DELINEATION OF IN-VITRO SPINAL KINETICS USING A ROBOTICS-BASED TESTING SYSTEM

by

Amy L. Loveless, M.S.

B.S.E. in Bioengineering, Arizona State University, 2001

Submitted to the Graduate Faculty of

the School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science in Bioengineering

University of Pittsburgh

2003

UNIVERSITY OF PITTSBURGH

SCHOOL OF ENGINEERING

This thesis was presented

by

Amy L. Loveless

It was defended on

July 22, 2003

and approved by

Rakié Cham, Ph.D., Department of Bioengineering

James D. Kang, MD, Department of Orthopaedic Surgery

Patrick J. Smolinski, Ph.D., Department of Mechanical Engineering

Thesis Advisor: Lars G. Gilbertson, Ph.D., Department of Bioengineering

DELINEATION OF IN-VITRO SPINAL KINETICS USING A ROBOTICS-BASED TESTING SYSTEM

Amy L. Loveless, M.S.

University of Pittsburgh, 2003

Delineation of the load-displacement characteristics of osteoligamentous spinal specimens has become fundamental to the investigation of spinal biomechanics. Traditionally, in-vitro kinetic parameters of the spine have been obtained through flexibility tests employing open or closed loop "load control" methods, or stiffness tests employing "displacement control" methods—each control method having attendant advantages and disadvantages. On the other hand, the combination load control and displacement control methods into a new, "hybrid control" method have advantages over load control or displacement control alone. Further, physical evidence such as presence of certain receptors suggests that the human body may employ a type of hybrid control method in the control of spinal movements.

In the present study, a robotics-based spine testing system with hybrid control was developed to delineate the in-vitro kinetics of lumbar spine specimens. The testing system was validated experimentally using a physical rigid-body-spring model of a spine specimen, as well as analytically by computer simulations in Matlab. For systematic study, the two components making up a hybrid control algorithm were analyzed separately: the outer "displacement control" loop, and the inner "load control" loop. The outer loop applies a rotation (e.g.,

flexion/extension) to the specimen, while the inner loop minimizes unwanted coupled forces (e.g., anterior/posterior shear and axial tension/compression).

The performance of existing standard hybrid control algorithms was tested in terms of a number of parameters, including peak force, work done to a specimen, and number of iterations. Based on these tests, a number of proposed changes to improve algorithm performance were identified. Updating the user-defined center of rotation (COR) to reflect a specimen's COR was found to improve performance of the displacement control part of the hybrid control algorithm, while using a more completely populated stiffness matrix improved performance of the load control part. The re-combination of the displacement control and load control loops into the fully constituted hybrid control algorithm revealed interesting interactions between these control components that suggest a basis for spinal dysfunction.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Lars G. Gilbertson, for all his help and guidance over the past two years. He was always willing to answer my questions and steer me in the right direction (and take people from the lab to Peter's for a much needed break!). I would also like to thank Dr. Pat Smolinski for ALL his help with simulation issues. I want to thank Kevin Bell. He was one of the first friends I made in Pittsburgh and we worked very closely together on the robot (good old Stuart). He, along with the rest of the Ferguson Lab, made Pittsburgh an enjoyable place to work and live. And last, but certainly not least, I want to thank my parents, Chuck and Wanda Loveless, and my fiancé, John Arthur, for listening to all my whining and complaining over the last two years. I guess now I'll just have to find something else to whine and complain about. I would have made my thanks much longer, but you all know I'm not good with the mushy stuff. I trust that you know how much I appreciate everyone without having to read it.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

ABBREVIATIONS

TCS = tool coordinate system

UFS = universal force-moment sensor

COORDINATE SYSTEMS

XYZ = global coordinate system

$(xyz)_0$ = local coordinate system of moveable rigid body M at position 0, before rigid body rotation/translation

$(xyz)_1$ = local coordinate system of moveable rigid body M at position 1, after rigid body rotation/translation

$(xyz)_{TCS0}$ = tool coordinate system at position 0

$(xyz)_{TCS1}$ = tool coordinate system at position 1

$(xyz)_{i0}$ = local coordinate system of node i at position 0

$(xyz)_{i1}$ = local coordinate system of node i at position 1

$(xyz)_j$ = local coordinate system of node j (fixed)

ϕ = incremental rotation of rigid body M about COR

dx, dy = rigid body translation of moveable rigid body M

θ_G = orientation of $(xyz)_0$ with respect to XYZ

θ_{COR} = orientation of $(xyz)_{TCS0}$ with respect to XYZ

θ_i = orientation of $(xyz)_{i0}$ with respect to $(xyz)_0$ and $(xyz)_{i1}$ with respect to $(xyz)_1$

θ_j = orientation of $(xyz)_j$ with respect to XYZ

P_{X0}, P_{Y0} = position of $(xyz)_0$ with respect to XYZ

COX_X, COR_Y = position of $(xyz)_{TCS0}$ (and $(xyz)_{TCS1}$) with respect to XYZ

TRANSFORMATIONS FOR RIGID BODY ANALYSIS

T_G^0 = transformation of $(xyz)_0$ with respect to XYZ

T_G^{TCS0} = transformation of $(xyz)_{TCS0}$ with respect to XYZ

T_{TCS0}^0 = transformation of $(xyz)_0$ with respect to $(xyz)_{TCS0}$

T_{TCS0}^{TCS1} = transformation of $(xyz)_{TCS1}$ with respect to $(xyz)_{TCS0}$

T_G^{TCS1} = transformation of $(xyz)_{TCS1}$ with respect to XYZ

T_{TCS1}^1 = transformation of $(xyz)_1$ with respect to $(xyz)_{TCS1}$

T_G^1 = transformation of $(xyz)_1$ with respect to XYZ

T_0^{i0} = transformation of $(xyz)_{i0}$ with respect to $(xyz)_0$

T_G^{i0} = transformation of $(xyz)_{i0}$ with respect to XYZ

T_1^{i1} = transformation of $(xyz)_{i1}$ with respect to $(xyz)_1$

T_G^{i1} = transformation of $(xyz)_{i1}$ with respect to XYZ

T_G^j = transformation of $(xyz)_j$ with respect to XYZ

T_{i0}^j = transformation of $(xyz)_j$ with respect to $(xyz)_{i0}$

T_{i1}^{j} = transformation of $(xyz)_j$ with respect to $(xyz)_{i1}$

TRANSFORMATIONS FOR ROBOTIC SYSTEM

T_G^{TCS} = transformation of TCS with respect to the base (global) robot coordinate system

T_{UFS}^{TCS} = transformation of TCS with respect to UFS coordinate system

T_G^{UFS} = transformation of UFS coordinate system with respect to base coordinate system

T_{UFS}^{i} = transformation of point of interest on superior vertebra with respect to UFS coordinate system

T_G^i = transformation of point of interest with respect to base coordinate system

T_0^i = transformation of point of interest with respect to centroid of superior vertebra

T_G^0 = transformation of centroid of superior vertebra with respect to base coordinate system

T_{UFS}^{0} = transformation of centroid of superior vertebra with respect to UFS coordinate system

1.0 INTRODUCTION

Delineation of the load-displacement characteristics of osteoligamentous spinal specimens has become fundamental to the investigation of spinal biomechanics and is key in understanding the effects of spinal pathologies and their clinical treatments. In the following sections, the basis for hybrid control as a testing algorithm is presented, as well as an examination of the two distinct loops of the general hybrid control algorithm our lab has chosen to employ.

1.1 Overview of Clinical Problems of Spine

Spinal disorders arising from injury, degeneration, aging or other causes is an expansive and expensive problem. Back pain is the second most prevalent reason for a physician visit, with nearly 13 million visits made annually specifically because of low back pain.[1] An estimated $20 billion is spent annually in medical expenses directly related to low back pain.[2] Treatments are far reaching, from a period of rest followed by a return to normal activities to chiropractic visits to surgery. For possible future clinical treatments of degenerative disc disease, research is being done to test the effectiveness of gene therapy.[3]

1.2 Spinal "Stability" vs. "Instability"

With severe degeneration or injury, one or more spinal segments can become unstable. There is no consensus on the definition of clinical instability, but many have offered their

opinions. Wyke described instability as abnormally large intervertebral motions that result in deformation to neural elements or abnormal deformations of the segment's soft tissue (as cited in Panjabi[4]), while White and Panjabi [5] define it more specifically as "the loss of the ability of the spine under physiologic loads to maintain its pattern of displacement so that there is no initial or additional neurological deficit, no major deformity, and no incapacitating pain".[5] Panjabi[4] conceptualized the spinal stabilizing system as consisting of three subsystems: passive (osteoligamentous spine), active (muscles and tendons), and control (neural elements and central nervous system). It has further been hypothesized that the neural control subsystem receives both position feedback and force feedback from various transducers located within the ligaments, tendons, and muscles, hence the spine may operate in some form of hybrid control mode.

1.3 In-Vitro Studies of Spinal Kinetics

Delineation of the load-displacement characteristics of osteoligamentous spinal specimens has become fundamental to the investigation of the biomechanics of the spine. Traditionally, in-vitro kinetic parameters of the spine have been obtained through biomechanical tests that are based on either the "flexibility method" or the "stiffness method".[6] In flexibility tests, loads (i.e., forces and moments) are applied singly[7,8] or in combination[9] to the free end of a spinal specimen and the resulting unconstrained three-dimensional displacements (i.e., translations and rotations) are measured. In stiffness tests, displacements are applied and the resulting loads are measured.[10,11] Kinetic parameters obtainable by these types of tests include specimen flexibility/stiffness coefficients useful for characterizing the biomechanics of the intact, injured, and stabilized spine.

An impetus behind the use of hybrid control for testing spinal kinetics is the controversy surrounding use of load-control versus displacement control methods for the biomechanical testing of spinal specimens[12]. The underlying hypothesis of work done previously[13] was that a combination of load control and displacement control methods within a hybrid control method would offer advantages over either load control or displacement control methods alone for the delineation of the highly nonlinear spinal kinetics.

1.3.1 Controversy: Load Control vs. Displacement Control

In addition to testing machines[7,8,14-24] and devices for measuring loads[25-27] and displacements[16,28-31], in vitro biomechanical testing of the spine requires implementation of a control method to govern the application of loads/displacements to a specimen. Flexibility tests employ open or closed loop "load control" methods, while stiffness tests employ "displacement control" methods. The relative advantages and disadvantages of load control and displacement control methods for the biomechanical testing of spinal specimens have been discussed by Goel et al.[12]. From a control perspective, it is apparent that load control is less appropriate than displacement control in low stiffness regions of the load-displacement curve such as the neutral zone (NZ) because large changes in displacement can occur with little or no change in applied load (**Figure 1**). On the other hand, displacement control is less appropriate than load control in high stiffness regions such as the elastic zone (EZ) because large changes in load can be produced by small changes in applied displacement. For the in-vitro biomechanical testing of spinal specimens, therefore, load control and displacement control methods are complementary (in that one method or the other is viewed as being more applicable in certain regions of the load-displacement curve).

