
      end
   end
end

% Code to correct for a single bug on Sensor 12 trial # 39

if regime == 3

   usepoints = [37,38,40,41,42,43,44,45];
   [p,s] = polyfit(loadstrim,squeeze(compdata(12,3,usepoints)),2);
   compdata_mod(12,3,39) = polyval(p,loads(3));

   figure
   for i = 1:length(badpnts)
      subplot(3,2,i); plot([0;loads],squeeze(compdata(1,1,badpnts(i)
- 8:badpnts(i))),'--r');
      hold on
```

```
      subplot(3,2,i);
plot([0;loads],squeeze(compdata_mod(1,1,badpnts(i) -
8:badpnts(i))),'.-b');
      hold off
   end
end


% compdata_mod[x,y,z] is formatted [1:14,1:3,1:243]
%   14 sensors
%   3 components x,y,z
%   243 load cases (27 zeros, and 216 loads)

% Plot Resultant Data to scan for Errors

figure
subplot(3,1,1); plot(squeeze(compdata_mod(:,1,:))')
title('X-Dir Position Data');
subplot(3,1,2); plot(squeeze(compdata_mod(:,2,:))')
title('Y-Dir Position Data');
subplot(3,1,3); plot(squeeze(compdata_mod(:,3,:))')
title('Z-Dir Position Data');

% Building a loadset to place against thetas

for i = 1:27
   loadset((i-1)*9 + 1:9*i) = [0;loads];
end

% Build Load Vectors
% LoadVect stores the x, y and z compenents of the applied forces
% This stage takes the cable sensors and converts the loading into X,
Y and Z components

LoadVect_opt = zeros(numofloads, 3);
LoadVect_0 = zeros(numofloads, 3);

fprintf('\nBeginning Load Vector Computations\n');

for i = 1:numofloads

   Lva = compdata(10,1:3,i) - compdata(9,1:3,i);
   Lvau = Lva/sqrt(dot(Lva,Lva));
   Fva = loadset(i)/2*Lvau;

   Lvb = compdata(8,1:3,i) - compdata(7,1:3,i);
   Lvbu = Lvb/sqrt(dot(Lvb,Lvb));
   Fvb = loadset(i)/2*Lvbu;

   LoadVect_opt(i,:) = Fva + Fvb;
end

% Must Convert the LoadVect from the OptoTrak Frame to the Robot
Frame


% Topt_0 converts data in the OptoTrak frame to the 0 frame
```

```
% Due to numerical roundoff issues, Topt_0 must be cleaned

R = Topt_0(1:3,1:3);
R1 =
[R(1,:)/sqrt(dot(R(1,:),R(1,:)));R(2,:)/sqrt(dot(R(2,:),R(2,:)));
R(3,:)/sqrt(dot(R(3,:),R(3,:)))];
R2 = [R1(:,1)/sqrt(dot(R1(:,1),R1(:,1))),R1(:,2)/sqrt(dot(R1(:,2),
R1(:,2))),
R1(:,3)/sqrt(dot(R1(:,3),R1(:,3)))];

for i = 1:numofloads
   LoadVect_0(i,:) = (Topt_0(1:3,1:3)*LoadVect_opt(i,:)')';
end

% Plot the Components of the Applied Load

figure
subplot(3,1,1); plot(LoadVect_opt(:,1));
title('Applied Load, X-Dir (Opt Frame)');
subplot(3,1,2); plot(LoadVect_opt(:,2));
title('Applied Load, Y-Dir (Opt Frame)');
subplot(3,1,3); plot(LoadVect_opt(:,3));
title('Applied Load, Z-Dir (Opt Frame)');

% Build a displacement matrix:   Disp(dx dy dz)
% I am only addressing the first 4 sensors at this point


% I must build the 4 coordinate frame from the data, and transform
back to 6th frame

% Build frame using points as follows
%
% Frame Number comes from point at the intersection of all three axis
% Z-Axis is orthoganol to plane of points
% X-Axis goes from the base point to the next point numerically
(CounterClockwise)

fprintf('\nBeginning Coordinate Frame Assignment and Data
Extraction\n');

Tdf_6(:,:,1) = Tdf1_6;
Tdf_6(:,:,2) = Tdf2_6;
Tdf_6(:,:,3) = Tdf3_6;
Tdf_6(:,:,4) = Tdf4_6;
dist = zeros(243,1);
Disp = zeros(numofloads,3);
Rot = zeros(numofloads,3);

for i = 1:27
   for j = 1:9
      % Generate the pose for all 4 sensors, then average them
      for k = 1:4
         if k == 1
            x1 = 2; y1 = 4;
         elseif k == 2
            x1 = 3; y1 = 1;
```

```
        elseif k == 3
           x1 = 4; y1 = 2;
        elseif k == 4
           x1 = 1; y1 = 3;
        end

        dfx = compdata_mod(x1,:,(i-1)*9 + 1) - compdata_mod(k,:,(i-
1)*9 + 1);
        dfx = dfx/sqrt(dot(dfx,dfx));
        dfy1 = compdata_mod(y1,:,(i-1)*9 + 1) - compdata_mod(k,:,(i-
1)*9 + 1);
        dfy1 = dfy1/sqrt(dot(dfy1,dfy1));
        dfz = cross(dfx,dfy1);
        dfz = dfz/sqrt(dot(dfz,dfz));
        dfy = cross(dfz,dfx);
        dfy = dfy/sqrt(dot(dfy,dfy));

        c1 = [dfx';0];
        c2 = [dfy';0];
        c3 = [dfz';0];
        c4 = [compdata_mod(k,:,(i-1)*9 + j)';1];

        Tdf_opt = [c1,c2,c3,c4];

        % Build current OptoTrak to Base Frame Transform

        sensor_14 = compdata_mod(14,:,9*(i-1) + j);
        sensor_11 = compdata_mod(11,:,9*(i-1) + j);
        sensor_12 = compdata_mod(12,:,9*(i-1) + j);

        x_base = sensor_12 - sensor_14;
        x_base = x_base/sqrt(dot(x_base,x_base));

        y_base1 = sensor_11 - sensor_14;
        y_base1 = y_base1/sqrt(dot(y_base1,y_base1));

        z_base = cross(x_base,y_base1);
        z_base = z_base/sqrt(dot(z_base,z_base));
        y_base = cross(z_base,x_base);
        y_base = y_base/sqrt(dot(y_base,y_base));

        col1_base = [x_base';0];
        col2_base = [y_base';0];
        col3_base = [z_base';0];
        col4_base = [sensor_14';1];

        Tbase_opt = [col1_base,col2_base,col3_base,col4_base];

        %Build current OptoTrak to 0 frame Transform

        Topt_0 = Tbase_0*inv(Tbase_opt);

        T6_opt = Tdf_opt*inv(Tdf_6(:,:,k));
        T6a_0(:,:,k) = Topt_0*T6_opt;

    end
```

```
      % Extract the displacement components
      dx = squeeze(T6a_0(1,4,:));
      dy = squeeze(T6a_0(2,4,:));
      dz = squeeze(T6a_0(3,4,:));
      dist(9*(i-1) + j) = sqrt(sum(dx)^2 + sum(dy)^2 + sum(dz)^2);

      % Build and extract the rotation components

      pitch = zeros(4,1);
      roll = zeros(4,1);
      yaw = zeros(4,1);

      for k = 1:4
         pitch(k) = asin(T6a_0(3,1,k));
         roll(k) = atan2(T6a_0(2,1,k),T6a_0(1,1,k));
         yaw(k) = atan2(T6a_0(3,2,k),T6a_0(3,3,k));
      end

      if j == 1
         pitch_0 = mean(pitch);
         roll_0 = mean(roll);
         yaw_0 = mean(yaw);
         dx_0 = mean(dx);
         dy_0 = mean(dy);
         dz_0 = mean(dz);
         Disp(9*(i-1) + j,:) = [0, 0, 0];
         Rot(9*(i-1) + j,:) = [0, 0, 0];
      else
         pitch_1 = mean(pitch) - pitch_0;
         roll_1 = mean(roll) - roll_0;
         yaw_1 = mean(yaw) - yaw_0;
         dx_1 = mean(dx) - dx_0;
         dy_1 = mean(dy) - dy_0;
         dz_1 = mean(dz) - dz_0;
         Disp(9*(i-1) + j,:) = [dx_1, dy_1, dz_1];
         Rot(9*(i-1) + j,:) = [roll_1, pitch_1, yaw_1];
      end
   end
end


% Make Pretty Pictures

figure
subplot(3,1,1); plot(Disp(:,1))
title('Displacement X-Dir');
subplot(3,1,2); plot(Disp(:,2))
title('Displacement Y-Dir');
subplot(3,1,3); plot(Disp(:,3))
title('Displacement Z-Dir');

figure
subplot(3,1,1); plot(Rot(:,1))
title('Roll (About X)');
subplot(3,1,2); plot(Rot(:,2))
title('Pitch (About Y)');
subplot(3,1,3); plot(Rot(:,3))
```

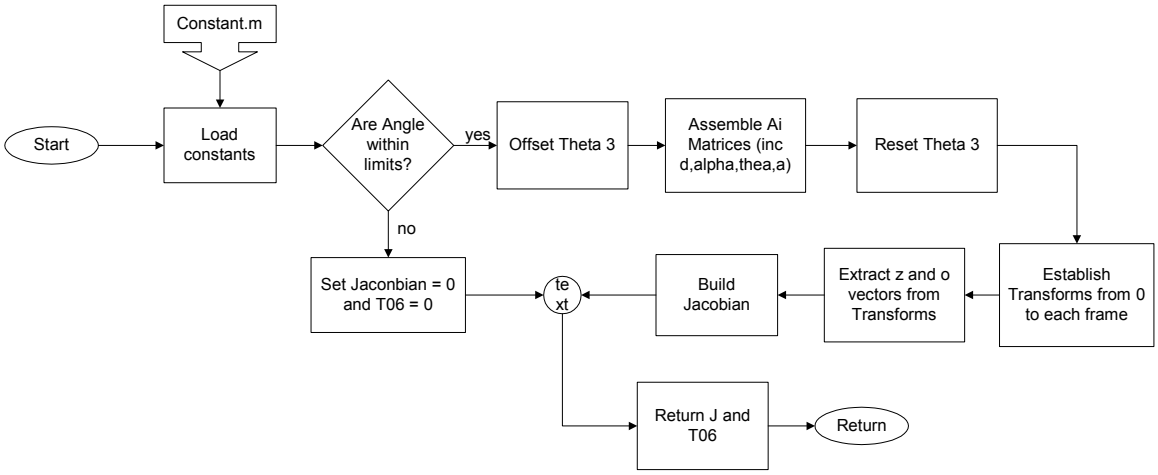
```
title('Yaw (About Z)');

figure
for i = 1:27
   subplot(3,1,1); plot(Rot(9*(i-1) + 1:9*(i-1) + 9,1))
   title('Roll (About X)');
   hold on
   subplot(3,1,2); plot(Rot(9*(i-1) + 1:9*(i-1) + 9,2))
   title('Pitch (About Y)');
   hold on
   subplot(3,1,3); plot(Rot(9*(i-1) + 1:9*(i-1) + 9,3))
   title('Yaw (About Z)');
   hold on
end
hold off

% Format OptData to write information out for Optimization
%  Fx  Fy  Fz  T1  T2  T3  T4  T5  T6  dx  dy  dz  rx  ry  rz

if regime == 1
   load c:\data\r1\theta.mat
elseif regime == 2
   load c:\data\r2\theta.mat
elseif regime == 3
   load c:\data\r3\theta.mat
end

tmp = Solutions';

if regime == 3

   for i = 1:27
      for j = 1:9
         if i == 1 | i == 19
            Theta(9*(i-1) + j,:) = tmp(1,:);
         elseif i == 2 |i == 20
            Theta(9*(i-1) + j,:) = tmp(2,:);
         elseif i == 6 | i == 18 | i == 22
            Theta(9*(i-1) + j,:) = tmp(3,:);
         elseif i == 7 | i == 17 | i == 21
            Theta(9*(i-1) + j,:) = tmp(4,:);
         elseif i == 5 | i == 23
            Theta(9*(i-1) + j,:) = tmp(5,:);
         elseif i == 4 | i == 10 | i == 27
            Theta(9*(i-1) + j,:) = tmp(6,:);
         elseif i == 3 | i == 11 | i == 26
            Theta(9*(i-1) + j,:) = tmp(7,:);
         elseif i == 9 | i == 12 | i == 24
            Theta(9*(i-1) + j,:) = tmp(8,:);
         elseif i == 8 | i == 13 | i == 25
            Theta(9*(i-1) + j,:) = tmp(9,:);
         elseif i == 14
            Theta(9*(i-1) + j,:) = tmp(10,:);
         elseif  i == 15
            Theta(9*(i-1) + j,:) = tmp(11,:);
         elseif i == 16
            Theta(9*(i-1) + j,:) = tmp(12,:);
```

```
         else
            fprintf('you goofed\n');
         end
      end
   end

else
   for i = 1:27
      for j = 1:9
         Theta(9*(i-1) + j,:) = tmp(i,:);
      end
   end
end

optdata = [LoadVect_0, Theta, Disp, Rot];

if regime == 1
   save c:\data\r1\expdata.mat optdata
elseif regime == 2
   save c:\data\r2\expdata.mat optdata
elseif regime == 3
   save c:\data\r3\expdata.mat optdata
end
```

10.5 Optimization and Compliance Codes

10.5.1 Compliant Forward Kinematics

- Code Filename: forwardk_v022502.m
- Code Description: Generalized forward kinematics code allowing input of all DH variables
- Inputs:
  i. Theta joint angles
  ii. Alpha joint angles
  iii. 'a' displacement distances
  iv. 'd' displacement distances
- Outputs:
  i. Transformation Matrix between 0 and 6th frames
  ii. Jacobian Matrix between 0 and 6th frame
- Application: This code is called from the optimization code to compute the deflected shape of the manipulator, based on modified DH terms.
- Notes: N/A



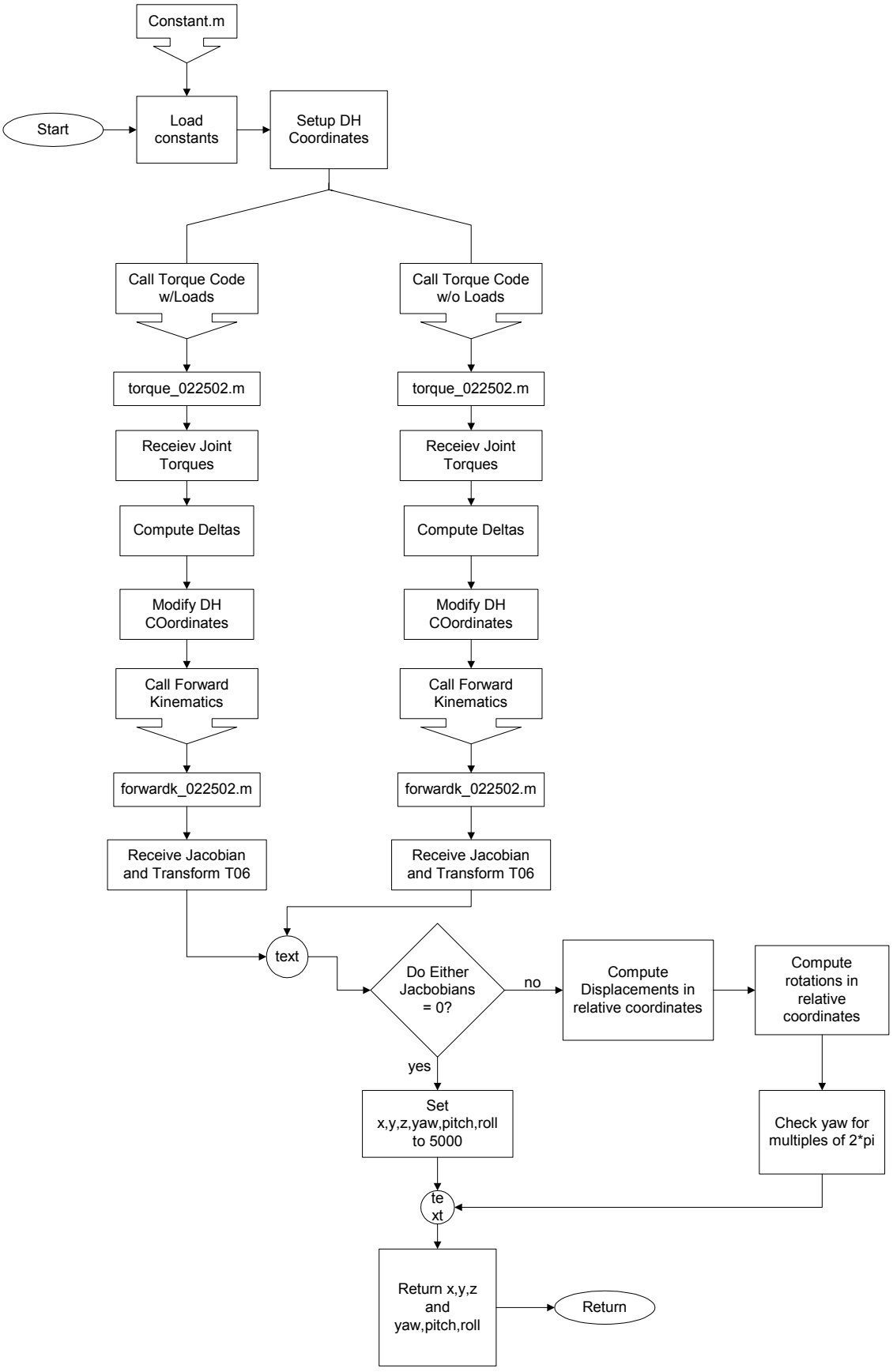

```
% Merlin Forward Kinematic Solver
% Mark W. Abbott
% October 5, 2001

% MATLAB code to compute the forward kinematics and Jacobian for 6-
DOF Merlin Robots
%
% Returns the Jacobian and the Transformation Matrix between frames
0 and 6
% If desired joint angles exceed mechanical limits, both output are
set to 0.
%

function [J,T06] = forwardk_v022502(t,a1,FLAG,a,d);

% ------------------------------------------------------------------
```

```
---
% User definable Settings

load c:\data\Constant.mat

% ------------------------------------------------------------------
---

% Test Angles to ensure they are within the mechanical limits

ta = t*180/pi;
Error = 0;

if ta(1) > t1up | ta(1) < t1lwr
   % fprintf('\nTheta 1 (%5.5g) is outside the work
envelope\n',ta(1));
    Error = 1;
end

if ta(2) < t2lwr | ta(2) > t2up
   % fprintf('\nTheta 2 (%5.5g) is outside the work
envelope\n',ta(2));
    Error = 1;
end

if ta(3) > t3up | ta(3) < t3lwr
   % fprintf('\nTheta 3 (%5.5g) is outside the work
envelope\n',ta(3));
    Error = 1;
end

if ta(5) > t5up | ta(5) < t5lwr
   % fprintf('\nTheta 5 (%5.5g) is outside the work
envelope\n',ta(5));
    Error = 1;
end

if Error == 0

    % Setting up Transformation Matrices from one frame to the next
A1 is from frame 0 to 1