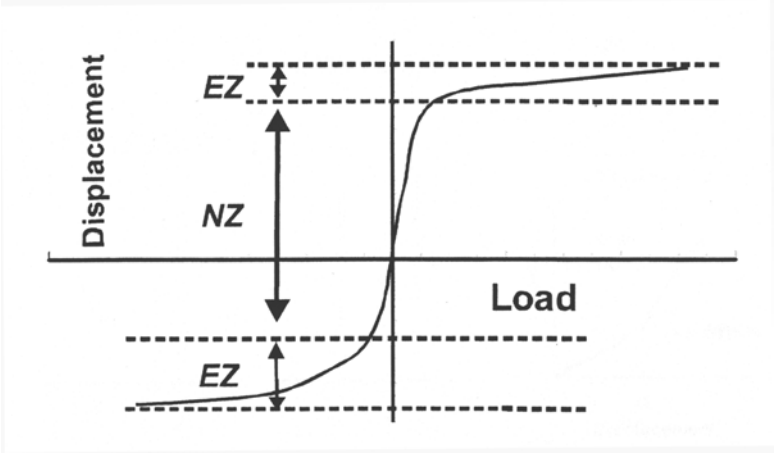


Figure 1 Idealized load-displacement curve

1.3.2 Hybrid Control

Hybrid control methods are a class of control algorithms that would appear to offer a potentially useful alternative to load control or displacement control for the biomechanical testing of spinal specimens. A hybrid control method combines aspects of load control and displacement control methods to achieve a new, "hybrid" method that is better suited to a particular application than either load control or displacement control alone. In the classical robotics literature, a rigorous formulation of the hybrid force/position control method has been performed by Raibert and Craig[32]. Hybrid control methods have been applied previously to the multi-DOF (degree-of-freedom) biomechanical testing of musculoskeletal joints (such as the knee) using a robotic/UFS (universal force-moment sensor) testing system[33-36]. Of particular interest are the hybrid control algorithms described by Fujie et al.[33] and Doehring [13] that enable the inherently position-controlled robot to achieve specified load targets in an iterative manner through incrementally applied displacements. At each position along the path of motion, the algorithm evaluates the relation between the change in specimen position (i.e., displacement) and the change in UFS-measured loads, and uses this relation to plan the application of the next

incremental displacement to achieve specified load targets. Control is thus based on the stiffness of the specimen, and because the stiffness estimates are regularly updated along the path of motion, this control algorithm appears to be well suited for delineation of the highly nonlinear in vitro kinetics of the spine throughout its entire range-of-motion.

2.0 BACKGROUND

2.1 Structure of Osteoligamentous Lumbar Spine

The function of the osteoligamentous spine is threefold: "(1) transfer the weights and the resultant bending moments of the head, trunk and any weights being lifted to the pelvis, (2) allow sufficient physiologic motions between these three body parts and (3) protect the spinal cord from injury".[5] The structure of a single functional spinal unit (FSU) is shown in **Figure 2**. The two bony vertebral bodies are separated by an intervertebral disc. The nucleus pulposus is the gelatinous center of the disc. The anulus fibrosus contains the nucleus with concentric layers of collagen.

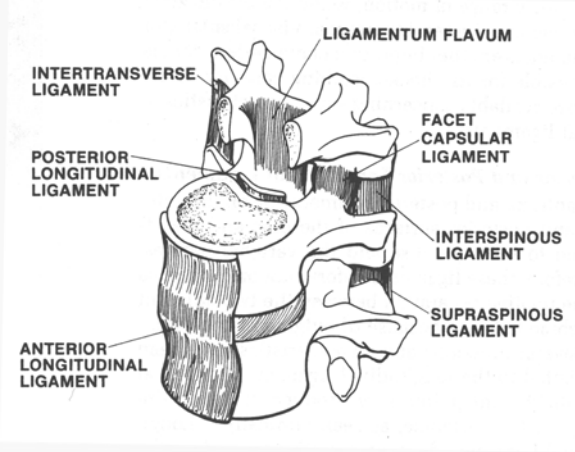


Figure 2 Osteoligamentous functional spinal unit (FSU)

3-dimensional joint motion is generally described as a combination of translations and rotations along and about a set of axes. The ISB recommends defining a nonorthogonal joint coordinate system based on the work of Grood and Suntay, in which two of the axes are defined using anatomical landmarks and the third "floating" axis is perpendicular to the first two.[37-39]

As applied to spinal motion segments, the e_1 axis is parallel to a line connecting similar landmarks on the bases of the right and left pedicles and points to the right, the e_3 axis passes through the centers of the upper and lower endplates and points cephalad and the e_2 axis is perpendicular to e_1 and e_3 (**Figure 3**). Flexion/extension is about the e_1 axis, left and right lateral bending is about the e_2 axis and left and right axial rotation is about the e_3 axis. The ISB also recognizes Panjabi's coordinate system. As shown in **Figure 4**, the x axis points left, the y axis points cephalad and the z axis points anterior. Flexion/extension is about the x axis, lateral bending is about the z axis and axial rotation is about the y axis. Our lab has chosen Panjabi's coordinate system to report data in for ease of comparison with other studies.

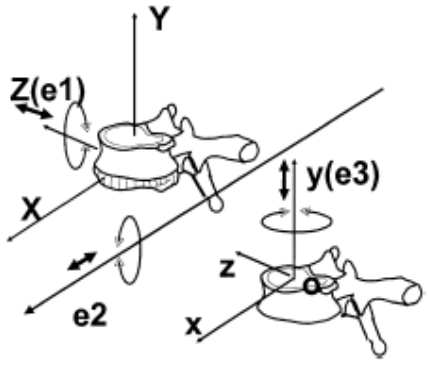


Figure 3 ISB spine joint coordinate system

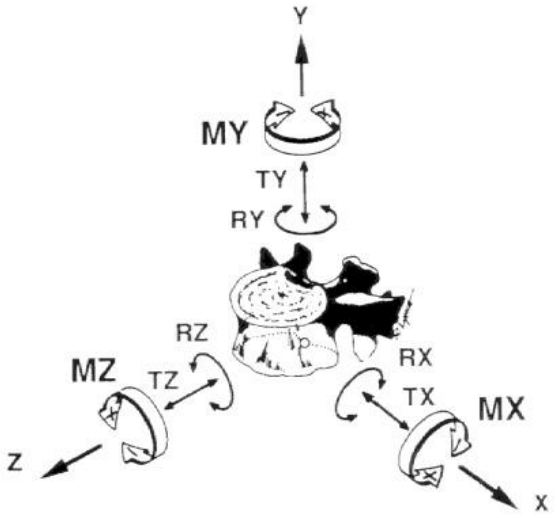


Figure 4 Panjabi spine coordinate system

2.2 Application of Hybrid Control to In-Vitro Biomechanical Testing

In the following paragraphs, representative limitations of displacement control and load control methods are contrasted with some of the apparent advantages of hybrid control methods.

2.2.1 Displacement Control Loop

A recognized limitation of displacement control methods for the biomechanical testing of spinal specimens is that rotational displacements are often prescribed about a fixed axis that is not the specimen's preferred axis of rotation—thereby resulting in large, "unphysiological" coupled loads[40]. A specimen's preferred axis of rotation is, of course, not known *a priori*, and a further complication is that the location of the preferred axis is not constant but changes throughout the path of passive motion. The hybrid control algorithm as described previously[13] mitigates this problem by permitting an adaptive, "floating" axis of rotation, as follows. The flexion/extension rotation increments applied within the applied rotation loop of the hybrid

control algorithm are prescribed about an axis perpendicular to the sagittal plane that passes through the user-specified center of rotation (COR), or the origin of the robot's tool coordinate system. If the user-chosen COR is not the specimen's preferred COR, the rotation does not result in the desired pure moment. Any coupled sagittal plane forces arising from an incremental rotation about this axis are relieved within the force minimization subroutine of hybrid control by incremental translations of the end-effector—automatically changing the location of the COR *globally*. Thus, following each applied rotational displacement increment, the axis of applied rotation moves incrementally to a position wherein residual coupled sagittal plane forces are minimized. The user-defined COR is not allowed to move with respect to the specimen's coordinate system, therefore, the COR is *locally* fixed.

2.2.2 Load Control Loop

A recognized limitation of load control methods for the biomechanical testing of spinal specimens is the difficulty of maintaining testing conditions in the neutral zone because the displacements can change with no change in the load input.[12] When *open-loop* load control tests are performed, the neutral zone is defined by the resting position of the specimen after the application of a series of loads in the degree-of-freedom of interest[6]— thus kinetics of the specimen within the neutral zone are not actually delineated. When *closed-loop* load control tests are performed, low stiffness of a specimen can put a high demand on the response characteristics of the control system — requiring the testing machine to respond to load control commands quickly, over long distances.[21] Unanticipated delays or overshoot are potential sources of load artifact generated by the response characteristics of a testing machine in a load control mode.[21] The hybrid control method described previously[13] is based on the *stiffness* of

the specimen, and because the stiffness estimates are regularly updated along the path of motion, the robotic/UFS testing system with hybrid control is able to adapt to the extreme range of stiffnesses presented by the highly nonlinear FSU—from near-zero stiffness in the "neutral zone" to high stiffness with facet joint contact and at the extremes of the "elastic zones." To simplify calculation of the local specimen stiffness matrix, only the diagonal terms of the matrix are calculated; the off-diagonal terms are set to zero. Delineation of the load-displacement response of specimens can be achieved throughout the entire flexion/extension range-of-motion—including the region of least stiffness or "neutral zone," the regions of increasing stiffness or "elastic zones," and the *transition* between these regions.

As mentioned above, the user-defined COR remains locally fixed. However, clinical data shows that the COR moves within the specimen during flexion/extension[41]. The amount of movement of the COR depends on the degree of flexion/extension and the extent of disc degeneration. An algorithm that does not account for this requires more iterations to minimize force during load control because the peak force may be higher than if the COR were allowed to move locally. In addition, setting the off-diagonal terms of the stiffness (flexibility) matrix to zero ignores the coupled stiffness terms. This attributes all the change in force in a certain direction to the translation in that direction, but the specimen is a highly complex, coupled system. To investigate the possibility of improving the current hybrid control algorithm, three specific aims will be accomplished.

3.0 SPECIFIC AIMS AND HYPOTHESES

3.1 Specific Aim 1

Develop analytical testing platform. This platform can be applied to testing control algorithms using well-defined rigid body-spring model of a lumbar functional spinal unit (FSU). Develop experimental testing platform. This platform may be used to experimentally test spinal specimens.

3.2 Specific Aim 2

Apply these platforms to the development of testing of new control methods. New control methods consist of changes to both the displacement control and load control loops.

3.2.1 Specific Aim 2a

To improve the displacement control loop, two methods of updating the user-defined COR are proposed. To calculate the preferred COR, three methods found in the literature will be investigated: Spiegelman and Woo[42], Crisco et al.[43] and Challis[44]. The first proposed method of updating the COR is a post hoc update in which the preferred COR will be calculated and stored for replay during the next flexion/extension cycle. The second proposed method is using feedback to update the COR. The preferred COR will be calculated every n degrees and updated for use during the next $n\phi$ degrees. It is hypothesized that allowing the COR to move locally will decrease the force resulting from rotation about a COR other than the preferred one, thereby reducing the number of iterations required to minimize force.