    A = zeros(4,4,6);

    t(3) = t(3) + pi/2;

    for i = 1:6
        A(:,:,i) = [cos(t(i)),-
sin(t(i))*cos(a1(i)),sin(t(i))*sin(a1(i)),a(i)*cos(t(i));...
                sin(t(i)),cos(t(i))*cos(a1(i)),-
cos(t(i))*sin(a1(i)),a(i)*sin(t(i));...
                0,sin(a1(i)),cos(a1(i)),d(i);0,0,0,1];
    end

    t(3) = t(3) - pi/2;

    A01 = squeeze(A(:,:,1));
```

```
    A12 = squeeze(A(:,:,2));
    A23 = squeeze(A(:,:,3));
    A34 = squeeze(A(:,:,4));
    A45 = squeeze(A(:,:,5));
    A56 = squeeze(A(:,:,6));

    % Establishing Transformations from frame 0 to each frame

    T01=A01;
    T02=T01*A12;
    T03=T02*A23;
    T04=T03*A34;
    T05=T04*A45;
    T06=T05*A56;

    % Extracting z and o vectors for Jacobian Development

    z0=[0;0;1];
    z1=T01(1:3,3);
    z2=T02(1:3,3);
    z3=T03(1:3,3);
    z4=T04(1:3,3);
    z5=T05(1:3,3);
    z6=T06(1:3,3);
    o0=[0;0;0];
    o1=T01(1:3,4);
    o2=T02(1:3,4);
    o3=T03(1:3,4);
    o4=T04(1:3,4);
    o5=T05(1:3,4);
    o6=T06(1:3,4);

    % Crossing vectors to create Jacobian

    J1=[cross(z0,o6 - o0);z0];
    J2=[cross(z1,o6 - o1);z1];
    J3=[cross(z2,o6 - o2);z2];
    J4=[cross(z3,o6 - o3);z3];
    J5=[cross(z4,o6 - o4);z4];
    J6=[cross(z5,o6 - o5);z5];

    J = [J1, J2, J3, J4, J5, J6];

else
%    fprintf('Due to the Errors above this point,\n');
%    fprintf('Calculation of the Forward Kinmatics has been
terminated\n\n');
    J = 0;  T06 = 0;
end
```

10.5.2 Compliance Model Algorithm

- Code Filename: compliant_v022502.m
- Code Description: Basic compliant model of the robot.
- Inputs:
    i. Stiffness Matrix, K
    ii. Loads applied to the end-effector, Endload
    iii. Theta joint angles, Theta
    iv. Directly Applied torques, apptor
- Outputs:
    i. Change in position and orientation from a no load state
- Application: This code is used by the objective function to calculate the change a given load causes in end-effector position and orientation.
- Notes: N/A


Constant.m
Start
Load constants
Setup DH Coordinates
Call Torque Code w/Loads
Call Torque Code w/o Loads
torque_022502.m
torque_022502.m
Receiev Joint Torques
Receiev Joint Torques
Compute Deltas
Compute Deltas
Modify DH COordinates
Modify DH COordinates
Call Forward Kinematics
Call Forward Kinematics
forwardk_022502.m
forwardk_022502.m
Receive Jacobian and Transform T06
Receive Jacobian and Transform T06
text
Do Either Jacbobians = 0?
no
Compute Displacements in relative coordinates
Compute rotations in relative coordinates
yes
Set x,y,z,yaw,pitch,roll to 5000
Check yaw for multiples of 2*pi
te xt
Return x,y,z and yaw,pitch,roll
Return

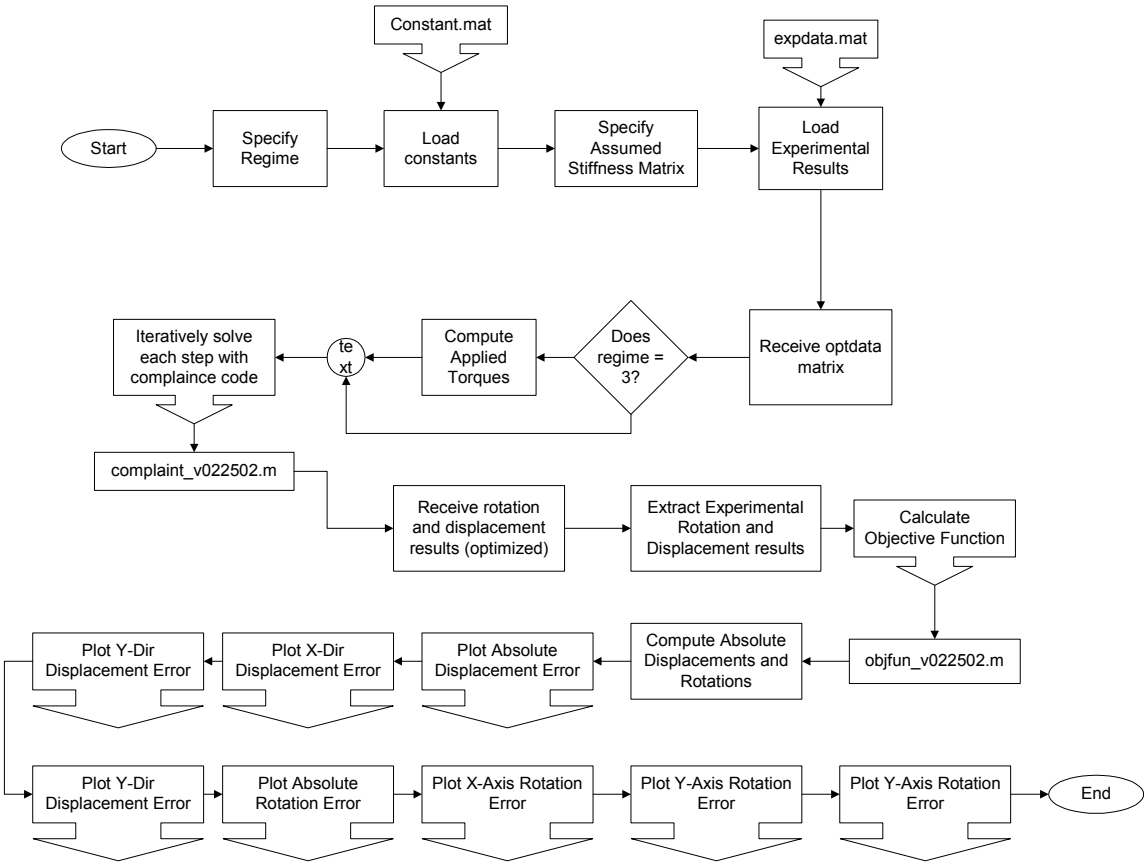
```
% Mark W. Abbott
% 10 - 26 - 2001

% Code to Predict the displaced form of the 6-DOF Merlin manipulator

function [ModPos] = Compliant_v022502(K,Endload,Theta,apptor);

% ------------------------------------------------------------------
---

load c:\data\Constant.mat

di = [d1;d2;0;d4;0;d6];
ai = [0;a2;0;0;0;0];
alphai = pi/2*[1;0;1;-1;1;0];

%------------------------------------------------------------------
---

OPTIONS = [];

% ------------------------------------------------------------------
% Begin Compliant Solution
% ------------------------------------------------------------------

for k = 1:2

    Jacobian = zeros(6,6);
    Jtorque = zeros(3,6);
    ModTheta = zeros(6,1);
    ModPos = zeros(3,1);
    Kx = K(:,1);
    Ky = K(:,2);
    Kz = K(:,3);

    % Calculate the Joint Torques

    if k == 1
        Jtorque = torque_v022502(Theta,Endload,OPTIONS,apptor);
    else
        Jtorque = torque_v022502(Theta,Endload*0,OPTIONS,apptor*0);
    end

    % Calculate the Changes in Angles

    ThetaDelta = squeeze(Jtorque(:,3))./Kz;
    AlphaDelta = squeeze(Jtorque(:,1))./Kx;
    ContribDelta = squeeze(Jtorque(:,2))./Ky;
    ContribDelta = ContribDelta(2:7);

    % Set to Zero because the current model is unable to accomodate
the loads

    ContribDelta(2) = 0;
    ContribDelta(6) = 0;
```

```
    % Apply Deltas to existing D-H variables

    ModTheta = Theta' + ThetaDelta(1:6) + ContribDelta;
    ModAlpha = alphai + AlphaDelta(2:7);
    Moda = ai;
    Modd = di;

    % Calculate the new End-effector Position

    [J1,T1] = forwardk_v022502(ModTheta,ModAlpha,OPTIONS,Moda,Modd);

    if k == 1
        J_load = J1;
        T_load = T1;
    else
        J_noload = J1;
        T_noload = T1;
    end
end

% Extract the Position Vector for Analysis

if J_noload == 0 | J_load == 0
    fprintf('Warning.  Jacobian is near singular.\n');
    x = 5000;
    y = 5000;
    z = 5000;
    pitch = 1000;
    roll = 1000;
    yaw = 1000;
else
    % Convert Positions Back to Robot Coordinates

    x = T_load(1,4) - T_noload(1,4);
    y = T_load(2,4) - T_noload(2,4);
    z = T_load(3,4) - T_noload(3,4);

    % I have a Rotation Matrix built into T1... it contains the 3
rotations I need
    % formulated as a yaw,pitch,roll wrist as in Spong and
Vidyasagar page 46

    roll = atan2(T_load(2,1),T_load(1,1)) -
atan2(T_noload(2,1),T_noload(1,1));
    yaw = atan2(T_load(3,2),T_load(3,3)) -
atan2(T_noload(3,2),T_noload(3,3));
    pitch = asin(-T_load(3,1)) - asin(-T_noload(3,1));

    if abs(yaw*180/pi) > 340 & abs(yaw*180/pi) < 370
        if yaw > 0
            yaw = yaw - 2*pi;
        else
            yaw = yaw + 2*pi;
        end
    end
end
```

```
ModPos = [x,y,z,roll,pitch,yaw];

return
```

10.5.3 Hand Stiffness Guess Routine

- Code Filename: Hand_stiff_guess_v022502.m
- Code Description: Code to test a given stiffness matrix without running the optimization routine.
- Inputs:
  i. Regime #
  ii. Stiffness Matrix, term by term
- Outputs:
  i. Objective Function Value
  ii. Plots
- Application: This code is use to do the hand stiffness optimization. A stiffness matrix is assumed and the resulting deflections of the end-effector are computed and displayed to see if the result is better or worse than the prior test.
- Notes: N/A



```
% Mark W. Abbott
% Feburary 15, 2002

% Code to run a hand generated stiffness matrix


clear all;
close all;
clc;

tic

% ============================================================

regime = 3;
filenum = 2234;
load c:\data\Constant.mat

Stiff_0x = 1E15;
Stiff_0y = 1E15;
Stiff_2y = 1E15;
Stiff_6z = 1E15;

Stiff_1x = 1E8;
Stiff_2x = 4E7;
Stiff_3x = 1E10;
Stiff_4x = 1E10;
Stiff_5x = 1E10;
Stiff_6x = 1E10;
Stiff_1y = 1E10;
Stiff_3y = 1E8;
Stiff_4y = 1E10;
Stiff_5y = 1E10;
Stiff_6y = 1E10;
Stiff_0z = 1E10;
Stiff_1z = 2E8;
Stiff_2z = 3E7;
Stiff_3z = 1E10;
Stiff_4z = 1E10;
Stiff_5z = 1E10;
% ============================================================

Stiff_x =
[Stiff_0x;Stiff_1x;Stiff_2x;Stiff_3x;Stiff_4x;Stiff_5x;Stiff_6x];
Stiff_y =
[Stiff_0y;Stiff_1y;Stiff_2y;Stiff_3y;Stiff_4y;Stiff_5y;Stiff_6y];
Stiff_z =
[Stiff_0z;Stiff_1z;Stiff_2z;Stiff_3z;Stiff_4z;Stiff_5z;Stiff_6z];
Stiff = [Stiff_x,Stiff_y,Stiff_z];


fprintf('Beginning to Compile Optimized Solution Set\n');

expdata = strcat('c:\data\r',num2str(regime),'\expdata.mat');
load(expdata);  % Inserts Experimental Deflections into workspace
localpath = strcat('c:\data\r',num2str(regime),'\sol_loop\');
```

```
apptor = zeros(243,3);
[r,c] = size(optdata);

if regime == 3
    Top_loop = 2*25.4;
    Side_loop = 1.75*25.4;
    for i = 1:r
        if i < 82
            apptor(i,:) = [0,0,0];
        elseif i < 163
            vect = [0,0,Side_loop];        % Zero Frame!
            apptor(i,:) = cross(vect,squeeze(optdata(i,1:3)));
        else
            vect = [0,0,Top_loop];
            apptor(i,:) = cross(vect,squeeze(optdata(i,1:3)));
        end
    end
end

for i = 1:27
    for j = 1:9
        k = 9*(i - 1) + j;
        Hold =
compliant_v022502(Stiff,[optdata(k,1:3)';0;0;0],optdata(k,4:9)*pi/180,
squeeze(apptor(k,:)));
        Rot_o(k,:) = Hold(4:6);
        Disp_o(k,:) = Hold(1:3);
    end
end

Rot_e = optdata(:,13:15);
Disp_e = optdata(:,10:12);

%obj =
objfun_v022502(Stiff,optdata,localpath,filenum,num2str(regime));

Disp_o_1 = sqrt(sum((Disp_o.^2)')');
Disp_e_1 = sqrt(sum((Disp_e.^2)')');
Rot_e_1 = sqrt(sum((Rot_e.^2)')');
Rot_o_1 = sqrt(sum((Rot_o.^2)')');

% Plot Absolute Results

figure
% subplot(3,1,1); plot(abs(Disp_e_1) - abs(Disp_o_1))
% hold on
% plot([81;81],[min(min(abs(Disp_e_1) - abs(Disp_o_1)));1.25*max(
max(abs(Disp_e_1) - abs(Disp_o_1)))],'-k');
% plot([162;162],[min(min(abs(Disp_e_1) - abs(Disp_o_1)));1.25*max(
max(abs(Disp_e_1) - abs(Disp_o_1)))],'-k');
% hold off
% title('Absolute Displacement Error');
% xlabel('Trial');
% ylabel('Deflection (mm)');

for i = 1:27
```

```
    subplot(2,1,1); plot([0;loads],abs(Disp_e_1(9*(i-1) + 1:9*(i-1) +
9)) - abs(Disp_o_1(9*(i-1) + 1:9*(i-1) + 9)));
    hold on
end
hold off
title('Absolute Displacement Error');
xlabel('Applied Load (N)');
ylabel('Deflection (mm)');

subplot(2,1,2); plot(Disp_e_1,'r')
hold on
plot(Disp_o_1,'-.b')
min1 = min([min(Disp_o_1) min(min(Disp_e_1))]);
max1 = max([max(Disp_o_1) 1.25*max(max(Disp_e_1))]);
plot([81;81],[min1;max1],'-k');
plot([162;162],[min1;max1],'-k');
hold off
legend('Experimental','Optimized');
title('Absolute Displacement');
xlabel('Trial');
ylabel('Deflection (mm)');


% Plot X-Axis Results

figure
% subplot(3,1,1); plot(abs(Disp_e(:,1)) - abs(Disp_o(:,1)))
% hold on
% plot([81;81],[min(min(abs(Disp_e(:,1)) -
abs(Disp_o(:,1))));1.25*max(max(
abs(Disp_e(:,1)) - abs(Disp_o(:,1))))],'-k');
% plot([162;162],[min(min(abs(Disp_e(:,1)) -
abs(Disp_o(:,1))));1.25*max(max(abs(Disp_e(:,1)) -
abs(Disp_o(:,1))))],'-k');
% hold off
% title('X-Axis Displacement Error');
% xlabel('Trial');
% ylabel('Deflection (mm)');

for i = 1:27
    subplot(2,1,1); plot([0;loads],abs(Disp_e(9*(i-1) + 1:9*(i-1) +
9,1)) -
abs(Disp_o(9*(i-1) + 1:9*(i-1) + 9,1)));
    hold on
end
hold off
title('X-Axis Displacement Error');
xlabel('Applied Load (N)');
ylabel('Deflection (mm)');

subplot(2,1,2); plot(Disp_e(:,1),'r')
hold on
plot(Disp_o(:,1),'-.b')
min1 = min([min(Disp_o(:,1)) min(min(Disp_e(:,1)))]);
max1 = max([max(Disp_o(:,1)) 1.25*max(max(Disp_e(:,1)))]);
plot([81;81],[min1;max1],'-k');
plot([162;162],[min1;max1],'-k');
```

```
hold off
legend('Experimental','Optimized');
title('X-Axis Displacement');
xlabel('Trial');
ylabel('Deflection (mm)');

% Plot Y=Axis Results

figure
% subplot(3,1,1); plot(abs(Disp_e(:,2)) - abs(Disp_o(:,2)))
% hold on
% plot([81;81],[min(min(abs(Disp_e(:,2)) -
abs(Disp_o(:,2))));1.25*max(max(
abs(Disp_e(:,2)) - abs(Disp_o(:,2))))],'-k');
% plot([162;162],[min(min(abs(Disp_e(:,2)) -
abs(Disp_o(:,2))));1.25*max(max(abs(Disp_e(:,2)) -
abs(Disp_o(:,2))))],'-k');
% hold off
% title('Y-Axis Displacement Error');
% xlabel('Trial');
% ylabel('Deflection (mm)');

for i = 1:27
    subplot(2,1,1); plot([0;loads],abs(Disp_e(9*(i-1) + 1:9*(i-1) +
9,2)) -
abs(Disp_o(9*(i-1) + 1:9*(i-1) + 9,2)));
    hold on
end
hold off
title('Y-Axis Displacement Error');
xlabel('Applied Load (N)');
ylabel('Deflection (mm)');

subplot(2,1,2); plot(Disp_e(:,2),'r')
hold on
plot(Disp_o(:,2),'-.b')
min1 = min([min(Disp_o(:,2)) min(min(Disp_e(:,2)))]);
max1 = max([max(Disp_o(:,2)) 1.25*max(max(Disp_e(:,2)))]);
plot([81;81],[min1;max1],'-k');
plot([162;162],[min1;max1],'-k');

hold off
legend('Experimental','Optimized');
title('Y-Axis Displacement');
xlabel('Trial');
ylabel('Deflection (mm)');