3.2.2 Specific Aim 2b

To improve the load control loop, the stiffness matrix will be fully populated. Three methods of calculating the full stiffness matrix are proposed to accomplish this. The first method is to perturb the rigid body in two orthogonal directions at each position, calculating all four terms in the 2x2 stiffness matrix. The second method is to limit the translations to the force minimized position in a stairstep fashion, calculating three terms in the 2x2 stiffness matrix at each position. The third method is a combination of the first two: three terms in the 2x2 matrix are calculated at each position by limiting the translations, while the fourth term is found by perturbing the rigid body after translating it. It is hypothesized that using a fully populated the stiffness matrix to calculate the translation necessary to minimize force will reduce the number of iterations required to reach the force minimized position and provide a more accurate description of specimen stiffness. The proposed methods of calculating the full stiffness matrix were based on the knowledge that the full matrix could not be calculated using one translation or perturbation, covered in more detail in section 6.2, and the hypothesis that the values of the terms in the matrix may be closely approximated using small perturbations or small translations.

4.0 DEVELOPMENT OF ANALYTICAL PLATFORM

The rigid body-spring model used experimentally is shown in **Figure 5**. Even though our rigid body-spring model is quite simple, it still exhibits complex, nonlinear behavior as a real specimen does. It was shown previously[13] that the model exhibits load-displacement characteristics with distinct neutral and elastic zones, analogous to a lumbar FSU. This thesis shows the nonlinearities present in load and stiffness data for our model and how the hybrid control algorithm handles such nonlinearities. Friis[45] and Wilke[46] are developing more sophisticated lumbar spine models. Our rigid body-spring model is used to validate experimental protocols. An analytical solution to the rigid body-spring model is thus needed to validate experimental results. This platform also provides a framework for formulating new clinical hypotheses, for example, a specimen with a painful (or injured) structure may minimize something other than force after the displacement control loop. Perhaps the specimen's natural reaction is to minimize the work done. To develop the analytical solution, a general rigid body-spring model consisting of two rigid bodies and one spring is presented.

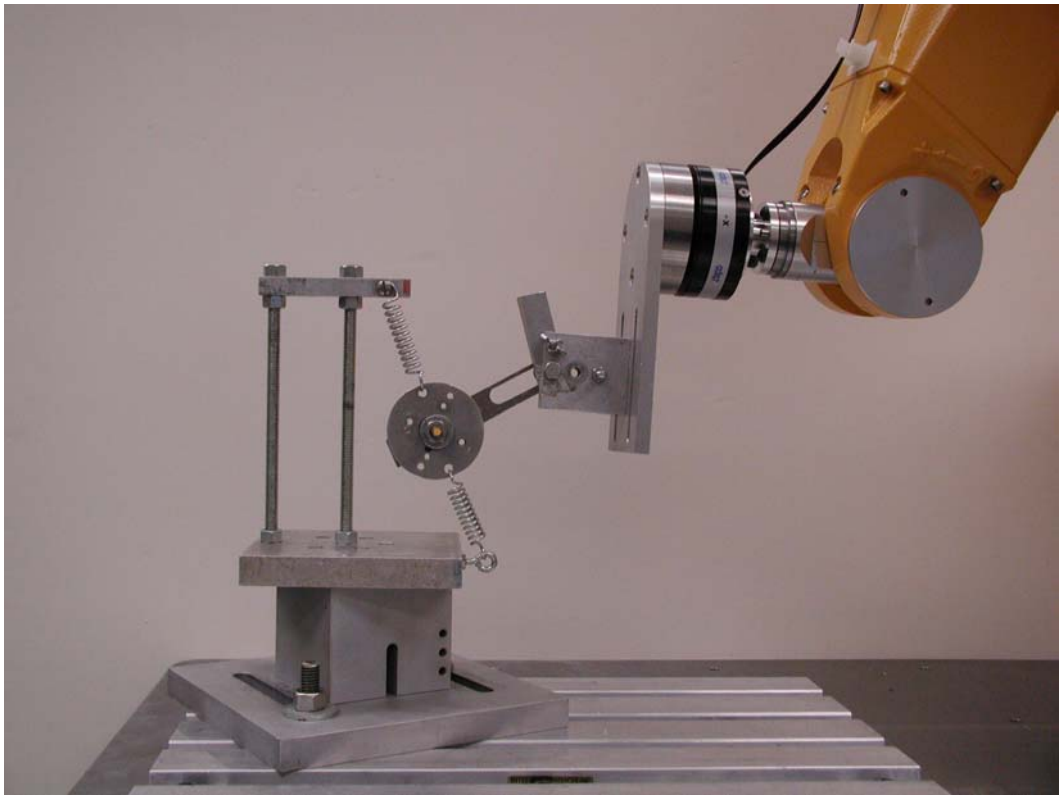


Figure 5 Rigid body-spring model

4.1 Description of General Rigid Body-Spring Model

Suppose there is a spring, spring i, connecting two rigid bodies (**Figure 6**). One rigid body (rigid body M) is allowed to move globally and the other (rigid body F) is fixed in space. As rigid body M rotates and translates away from its equilibrium position, forces and moments due to spring i are created. We confine the rigid body-spring model to planar motion, so there are three degrees of freedom: a rotation about the z axis and two translations in the xy-plane. In order to fully describe the model's kinematics and kinetics, three points are defined. The origin of a local coordinate system, xyz, is defined on rigid body M at some point P. One end of spring i is connected to rigid body M at node i, the origin of coordinate system $(xyz)_i$. The

other end of the spring is connected to rigid body F at node j, the origin of coordinate system $(xyz)_j$. The homogeneous transformation describing the position and orientation of $(xyz)_i$ with respect to xyz is constant throughout rigid body motion. If the body is not rigid, then the transformation is not constant. In this case, individual nodes must be tracked or deformable body principles must be applied to correct for rigid body deformation. Point P is the same thing as a node, but for clarity later, it is differentiated from the other nodes by calling it a point.

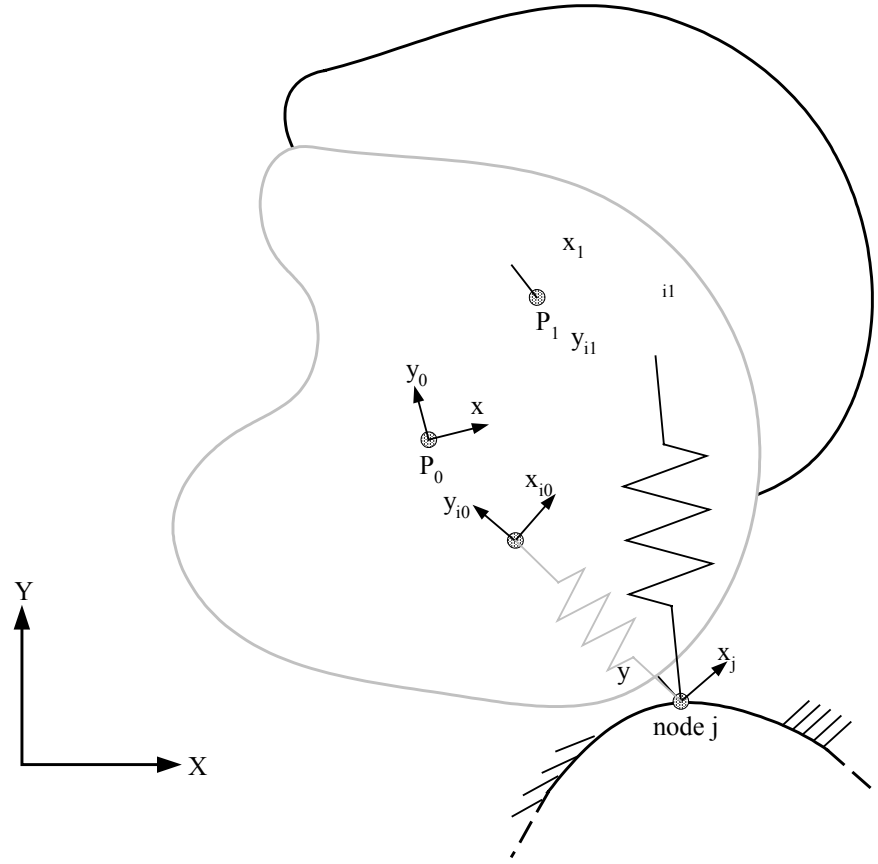


Figure 6 General rigid body-spring model

The general rigid body-spring model can be likened to a lumbar FSU. The rigid body M represents the superior vertebra and rigid body F represents the inferior vertebra. Point P

represents the center of the superior vertebra, node i represents a point on the superior insertion site of a ligament, node j represents a point on the inferior insertion site of the ligament and spring i loosely represents the ligament itself. More complex representations of ligaments are available in the literature, but our interest lies in developing the general rigid body-spring model kinematics for an n (linear elastic) spring system, leading to analytical expressions for the loads and stiffness coefficients developed during general rigid body motion. Additional nodes on either vertebra may be defined. For example, suppose we want to define more nodes on the insertion sites of a ligament as a better approximation of ligament deformation. The only restriction on defining nodes is that they are confined to the vertebra they are measured with respect to, i.e., nodes on the superior vertebra must be measured with respect to the superior vertebra's coordinate system because of the rigid body assumption.

4.2 General Closed Form Solution

Movement of nodes, including point P, and all loads are referred to the global coordinate system for purposes of simulation. Nodal displacements and loads may be reported in any coordinate system, for example, the rigid body's local coordinate system, as is done experimentally. Because the coordinate system set at the COR is will be allowed to move both locally and globally (discussed in later sections), loads and displacements should not be reported in this coordinate system. To describe rigid body motion and the resulting loads, several homogeneous transformations must be known. In the following transformations, the subscript is the coordinate system that the superscript coordinate system is measured with respect to, for example, T_A^B is the transformation of frame B with respect to frame A. Also, the convention

$T_A^B = \begin{bmatrix} & [R] & & [d] \\ 0 & 0 & 0 & 1 \end{bmatrix}$ is used, where $[R]$ is the rotation matrix describing the orientation of frame B with respect to frame A, $[d]$ is the position vector describing the distance from the origin of frame A to the origin of frame B measured in frame A coordinates and the row vector $[0 \quad 0 \quad 0 \quad 1]$ is added for mathematical convenience.

4.2.1 Homogeneous Transformation of $(xyz)_0$ with Respect to XYZ

At initial position 0, before rigid body motion, point P is denoted P_0.

$$T_G^0 = \begin{bmatrix} c_G & -s_G & 0 & P_{0X} \\ s_G & c_G & 0 & P_{0Y} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where (P_{0X}, P_{0Y}) is the initial global position of P_0 and θ_G is the initial orientation of $(xyz)_0$ with respect to XYZ (**Figure 7**).

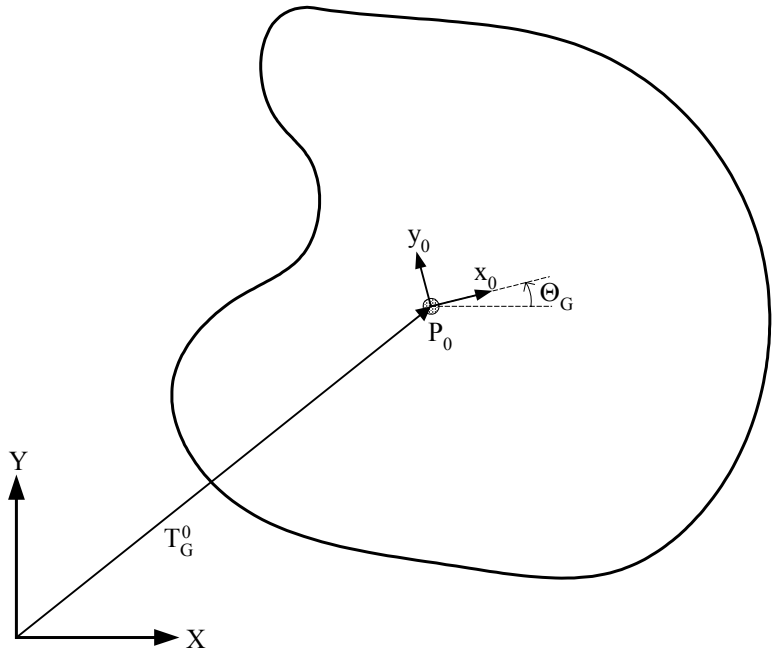


Figure 7 Homogeneous transformation of $(xyz)_0$ with respect to XYZ

4.2.2 Homogeneous Transformation of $(xyz)_{TCS0}$ with Respect to XYZ

Experimentally, the COR is the origin of the robot's tool coordinate system (TCS). For development of the general rigid body-spring model, the coordinate system $(xyz)_{COR}$ is used interchangeably with $(xyz)_{TCS}$. At position 0, the COR is denoted COR_0.

$$T_G^{TCS0} = \begin{bmatrix} c_{COR} & -s_{COR} & 0 & COR_{0X} \\ s_{COR} & c_{COR} & 0 & COR_{0Y} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where (COR_{0X}, COR_{0Y}) is the initial global position of COR_0 and θ_{COR} is the initial orientation of $(xyz)_{TCS0}$ with respect to XYZ (**Figure 8**).

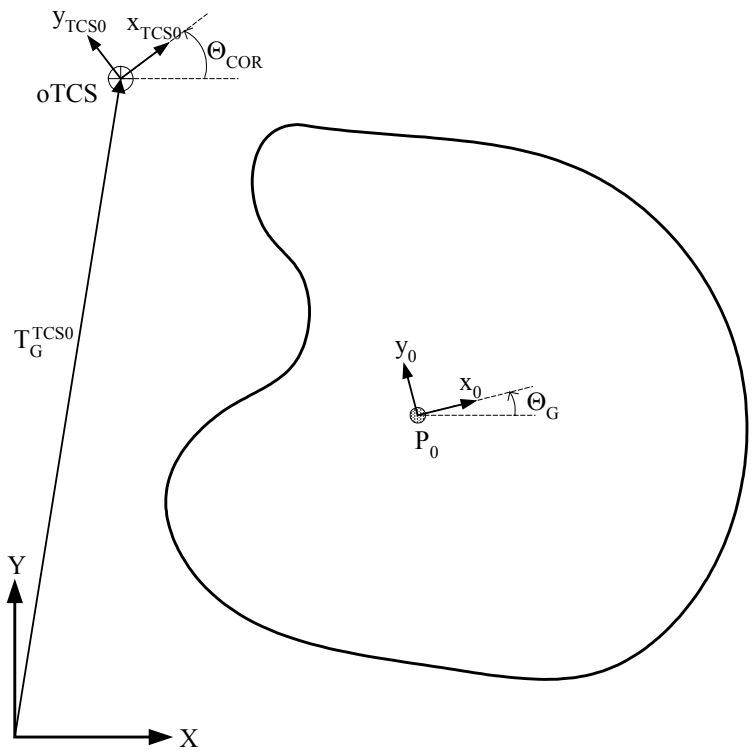


Figure 8 Homogeneous transformation of $(xyz)_{TCS0}$ with respect to XYZ

4.2.3 Homogeneous Transformation of $(xyz)_0$ with Respect to $(xyz)_{TCS0}$

$$T_{TCS0}^{0} = \left(T_G^{TCS0}\right)^{-1} T_G^0$$

See **Figure 9**.

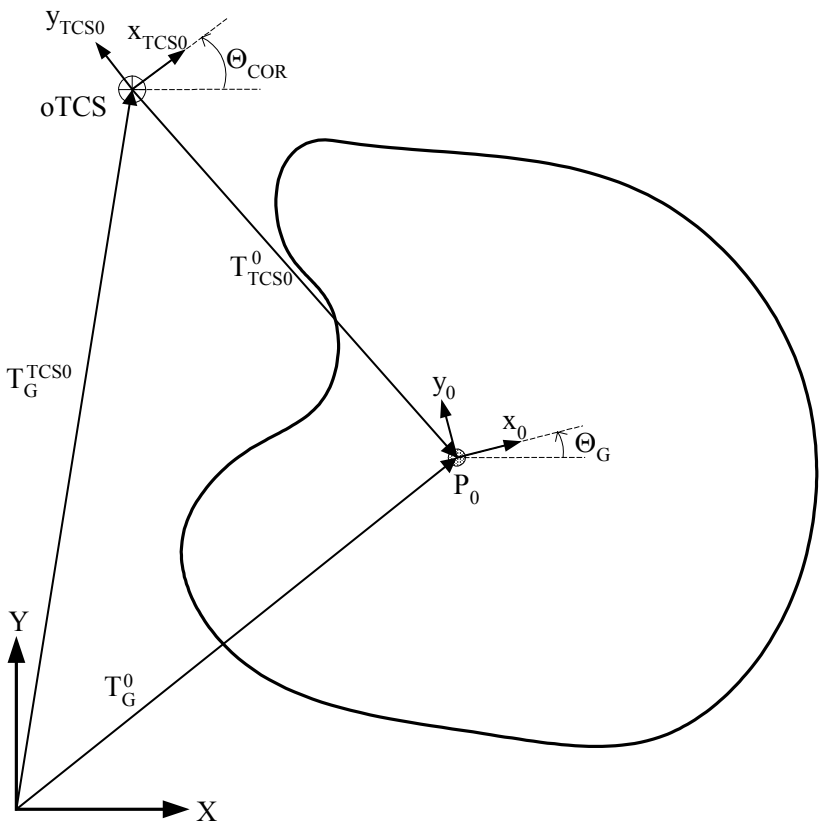


Figure 9 Homogeneous transformation of $(xyz)_0$ with respect to $(xyz)_{TCS0}$

4.2.4 Homogeneous Transformation of $(xyz)_{TCS1}$ with Respect to $(xyz)_{TCS0}$

$$T_{TCS0}^{TCS1} = \begin{bmatrix} c_\phi & -s_\phi & 0 & dx \\ s_\phi & c_\phi & 0 & dy \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

At position 1, after rigid body motion, the TCS is denoted TCS_1. During the displacement control loop, the rigid body rotates about the COR by ϕ degrees, but does not translate (**Figure 10**). Hence, frame TCS_0 rotates about its origin with no translation: $(dx, dy) \to 0$. During the load control loop, the rigid body translates by (dx, dy), but does not

rotate: $\phi \to 0$. We can think of the relationship between frame TCS and frame P as an imaginary rigid link. If point P translates by (dx, dy), then so does the COR. (dx, dy) can either be added to (P_X, P_Y) and (COR_X, COR_Y) or equivalently it can be inserted into T_{TCS0}^{TCS1}. (dx, dy) is inserted in T_{TCS0}^{TCS1} for consistency. Now that (dx, dy) has been used here, it is not used elsewhere.

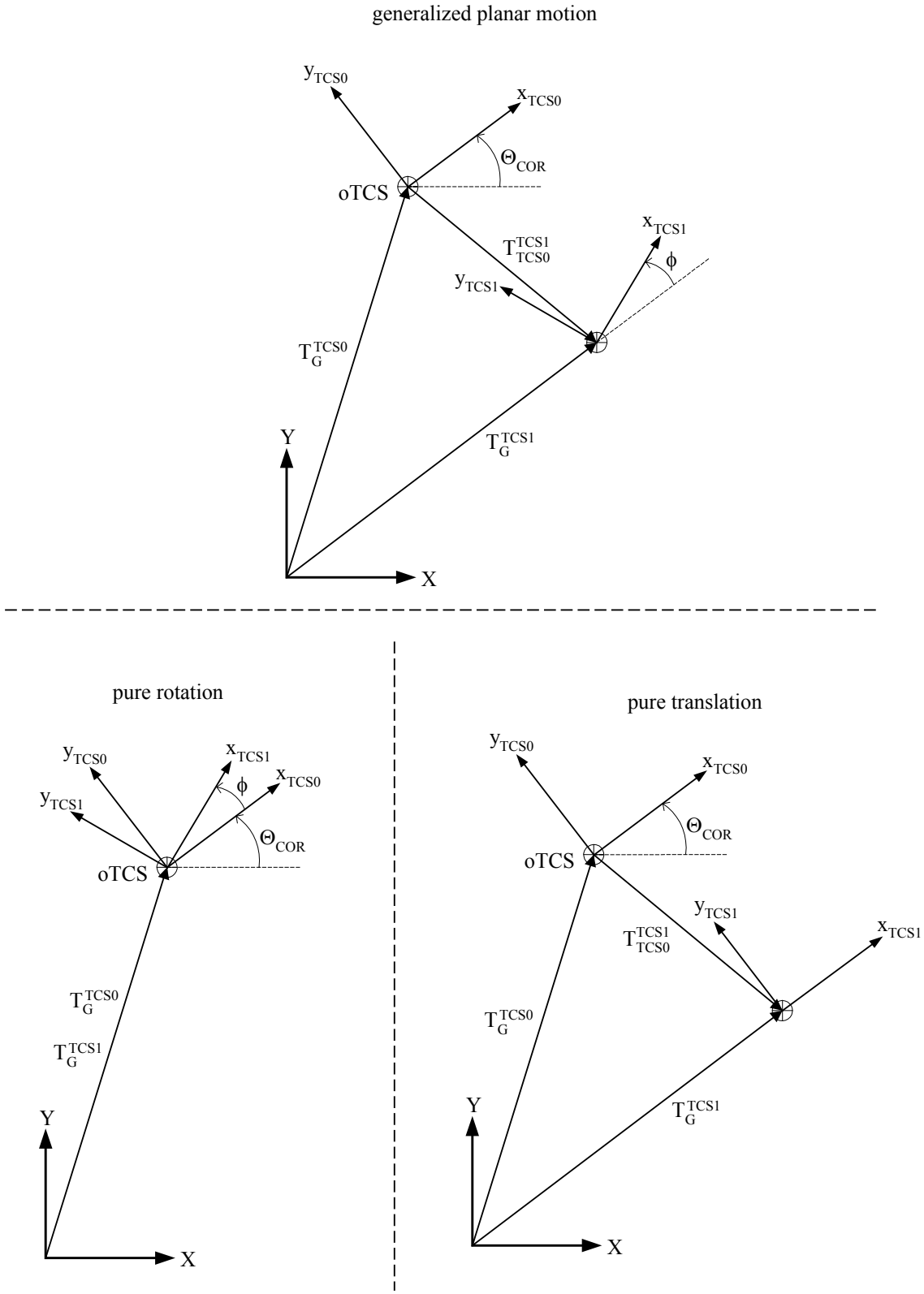


Figure 10 Homogeneous transformation of $(xyz)_{TCS1}$ with respect to $(xyz)_{TCS0}$