% Plot Z-Axis Results

figure
% subplot(3,1,1); plot(abs(Disp_e(:,3)) - abs(Disp_o(:,3)))
% hold on
% plot([81;81],[min(min(abs(Disp_e(:,3)) -
abs(Disp_o(:,3))));1.25*max(max(abs
(Disp_e(:,3)) - abs(Disp_o(:,3))))],'-k');
% plot([162;162],[min(min(abs(Disp_e(:,3)) -
abs(Disp_o(:,3))));1.25*max(max(abs(Disp_e(:,3)) -
```

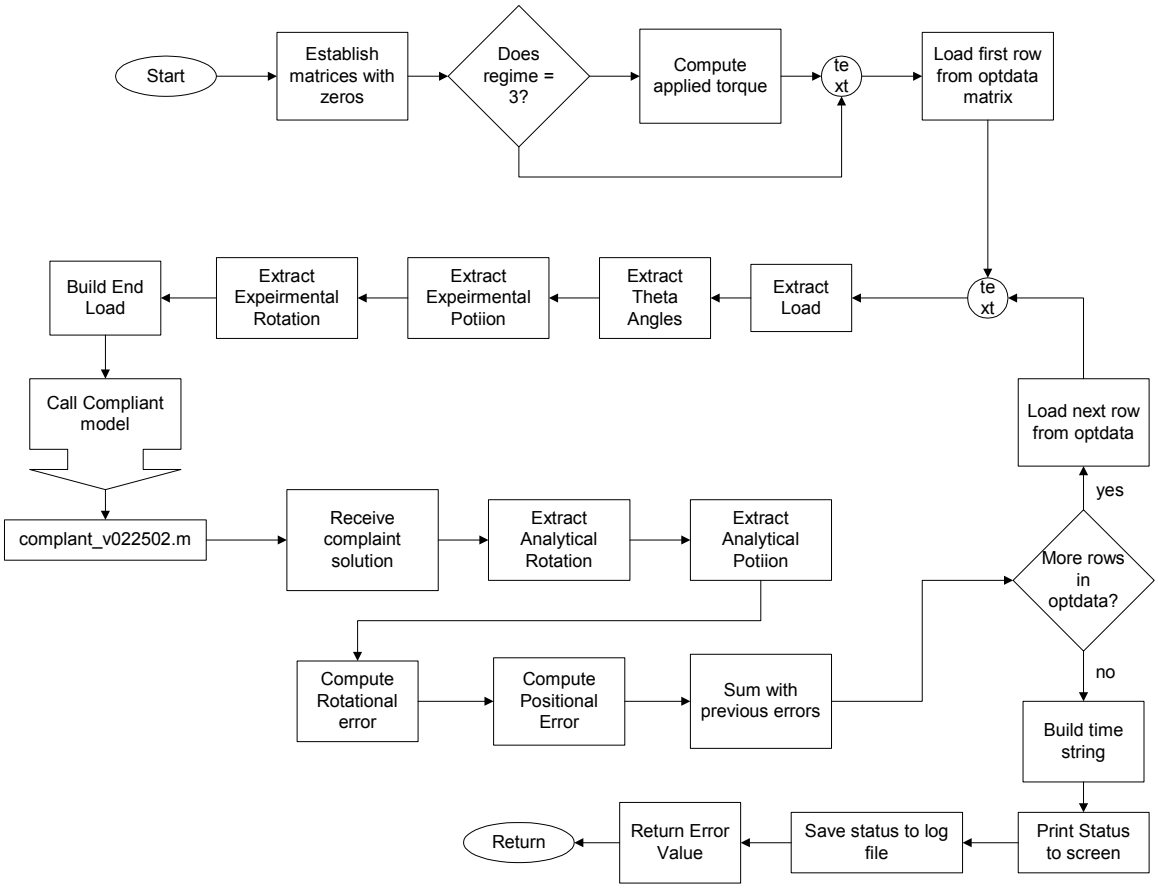
```
abs(Disp_o(:,3)))))],'-k');
% hold off
% title('Z-Axis Displacement Error');
% xlabel('Trial');
% ylabel('Deflection (mm)');

for i = 1:27
    subplot(2,1,1); plot([0;loads],abs(Disp_e(9*(i-1) + 1:9*(i-1) +
9,3)) -
abs(Disp_o(9*(i-1) + 1:9*(i-1) + 9,3)));
    hold on
end
hold off
title('Z-Axis Displacement Error');
xlabel('Applied Load (N)');
ylabel('Deflection (mm)');

subplot(2,1,2); plot(Disp_e(:,3),'r')
hold on
plot(Disp_o(:,3),'-.b')
min1 = min([min(Disp_o(:,3)) min(min(Disp_e(:,3)))]);
max1 = max([max(Disp_o(:,3)) 1.25*max(max(Disp_e(:,3)))]);
plot([81;81],[min1;max1],'-k');
plot([162;162],[min1;max1],'-k');

hold off
legend('Experimental','Optimized');
title('Z-Axis Displacement');
xlabel('Trial');
ylabel('Deflection (mm)');

toc

%
%        % Plot Absolute Rotation Results
%
%     figure
%     %subplot(3,1,1); plot(abs(Rot_e_1) - abs(Rot_o_1))
%     %title('Absolute Rotation Error (abs(Experimental) -
abs(Optimized))');
%     %xlabel('Trial');
%     %ylabel('Deflection (mm)');
%
%     for i = 1:27
%         subplot(2,1,1); plot([0;loads],abs(Rot_e_1(9*(i-1) + 1:9*(i-
1) + 9)) - abs(Rot_o_1(9*(i-1) + 1:9*(i-1) + 9)));
%         hold on
%     end
%     hold off
%     title('Absolute Rotation Error');
%     xlabel('Applied Load (N)');
%     ylabel('Rotation (rad)');
%
%     subplot(2,1,2); plot(Rot_e_1,'r')
%     hold on
%     plot(Rot_o_1,'-.b')
%     hold off
```

```
%     legend('Experimental','Optimized');
%     title('Absolute Rotation');
%     xlabel('Trial');
%     ylabel('Rotation (rad)');
%
%         % Plot X-Axis Results
%
%     figure
%     %subplot(3,1,1); plot(abs(Rot_e(:,1)) - abs(Rot_o(:,1)))
%     %title('X-Axis Rotation Error (abs(Experimental) -
abs(Optimized))');
%     %xlabel('Trial');
%     %ylabel('Deflection (mm)');
%
%     for i = 1:27
%         subplot(2,1,1); plot([0;loads],abs(Rot_e(9*(i-1) + 1:9*(i-1)
+ 9,1)) - abs(Rot_o(9*(i-1) + 1:9*(i-1) + 9,1)));
%         hold on
%     end
%     hold off
%     title('X-Axis Rotation Error');
%     xlabel('Applied Load (N)');
%     ylabel('Rotation (rad)');
%
%     subplot(2,1,2); plot(Rot_e(:,1),'r')
%     hold on
%     plot(Rot_o(:,1),'-.b')
%     hold off
%     legend('Experimental','Optimized');
%     title('X-Axis Rotation');
%     xlabel('Trial');
%     ylabel('Rotation (rad)');
%
%     % Plot Y-Axis Results
%
%     figure
%     %subplot(3,1,1); plot(abs(Rot_e(:,2)) - abs(Rot_o(:,2)))
%     %title('Y-Axis Rotation Error (abs(Experimental) -
abs(Optimized))');
%     %xlabel('Trial');
%     %ylabel('Deflection (mm)');
%
%     for i = 1:27
%         subplot(2,1,1); plot([0;loads],abs(Rot_e(9*(i-1) + 1:9*(i-1)
+ 9,2)) - abs(Rot_o(9*(i-1) + 1:9*(i-1) + 9,2)));
%         hold on
%     end
%     hold off
%     title('Y-Axis Rotation Error');
%     xlabel('Applied Load (N)');
%     ylabel('Rotation (rad)');
%
%     subplot(2,1,2); plot(Rot_e(:,2),'r')
%     hold on
%     plot(Rot_o(:,2),'-.b')
%     hold off
%     legend('Experimental','Optimized');
```

```
%     title('Y-Axis Rotation');
%     xlabel('Trial');
%     ylabel('Rotation (rad)');
%
%     % Plot Z-Axis Results
%
%     figure
%     %subplot(3,1,1); plot(abs(Rot_e(:,3)) - abs(Rot_o(:,3)))
%     %title('Z-Axis Rotation Error (abs(Experimental) -
abs(Optimized))');
%     %xlabel('Trial');
%     %ylabel('Deflection (mm)');
%
%     for i = 1:27
%         subplot(2,1,1); plot([0;loads],abs(Rot_e(9*(i-1) + 1:9*(i-1)
+ 9,3)) - abs(Rot_o(9*(i-1) + 1:9*(i-1) + 9,3)));
%         hold on
%     end
%     hold off
%     title('Z-Axis Rotation Error');
%     xlabel('Applied Load (N)');
%     ylabel('Rotation (rad)');
%
%     subplot(2,1,2); plot(Rot_e(:,3),'r')
%     hold on
%     plot(Rot_o(:,3),'-.b')
%     hold off
%     legend('Experimental','Optimized');
%     title('Z-Axis Rotation');
%     xlabel('Trial');
%     ylabel('Rotation (rad)');
```

10.5.4 Objective Function

- Code Filename: objfun_v022502.m
- Code Description: Objective Function for full load optimization problems.
- Inputs:
    i. Stiffness Matrix, x0
    ii. OptoTrak results, optdata
    iii. Local file storage path, localpath
    iv. Log File Number, filenum
    v. Regime being solved, regime
- Outputs:
    i. Objective Function Value, f
- Application: This is the primary objective function for optimization solutions based on the entire load set.
- Notes: N/A

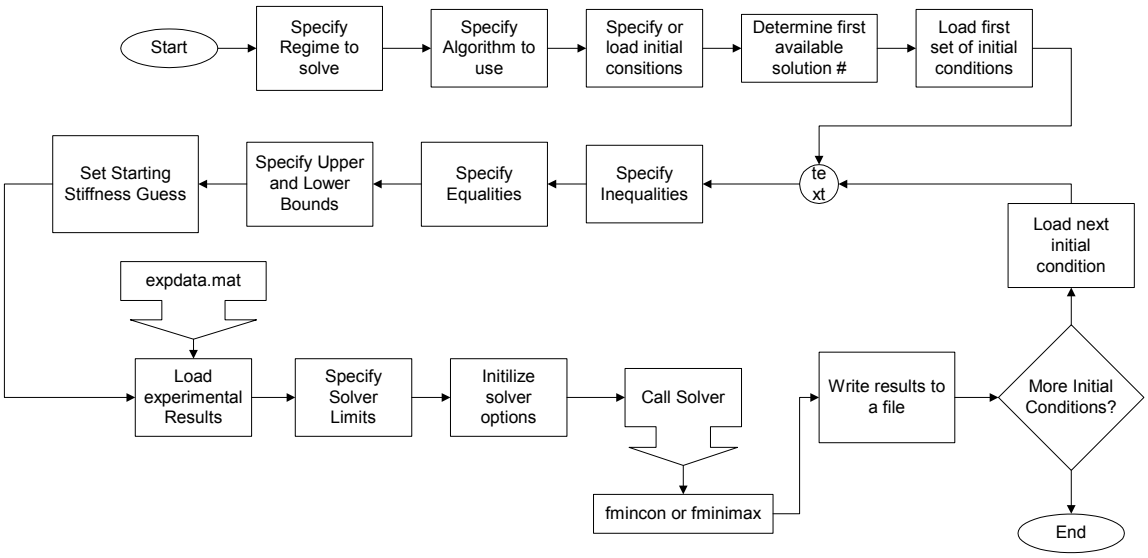



```
% Mark W. Abbott
% Feburary 21, 2002

% Objective Function for Optimization with applied torques

function f = objfun_v4(x0,optdata,localpath,filenum,regime)

[r,c] = size(optdata);
Load = zeros(1,3);
Theta = zeros(1,6);
ExpPos = zeros(1,3);
ExpRot = zeros(1,3);
Error = 0;
Error1 = 0;
Error2 = 0;
Count = 1;
apptor = zeros(243,3);

if regime == 3
    Top_loop = 1.5*25.4;            % Measured Quantity, Problem
Specific
    Side_loop = 0.875*25.4;         % Measured Quantity, Problem
Specific
    for i = 1:r
        if i < 82
            apptor(i,:) = [0,0,0];
```

```
        elseif i < 163
            vect = [0,0,Side_loop];        % Zero Frame!
            apptor(i,:) = cross(vect,squeeze(optdata(i,1:3)));
        else
            vect = [0,0,Top_loop];
            apptor(i,:) = cross(vect,squeeze(optdata(i,1:3)));
        end
    end
end

for i = 1:r
        Load = optdata(i,1:3);
        Theta = optdata(i,4:9)*(pi/180);
        ExpPos = optdata(i,10:12);
        ExpRot = optdata(i,13:15);
        EndLoad = [Load';0;0;0];
        Hold =
compliant_v022502(x0,EndLoad,Theta,squeeze(apptor(i,:)));
        AnaPos = Hold(1:3);
        AnaRot = Hold(4:6);
        Pos1 = abs(abs(ExpPos(1)) - abs(AnaPos(1))) + 1;
        Pos2 = abs(abs(ExpPos(2)) - abs(AnaPos(2))) + 1;
        Pos3 = abs(abs(ExpPos(3)) - abs(AnaPos(3))) + 1;
        Error1 = Pos1^2 + Pos2^2 + Pos3^2;
        Rot1 = abs(abs(ExpRot(1)) - abs(AnaRot(1))) + 1;
        Rot2 = abs(abs(ExpRot(2)) - abs(AnaRot(2))) + 1;
        Rot3 = abs(abs(ExpRot(3)) - abs(AnaRot(3))) + 1;
        Error2 = Rot1^2 + Rot2^2 + Rot3^2;
        Error = Error + Error1 + Error2;
end

temp = fix(clock);
fprintf('\n%g.%g.%g  %g.%g.%g   %g Objective function Current Value:
%g',temp(2),temp(3),temp(1),temp(4),temp(5),temp(6),filenum, Error);

f = Error;

if filenum < 10
    strnum = strcat('0',num2str(filenum));
else
    strnum = num2str(filenum);
end

optimlog = strcat(localpath,strnum,'optimlog.txt');

fid = fopen(optimlog,'a');
fseek(fid,0,'eof');
junk = fprintf(fid,'\n%g.%g.%g  %g.%g.%g  Objective function Current
Value:  %g\n',
temp(2),temp(3),temp(1),temp(4),temp(5),temp(6),Error);
junk = fprintf(fid,'     x(01-07) %8.8g  %8.8g  %8.8g  %8.8g  %8.8g
%8.8g  %8.8g\n',x0(1),x0(2),x0(3),x0(4),x0(5),x0(6),x0(7));
junk = fprintf(fid,'     x(08-14) %8.8g  %8.8g  %8.8g  %8.8g  %8.8g
%8.8g  %8.8g\n',x0(8),x0(9),x0(10),x0(11),x0(12),x0(13),x0(14));
junk = fprintf(fid,'     x(15-21) %8.8g  %8.8g  %8.8g  %8.8g  %8.8g
%8.8g  %8.8g\n',x0(15),x0(16),x0(17),x0(18),x0(19),x0(20),x0(21));
fclose(fid);
```

10.5.5 Optimize Routine

- Code Filename: optimize_v022502.m
- Code Description: Optimization front end for full load set optimization problems.
- Inputs:
  i. All Optimization solver required inputs
  ii. Regime to be solved
  iii. Optimization routine to use to solve problem
  iv. Selection of Stiffness matrices to use as initial conditions for solver
- Outputs:
  i. Converged stiffness matrix
- Application: This is the primary optimization front end and is used to control the optimization process.
- Notes:




```
% Mark W. Abbott
% Feburary 21, 2002

clear all;
close all;
clc;


%=======================================================

regime = 3;
mincon = 1;
minimax = 0;

% Establish the matrix of start conditions here
% This should be a 7 x 3 x n matrix
```

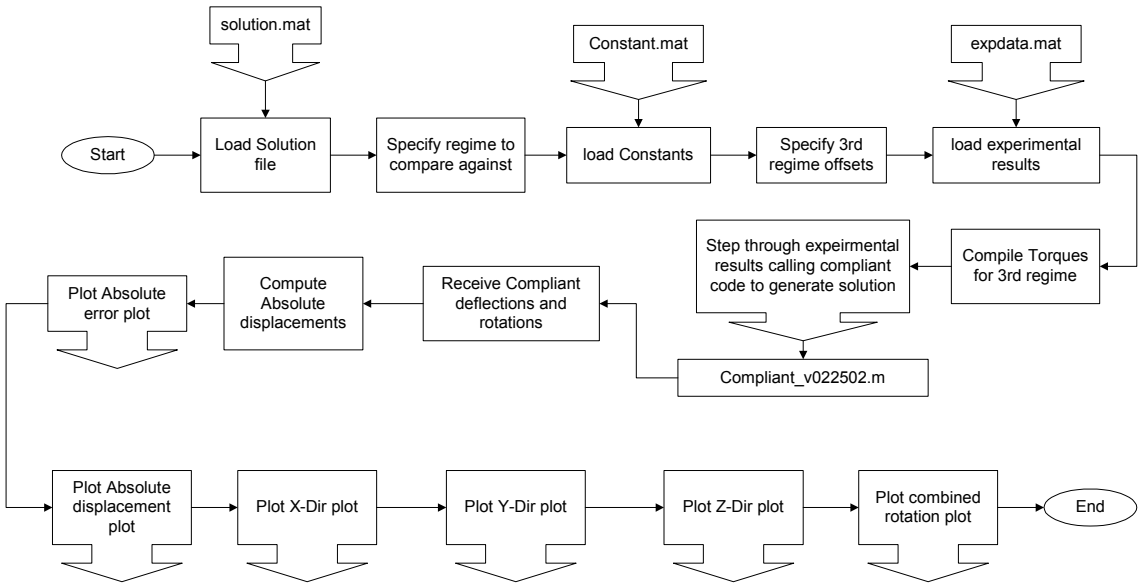
```
%   There are 6 links
%   There are 3 Stiffnesses
%   There are n different start cases

load c:\data\r1\sol_loop\solution_final.mat
Kmatrix(:,:,1) = x;

load c:\data\r2\sol_loop\solution_final.mat
Kmatrix(:,:,2) = x;

Kmatrix(:,:,3) = 1E10*ones(7,3);
Kmatrix(2,1,3) = 1E8;
Kmatrix(2,2,3) = 4E7;
Kmatrix(4,2,3) = 1E8;
Kmatrix(2,3,3) = 2E8;
Kmatrix(3,3,3) = 3E7;

Kmatrix(:,:,4) = 1E7*ones(7,3);

Kmatrix(:,:,5) = 1E8*ones(7,3);

Kmatrix(:,:,6) = 1E9*ones(7,3);

%=========================================================

localpath = strcat('c:\data\r',num2str(regime),'\sol_loop\');

i = 1;
holdval = 0;
while i < 1000
    numstr = num2str(i);
    if i < 100
        numstr = strcat('0',numstr);
    end
    if i < 10
        numstr = strcat('0',numstr);
    end
    filename = strcat(localpath,'solution',numstr,'.mat');
    testval = fopen(filename);
    if testval == -1
        holdval = i;
        i = 1000;
    end
    i = i + 1;
end

[a,b,c] = size(Kmatrix);

for j = 1:c

    % Optimization test routine

    % Contrained Optimization with Bounds, includes Numerical
Gradients

    % Linear Inequalities of the form A*x <= B
    A = [];
```

```
    B = [];

    % Linear Equalities of the form Aeq*x = Beq
    Aeq = [];
    Beq = [];

    % Upper and Lower Bounds for Variables
    lb = 10^2*ones(7,3);
    ub = 10^20*ones(7,3);

    % Starting Guess
    x0 = squeeze(Kmatrix(:,:,j));

    expdata = strcat('c:\data\r',num2str(regime),'\expdata.mat');

    load(expdata);  % Inserts Experimental Deflections into
workspace
    %  Fx  Fy  Fz  T1  T2  T3  T4  T5  T6  dx  dy  dz  rx  ry  rz