4.2.5 Homogeneous Transformation of $(xyz)_{TCS1}$ with Respect to XYZ

Because the relationship between frame P and frame TCS is constant, the global position of the COR must be updated to reflect changes in position of point P (**Figure 10**).

$$T_G^{TCS1} = T_G^{TCS0} T_{TCS0}^{TCS1}$$

4.2.6 Homogeneous Transformation of $(xyz)_1$ with Respect to $(xyz)_{TCS1}$

As noted above, the relationship between frame P and frame TCS is constant. Therefore, $(xyz)_1$ has the same relative position and orientation from $(xyz)_{TCS1}$ as $(xyz)_0$ has from $(xyz)_{TCS0}$ (**Figure 11**):

$$T_{TCS1}^{1} = T_{TCS0}^{0}.$$

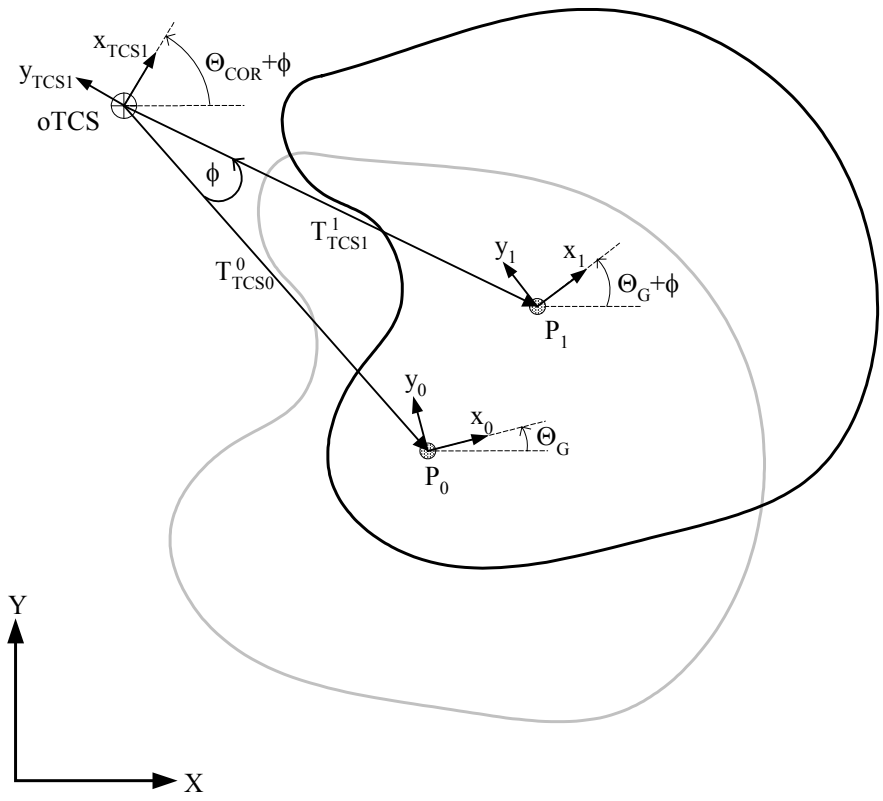


Figure 11 Homogeneous transformation of $(xyz)_1$ with respect to $(xyz)_{TCS1}$

4.2.7 Homogeneous Transformation of $(xyz)_1$ with Respect to XYZ

At final position 1, point P is denoted P_1.

$$T_G^1 = T_G^{TCS1} T_{TCS1}^1 = \begin{bmatrix} c_{G\phi} & -s_{G\phi} & 0 & P_{1X} \\ s_{G\phi} & c_{G\phi} & 0 & P_{1Y} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where (P_{1X}, P_{1Y}) is the global position of P_1 and $G\phi = \theta_G + \phi$ is the orientation of $(xyz)_1$ with respect to XYZ (**Figure 12**).

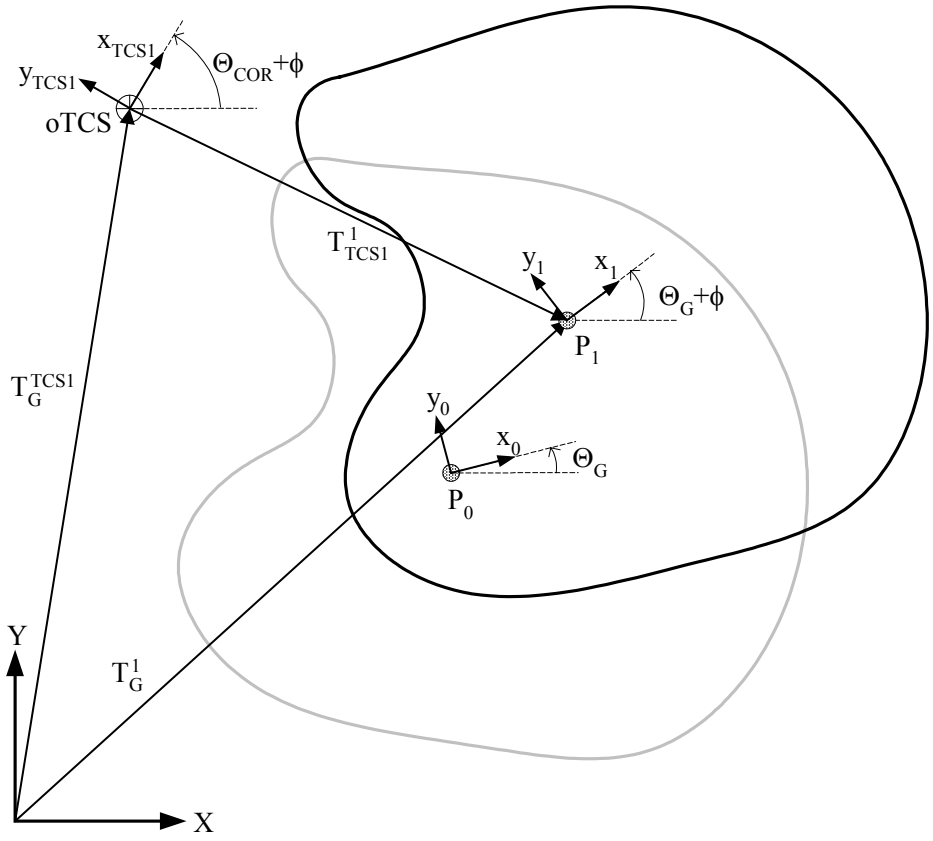


Figure 12 Homogeneous transformation of $(xyz)_1$ with respect to XYZ

4.2.8 Homogeneous Transformation of $(xyz)_{i0}$ with Respect to $(xyz)_0$

Now that the global position of point P is known before and after rigid body motion, the resulting global motion of node i is considered. The following transformations are easily extended to any number of nodes on rigid body M. Note that because the position and orientation of node i remains fixed relative to xyz, there is no subscript on i_x and i_y to differentiate between position 0 and position 1.

$$T_0^{i0} = \begin{bmatrix} c_i & -s_i & 0 & i_x \\ s_i & c_i & 0 & i_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where (i_x, i_y) is the local position of node i and θ_i is the orientation of $(xyz)_i$ with respect to xyz (**Figure 13**).

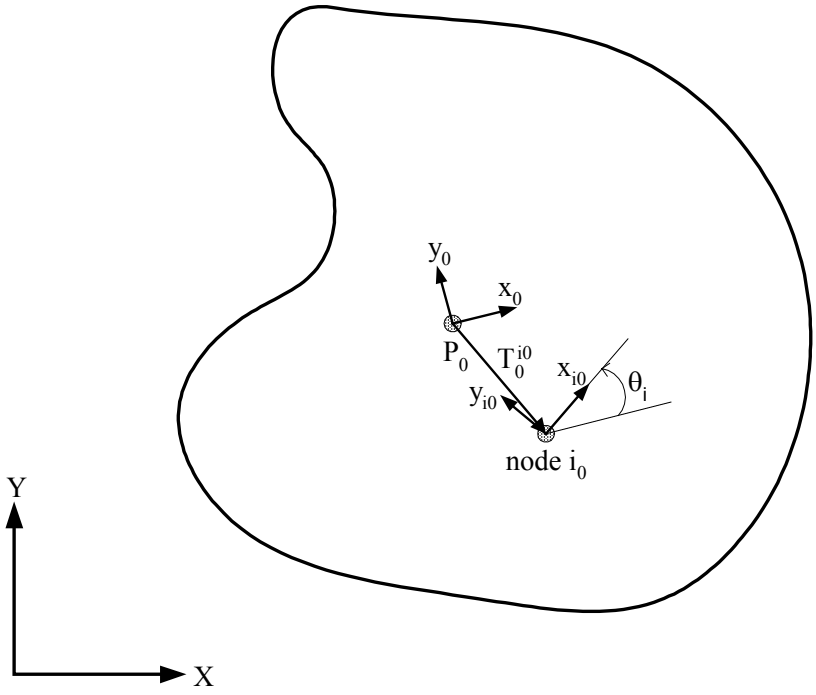


Figure 13 Homogeneous transformation of $(xyz)_{i0}$ with respect to $(xyz)_0$

4.2.9 Homogeneous Transformation of $(xyz)_{i0}$ with Respect to XYZ

The global position and orientation of node i at initial position 0 is described by

$$T_G^{i0} = T_G^0 T_0^{i0} = \begin{bmatrix} c_{Gi} & -s_{Gi} & 0 & i_{0X} \\ s_{Gi} & c_{Gi} & 0 & i_{0Y} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where (i_{0X}, i_{0Y}) is the initial global position of node i and $Gi = \theta_G + \theta_i$ is the orientation of $(xyz)_{i0}$ with respect to XYZ (**Figure 14**). Even though the position and orientation of node i remains fixed locally from position 0 to position 1, its global position changes.