    % Termination Value for the Objective Function
    ObjLimit = 10^-10;

    % Termination Value for X increment
    Xincr = 10^-10;

    % Determines the Maximum Number of Function Evaluations
    maxeval = [400000];

    % Sets the Maxmimum # of Iterations
    iter = [100000];

    % Set Options
    OPTIONS =
optimset('LargeScale','off','Tolfun',ObjLimit,'TolX',Xincr,'MaxFunEv
als',maxeval,'MaxIter',iter,'Diagnostics','on');             %
Specifies a medium-scale solver

    % Call Optimization Routine
    if mincon == 1
        [x,fval,exitflag,output,lambda,grad] =
fmincon('objfun_v022502',x0,A,B,Aeq,Beq,lb,ub,[],OPTIONS,optdata,
localpath,j,regime)
  % Calls the Optimization function
    elseif minimax == 1
        [x,fval,exitflag,output,lambda,grad] =
fminimax('objfun_v022502',x0,A,B,Aeq,Beq,lb,ub,[],OPTIONS,optdata,
localpath,j,regime)
  % Calls the Optimization function
    end
    % Exitflag Notes
    % > 0 -->  Converged to a solution
    % = 0 -->  Reached Maximum # of Function Evaluations
    % < 0 -->  Did not Converge on a Solution

    numstr = num2str(holdval);
    if holdval < 100
        numstr = strcat('0',numstr);
```

```
    end
    if holdval < 10
        numstr = strcat('0',numstr);
    end

    filename2 = strcat(localpath,'solution',numstr,'.mat');
    save(filename2,'x0','x','grad')
    holdval = holdval + 1;
end

fclose all;
```

10.5.6 Thesis Plot Routine

- Code Filename: thesis_plot_v030102.m
- Code Description: Code to build plots for thesis presentation.
- Inputs:
    i. Proposed stiffness matrix
    ii. Regime to solve
- Outputs:
    i. Plots comparing results
    ii. Objective function value
- Application: This code computes the resulting plots based on a given stiffness matrix. The output from this code is presented throughout the thesis and provides the best comparison of the quality of the results from the optimization.
- Notes: N/A




```
% Mark W. Abbott
% March 2, 2001

% Code to generate plots from optimzation output
```

```
clear all;
close all;
clc;

%=============================================================

path2 = 'c:\data\r3\r3_1k_full\1827_+_solution003.mat';
regime = 3;
filenum = 1234;

load c:\data\Constant.mat;
top_loop_offset = 1.5;      % Offset from 6th frame to Top Loaded
Cases
side_loop_offset = 0.875;   % Offset from 6th frame to Side Loaded
Cases

%=============================================================

testval = fopen(path2);

if testval == -1

    fprintf('WARNING:  Solution.mat file not found! ...  Aborting.');

else

    load(path2);
    expdata = strcat('c:\data\r',num2str(regime),'\expdata.mat');
    load(expdata);  % Inserts Experimental Deflections into workspace


    fclose all;

    Rot_o = zeros(243,3);
    Disp_o = zeros(243,3);

    fprintf('Beginning to Compile Optimized Solution Set\n');

    apptor = zeros(243,3);
    [r,c] = size(optdata);

    % Build Torque due to offset Loads for 3rd Regime

    if regime == 3
        Top_loop = top_loop_offset*25.4;
        Side_loop = side_loop_offset*25.4;
        for i = 1:r
            if i < 82
                apptor(i,:) = [0,0,0];
            elseif i < 163
                vect = [0,0,Side_loop];          % Zero Frame!
                apptor(i,:) = cross(vect,squeeze(optdata(i,1:3)));
            else
                vect = [0,0,Top_loop];           % Zero Frame!
                apptor(i,:) = cross(vect,squeeze(optdata(i,1:3)));
```

```
            end
        end
    end

    % Call Compliance code to generate optimized solutions

    for i = 1:27
        for j = 1:9
            k = 9*(i - 1) + j;
            Hold =
compliant_v022502(x,[optdata(k,1:3)';0;0;0],optdata(k,4:9)*pi/180,
squeeze(apptor(k,:)));
            Rot_o(k,:) = Hold(4:6);
            Disp_o(k,:) = Hold(1:3);
        end
    end

    Rot_e = optdata(:,13:15);
    Disp_e = optdata(:,10:12);

    localpath = strcat('c:\data\r',num2str(regime),'\sol_loop\');

    %obj =
objfun_v022502(x,optdata,localpath,filenum,num2str(regime));


    % Compile X, Y and Z direction Data into a single result via sqrt
of
sum of squares

    Disp_o_1 = sqrt(sum((Disp_o.^2)')');
    Disp_e_1 = sqrt(sum((Disp_e.^2)')');

    % Plot Error Plot

    absDispError = abs(Disp_e_1 - Disp_o_1);
    perDispError = 100*absDispError/abs(Disp_e_1);

    figure
    plot(perDispError);
    hold on

plot([81;81],[min(min(perDispError));max(max(perDispError))],':');

plot([162;162],[min(min(perDispError));max(max(perDispError))],':');
    hold off
    xlabel('Trial');
    ylabel('Percentage Deflection Error');
    %title('Percentage Error of Absolute Displacement');

    % Plot Absolute Displacement Results

    figure

    for i = 1:27
        subplot(2,1,1); plot([0;loads],abs(Disp_e_1(9*(i-1) + 1:9*(i-
1) + 9)) - abs(Disp_o_1(9*(i-1) + 1:9*(i-1) + 9)));
```

```
        hold on
    end
    hold off
    title('Absolute Displacement Error');
    xlabel('Applied Load (N)');
    ylabel('Deflection (mm)');

    subplot(2,1,2); plot(Disp_e_1,'r')
    hold on
    plot(Disp_o_1,'-.b')
    plot([81;81],[min(min(Disp_o_1));1.25*max(max(Disp_o_1))],'-k');
    plot([162;162],[min(min(Disp_o_1));1.25*max(max(Disp_o_1))],'-
k');
    hold off
    legend('Experimental','Optimized');
    title('Absolute Displacement');
    xlabel('Trial');
    ylabel('Deflection (mm)');

    % Plot X-Axis Results

    figure
    for i = 1:27
        subplot(2,1,1); plot([0;loads],abs(Disp_e(9*(i-1) + 1:9*(i-1)
+ 9,1)) - abs(Disp_o(9*(i-1) + 1:9*(i-1) + 9,1)));
        hold on
    end
    hold off
    title('X-Axis Displacement Error');
    xlabel('Applied Load (N)');
    ylabel('Deflection (mm)');

    subplot(2,1,2); plot(Disp_e(:,1),'r')
    hold on
    plot(Disp_o(:,1),'-.b')
    min1 = min([min(Disp_o(:,1)) min(min(Disp_e(:,1)))]);
    max1 = 1.25*max([max(Disp_o(:,1)) max(max(Disp_e(:,1)))]);
    plot([81;81],[min1;max1],'-k');
    plot([162;162],[min1;max1],'-k');
    hold off
    legend('Experimental','Optimized');
    title('X-Axis Displacement');
    xlabel('Trial');
    ylabel('Deflection (mm)');

    figure
    for i = 1:27
        subplot(2,1,1); plot([0;loads],abs(Rot_e(9*(i-1) + 1:9*(i-1)
+ 9,1)) - abs(Rot_o(9*(i-1) + 1:9*(i-1) + 9,1)));
        hold on
    end
    hold off
    title('X-Axis Rotation Error');
    xlabel('Applied Load (N)');
    ylabel('Rotation (rad)');

    subplot(2,1,2); plot(Rot_e(:,1),'r')
```

```
    hold on
    plot(Rot_o(:,1),'-.b')
    min1 = min([min(Rot_o(:,1)) min(min(Rot_e(:,1)))]);
    max1 = 1.25*max([max(Rot_o(:,1)) max(max(Rot_e(:,1)))]);
    plot([81;81],[min1;max1],'-k');
    plot([162;162],[min1;max1],'-k');
    hold off
    legend('Experimental','Optimized');
    title('X-Axis Rotation');
    xlabel('Trial');
    ylabel('Rotation (rad)');



    % Plot Y-Axis Results

    figure
    for i = 1:27
        subplot(2,1,1); plot([0;loads],abs(Disp_e(9*(i-1) + 1:9*(i-1)
+ 9,2)) - abs(Disp_o(9*(i-1) + 1:9*(i-1) + 9,2)));
        hold on
    end
    hold off
    title('Y-Axis Displacement Error');
    xlabel('Applied Load (N)');
    ylabel('Deflection (mm)');

    subplot(2,1,2); plot(Disp_e(:,2),'r')
    hold on
    plot(Disp_o(:,2),'-.b')
    min1 = min([min(Disp_o(:,2)) min(min(Disp_e(:,2)))]);
    max1 = 1.25*max([max(Disp_o(:,2)) max(max(Disp_e(:,2)))]);
    plot([81;81],[min1;max1],'-k');
    plot([162;162],[min1;max1],'-k');
    legend('Experimental','Optimized');
    title('Y-Axis Displacement');
    xlabel('Trial');
    ylabel('Deflection (mm)');

    figure
    for i = 1:27
        subplot(2,1,1); plot([0;loads],abs(Rot_e(9*(i-1) + 1:9*(i-1)
+ 9,2)) - abs(Rot_o(9*(i-1) + 1:9*(i-1) + 9,2)));
        hold on
    end
    hold off
    title('Y-Axis Rotation Error');
    xlabel('Applied Load (N)');
    ylabel('Rotation (rad)');

    subplot(2,1,2); plot(Rot_e(:,2),'r')
    hold on
    plot(Rot_o(:,2),'-.b')
    min1 = min([min(Rot_o(:,2)) min(min(Rot_e(:,2)))]);
    max1 = 1.25*max([max(Rot_o(:,2)) max(max(Rot_e(:,2)))]);
    plot([81;81],[min1;max1],'-k');
    plot([162;162],[min1;max1],'-k');
```

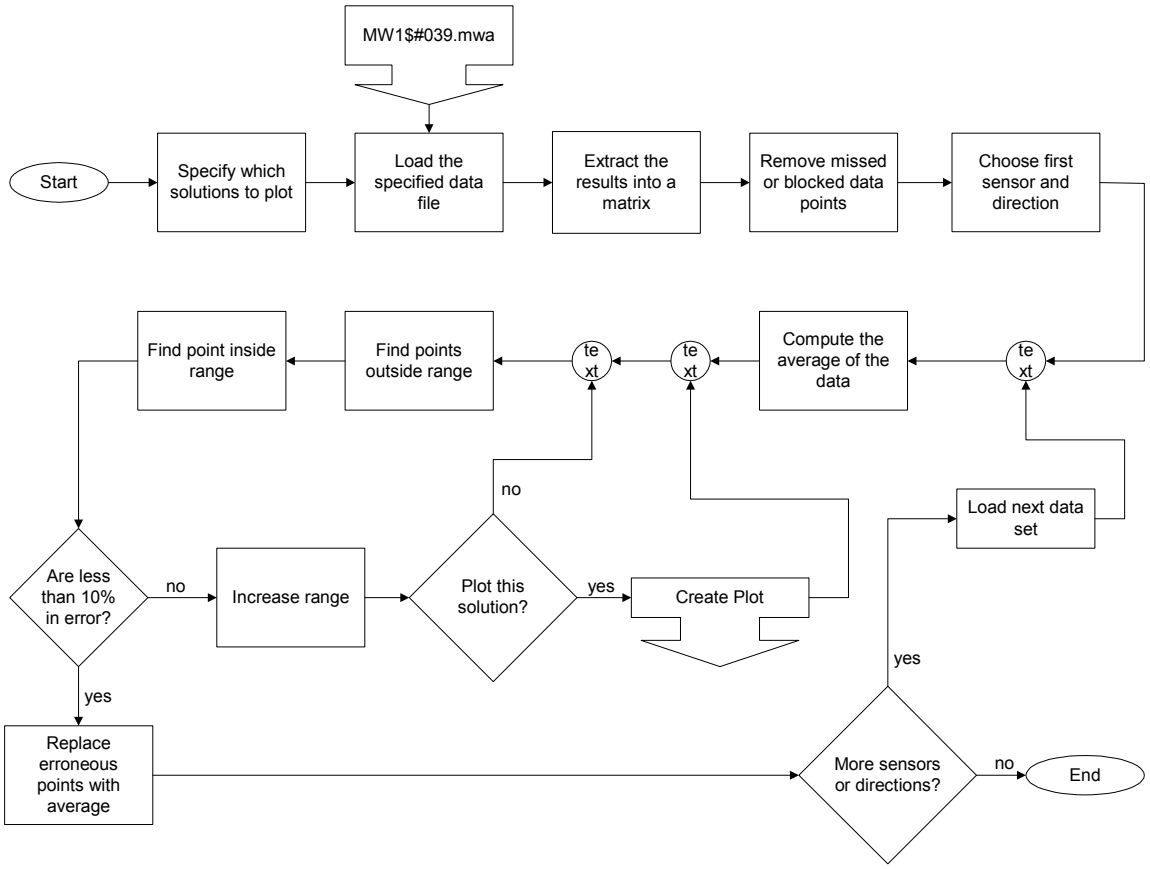
```
     hold off
    legend('Experimental','Optimized');
    title('Y-Axis Rotation');
    xlabel('Trial');
    ylabel('Rotation (rad)');



    % Plot Z-Axis Results

    figure
    for i = 1:27
        subplot(2,1,1); plot([0;loads],abs(Disp_e(9*(i-1) + 1:9*(i-1)
+ 9,3)) - abs(Disp_o(9*(i-1) + 1:9*(i-1) + 9,3)));
        hold on
    end
    hold off
    title('Z-Axis Displacement Error');
    xlabel('Applied Load (N)');
    ylabel('Deflection (mm)');

    subplot(2,1,2); plot(Disp_e(:,3),'r')
    hold on
    plot(Disp_o(:,3),'-.b')
    min1 = min([min(Disp_o(:,3)) min(min(Disp_e(:,3)))]);
    max1 = 1.25*max([max(Disp_o(:,3)) max(max(Disp_e(:,3)))]);
    plot([81;81],[min1;max1],'-k');
    plot([162;162],[min1;max1],'-k');
    hold off
    legend('Experimental','Optimized');
    title('Z-Axis Displacement');
    xlabel('Trial');
    ylabel('Deflection (mm)');

    figure
    for i = 1:27
        subplot(2,1,1); plot([0;loads],abs(Rot_e(9*(i-1) + 1:9*(i-1)
+ 9,3)) - abs(Rot_o(9*(i-1) + 1:9*(i-1) + 9,3)));
        hold on
    end
    hold off
    title('Z-Axis Rotation Error');
    xlabel('Applied Load (N)');
    ylabel('Rotation (rad)');

    subplot(2,1,2); plot(Rot_e(:,3),'r')
    hold on
    plot(Rot_o(:,3),'-.b')
    min1 = min([min(Rot_o(:,3)) min(min(Rot_e(:,3)))]);
    max1 = 1.25*max([max(Rot_o(:,3)) max(max(Rot_e(:,3)))]);
    plot([81;81],[min1;max1],'-k');
    plot([162;162],[min1;max1],'-k');
    hold off
    legend('Experimental','Optimized');
    title('Z-Axis Rotation');
    xlabel('Trial');
    ylabel('Rotation (rad)');
```

```
    % Make Combined Rotation Plot

    figure
    for i = 1:27
        subplot(3,2,1); plot([0;loads],abs(Rot_e(9*(i-1) + 1:9*(i-1)
+ 9,1)) - abs(Rot_o(9*(i-1) + 1:9*(i-1) + 9,1)));
        hold on
    end
    hold off
    title('Rotation Error');
    %xlabel('Applied Load (N)');
    ylabel('X-Axis Rotation (rad)');

    subplot(3,2,2); plot(Rot_e(:,1),'r')
    hold on
    plot(Rot_o(:,1),'-.b')
    min1 = min([min(Rot_o(:,1)) min(min(Rot_e(:,1)))]);
    max1 = 1.25*max([max(Rot_o(:,1)) max(max(Rot_e(:,1)))]);
    plot([81;81],[min1;max1],'-k');
    plot([162;162],[min1;max1],'-k');
    hold off
    legend('Experimental','Optimized',0);
    title('Rotation');
    %xlabel('Trial');
    %ylabel('Rotation (rad)');

    for i = 1:27
        subplot(3,2,3); plot([0;loads],abs(Rot_e(9*(i-1) + 1:9*(i-1)
+ 9,2)) - abs(Rot_o(9*(i-1) + 1:9*(i-1) + 9,2)));
        hold on
    end
    hold off
    %title('Y-Axis Rotation Error');
    %xlabel('Applied Load (N)');
    ylabel('Y-Axis Rotation (rad)');

    subplot(3,2,4); plot(Rot_e(:,2),'r')
    hold on
    plot(Rot_o(:,2),'-.b')
    min1 = min([min(Rot_o(:,2)) min(min(Rot_e(:,2)))]);
    max1 = 1.25*max([max(Rot_o(:,2)) max(max(Rot_e(:,2)))]);
    plot([81;81],[min1;max1],'-k');
    plot([162;162],[min1;max1],'-k');
    hold off
    legend('Experimental','Optimized',0);
    %title('Y-Axis Rotation');
    %ylabel('Rotation (rad)');
    %xlabel('Trial');


    for i = 1:27
        subplot(3,2,5); plot([0;loads],abs(Rot_e(9*(i-1) + 1:9*(i-1)
+ 9,3)) - abs(Rot_o(9*(i-1) + 1:9*(i-1) + 9,3)));
        hold on
    end
    hold off
```

```
    %title('Z-Axis Rotation Error');
    xlabel('Applied Load (N)');
    ylabel('Z-Axis Rotation (rad)');

    subplot(3,2,6); plot(Rot_e(:,3),'r')
    hold on
    plot(Rot_o(:,3),'-.b')
    min1 = min([min(Rot_o(:,3)) min(min(Rot_e(:,3)))]);
    max1 = 1.25*max([max(Rot_o(:,3)) max(max(Rot_e(:,3)))]);
    plot([81;81],[min1;max1],'-k');
    plot([162;162],[min1;max1],'-k');
    hold off
    legend('Experimental','Optimized',0);
    %title('Z-Axis Rotation');
    xlabel('Trial');
    %ylabel('Rotation (rad)');

end
```

10.5.7 Thesis Plot Routine for Experimental Number Reduction

- Code Filename: TP_exp_redux.m
- Code Description: Code to display how the confidence banding functions.
- Inputs:
    i. A data file
- Outputs:
    i. Plots
- Application: Used to generate plots for thesis to show how the confidence banding algorithm works.
- Notes: N/A




```
% Mark W. Abbott
% February 26, 2002

% Code to generate Plots to explain how the confidence banding works

clear all;
close all;
clc;


% ============================================================

% Establish data matrix and open file

x = 12;
y = 1;

data = zeros(14,3,400);
fid = fopen('c:\data\r3\orig_data\MW1$#039.mwa');


% ============================================================

titletrash = fgetl(fid);