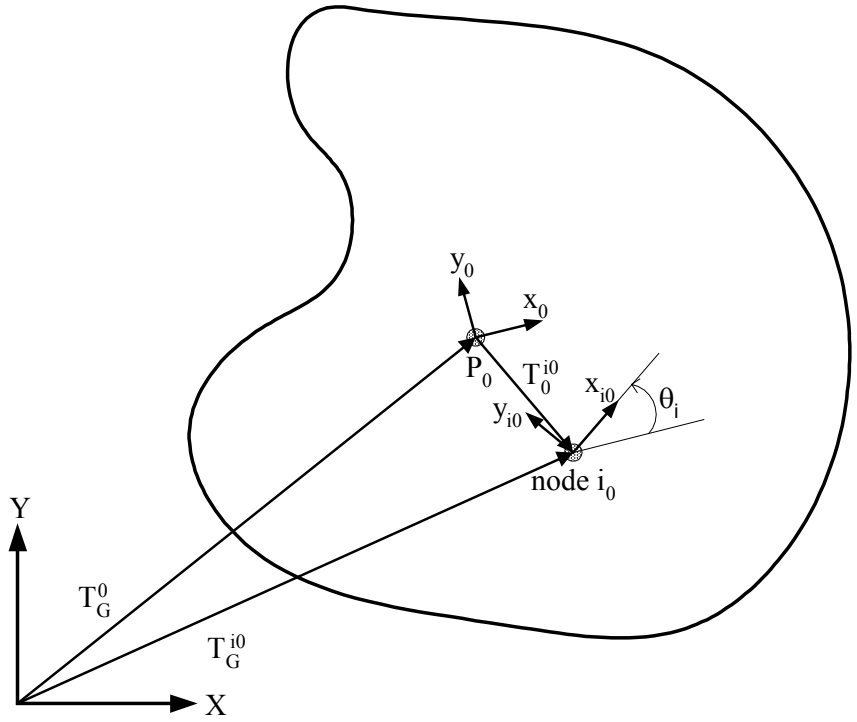


Figure 14 Homogeneous transformation of $(xyz)_{i0}$ with respect to XYZ

4.2.10 Homogeneous Transformation of $(xyz)_{i1}$ with Respect to $(xyz)_1$

Because of the rigid body assumption, the position and orientation of $(xyz)_{i1}$ with respect to $(xyz)_1$ is the same as $(xyz)_{i0}$ with respect to $(xyz)_0$ (**Figure 15**). Therefore,

$$T_1^{i1} = T_0^{i0}.$$

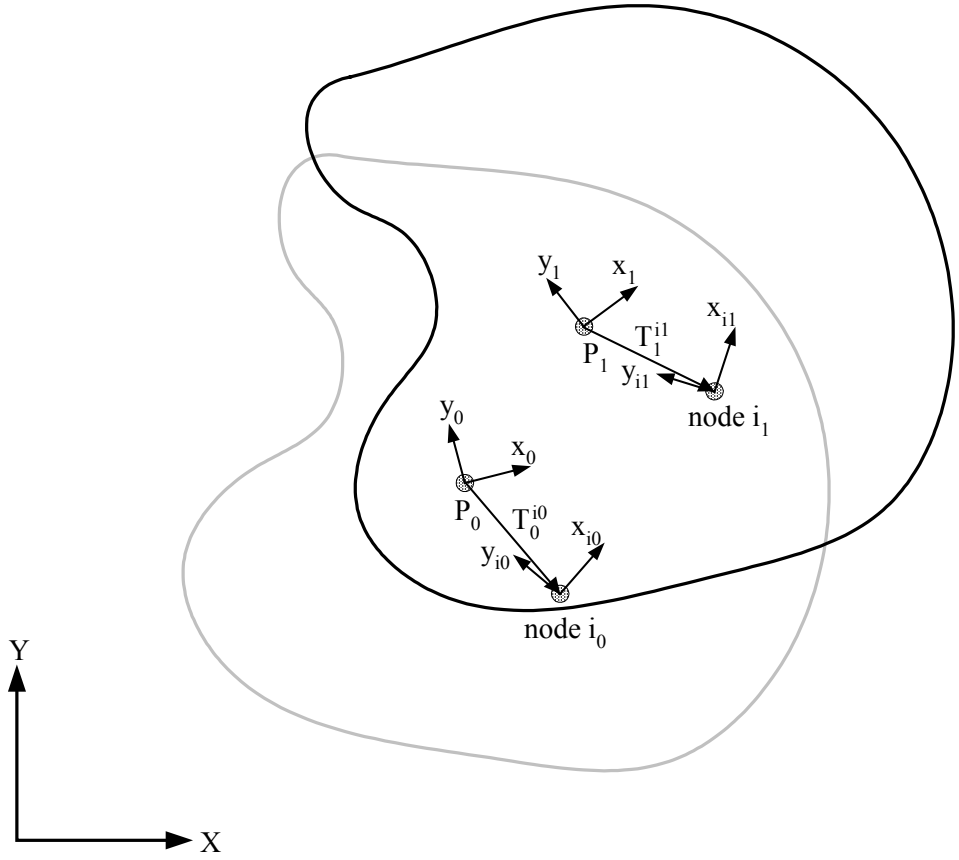


Figure 15 Homogeneous transformation of $(xyz)_{i1}$ with respect to $(xyz)_1$

4.2.11 Homogeneous Transformation of $(xyz)_{i1}$ with Respect to XYZ

The global position and orientation of node i at position 1 is described by

$$T_G^{i1} = T_G^1 T_1^{i1} = \begin{bmatrix} c_{Gi\phi} & -s_{Gi\phi} & 0 & i_{1X} \\ s_{Gi\phi} & c_{Gi\phi} & 0 & i_{1Y} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where (i_{1X}, i_{1Y}) is the global position of node i at position 1 and $Gi\phi = \theta_G + \theta_i + \phi$ is the orientation of $(xyz)_{i1}$ with respect to XYZ (**Figure 16**).

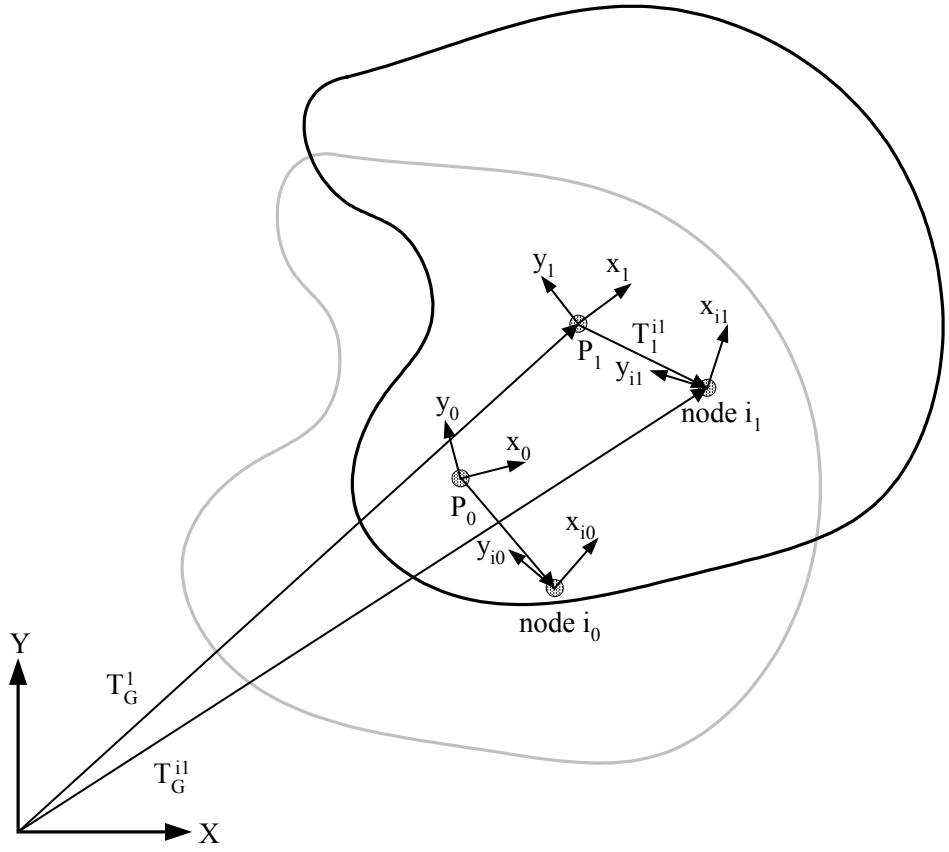


Figure 16 Homogeneous transformation of $(xyz)_{i1}$ with respect to XYZ

4.2.12 Homogeneous Transformation of $(xyz)_j$ with Respect to XYZ

The global position and orientation of frame j on the fixed rigid body is known through transformations similar to those shown above. For simplicity, the transformations leading to the global position and orientation of node j are not shown. Experimentally, we must solve for these coordinates using coordinate transformations. In simulations, we can define the global coordinates of frame j and bypass the transformations necessary to calculate them.

$$T_G^j = \begin{bmatrix} c_j & -s_j & 0 & j_X \\ s_j & c_j & 0 & j_Y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where (j_X, j_Y) are the global coordinates of node j and θ_j is the orientation of $(xyz)_j$ with respect to XYZ (**Figure 17**).

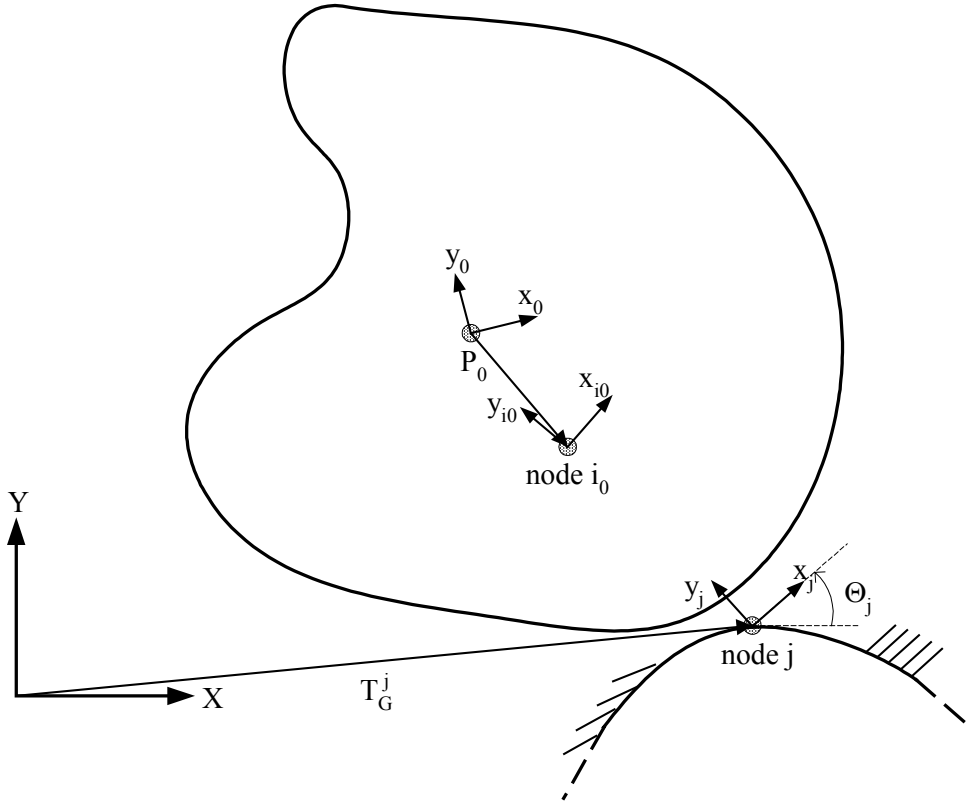


Figure 17 Homogeneous transformation of $(xyz)_j$ with respect to XYZ

4.2.13 Homogeneous Transformation of $(xyz)_j$ with Respect to $(xyz)_{i0}$

In order to fully define the loads acting on the rigid body due to spring i, we must know the line of action of spring force. At any position of rigid body M, the direction of spring force

is along a line between nodes i and j. Therefore, the transformation between frame i and frame j must be known. At initial position 0,

$$T_{i0}^{j} = \left(T_{G}^{i0}\right)^{-1} T_{G}^{j}.$$

See **Figure 18**.