% Fill data matrix
```

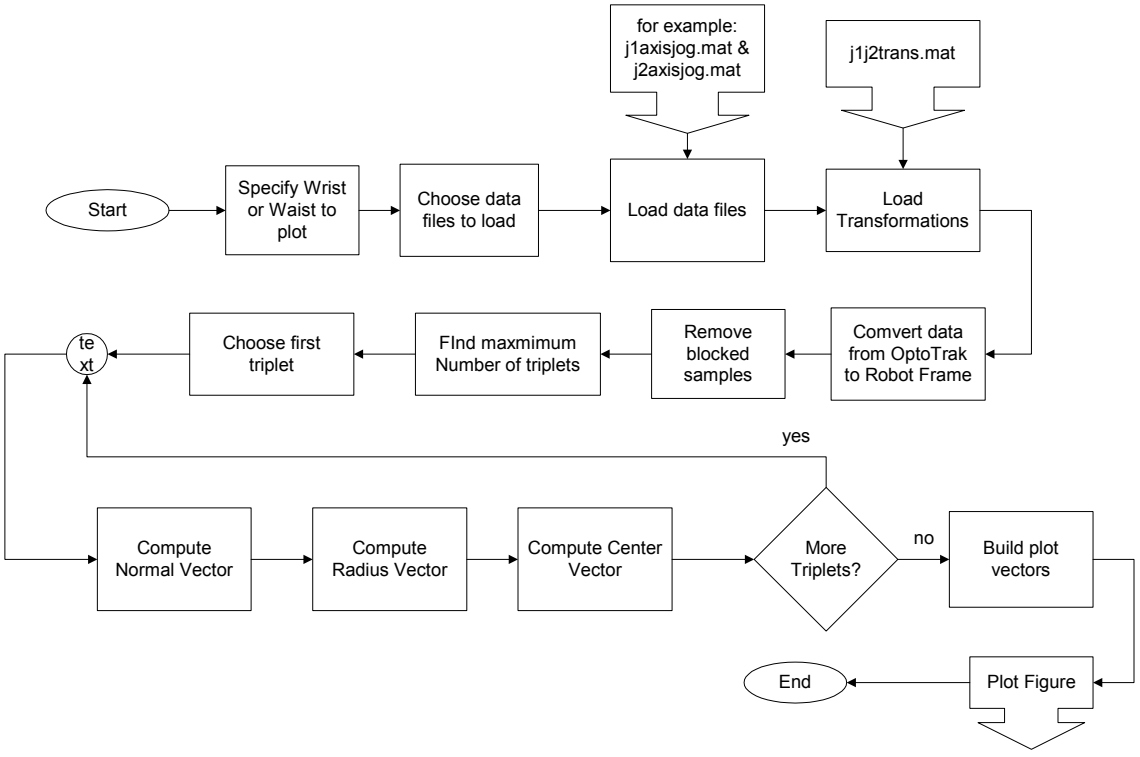
```
for i = 1:400
    for j = 1:14
        tline = fgetl(fid);
        tlinenum = str2num(tline);
        data(j,:,i) = tlinenum(length(tlinenum) -
2:length(tlinenum));
    end
end

fclose(fid);
data1 = data;

%figure
%plot(squeeze(data(12,1,:)))
data_orig = data;

% Cleanse Data of missed points etc

for i = 1:14
    for j = 1:3
        good = find(abs(data(i,j,:)) < 10^10);
        bad = find(abs(data(i,j,:)) > 10^10);
        if length(bad) < 0.1*400
            ave_tmp = 0;
            for m = 1:length(good)
                ave_tmp = ave_tmp + data(i,j,good(m));
            end

            ave_tmp = ave_tmp/length(good);

            for m = 1:length(bad)
                data(i,j,bad(m)) = ave_tmp;
                fprintf('\nReplacing data(%g,%g,%g) due to blocked
sensor',i,j,bad(m));
            end
        else
            fprintf('\nWarning:  data(%g,%g,:) has too many
blocked....',i,j);

            data(i,j,:) = 0;
            fprintf('\n%g points in error:  Data set to
zero.',length(bad));
        end
    end
end

%figure
%plot(squeeze(data(12,1,:)))
data_blk = data;

% Cleanse Data of erroneus Measurements etc

range = 1;

for i = 1:14
    for j = 1:3
        range = 1;
```

```
        while range <= 5
            ave_orig = mean(data(i,j,:));
            data_tmp = data(i,j,:) - ave_orig;
            good1 = find(abs(data_tmp) <= range);
            bad1 = find(abs(data_tmp) > range);
            if length(bad1) <= 0.1*400 & length(bad1) ~= 0
                sum_tmp = 0;
                for m = 1:length(good1)
                    sum_tmp = sum_tmp + data(i,j,good1(m));
                end
                ave_tmp = sum_tmp/length(good1);
                for m = 1:length(bad1)
                    data(i,j,bad1(m)) = ave_tmp;
                    fprintf('\nReplacing data(%g,%g,%g) due to
erroneus measurement   %g   %g',i,j,bad1(m),ave_orig,ave_tmp);
                end
                range = 6;
            elseif length(bad1) > 0.1*400
                range = range + 0.5;
                if range > 5
                    fprintf('\Critical Error:  data(%g,%g,:) has too
points (%g) many outside range....',i,j,length(bad1));
                end
                fprintf('\nModifying Range Variable: %g %g %g',
range, i ,j);
            elseif length(bad1) == 0
                range = 6;
            end
            if i == x & j == y
                figure
                plot(squeeze(data_tmp))
                title1 = strcat('Data Spread with Range =
',num2str(range));
                title(title1);
                xlabel('Sample Number');
                ylabel('Distance From Average (mm)');
                hold on
                plot(range*ones(400,1),'g')
                plot(-range*ones(400,1),'g')
                axis([0 400 -5 5])
                hold off
                figure
                plot(squeeze(data(x,y,:)))
                title('Corrected Data');
                xlabel('Sample Number');
                ylabel('Distance (mm)')
                hold on
                if range == 1
                    ave = ave_orig;
                else
                    ave = ave_tmp;
                end
                plot(ave*ones(400,1),'--r');
            end
        end
    end
end
```

```
data_fix = data;

%figure
%plot(squeeze(data_blk(12,1,:)))
%hold on
%plot(ave_orig*ones(400,1),'r')
```

10.5.8 Thesis Plot Routine for Experimental Coordinate Transforms

- Code Filename: TP_xp_trans.m
- Code Description: Plot routine for experimental motions
- Inputs:
  i. Two wrist test results
  ii. Two arm test results
- Outputs:
  i. Plots
- Application: Used to generate basis plot for thesis figures dealing with the results from the tests to build coordinate transforms between optotrak and robot coordinate systems.
- Notes: N/A



```
% Mark W. Abbot
% Code to Plot the pathes resulting from the wrist and arm motions.

clear all;
close all;
clc;

% ============================================================

Wrist = 0;
Waist = 1;

% ============================================================

if Wrist == 1 & Waist == 0
    d = 3;
    e = 4;
elseif Wrist == 0 & Waist == 1
    d = 1;
    e = 2;
else
    fprintf('Please Select either the Wrist or the Waist
Joints...');
end

figure

for q = d:e

    if q == 1
        load c:\data\j1axisjog\backup.mat
    elseif q == 2
        load c:\data\j2axisjog\backup.mat
    elseif q == 3
        load c:\data\j5axisjog\backup.mat
    elseif q == 4
        load c:\data\j6axisjog\backup.mat
    end

    load c:\data\j1j2trans.mat

    data1 = compdata(1:4,:,:);
    data2 = compdata(11:14,:,:);
    data = [data1;data2];
    [a,b,c] = size(data);

    % Convert Data from OptoTrak to Robot Zero Frame

    for i = 1:a
        for j = 1:c
            tmp = Topt_0*[data(i,:,j),1]';
            data(i,:,j) = tmp(1:3);
        end
    end

    length = c;
    removed = 0;
```

```
% This removes Bad samples from the averaged data

for i = 1:4
    for j = 1:3
        k = 1;
        while k < length - removed + 1
            if abs(data(i,j,k)) > 10^5
                data(:,:,k) = [];
                removed = removed + 1;
            end
            k = k + 1;
        end
    end
end

% Find Maximum Number of triplets

[a1,b1,c1] = size(data);
go = 1;
num = 0;
while go == 1
    if num <= c1 - 3
        num = num + 3;
    else
        go = 0;
    end
end

Vect1a = zeros(1,3);
Vect1b = zeros(1,3);
Pnt1 = zeros(1,3);
Pnt2 = zeros(1,3);
Pnt3 = zeros(1,3);
Norm1 = zeros(1,3);
Normal = zeros(4,num/3,3);
Radius = zeros(4,num/3);
Center = zeros(4,num/3,3);
Half = zeros(4,num/3);
Line = zeros(4,num/3);
Point1 = zeros(4,num/3,3);
Point2 = zeros(4,num/3,3);
Point3 = zeros(4,num/3,3);

for j = 1:4
    for i = 1:num/3

        % Find Normal Vectors
        Pnt1 = data(j,1:3,i);
        Pnt2 = data(j,1:3,i + num/3);
        Pnt3 = data(j,1:3,i + 2*num/3);
        Vect1a = Pnt2 - Pnt1;
        Vect1b = Pnt2 - Pnt3;
        Norm1 = cross(Vect1a,Vect1b);
        Norm1 = Norm1/sqrt(dot(Norm1,Norm1'));
        Normal(j,i,:) = Norm1;
        Point1(j,i,:) = Pnt1;
```

```
            Point2(j,i,:) = Pnt2;
            Point3(j,i,:) = Pnt3;

            % Find Radius
            Side1 = Pnt3 - Pnt2;
            Side2 = Pnt1 - Pnt2;
            Side3 = Pnt3 - Pnt1;
            Mag1 = sqrt(dot(Side1,Side1'));
            Mag2 = sqrt(dot(Side2,Side2'));
            Mag3 = sqrt(dot(Side3,Side3'));
            areaV = 1/2*cross(Side2,Side1);
            area = sqrt(dot(areaV,areaV'));
            R = Mag1*Mag2*Mag3/(4*area);
            Radius(j,i) = R;

            % Find Center Points
            Half1 = Side1/2;
            Line1 = cross(-Norm1,Half1);
            alpha = acos((Mag1^2-2*R^2)/(-2*R^2))/2;
            Rscale = R*cos(alpha);
            Center(j,i,:) = Pnt2 + Half1 +
Rscale*Line1/sqrt(dot(Line1,Line1'));
        end
    end

    p1x = [Point1(1,:,1),Point2(1,:,1),Point3(1,:,1)];
    p1y = [Point1(1,:,2),Point2(1,:,2),Point3(1,:,2)];
    p1z = [Point1(1,:,3),Point2(1,:,3),Point3(1,:,3)];

    p2x = [Point1(2,:,1),Point2(2,:,1),Point3(2,:,1)];
    p2y = [Point1(2,:,2),Point2(2,:,2),Point3(2,:,2)];
    p2z = [Point1(2,:,3),Point2(2,:,3),Point3(2,:,3)];

    p3x = [Point1(3,:,1),Point2(3,:,1),Point3(3,:,1)];
    p3y = [Point1(3,:,2),Point2(3,:,2),Point3(3,:,2)];
    p3z = [Point1(3,:,3),Point2(3,:,3),Point3(3,:,3)];

    p4x = [Point1(4,:,1),Point2(4,:,1),Point3(4,:,1)];
    p4y = [Point1(4,:,2),Point2(4,:,2),Point3(4,:,2)];
    p4z = [Point1(4,:,3),Point2(4,:,3),Point3(4,:,3)];

    c1x = [Center(1,:,1),Center(1,:,1),Center(1,:,1)];
    c1y = [Center(1,:,2),Center(1,:,2),Center(1,:,2)];
    c1z = [Center(1,:,3),Center(1,:,3),Center(1,:,3)];

    c2x = [Center(2,:,1),Center(2,:,1),Center(2,:,1)];
    c2y = [Center(2,:,2),Center(2,:,2),Center(2,:,2)];
    c2z = [Center(2,:,3),Center(2,:,3),Center(2,:,3)];

    c3x = [Center(3,:,1),Center(3,:,1),Center(3,:,1)];
    c3y = [Center(3,:,2),Center(3,:,2),Center(3,:,2)];
    c3z = [Center(3,:,3),Center(3,:,3),Center(3,:,3)];

    c4x = [Center(4,:,1),Center(4,:,1),Center(4,:,1)];
    c4y = [Center(4,:,2),Center(4,:,2),Center(4,:,2)];
    c4z = [Center(4,:,3),Center(4,:,3),Center(4,:,3)];
```

```
    figure
    plot3(p1x,p1y,p1z,'r');
    hold on
    plot3(p2x,p2y,p2z,'r');
    plot3(p3x,p3y,p3z,'r');
    plot3(p4x,p4y,p4z,'r');
    plot3(c1x,c2y,c1z,'og');
    plot3(c2x,c2y,c2z,'og');
    plot3(c3x,c3y,c3z,'og');
    plot3(c4x,c4y,c4z,'og');
    hold off
    view(3)
    axis('square')

end
```

10.5.9 Low End Hand Stiffness Guess Routine

- Code Filename: Hand_stiff_guess_le_v022502.m
- Code Description: Reduced load set hand stiffness guess code.
- Inputs:
    i. Regime to solve
    ii. Stiffness matrix, term by term to use
- Outputs:
    i. Objective Function Value
    ii. Plots of the resulting performance
- Application: Used to solve for the resulting deflections and rotations of the model based on an assumed stiffness matrix.
- Notes: N/A

```
% Mark W. Abbott
% Feburary 15, 2002

% Code to run a hand generated stiffness matrix


clear all;
close all;
clc;

tic

% ========================================================

regime = 3;
filenum = 2234;
load c:\data\Constant.mat

Stiff_0x = 1E15;
Stiff_0y = 1E15;
Stiff_2y = 1E15;
Stiff_6z = 1E15;
```

```
Stiff_1x = 1E8;
Stiff_2x = 4E7;
Stiff_3x = 1E10;
Stiff_4x = 1E10;
Stiff_5x = 1E10;
Stiff_6x = 1E10;
Stiff_1y = 1E10;
Stiff_3y = 1E8;
Stiff_4y = 1E10;
Stiff_5y = 1E10;
Stiff_6y = 1E10;
Stiff_0z = 1E10;
Stiff_1z = 2E8;
Stiff_2z = 3E7;
Stiff_3z = 1E10;
Stiff_4z = 1E10;
Stiff_5z = 1E10;
% ============================================================

Stiff_x =
[Stiff_0x;Stiff_1x;Stiff_2x;Stiff_3x;Stiff_4x;Stiff_5x;Stiff_6x];
Stiff_y =
[Stiff_0y;Stiff_1y;Stiff_2y;Stiff_3y;Stiff_4y;Stiff_5y;Stiff_6y];
Stiff_z =
[Stiff_0z;Stiff_1z;Stiff_2z;Stiff_3z;Stiff_4z;Stiff_5z;Stiff_6z];
Stiff = [Stiff_x,Stiff_y,Stiff_z];

fprintf('Beginning to Compile Optimized Solution Set\n');

expdata = strcat('c:\data\r',num2str(regime),'\expdata.mat');
load(expdata);  % Inserts Experimental Deflections into workspace
localpath = strcat('c:\data\r',num2str(regime),'\sol_loop\');


apptor = zeros(243,3);
[r,c] = size(optdata);

if regime == 3
    Top_loop = 2*25.4;
    Side_loop = 1.75*25.4;
    for i = 1:r
        if i < 82
            apptor(i,:) = [0,0,0];
        elseif i < 163
            vect = [0,0,Side_loop];         % Zero Frame!
            apptor(i,:) = cross(vect,squeeze(optdata(i,1:3)));
        else
            vect = [0,0,Top_loop];
            apptor(i,:) = cross(vect,squeeze(optdata(i,1:3)));
        end
    end
end


step = 1;

for i = 1:27
```

```
    for j = 1:9
        k = 9*(i - 1) + j;
        tmp = k;
        while tmp > 9
            tmp = tmp - 9;
        end
        if tmp < 7
            Hold =
compliant_v022502(Stiff,[optdata(k,1:3)';0;0;0],optdata(k,4:9)*pi/180,
squeeze(apptor(k,:)));
            Rot_o(step,:) = Hold(4:6);
            Disp_o(step,:) = Hold(1:3);
            Rot_e(step,:) = optdata(k,13:15);
            Disp_e(step,:) = optdata(k,10:12);
            step = step + 1;
        end
    end
end


obj =
objfun_le_v022502(Stiff,optdata,localpath,filenum,num2str(regime));

% Plot X-Axis Results

figure
subplot(3,1,1); plot(abs(Disp_e(:,1)) - abs(Disp_o(:,1)))
title('X-Axis Displacement Error (abs(Experimental) -
abs(Optimized))');
xlabel('Trial');
ylabel('Deflection (mm)');

le_loads = [0;loads(1:5)];

for i = 1:27
    subplot(3,1,2); plot(le_loads,abs(Disp_e(6*(i-1) + 1:6*(i-1) +
6,1)) -
abs(Disp_o(6*(i-1) + 1:6*(i-1) + 6,1)));
    hold on
end
hold off
title('X-Axis Displacement Error (abs(Experimental) -
abs(Optimized))');
xlabel('Applied Load (N)');
ylabel('Deflection (mm)');

subplot(3,1,3); plot(Disp_e(:,1),'r')
hold on
plot(Disp_o(:,1),'-.b')
hold off
legend('Experimental','Optimized');
title('X-Axis Displacement');
xlabel('Trial');
ylabel('Deflection (mm)');

% Plot Y=Axis Results
```

```
figure
subplot(3,1,1); plot(abs(Disp_e(:,2)) - abs(Disp_o(:,2)))
title('Y-Axis Displacement Error (abs(Experimental) -
abs(Optimized))');
xlabel('Trial');
ylabel('Deflection (mm)');

for i = 1:27
    subplot(3,1,2); plot(le_loads,abs(Disp_e(6*(i-1) + 1:6*(i-1) +
6,2)) -
abs(Disp_o(6*(i-1) + 1:6*(i-1) + 6,2)));
    hold on
end
hold off
title('Y-Axis Displacement Error (abs(Experimental) -
abs(Optimized))');
xlabel('Applied Load (N)');
ylabel('Deflection (mm)');

subplot(3,1,3); plot(Disp_e(:,2),'r')
hold on
plot(Disp_o(:,2),'-.b')
hold off
legend('Experimental','Optimized');
title('Y-Axis Displacement');
xlabel('Trial');
ylabel('Deflection (mm)');

% Plot Z-Axis Results

figure
subplot(3,1,1); plot(abs(Disp_e(:,3)) - abs(Disp_o(:,3)))
title('Z-Axis Displacement Error (abs(Experimental) -
abs(Optimized))');
xlabel('Trial');
ylabel('Deflection (mm)');

for i = 1:27
    subplot(3,1,2); plot(le_loads,abs(Disp_e(6*(i-1) + 1:6*(i-1) +
6,3)) -
abs(Disp_o(6*(i-1) + 1:6*(i-1) + 6,3)));
    hold on
end
hold off
title('Z-Axis Displacement Error (abs(Experimental) -
abs(Optimized))');
xlabel('Applied Load (N)');
ylabel('Deflection (mm)');

subplot(3,1,3); plot(Disp_e(:,3),'r')
hold on
plot(Disp_o(:,3),'-.b')
hold off
legend('Experimental','Optimized');
title('Z-Axis Displacement');
xlabel('Trial');
ylabel('Deflection (mm)');
```

10.5.10 Low End Objective Function

- Code Filename: objfun_le_v022502.m
- Code Description: Objective Function for reduced data set solutions.
- Inputs:
  i. Stiffness matrix, x0
  ii. OptoTrak results, optdata
  iii. Local path to store files, localpath
  iv. Filenum to start incremementing with, filenum
  v. Regime to solve, regmie
- Outputs:
  i. Objective Function Value, f
- Application: This is the objective function used to solve the reduced data set optimization problems.
- Notes: N/A