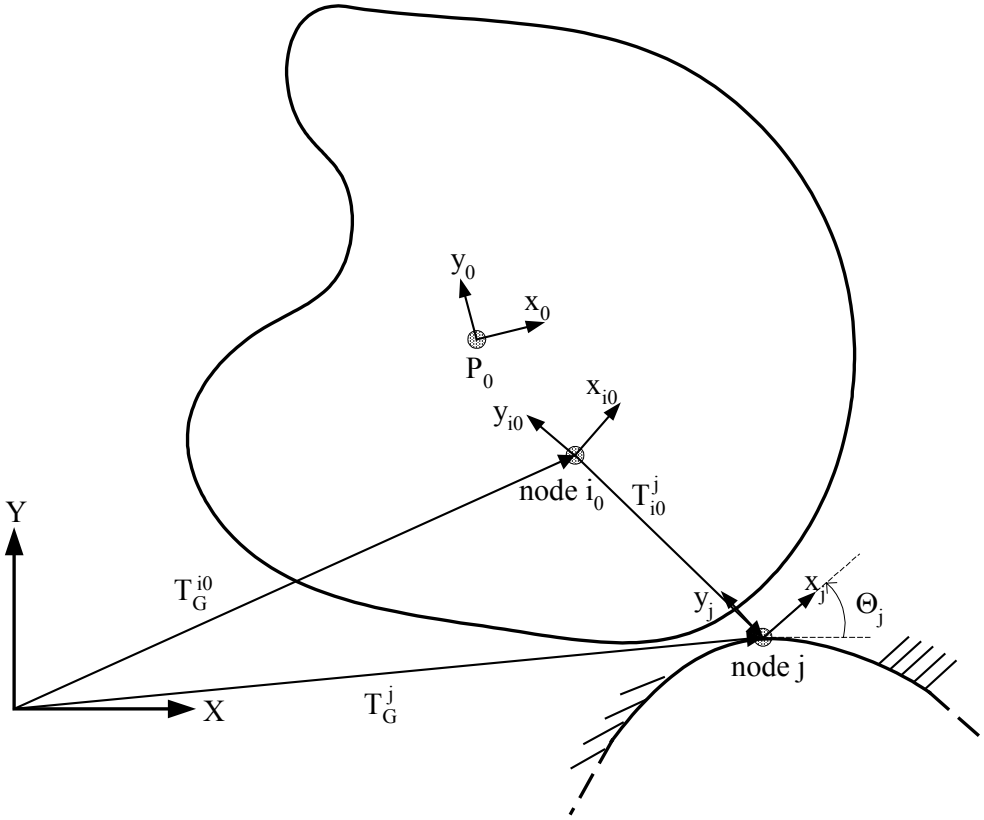


Figure 18 Homogeneous transformation of $\left(xyz\right)_{j}$ with respect to $\left(xyz\right)_{i0}$

4.2.14 Homogeneous Transformation of $\left(xyz\right)_{j}$ with Respect to $\left(xyz\right)_{i1}$

At position 1,

$$T_{i1}^{j} = \left(T_{G}^{i1}\right)^{-1} T_{G}^{j}.$$

See **Figure 19**.

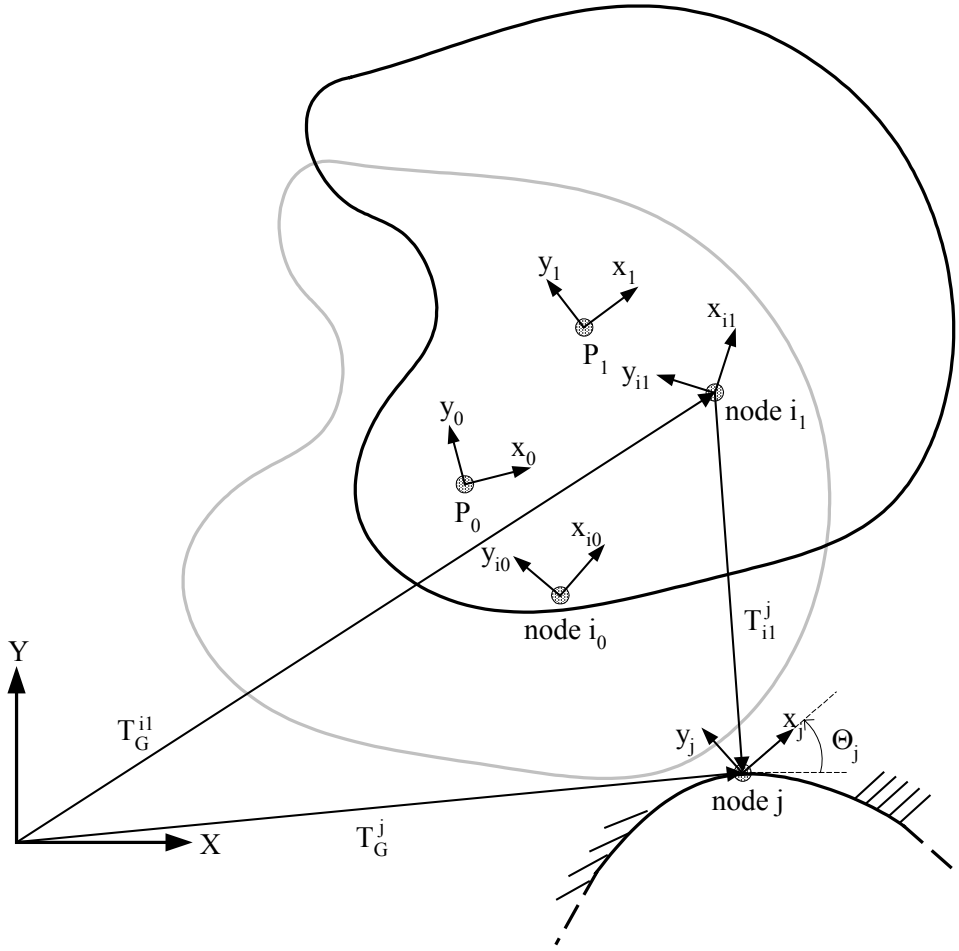


Figure 19 Homogeneous transformation of $(xyz)_j$ with respect to $(xyz)_{i1}$

4.2.15 Change in Length of Spring Attached to Node i and Fixed Node j

To find the loads acting on rigid body M, we must know the elongation of spring i, δ_i. The length of spring i at the initial position is the magnitude of the position vector of T_{i0}^{J}, $\left|\vec{l}_{i0}\right|$, defined in the $(xyz)_{i0}$ coordinate system. The elongation of spring i at the initial position is $\delta_{i0} = \left|\vec{l}_{i0}\right| - \ell_r$, where ℓ_r is the resting length of the spring. The length of spring i at the final

position is the magnitude of the position vector of T_{i1}^{j}, $\left|\vec{l}_{i1}\right|$, defined in the $(xyz)_{i1}$ coordinate system. The elongation of spring i at the final position is $\delta_{i1} = \left|\vec{l}_{i1}\right| - \ell_r$.

4.2.16 Loads on Rigid Body Due to Spring i

The total force acting on rigid body M at node i due to spring i at the initial and final positions are $\vec{f}_{i0}$ and $\vec{f}_{i1}$, respectively.

$$\vec{f}_{i0} = k_i \delta_{i0} \frac{\vec{l}_{i0}}{\left|\vec{l}_{i0}\right|}$$

$$\vec{f}_{i1} = k_i \delta_{i1} \frac{\vec{l}_{i1}}{\left|\vec{l}_{i1}\right|},$$

where k_i is the spring constant of spring i. $\vec{f}_{i0}$ and $\vec{f}_{i1}$ are known in the $(xyz)_{i0}$ and $(xyz)_{i1}$ coordinate systems, respectively, because $\vec{l}_{i0}$ and $\vec{l}_{i1}$ are defined in those coordinate systems. We want to know $\vec{f}_{i0}$ and $\vec{f}_{i1}$ in the global coordinate system, so we use transformations to convert them to the global coordinate system.

$$\begin{Bmatrix} \left(\vec{F}_{i0}\right)_X \\ \left(\vec{F}_{i0}\right)_Y \\ 0 \end{Bmatrix} = \begin{bmatrix} c_{Gi} & -s_{Gi} & 0 \\ s_{Gi} & c_{Gi} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} \left(\vec{f}_{i0}\right)_{i0x} \\ \left(\vec{f}_{i0}\right)_{i0y} \\ 0 \end{Bmatrix}$$

$$\begin{Bmatrix} \left(\vec{F}_{i1}\right)_X \\ \left(\vec{F}_{i1}\right)_Y \\ 0 \end{Bmatrix} = \begin{bmatrix} c_{Gi\phi} & -s_{Gi\phi} & 0 \\ s_{Gi\phi} & c_{Gi\phi} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} \left(\vec{f}_{i1}\right)_{i1x} \\ \left(\vec{f}_{i1}\right)_{i1y} \\ 0 \end{Bmatrix},$$

where $\vec{f}_{i0}$ and $\vec{f}_{i1}$ are broken into local x and y components and $\vec{F}_{i0}$ and $\vec{F}_{i1}$ are broken into global X and Y components.

If the force acting at node i due to spring i, $\vec{F}_i$, is replaced by an equal force acting at point P, a couple $\vec{M}_i$ is necessary to make sure the external effects of the original force on rigid body M are not changed (**Figure 20**).

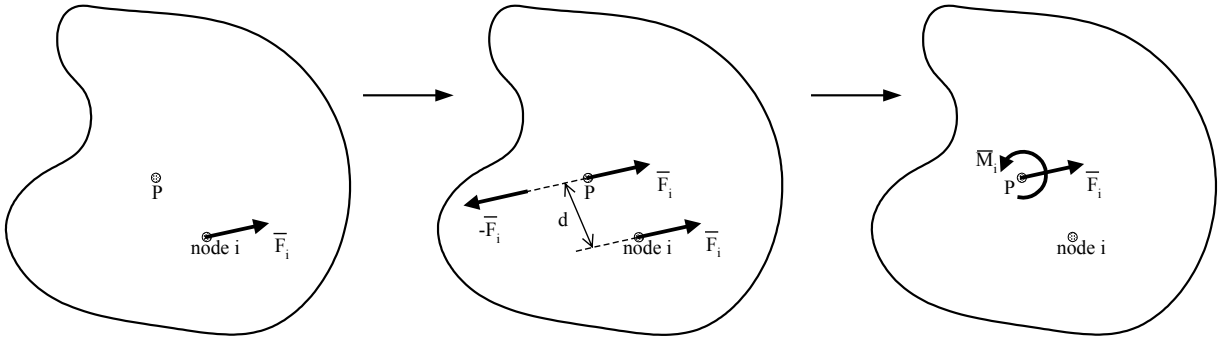


Figure 20 Force-couple equivalent

If $\vec{F}_{i0}$ acts at point P_0, $\vec{M}_{i0} = \vec{R}_{i0} \times \vec{F}_{i0}$, where $\vec{R}_{i0}$ is the position vector of T_0^{i0} transformed into the global coordinate system (**Figure 21**).

$$\vec{R}_{i0} = \begin{bmatrix} c_G & -s_G & 0 \\ s_G & c_G & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} i_x \\ i_y \\ 0 \end{Bmatrix}$$