```
% Mark W. Abbott
% Feburary 21, 2002

% Objective Function for Optimization with applied torques

function f = objfun_le_v022502(x0,optdata,localpath,filenum,regime)

[r,c] = size(optdata);
Load = zeros(1,3);
Theta = zeros(1,6);
ExpPos = zeros(1,3);
ExpRot = zeros(1,3);
Error = 0;
Error1 = 0;
Error2 = 0;
Count = 1;
apptor = zeros(243,3);

if regime == 3
    Top_loop = 1.5*25.4;            % Measured Quantity, Problem
Specific
    Side_loop = 0.875*25.4;         % Measured Quantity, Problem
Specific
    for i = 1:r
        if i < 82
            apptor(i,:) = [0,0,0];
        elseif i < 163
            vect = [0,0,Side_loop];        % Zero Frame!
            apptor(i,:) = cross(vect,squeeze(optdata(i,1:3)));
        else
            vect = [0,0,Top_loop];
            apptor(i,:) = cross(vect,squeeze(optdata(i,1:3)));
        end
    end
end
```

```
for i = 1:r
    tmp = i;            % Code to ignore cases above 25 pounds
    while tmp > 9
        tmp = tmp - 9;
    end
    if tmp < 7
        Load = optdata(i,1:3);
        Theta = optdata(i,4:9)*(pi/180);
        ExpPos = optdata(i,10:12);
        ExpRot = optdata(i,13:15);
        EndLoad = [Load';0;0;0];
        Hold =
compliant_v022502(x0,EndLoad,Theta,squeeze(apptor(i,:)));
        AnaPos = Hold(1:3);
        AnaRot = Hold(4:6);
        Pos1 = abs(abs(ExpPos(1)) - abs(AnaPos(1))) + 1;
        Pos2 = abs(abs(ExpPos(2)) - abs(AnaPos(2))) + 1;
        Pos3 = abs(abs(ExpPos(3)) - abs(AnaPos(3))) + 1;
        Error1 = Pos1^2 + Pos2^2 + Pos3^2;
        Rot1 = abs(abs(ExpRot(1)) - abs(AnaRot(1))) + 1;
        Rot2 = abs(abs(ExpRot(2)) - abs(AnaRot(2))) + 1;
        Rot3 = abs(abs(ExpRot(3)) - abs(AnaRot(3))) + 1;
        Error2 = Rot1^2 + Rot2^2 + Rot3^2;
        Error = Error + Error1 + Error2;
    end
end

temp = fix(clock);
fprintf('\n%g.%g.%g  %g.%g.%g   %g Objective function Current Value:
%g',temp(2),temp(3),temp(1),temp(4),temp(5),temp(6),filenum, Error);

f = Error;

if filenum < 10
    strnum = strcat('0',num2str(filenum));
else
    strnum = num2str(filenum);
end

optimlog = strcat(localpath,strnum,'optimlog.txt');

fid = fopen(optimlog,'a');
fseek(fid,0,'eof');
junk = fprintf(fid,'\n%g.%g.%g  %g.%g.%g  Objective function Current
Value:  %g\n',
temp(2),temp(3),temp(1),temp(4),temp(5),temp(6),Error);
junk = fprintf(fid,'     x(01-07) %8.8g  %8.8g  %8.8g  %8.8g  %8.8g
%8.8g  %8.8g\n',x0(1),x0(2),x0(3),x0(4),x0(5),x0(6),x0(7));
junk = fprintf(fid,'     x(08-14) %8.8g  %8.8g  %8.8g  %8.8g  %8.8g
%8.8g  %8.8g\n',x0(8),x0(9),x0(10),x0(11),x0(12),x0(13),x0(14));
junk = fprintf(fid,'     x(15-21) %8.8g  %8.8g  %8.8g  %8.8g  %8.8g
%8.8g  %8.8g\n',x0(15),x0(16),x0(17),x0(18),x0(19),x0(20),x0(21));
fclose(fid);
```

10.5.11 Low End Optimize Routine

- Code Filename: optimize_le_v022502.m
- Code Description: Reduced Data set optimization front-end.
- Inputs:
  i. All Optimization solver required inputs
  ii. Regime to be solved
  iii. Optimization routine to use to solve problem
  iv. Selection of Stiffness matrices to use as initial conditions for solver
- Outputs:
  i. Converged stiffness matrix
- Application: This is the primary optimization front end and is used to control the optimization process.
- Notes: This code only differs from the full data set optimization code by a few lines, and therefore only those linjes will be mentioned here. The remaining code is shown in 10.5.5 .

```
    % Set Options
    OPTIONS =
optimset('LargeScale','off','Tolfun',ObjLimit,'TolX',Xincr,
'MaxFunEvals',
maxeval,'MaxIter',iter,'Diagnostics','on');
% Specifies a medium-scale solver

    % Call Optimization Routine
    if mincon == 1
        [x,fval,exitflag,output,lambda,grad] =
fmincon('objfun_le_v022502',x0,A,B,Aeq,Beq,lb,ub,[],OPTIONS,optdata,
localpath,j,regime)  % Calls the Optimization function
    elseif minimax == 1
        [x,fval,exitflag,output,lambda,grad] =
fminimax('objfun__le_v022502',x0,A,B,Aeq,Beq,lb,ub,[],OPTIONS,optdata
,localpath,j,regime)  % Calls the Optimization function
    end
    % Exitflag Notes
```

10.5.12 Low End Thesis Plot Routine

- Code Filename: thesis_plot_le_v030102.m
- Code Description: Code to build plots for thesis presentation.
- Inputs:
  i. Proposed stiffness matrix
  ii. Regime to solve
- Outputs:
  i. Plots comparing results
  ii. Objective function value
- Application: This code computes the resulting plots based on a given stiffness matrix. The output from this code is presented throughout the

thesis and provides the best comparison of the quality of the results from the optimization.

- Notes: This code is very similar to the full results thesis plot routine, however, there are small changes throughout the code it will be included in its entirety here.

```
% Mark W. Abbott
% March 1, 2001

% Code to generate plots from optimzation output custom tailored for
the Thesis


clear all;
close all;
clc;

%================================================================

path2 = 'c:\data\r3\r3_1k_part\1034.77_+_solution002.mat';
regime = 3;
filenum = 1234;

load c:\data\Constant.mat;
top_loop_offset = 1.5;      % Offset from 6th frame to Top Loaded
Cases
side_loop_offset = 0.875;   % Offset from 6th frame to Side Loaded
Cases

%================================================================

testval = fopen(path2);

if testval == -1

    fprintf('WARNING:  Solution.mat file not found! ...
Aborting.');

else

    load(path2);
    expdata = strcat('c:\data\r',num2str(regime),'\expdata.mat');
    load(expdata);  % Inserts Experimental Deflections into
workspace

    fclose all;

    Rot_o = zeros(162,3);
    Disp_o = zeros(162,3);

    fprintf('Beginning to Compile Optimized Solution Set\n');

    apptor = zeros(243,3);
    [r,c] = size(optdata);
```

```
    % Build Torque due to offset Loads for 3rd Regime

    if regime == 3
        Top_loop = top_loop_offset*25.4;
        Side_loop = side_loop_offset*25.4;
        for i = 1:r
            if i < 82
                apptor(i,:) = [0,0,0];
            elseif i < 163
                vect = [0,0,Side_loop];          % Zero Frame!
                apptor(i,:) = cross(vect,squeeze(optdata(i,1:3)));
            else
                vect = [0,0,Top_loop];           % Zero Frame!
                apptor(i,:) = cross(vect,squeeze(optdata(i,1:3)));
            end
        end
    end

    % Call Compliance code to generate optimized solutions

    step = 1;

    for i = 1:27
        for j = 1:9
            k = 9*(i - 1) + j;
            tmp = k;
            while tmp > 9
                tmp = tmp - 9;
            end
            if tmp < 7
                Hold =
compliant_v022502(x,[optdata(k,1:3)';0;0;0],optdata(k,4:9)*pi/180,sq
ueeze(
apptor(k,:)));
                Rot_o(step,:) = Hold(4:6);
                Disp_o(step,:) = Hold(1:3);
                Rot_e(step,:) = optdata(k,13:15);
                Disp_e(step,:) = optdata(k,10:12);
                step = step + 1;
            end

        end
    end


    localpath = strcat('c:\data\r',num2str(regime),'\sol_loop\');

    %obj =
objfun_le_v022502(x,optdata,localpath,filenum,num2str(regime));

    le_loads = [0;loads(1:5)];

    % Compile X, Y and Z direction Data into a single result via
sqrt of
sum of squares

    Disp_o_1 = sqrt(sum((Disp_o.^2)')');
```

```
    Disp_e_1 = sqrt(sum((Disp_e.^2)')');

    % Plot Error Plot

    absError = abs(Disp_e_1 - Disp_o_1);
    perError = 100*absError/abs(Disp_e_1);

    figure
    plot(perError);
    hold on
    plot([54;54],[min(min(perError));max(max(perError))],':');
    plot([108;108],[min(min(perError));max(max(perError))],':');
    hold off
    xlabel('Trial');
    ylabel('Percentage Deflection Error');
    %title('Percentage Error of Absolute Displacement');


    % Plot Overall Results

    figure
    for i = 1:27
        subplot(2,1,1); plot(le_loads,abs(Disp_e_1(6*(i-1) + 1:6*(i-
1) + 6)) - abs(Disp_o_1(6*(i-1) + 1:6*(i-1) + 6)));
        hold on
    end

    hold off
    title('Absolute Displacement Error');
    xlabel('Applied Load (N)');
    ylabel('Deflection (mm)');

    subplot(2,1,2); plot(Disp_e_1,'r')
    hold on
    plot(Disp_o_1,'-.b')
    min1 = min([min(Disp_o_1) min(min(Disp_e_1))]);
    max1 = 1.25*max([max(Disp_o_1) max(max(Disp_e_1))]);
    plot([54;54],[min1;max1],'-k');
    plot([108;108],[min1;max1],'-k');
    hold off
    legend('Experimental','Optimized');
    title('Absolute Displacement');
    xlabel('Trial');
    ylabel('Deflection (mm)');

    % Plot X-Axis Results

    figure
    le_loads = [0;loads(1:5)];
    for i = 1:27
        subplot(2,2,1); plot(le_loads,abs(Disp_e(6*(i-1) + 1:6*(i-1)
+ 6,1)) - abs(Disp_o(6*(i-1) + 1:6*(i-1) + 6,1)));
        hold on
    end
    hold off
    title('X-Axis Displacement Error');
    xlabel('Applied Load (N)');
```

```
    ylabel('Deflection (mm)');

    subplot(2,2,3); plot(Disp_e(:,1),'r')
    hold on
    plot(Disp_o(:,1),'-.b')
    min1 = min([min(Disp_o(:,1)) min(min(Disp_e(:,1)))]);
    max1 = 1.25*max([max(Disp_o(:,1)) max(max(Disp_e(:,1)))]);
    plot([54;54],[min1;max1],'-k');
    plot([108;108],[min1;max1],'-k');
    hold off
    legend('Experimental','Optimized');
    title('X-Axis Displacement');
    xlabel('Trial');
    ylabel('Deflection (mm)');

    for i = 1:27
        subplot(2,2,2); plot(le_loads,abs(Rot_e(6*(i-1) + 1:6*(i-1)
+ 6,1)) - abs(Rot_o(6*(i-1) + 1:6*(i-1) + 6,1)));
        hold on
    end
    hold off
    title('X-Axis Rotation Error');
    xlabel('Applied Load (N)');
    ylabel('Rotation (rad)');

    subplot(2,2,4); plot(Rot_e(:,1),'r')
    hold on
    plot(Rot_o(:,1),'-.b')
    min1 = min([min(Rot_o(:,1)) min(min(Rot_e(:,1)))]);
    max1 = 1.25*max([max(Rot_o(:,1)) max(max(Rot_e(:,1)))]);
    plot([54;54],[min1;max1],'-k');
    plot([108;108],[min1;max1],'-k');
    hold off
    legend('Experimental','Optimized');
    title('X-Axis Rotation');
    xlabel('Trial');
    ylabel('Rotation (rad)');

    % Plot Y-Axis Results

    figure
    for i = 1:27
        subplot(2,2,1); plot(le_loads,abs(Disp_e(6*(i-1) + 1:6*(i-1)
+ 6,2)) - abs(Disp_o(6*(i-1) + 1:6*(i-1) + 6,2)));
        hold on
    end
    hold off
    title('Y-Axis Displacement Error');
    xlabel('Applied Load (N)');
    ylabel('Deflection (mm)');

    subplot(2,2,3); plot(Disp_e(:,2),'r')
    hold on
    plot(Disp_o(:,2),'-.b')
    min1 = min([min(Disp_o(:,2)) min(min(Disp_e(:,2)))]);
    max1 = 1.25*max([max(Disp_o(:,2)) max(max(Disp_e(:,2)))]);
    plot([54;54],[min1;max1],'-k');
```

```
    plot([108;108],[min1;max1],'-k');
    hold off
    legend('Experimental','Optimized');
    title('Y-Axis Displacement');
    xlabel('Trial');
    ylabel('Deflection (mm)');

    for i = 1:27
        subplot(2,2,2); plot(le_loads,abs(Rot_e(6*(i-1) + 1:6*(i-1)
+ 6,2)) - abs(Rot_o(6*(i-1) + 1:6*(i-1) + 6,2)));
        hold on
    end
    hold off
    title('Y-Axis Rotation Error');
    xlabel('Applied Load (N)');
    ylabel('Rotation (rad)');

    subplot(2,2,4); plot(Rot_e(:,2),'r')
    hold on
    plot(Rot_o(:,2),'-.b')
    min1 = min([min(Rot_o(:,2)) min(min(Rot_e(:,2)))]);
    max1 = 1.25*max([max(Rot_o(:,2)) max(max(Rot_e(:,2)))]);
    plot([54;54],[min1;max1],'-k');
    plot([108;108],[min1;max1],'-k');
    hold off
    legend('Experimental','Optimized');
    title('Y-Axis Rotation');
    xlabel('Trial');
    ylabel('Rotation (rad)');



    % Plot Z-Axis Results

    figure
    for i = 1:27
        subplot(2,2,1); plot(le_loads,abs(Disp_e(6*(i-1) + 1:6*(i-1)
+ 6,3)) - abs(Disp_o(6*(i-1) + 1:6*(i-1) + 6,3)));
        hold on
    end
    hold off
    title('Z-Axis Displacement Error');
    xlabel('Applied Load (N)');
    ylabel('Deflection (mm)');

    subplot(2,2,3); plot(Disp_e(:,3),'r')
    hold on
    plot(Disp_o(:,3),'-.b')
    min1 = min([min(Disp_o(:,3)) min(min(Disp_e(:,3)))]);
    max1 = 1.25*max([max(Disp_o(:,3)) max(max(Disp_e(:,3)))]);
    plot([54;54],[min1;max1],'-k');
    plot([108;108],[min1;max1],'-k');
    hold off
    legend('Experimental','Optimized');
    title('Z-Axis Displacement');
    xlabel('Trial');
    ylabel('Deflection (mm)');
```

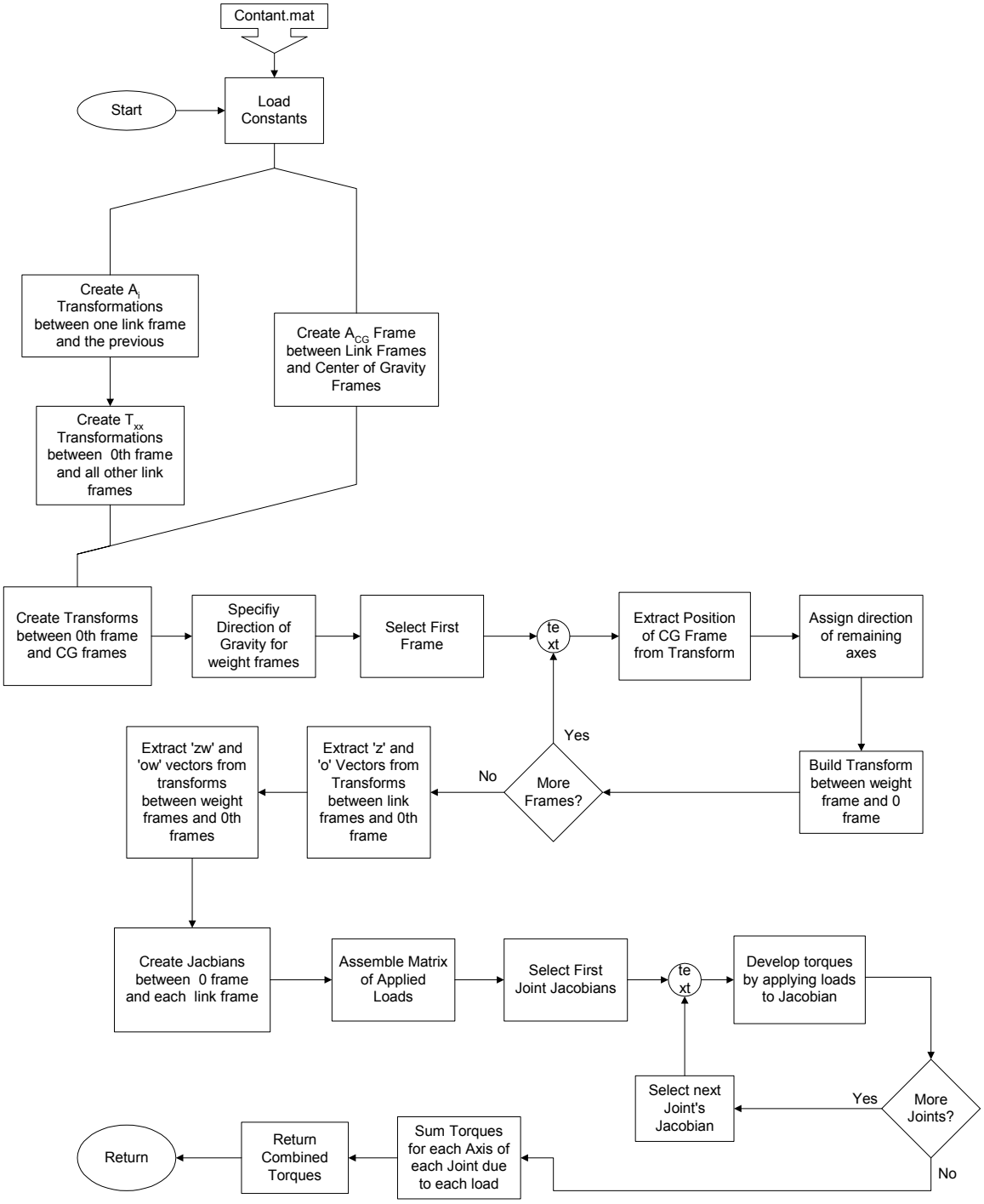
```
    for i = 1:27
        subplot(2,2,2); plot(le_loads,abs(Rot_e(6*(i-1) + 1:6*(i-1)
+ 6,3)) - abs(Rot_o(6*(i-1) + 1:6*(i-1) + 6,3)));
        hold on
    end
    hold off
    title('Z-Axis Rotation Error');
    xlabel('Applied Load (N)');
    ylabel('Rotation (rad)');

    subplot(2,2,4); plot(Rot_e(:,3),'r')
    hold on
    plot(Rot_o(:,3),'-.b')
    min1 = min([min(Rot_o(:,3)) min(min(Rot_e(:,3)))]);
    max1 = 1.25*max([max(Rot_o(:,3)) max(max(Rot_e(:,3)))]);
    plot([54;54],[min1;max1],'-k');
    plot([108;108],[min1;max1],'-k');
    hold off
    legend('Experimental','Optimized');
    title('Z-Axis Rotation');
    xlabel('Trial');
    ylabel('Rotation (rad)');

    % Make Combined Rotation Plot

    figure
    for i = 1:27
        subplot(3,2,1); plot(le_loads,abs(Rot_e(6*(i-1) + 1:6*(i-1)
+ 6,1)) - abs(Rot_o(6*(i-1) + 1:6*(i-1) + 6,1)));
        hold on
    end
    hold off
    title('Rotation Error');
    %xlabel('Applied Load (N)');
    ylabel('X-Axis Rotation (rad)');

    subplot(3,2,2); plot(Rot_e(:,1),'r')
    hold on
    plot(Rot_o(:,1),'-.b')
    min1 = min([min(Rot_o(:,1)) min(min(Rot_e(:,1)))]);
    max1 = 1.25*max([max(Rot_o(:,1)) max(max(Rot_e(:,1)))]);
    plot([54;54],[min1;max1],'-k');
    plot([108;108],[min1;max1],'-k');
    hold off
    legend('Experimental','Optimized');
    title('Rotation');
    %xlabel('Trial');
    %ylabel('X-Axis Rotation (rad)');

    for i = 1:27
        subplot(3,2,3); plot(le_loads,abs(Rot_e(6*(i-1) + 1:6*(i-1)
+ 6,2)) - abs(Rot_o(6*(i-1) + 1:6*(i-1) + 6,2)));
        hold on
    end
    hold off
    %title('Y-Axis Rotation Error');
```

```
    %xlabel('Applied Load (N)');
    ylabel('Y-Axis Rotation (rad)');

    subplot(3,2,4); plot(Rot_e(:,2),'r')
    hold on
    plot(Rot_o(:,2),'-.b')
    min1 = min([min(Rot_o(:,2)) min(min(Rot_e(:,2)))]);
    max1 = 1.25*max([max(Rot_o(:,2)) max(max(Rot_e(:,2)))]);
    plot([54;54],[min1;max1],'-k');
    plot([108;108],[min1;max1],'-k');
    hold off
    legend('Experimental','Optimized');
    %title('Y-Axis Rotation');
    %xlabel('Trial');


    for i = 1:27
        subplot(3,2,5); plot(le_loads,abs(Rot_e(6*(i-1) + 1:6*(i-1)
+ 6,3)) - abs(Rot_o(6*(i-1) + 1:6*(i-1) + 6,3)));
        hold on
    end
    hold off
    %title('Z-Axis Rotation Error');
    xlabel('Applied Load (N)');
    ylabel('Z-Axis Rotation (rad)');

    subplot(3,2,6); plot(Rot_e(:,3),'r')
    hold on
    plot(Rot_o(:,3),'-.b')
    min1 = min([min(Rot_o(:,3)) min(min(Rot_e(:,3)))]);
    max1 = 1.25*max([max(Rot_o(:,3)) max(max(Rot_e(:,3)))]);
    plot([54;54],[min1;max1],'-k');
    plot([108;108],[min1;max1],'-k');
    hold off
    legend('Experimental','Optimized');
    %title('Z-Axis Rotation');
    xlabel('Trial');
    %ylabel('Rotation (rad)');


end
```

10.5.13 Torque Solver

- Code Filename: torque_v022502.m
- Code Description: This code solves the kinematic torque problem.
- Inputs:
    i. Theta joint angles, t
    ii. Forces applied to the end-effector, Loadend
    iii. Torques applied to the end-effector, apptor
- Outputs:
    i. Torques developed about each axis of the DH frames

- Application: The torque code solves for the loads applied to each DH frame based on the manipulator kinematics.
- Notes: N/A




```
% Merlin Forward Kinematic Solver
% Mark W. Abbott
% October 24, 2001, revised on February 7, 2002
```

```
% MATLAB code to compute the Torques for 6-DOF Merlin Robots
%

%clear all;
%close all;
%clc;

function [Torque] = torque_v4(t,Loadend,FLAG,apptor);

% t = theta values for computation (radians)
% disp = Locations of weight frames relative to 'primary' link frame
% Loadend = Applied Load at end-effector

% ------------------------------------------------------------------
-----------------------------
% User definable Settings

load c:\data\Constant.mat

% ------------------------------------------------------------------
-----------------------------

% Setting up Transformation Matrices from one frame to the next.
A(:,:,i) is from frame i-1 to i

A = zeros(4,4,6);
A(:,:,1) = [cos(t(1)),0,sin(t(1)),0;sin(t(1)),0,-
cos(t(1)),0;0,1,0,d1;0,0,0,1];
A(:,:,2) = [cos(t(2)),-
sin(t(2)),0,a2*cos(t(2));sin(t(2)),cos(t(2)),0,a2*sin(t(2));0,0,1,d2;
0,0,0,1];
A(:,:,3) = [-
sin(t(3)),0,cos(t(3)),0;cos(t(3)),0,sin(t(3)),0;0,1,0,0;0,0,0,1];
A(:,:,4) = [cos(t(4)),0,-sin(t(4)),0;sin(t(4)),0,cos(t(4)),0;0,-
1,0,d4;0,0,0,1];
A(:,:,5) = [cos(t(5)),0,sin(t(5)),0;sin(t(5)),0,-
cos(t(5)),0;0,1,0,0;0,0,0,1];
A(:,:,6) = [cos(t(6)),-
sin(t(6)),0,0;sin(t(6)),cos(t(6)),0,0;0,0,1,d6;0,0,0,1];

% Setting up the frames from each link frame to the center of gravity
frame
% This is based on the assumption that all CG frames have the same
pose as their parent frames

ACG = zeros(4,4,6);

for i = 1:6
    ACG(:,:,i) = [1,0,0,dx(i);0,1,0,dy(i);0,0,1,dz(i);0,0,0,1];
end

% Establishing Transformations from frame 0 to each parent frame

T0 = zeros(4,4,6);

for i = 1:6
    if i == 1
```

```
        T0(:,:,i) = A(:,:,i);
    else
        T0(:,:,i) = T0(:,:,i - 1)*A(:,:,i);
    end
end

% Establishing Transformations from frame 0 to each CG Frame

TCG = zeros(4,4,6);
for i = 1:6
    TCG(:,:,i) = T0(:,:,i)*ACG(:,:,i);
end

% Creating Matrices to align the weight with gravity

yw = [0;0;-1];      % Defines the direction of the load for all
weight frames
T0W = zeros(4,4,6);

for i = 1:6

    % Extract the x, y and z between 0 and CG frame

    xcg = TCG(1:3,1,i);
    ycg = TCG(1:3,2,i);
    zcg = TCG(1:3,3,i);

    % Generate the Zw vector based on required cross products

    if abs(xcg) == abs(yw)
        zw = cross(zcg,yw)/sqrt(dot(cross(zcg,yw),cross(zcg,yw)));
    else
        zw = cross(xcg,yw)/sqrt(dot(cross(xcg,yw),cross(xcg,yw)));
    end

    % Generate the Xw vector from Yw and Zw

    xw = cross(yw,zw);

    % Build the Transformation from 0 to the new weight frame

    T0W(:,:,i) =
[xw(1),yw(1),zw(1),TCG(1,4,i);xw(2),yw(2),zw(2),TCG(2,4,i);xw(3),
yw(3),zw(3),TCG(3,4,i);0,0,0,1];
end


% Extracting z and o vectors for Jacobian Development
% Index 1 corresponds to Frame 0, Therefore there are 7

x = zeros(3,7);
y = zeros(3,7);
z = zeros(3,7);
zw = zeros(3,7);
o = zeros(3,7);
ow = zeros(3,7);
```

```
x(:,1) = [1;0;0];
y(:,1) = [0;1;0];
z(:,1) = [0;0;1];
zw(:,1) = [0;0;0];

for i = 2:7
    x(:,i) = T0(1:3,1,i - 1);
    y(:,i) = T0(1:3,2,i - 1);
    z(:,i) = T0(1:3,3,i - 1);
    zw(:,i) = T0W(1:3,3,i - 1);
    o(:,i) = T0(1:3,4,i - 1);
    ow(:,i) = T0W(1:3,4,i - 1);
end

% Begin building Jacobians for Torque Calculation

Jwz = zeros(6,6,6);
Jwx = zeros(6,6,6);
Jwy = zeros(6,6,6);
J06z = zeros(6,6);
J06x = zeros(6,6);
J06y = zeros(6,6);

% i goes from 1 - 6, denoting which load is being addressed
% k is i + 1, this accounts for the numbering difference in ow,
and sets it static for each loop
% j goes from the 1 to i, computing the columns as it goes

for i = 1:6
    k = i + 1;
    for j = 1:i
        Jwx(:,j,i) = [cross(x(:,j),ow(:,k) - o(:,j));x(:,j)];
        Jwy(:,j,i) = [cross(y(:,j),ow(:,k) - o(:,j));y(:,j)];
        Jwz(:,j,i) = [cross(z(:,j),ow(:,k) - o(:,j));z(:,j)];
        if i == 6
            J06x(:,j) = [cross(x(:,j),o(:,k) - o(:,j));x(:,j)];
            J06y(:,j) = [cross(y(:,j),o(:,k) - o(:,j));y(:,j)];
            J06z(:,j) = [cross(z(:,j),o(:,k) - o(:,j));z(:,j)];
        end

    end
end

% Build defined matrix of loads to be applied

Loadw = [zeros(1,6);zeros(1,6);-
weight;zeros(1,6);zeros(1,6);zeros(1,6)];
Load = [Loadw,Loadend];

% Build defined matrix of torques to be applied

Torkx = zeros(6,7);
Torky = zeros(6,7);
Torkz = zeros(6,7);

% Builds Torkx(i,j)
%   i is the ...
```

```
for i = 1:6
    Torkx(:,i) = Jwx(:,:,i)'*Load(:,i);
    Torky(:,i) = Jwy(:,:,i)'*Load(:,i);
    Torkz(:,i) = Jwz(:,:,i)'*Load(:,i);
end

Torkx(:,7) = J06x'*Load(:,7);
Torky(:,7) = J06y'*Load(:,7);
Torkz(:,7) = J06z'*Load(:,7);

Torquex = sum(Torkx')';
Torquey = sum(Torky')';
Torquez = sum(Torkz')';

Torque = [Torquex,Torquey,Torquez;apptor];

return
```

10.5.14 Objective Function for Planar Fit

- Code Filename: objfun_pln.m
- Code Description: Objective Function for fitting 4 points to a plane.
- Inputs:
    i. Initial guess of planar constants, x
    ii. Four data points to fit to, Pnts
- Outputs:
    i. Objective Function, f
- Application: This code is used to fit the four center points to a plane during the generation of transformation matrices.
- Notes: N/A

```
% Mark W. Abbott
% December 3, 2001

% Objective Function for Optimization

function f = objfun_pln(x,Pnts)

%f = exp(x(1))*(4*x(1)^2 + 2*x(2)^2 + 4*x(1)*x(2) + 2*x(2) + 1);
%Objective Function

f1 = x(1)*Pnts(1,1) + x(2)*Pnts(1,2) + x(3)*Pnts(1,3) - x(4);
f2 = x(1)*Pnts(2,1) + x(2)*Pnts(2,2) + x(3)*Pnts(2,3) - x(4);
f3 = x(1)*Pnts(3,1) + x(2)*Pnts(3,2) + x(3)*Pnts(3,3) - x(4);
f4 = x(1)*Pnts(4,1) + x(2)*Pnts(4,2) + x(3)*Pnts(4,3) - x(4);

f = sqrt(f1^2 + f2^2 + f3^2 + f4^2);
```

10.5.15 Raw Data Plot Routine

- Code Filename: raw_data_plot_v031402.m
- Code Description: Code to plot uncleaned data
- Inputs:
    i. N/A
- Outputs:
    i. Plots
- Application: This code is used to view results from expdata.m, without modifications.
- Notes: N/A

```
% Mark W. Abbott
% March 14, 2002

% Code to generate raw data plots for thesis

clear all;
close all;
clc;

load c:\data\constant.mat;
load c:\data\r3\expdata.mat;

xaxis = 1:1:243;

figure
plot(xaxis,squeeze(optdata(:,10)),'--r')
hold on
plot(xaxis,squeeze(optdata(:,11)),'-.g')
plot(xaxis,squeeze(optdata(:,12)),'b')
plot([81;81],[min(min(optdata(:,10:12)));max(max(optdata(:,10:12)))],
':c');
plot([162;162],[min(min(optdata(:,10:12)));max(max(optdata(:,10:12)))
],':c');
hold off
xlabel('Data Sample');
ylabel('Distance (mm)');
%title('Displacement Raw Data');
legend('X-Dir','Y-Dir','Z-Dir',0);

figure
plot(xaxis,squeeze(optdata(:,13)),'--r')
hold on
plot(xaxis,squeeze(optdata(:,14)),'-.g')
plot(xaxis,squeeze(optdata(:,15)),'b')
plot([81;81],[min(min(optdata(:,13:15)));max(max(optdata(:,13:15)))],
':c');
plot([162;162],[min(min(optdata(:,13:15)));max(max(optdata(:,13:15)))
],':c');
hold off
```

```
axis([1 243 -5E-5 5E-5])
xlabel('Data Sample');
ylabel('Rotation (rad)');
%title('Rotation Raw Data');
legend('X-Axis','Y-Axis','Z-Axis',0);

figure
for i = 1:27
    subplot(3,1,1); plot([0;loads],optdata(9*(i-1) + 1:9*(i-1) +
9,10))
    hold on
end
xlabel('Applied Load (N)');
ylabel('Displacement (mm)');
title('X-Dir Displacement');
for i = 1:27
    subplot(3,1,2); plot([0;loads],optdata(9*(i-1) + 1:9*(i-1) +
9,11))
    hold on
end
xlabel('Applied Load (N)');
ylabel('Displacement (mm)');
title('Y-Dir Displacement');
for i = 1:27
    subplot(3,1,3); plot([0;loads],optdata(9*(i-1) + 1:9*(i-1) +
9,12))
    hold on
end
xlabel('Applied Load (N)');
ylabel('Displacement (mm)');
title('Z-Dir Displacement');

figure
for i = 1:27
    subplot(3,1,1); plot([0;loads],optdata(9*(i-1) + 1:9*(i-1) +
9,13))
    hold on
end
xlabel('Applied Load (N)');
ylabel('Rotation (rad)');
title('X-Axis Rotation');
for i = 1:27
    subplot(3,1,2); plot([0;loads],optdata(9*(i-1) + 1:9*(i-1) +
9,14))
    hold on
end
xlabel('Applied Load (N)');
ylabel('Rotation (rad)');
title('Y-Axis Rotation');
for i = 1:27
    subplot(3,1,3); plot([0;loads],optdata(9*(i-1) + 1:9*(i-1) +
9,15))
    hold on
end
xlabel('Applied Load (N)');
ylabel('Rotation (rad)');
title('Z-Axis Rotation');
```

10.5.16 Merlin FK – IK Test Routine

- Code Filename: Merlin_test.m
- Code Description: Test Code for IK and FK
- Inputs:
    i. Initial Joint Angles
- Outputs:
    i. All Solutions to meet joint angles
- Application: This code is used to verify the forward and inverse kinematics routines are fully functional.
- Notes: N/A

```
% Test Code for Evaluating Forward and Inverse Kinematic Subfunctions
% Mark W. Abbott
% October 5, 2001

% MATLAB code to test and verify both the forward and inverse
kinematic modules
%   as well as provide a general basis for theiur use.
%
% Based on a specified set of input thetas (theta_test), the
resulting transformation
%   is constructed and then use to verify the inverse kinematics
ability to solve the problem.
% A total of 8 solutions are possible, however due to mechanical
limits, not all will
%   be functional at every position.
%

clear all;
close all;
clc;

% ------------------------------------------------------------------
% User Definable settings


d1 = 45.46*25.4;        % Distance from zero frame to first frame
(from floor to shoulder)
d2 = 11.9*25.4;          % Offset from center of robot trunk to
outside of shoulder
a2 = 17.375*25.4;        % Length of Link from shoulder to elbow
d4 = 17.25*25.4;         % Length of Link from elbow to wrist
d6 = 3.5*25.4;           % Distance from center of wrist to tool
faceplate

theta_test = [0;40;25;0;25;0]  % Desired Joint Angles for Test
Sequence

Error_Limit = 10^-10;   % Defines the level at which error between
transformation
                        % matrices are assumed to be zero
```

```
% ------------------------------------------------------------------

% Test Configuration

fprintf('Generating Initial Seed Solution from Forward
Kinematics\n');
fprintf('Input Theta = ');

% Input string in format {theta1;theta2;theta3;theta4;theta5;theta6]

theta_test = theta_test*pi/180;

OPTIONS=[];
[J,T06] = Merlin_FK(theta_test,d1,OPTIONS,d2,a2,d4,d6);

if J == 0 & T06 == 0
    fprintf('\nUnable to reach Point:  Bypassing Inverse
Kinematics\n\n');
    fprintf('This occurs when one of the joints is requested to
position\n');
    fprintf('itself beyond mechanical stops.\n');
else
    fprintf('Solving Inverse Kinematics for Input\n');

    % Extract rotation and position matrices for inverse solver

    R = T06(1:3,1:3)
    d = T06(1:3,4)

    OPTIONS = [];
    Verify = Merlin_IK(R,d,OPTIONS,d1,d2,a2,d4,d6);

    Output = Verify*180/pi;

    % Force very small terms to zero.

    for x = 1:6
        for y = 1:8
            if abs(Output(x,y)) < 10^-8
                Output(x,y) = 0;
            end
        end
    end

    % Begin Verifying Solutions with Forward Kinematics

    CK = zeros(8,1);

    [J_1,T06_1] = Merlin_FK(Verify(:,1),d1,OPTIONS,d2,a2,d4,d6);
    Check1 = T06 - T06_1;
    CK(1) = max(max(abs(Check1)));

    [J_2,T06_2] = Merlin_FK(Verify(:,2),d1,OPTIONS,d2,a2,d4,d6);
    Check2 = T06 - T06_2;
    CK(2) = max(max(abs(Check2)));
```

```
    [J_3,T06_3] = Merlin_FK(Verify(:,3),d1,OPTIONS,d2,a2,d4,d6);
    Check3 = T06 - T06_3;
    CK(3) = max(max(abs(Check3)));

    [J_4,T06_4] = Merlin_FK(Verify(:,4),d1,OPTIONS,d2,a2,d4,d6);
    Check4 = T06 - T06_4;
    CK(4) = max(max(abs(Check4)));

    [J_5,T06_5] = Merlin_FK(Verify(:,5),d1,OPTIONS,d2,a2,d4,d6);
    Check5 = T06 - T06_5;
    CK(5) = max(max(abs(Check5)));

    [J_6,T06_6] = Merlin_FK(Verify(:,6),d1,OPTIONS,d2,a2,d4,d6);
    Check6 = T06 - T06_6;
    CK(6) = max(max(abs(Check6)));

    [J_7,T06_7] = Merlin_FK(Verify(:,7),d1,OPTIONS,d2,a2,d4,d6);
    Check7 = T06 - T06_7;
    CK(7) = max(max(abs(Check7)));

    [J_8,T06_8] = Merlin_FK(Verify(:,8),d1,OPTIONS,d2,a2,d4,d6);
    Check8 = T06 - T06_8;
    CK(8) = max(max(abs(Check8)));

    % Establish FKE vector to regulate output to valid solutions
only.

    FKE = zeros(8,1);

    if J_1 == 0 & T06_1 == 0
        FKE(1) = 1;
    end
    if J_2 == 0 & T06_2 == 0
        FKE(2) = 1;
    end
    if J_3 == 0 & T06_3 == 0
        FKE(3) = 1;
    end
    if J_4 == 0 & T06_4 == 0
        FKE(4) = 1;
    end
    if J_5 == 0 & T06_5 == 0
        FKE(5) = 1;
    end
    if J_6 == 0 & T06_6 == 0
        FKE(6) = 1;
    end
    if J_7 == 0 & T06_7 == 0
        FKE(7) = 1;
    end
    if J_8 == 0 & T06_8 == 0
        FKE(8) = 1;
    end

    % Force very small errors to zero for output

    for i = 1:8
```

```
        if CK(i) < Error_Limit
            CK(i) = 0;
        end
    end

    % Begin outputting valid solutions

    fprintf('\nThe Valid Kinematic Solutions are:\n');
    if FKE(1) == 0 & FKE(2) == 0
        fprintf('Solution 1 and 2:\n  Theta 1 = %10.4g\t\t    Theta 1
= %10.4g\n  Theta 2 = %10.4g\t\t    Theta 2 = %10.4g\n  Theta 3 =
%10.4g\t\t    Theta 3 =
%10.4g\n',Output(1,1),Output(1,2),Output(2,1),Output(2,2),Output(3,1)
,Output(3,2));
        fprintf('  Theta 4 = %10.4g\t\t    Theta 4 = %10.4g\n  Theta
5 = %10.4g\t\t    Theta 5 = %10.4g\n  Theta 6 = %10.4g\t\t    Theta 6
=
%10.4g\n',Output(4,1),Output(4,2),Output(5,1),Output(5,2),Output(6,1)
,Output(6,2));
        fprintf('  Max Error = %8.4g\t\t    Max Error =
%8.4g\n',CK(1), CK(2));
    else
        fprintf('Unfortunatly, Solutions 1 and 2 are beyond physical
Limitations\n');
    end
    if FKE(3) == 0 & FKE(4) == 0
        fprintf('Solution 3 and 4:\n  Theta 1 = %10.4g\t\t    Theta 1
= %10.4g\n  Theta 2 = %10.4g\t\t    Theta 2 = %10.4g\n  Theta 3 =
%10.4g\t\t    Theta 3 =
%10.4g\n',Output(1,3),Output(1,4),Output(2,3),Output(2,4),Output(3,3)
,Output(3,4));
        fprintf('  Theta 4 = %10.4g\t\t    Theta 4 = %10.4g\n  Theta
5 = %10.4g\t\t    Theta 5 = %10.4g\n  Theta 6 = %10.4g\t\t    Theta 6
=
%10.4g\n',Output(4,3),Output(4,4),Output(5,3),Output(5,4),Output(6,3)
,Output(6,4));
        fprintf('  Max Error = %8.4g\t\t    Max Error =
%8.4g\n',CK(3), CK(4));
    else
        fprintf('Unfortunatly, Solutions 3 and 4 are beyond physical
Limitations\n');
    end
    if FKE(5) == 0 & FKE(6) == 0
        fprintf('Solution 5 and 6:\n  Theta 1 = %10.4g\t\t    Theta 1
= %10.4g\n  Theta 2 = %10.4g\t\t    Theta 2 = %10.4g\n  Theta 3 =
%10.4g\t\t    Theta 3 =
%10.4g\n',Output(1,5),Output(1,6),Output(2,5),Output(2,6),Output(3,5)
,Output(3,6));
        fprintf('  Theta 4 = %10.4g\t\t    Theta 4 = %10.4g\n  Theta
5 = %10.4g\t\t    Theta 5 = %10.4g\n  Theta 6 = %10.4g\t\t    Theta 6
=
%10.4g\n',Output(4,5),Output(4,6),Output(5,5),Output(5,6),Output(6,5)
,Output(6,6));
        fprintf('  Max Error = %8.4g\t\t    Max Error =
%8.4g\n',CK(5), CK(6));
    else
        fprintf('Unfortunatly, Solutions 5 and 6 are beyond physical
```

```
Limitations\n');
    end
    if FKE(7) == 0 & FKE(8) == 0
        fprintf('Solution 7 and 8:\n  Theta 1 = %10.4g\t\t    Theta 1
= %10.4g\n  Theta 2 = %10.4g\t\t    Theta 2 = %10.4g\n  Theta 3 =
%10.4g\t\t    Theta 3 =
%10.4g\n',Output(1,7),Output(1,8),Output(2,7),Output(2,8),Output(3,7)
,Output(3,8));
        fprintf('  Theta 4 = %10.4g\t\t    Theta 4 = %10.4g\n  Theta
5 = %10.4g\t\t    Theta 5 = %10.4g\n  Theta 6 = %10.4g\t\t    Theta 6
=
%10.4g\n',Output(4,7),Output(4,8),Output(5,7),Output(5,8),Output(6,7)
,Output(6,8));
        fprintf('  Max Error = %8.4g\t\t    Max Error =
%8.4g\n',CK(7), CK(8));
    else
        fprintf('Unfortunatly, Solutions 7 and 8 are beyond physical
Limitations\n');
    end

end
```

10.6 Codes Validation Results

10.6.1 Validation Positions and Load cases

This technique is built into torque.m code for use with the Merlin 6-DOF robot. It was necessary to verify the functionality of the method against hand calculations prior to solving the optimization problems. The following section presents the code verification that was completed to ensure accurate results.

The first task was to quantify the torques due to the internal loads of the manipulator.

The manipulator weights are defined as follows:

Link 1(Transmission Cluster) weighs 288 lbs and is assumed to be centered 5 inches behind the shoulder axis.
Link 2 (inner arm and counter weights) weighs 240 lbs and is assumed to centered about the shoulder axis.
Link 3 (Outer arm) weighs 90 lbs and its weight is assumed to be centered 1 inch behind the elbow axis.
The wrist is assumed to weigh an additional 25 lbs and its weight is approximated as centered about the wrist center.

Torques are reported in a six-by-three matrix, where each of the rows (6) corresponds to a coordinate frame starting with 0 and going up to 5. It is assumed the external loads are known in the 6^{th} frame and therefore they are not necessary to compute. The columns correspond to the X, Y and Z axis of each coordinate frame.

With the arm angles at [0,0,0,0,-90,0] the torque matrix in Table 10.1 can be developed. After entering link lengths, the matrix is shown in Table 10.2.

Table 10.1 Torque Verification due to Internal Weights

i	X_i	Y_i	Z_i
0	$355lb*d_2$	$288lb*(-127mm) + 90lb*(a_2-25.4) + 25lb*(a_2+d_4)$	0
1	$355lb*d_2$	0	$-90lb*(a_2-25.4) - 25lb*(a_2+d_4)$
2	0	0	$-25lb*(d_4 – 25.4)$
3	0	$-25*d_4$	0
4	0	0	0
5	0	0	0

Table 10.2 Torque Verification due to Internal Weights Simplified

i	X_i	Y_i	Z_i
0	4.7709E5	1.0157E5	0
1	4.7709E5	0	-2.642E5
2	0	0	-3.854E4
3	0	-4.87E4	0
4	0	0	0
5	0	0	0

The computer code shows perfect agreement with the internal weight load case. Knowing the torques due to the internal loads, these can be subtracted from the final torque values to isolate the torques induced by external loads applied at the tool tip, as follows:

In the same configuration as before, assume a 45 N (10 lb) load is applied to frame 6, normal to the floor. The resulting torques are shown in Table 10.3 and Table 10.4.

Table 10.3 Torque Validation due to 10 lb Load in Z direction

i	X_i	Y_i	Z_i
0	-10lb*d_2	-10lb*(a_2+d_4)	0
1	-10lb*d_2	0	10lb*(a_2+d_4)
2	0	0	10lb*d_4
3	0	10lb*d_4	0
4	0	0	0
5	0	0	0

Table 10.4 Torque Validation due to 10 lb Load in Z direction Simplified

i	X_i	Y_i	Z_i
0	-1.3445E4	-3.9121E4	0
1	-1.3445E4	0	3.9121E4
2	0	0	1.9490E4
3	0	1.9490E4	0
4	0	0	0
5	0	0	0

Similarly, if a 10 lb load is applied in the Y direction, the resulting torque matrix is shown in Table 10.5 and Table 10.6.

Table 10.5 Torque Validation due to 10 lb Load in Y direction

i	X_i	Y_i	Z_i
0	-10lb*(d_1-d_6)	0	10lb*(a_2 + d_4)
1	10*d_6	10lb*(a_2 + d_4)	0
2	10*d_6	10lb*d_4	0
3	10*d_4	0	10lb*d_6
4	0	-10lb*d_6	0
5	10*d_6	0	0

Table 10.6 Torque Validation due to 10 lb Load in Y direction Simplified

i	X_i	Y_i	Z_i
0	-4.7387E4	0	3.9104E4
1	3.953E3	3.9104E4	0
2	3.953E3	1.9481E4	0
3	1.9481E4	0	3.953E3
4	0	-3.952E3	0
5	3.953E3	0	0

Based on these cases, the torque.m code is shown to properly compute the torques for each coordinate frame given both internal and external loads.

10.7 Data Correlation with Experimental Test

The OPTOTRAK writes all measured results to a proprietary format binary file. Using an included transformation program, the results were converted into three-dimensional data and written to ASCII text files named in such a way to correspond to the experiment. The following section details the formatting and order of the results.

Each regime of testing includes 243 individual experiments. These experiments correspond to twenty-seven locations with nine loads at each location. The loads are always applied in order from smallest to largest, reprinted from Table 4.3, in Table 10.7. Therefore, the first nine tests in any regime correspond to a complete series of loads and the tenth load begins another complete series of loads. The twenty-seven test locations are different for each regime and are presented again in Table 10.8, in robot coordinates, reprinted from Table 4.2.

Table 10.7 Experimental Loads

Number	Pounds	Newtons
1	0	0
2	5.3	24.1
3	10.3	46.8
4	15.3	69.5
5	20.3	92.3
6	23.7	107.7
7	50.0	227.3
8	76.3	346.8
9	100.3	455.9

Table 10.8 Experimental Data Collection Points

Point	Regime 1			Regime 2			Regime 3		
	X	Y	Z	X	Y	Z	X	Y	Z
1	22	-4	-10	15.5	12	-12	25	-24	4.5
2	22	0	-10	24.4	12	-12	12	-24	4.5
3	22	4	-10	33.3	12	-12	12	-24	-10

4	26	-4	-10	17	12	0	25	-24	-10
5	26	0	-10	25.5	12	0	20	-8.3	0
6	26	4	-10	34	12	0	25	16	4.5
7	30	-4	-10	24	12	12	12	16	4.5
8	30	0	-10	19.5	12	12	12	16	-10
9	30	4	-10	15	12	12	25	16	-10
10	22	-4	-6	15.5	0	12	25	-24	-10
11	22	0	-6	21	0	12	12	-24	-10
12	22	4	-6	27	0	12	25	16	-10
13	26	-4	-6	36	0	0	12	16	-10
14	26	0	-6	28.5	0	0	20	-8.3	**-9.5**
15	26	4	-6	21	0	0	25	-24	**-3**
16	30	-4	-6	19.5	0	-12	12	-24	**-2.3**
17	30	0	-6	27.3	0	-12	12	16	4.5
18	30	4	-6	35	0	-12	25	16	4.5
19	22	-4	-2	33	-12	-12	25	-24	4.5
20	22	0	-2	24.3	-12	-12	12	-24	4.5
21	22	4	-2	15.5	-12	-12	12	16	4.5
22	26	-4	-2	17	-12	0	25	16	4.5
23	26	0	-2	25.5	-12	0	20	-8.3	0
24	26	4	-2	34	-12	0	25	16	-10
25	30	-4	-2	24	-12	12	12	16	-10
26	30	0	-2	17	-12	12	12	-24	-10
27	30	4	-2	10	-12	12	25	-24	-10

** Red Text indicates the 445 N test failed

** Bold text indicates a point is moved slightly to allow loading

Knowing the physical position in robot coordinates and the loads applied does not determine which experimental results file corresponds to each test. This correlation allows the OPTOTRAK's result files to be referenced back to the physical experiment. Figure Table 10.9 presents the relationships between the physical location in the workspace and the actual sequence of testing. The location shown in Figure Table 10.9 is synonymous with the location in the prior table. Regimes 1 and 2 share the same order of testing, while regime 3 is quite a bit different, as shown. The *Zero Load* and *Max Load* columns correspond to the number of the test for the two extreme tests for any given location. Furthermore, the third regime includes details about the direction of the applied load for each test. All loads in the first and second regime were directed vertically.

Table 10.9 Workspace Location and Testing Sequence

	Regime 1 & 2		Regime 3		
Location	Zero Load	Max Load	Zero Load	Max Load	Load Direction
1	1	9	1	9	Vertical
2	10	18	10	18	Vertical
3	19	27	46	54	Vertical
4	28	36	55	63	Vertical
5	37	45	37	45	Vertical
6	46	54	28	36	Vertical
7	55	63	19	27	Vertical
8	64	72	73	81	Vertical
9	73	81	64	72	Vertical
10	82	90	127	135	Backwards
11	91	99	136	145	Backwards
12	100	108	154	162	Backwards
13	109	117	145	153	Backwards
14	118	126	118	127	Backwards
15	127	135	82	90	Backwards
16	136	144	91	99	Backwards
17	145	153	100	108	Backwards
18	154	162	109	117	Backwards
19	163	171	163	171	Sideways
20	172	180	172	180	Sideways
21	181	189	190	198	Sideways
22	190	198	181	189	Sideways
23	199	207	199	207	Sideways
24	208	216	235	243	Sideways
25	217	225	226	234	Sideways
26	226	234	208	216	Sideways
27	235	243	212	225	Sideways

All OPTOTRAK results are written to ASCII test files, and the first page of the results from the first test of the first regime is shown in Table 10.10. Each of the 243 files for a regime have four hundred entries, corresponding to ten seconds of data at twenty Hz. The *Frame* number identifies which of the four hundred samples is being shown, while the *Item* corresponds to the sensor number. The locations of all sensors are shown in Section 4.6 . Sensors five and six were installed on the arm but not required beyond initial calibration experiments. The last three columns, *S1, S2,* and *S3* correspond to the X, Y and Z location of the sensor in OPTOTRAK coordinates, respectively.

Table 10.10 Example OPTOTRAK results

Frame	Item	S1	S2	S3
0	1	-266.604	-359.143	-3647.888
	2	-187.036	-358.548	-3649.707
	3	-186.491	-277.915	-3647.759
	4	-266.615	-277.941	-3646.196
	5	422.312	-118.490	-3729.215
	6	5.216	13.097	-4045.277
	7	18.006	-430.457	-3702.506
	8	557.204	-385.782	-3703.422
	9	28.156	-205.804	-3694.963
	10	541.360	-270.402	-3698.996
	11	-959.299	-847.553	-3806.592
	12	-954.700	23.671	-3785.868
	13	-460.921	-413.846	-4186.149
	14	-928.243	-410.161	-4177.549
1	1	-266.616	-359.156	-3647.936
	2	-187.046	-358.556	-3649.738
	3	-186.498	-277.927	-3647.838
	4	-266.621	-277.945	-3646.224
	5	422.317	-118.494	-3729.212
	6	5.212	13.104	-4045.228
	7	18.006	-430.478	-3702.577
	8	557.210	-385.786	-3703.453
	9	28.151	-205.814	-3694.998
	10	541.356	-270.393	-3698.953
	11	-959.323	-847.583	-3806.671
	12	-954.670	23.682	-3785.755
	13	-460.930	-413.854	-4186.195
	14	-928.203	-410.124	-4177.373
2	1	-266.605	-359.150	-3647.911
	2	-187.039	-358.555	-3649.733
	3	-186.488	-277.919	-3647.778
	4	-266.619	-277.945	-3646.238
	5	422.326	-118.490	-3729.278
	6	5.212	13.115	-4045.072
	7	18.007	-430.459	-3702.502
	8	557.196	-385.778	-3703.422
	9	28.150	-205.813	-3694.992
	10	541.347	-270.389	-3698.932
	11	-959.333	-847.592	-3806.676
	12	-954.682	23.682	-3785.772
	13	-460.935	-413.859	-4186.214
	14	-928.206	-410.118	-4177.337
3	1	-266.602	-359.138	-3647.862
	2	-187.043	-358.563	-3649.763
	3	-186.487	-277.902	-3647.708
	4	-266.614	-277.942	-3646.206
	5	422.320	-118.486	-3729.219
	6	5.216	13.115	-4045.109
	7	17.982	-430.441	-3702.434
	8	557.191	-385.774	-3703.412
	9	28.151	-205.797	-3694.924
	10	541.344	-270.387	-3698.938
	11	-959.295	-847.549	-3806.564
	12	-954.681	23.684	-3785.784
	13	-460.933	-413.859	-4186.224
	14	-928.203	-410.119	-4177.346

In summary, this section provides the information to relate a specific data sample to the load and location in the workspace. Furthermore, the layout of the data files is presented.

Vita

Mark William Abbott was born on May 20, 1977 in Poughkeepsie, New York. Mark's family moved six times through his childhood before he completed high school at Bloomingdale Senior High in Valrico, Florida. At an early point in his life, Mark was given a box of Legos, beginning the journey that led him to attend Tennessee Technological University in Cookeville, Tennessee to pursue a Bachelors degree in Mechanical Engineering. Upon completion of his undergraduate work in May 2000, Mark married Allison Westbrook, and they went on to attend Virginia Polytechnic Institute and State University in Blacksburg, Virginia where Mark completed his Masters degree in Mechanical Engineering in May of 2002. After graduation, Mark and Allison will relocate to Houston, Texas to start work for ExxonMobil.