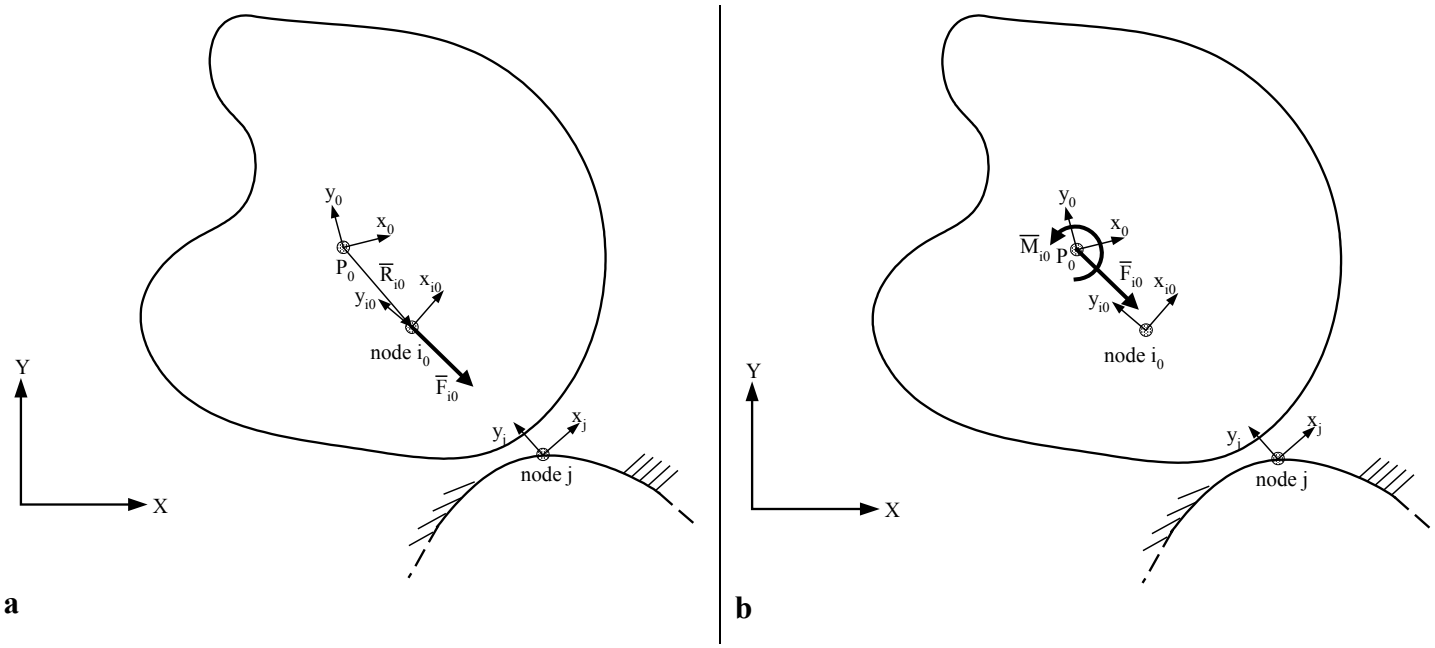


Figure 21 (**a**) force due to spring i at position 0, (**b**) equivalent force-couple system at position 0

If $\vec{F}_{i1}$ acts at point P_1, $\vec{M}_{i1} = \vec{R}_{i1} \times \vec{F}_{i1}$, where $\vec{R}_{i1}$ is the position vector of T_1^{i1} transformed into the global coordinate system (**Figure 22**).

$$\vec{R}_{i1} = \begin{bmatrix} c_{G\phi} & -s_{G\phi} & 0 \\ s_{G\phi} & c_{G\phi} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} i_x \\ i_y \\ 0 \end{Bmatrix}$$

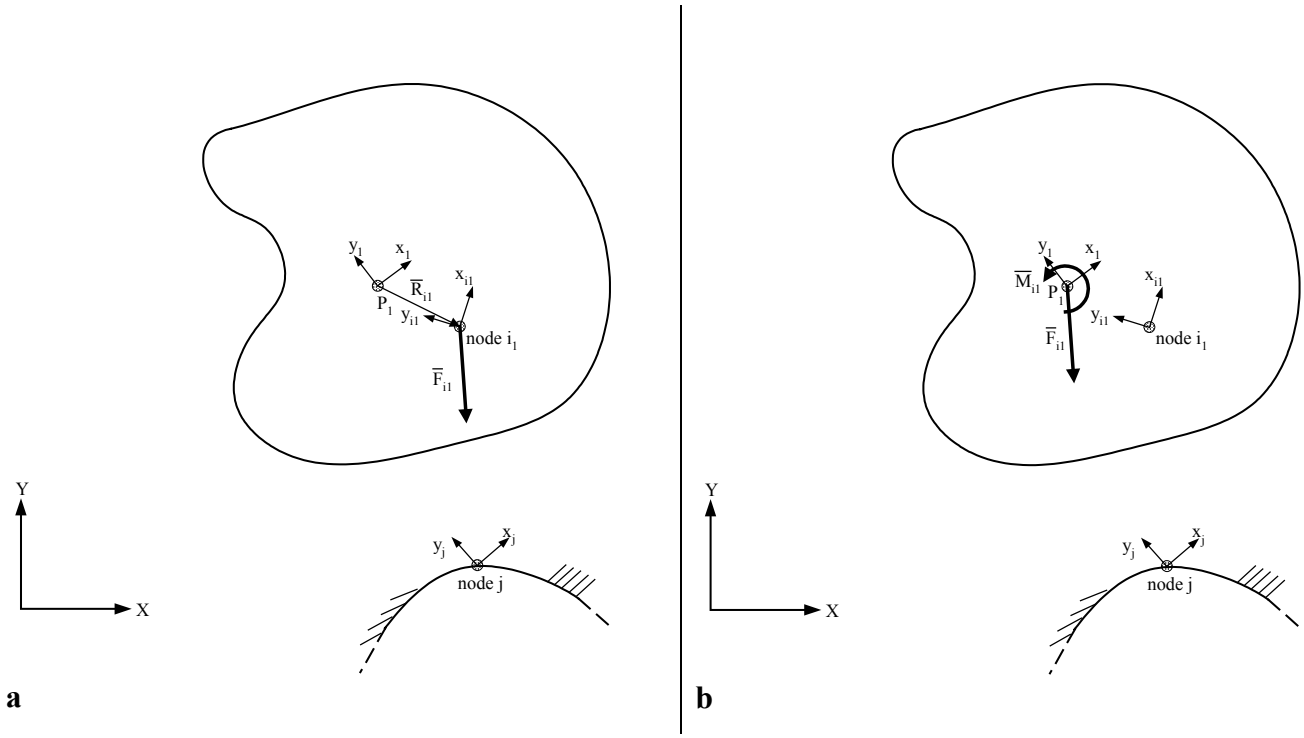


Figure 22 (**a**) force due to spring i at position 1, (**b**) equivalent force-couple system at position 1

To break forces and moment at points P_0 and P_1 into global X and Y components,

$$\begin{Bmatrix} \left(\vec{F}_{i0}\right)_X \\ \left(\vec{F}_{i0}\right)_Y \\ \left(\vec{M}_{i0}\right)_Z \end{Bmatrix} = \begin{Bmatrix} \vec{F}_{i0} \cdot \hat{i} \\ \vec{F}_{i0} \cdot \hat{j} \\ \left(\vec{R}_{i0} \times \vec{F}_{i0}\right) \cdot \hat{k} \end{Bmatrix}$$

$$\begin{Bmatrix} \left(\vec{F}_{i1}\right)_X \\ \left(\vec{F}_{i1}\right)_Y \\ \left(\vec{M}_{i1}\right)_Z \end{Bmatrix} = \begin{Bmatrix} \vec{F}_{i1} \cdot \hat{i} \\ \vec{F}_{i1} \cdot \hat{j} \\ \left(\vec{R}_{i1} \times \vec{F}_{i1}\right) \cdot \hat{k} \end{Bmatrix},$$

where $\hat{i}$, $\hat{j}$ and $\hat{k}$ are unit vectors in the global X, Y and Z directions, respectively.

The forces and moments acting on the rigid body due to springs $i = 1, 2, \ldots, n$ may be summed to find the total loads on the body: $\vec{F} = \sum_{i=1}^{n} \vec{F}_i$, $\vec{M} = \sum_{i=1}^{n} \vec{M}_i$.

4.2.17 Global Stiffness Matrix, $\underline{K}$

After developing the analytical solution for loads arising from rigid body motion, it is necessary to find the analytical solution for the global stiffness matrix for use during load control. The analytical *equations* for each term in the matrix are valid at any position of rigid body M even though the *values* of the terms are only valid over small ranges of motion. Consequently, translations of rigid body M during load control should be limited because the calculated displacement depends on local stiffness values. A large translation may move the rigid body outside the region of constant local stiffness. We take the partial differential of the analytical expressions for F_X, F_Y and M_Z with respect to P_X, P_Y and P_Φ to find the global stiffness matrix, $\underline{K}$:

$$\underline{K} = \begin{bmatrix} K_{XX} & K_{XY} & K_{XZ} \\ K_{YX} & K_{YY} & K_{YZ} \\ K_{ZX} & K_{ZY} & K_{ZZ} \end{bmatrix}$$

$$\begin{Bmatrix} \Delta F_X \\ \Delta F_Y \\ \Delta M_Z \end{Bmatrix} = \underline{K} \begin{Bmatrix} \Delta P_X \\ \Delta P_Y \\ \Delta P_\Phi \end{Bmatrix} = \begin{bmatrix} \frac{\partial F_X}{\partial P_X} & \frac{\partial F_X}{\partial P_Y} & \frac{\partial F_X}{\partial P_\Phi} \\ \frac{\partial F_Y}{\partial P_X} & \frac{\partial F_Y}{\partial P_Y} & \frac{\partial F_Y}{\partial P_\Phi} \\ \frac{\partial M_Z}{\partial P_X} & \frac{\partial M_Z}{\partial P_Y} & \frac{\partial M_Z}{\partial P_\Phi} \end{bmatrix} \begin{Bmatrix} \Delta P_X \\ \Delta P_Y \\ \Delta P_\Phi \end{Bmatrix},$$

where P_X and P_Y are the global coordinates of point P at any position of rigid body M, P_Φ is the orientation of xyz with respect to XYZ at any position and F_X, F_Y and M_Z are the total force and moment acting on rigid body M at any position.

Ren et al.[47] also used partial derivates to calculate the tangent stiffness matrix for their rigid body-spring model. They used three rigid bodies connected by springs that were allowed to

translate and rotate in the XY-plane. Rigid body e was connected to rigid body $e+1$ by three springs: an axial spring (spring constant K_e^a), a shear spring (spring constant K_s^e) and a bending spring (spring constant K_e^θ). These spring constants were collectively referred to as K_e. Similarly, rigid body $e-1$ was connected to rigid body e by three springs with spring constant K_{e-1}. They developed equations to describe the sum of the forces and moments acting on rigid body e and took the partial derivatives of these expressions with respect to the translational and rotational motion of the centroid of each rigid body to find the tangent stiffness matrix. They did not make any assumptions while developing their analytical stiffness matrix, so their method is completely general for any planar rigid body motion. Their method is very similar to what is done above. An axial spring connects two rigid bodies between node j, located on a fixed rigid body ($e-1$), and node i, located on a rigid body (e) that is allowed to move in the same plane as that in Ren et al. In this case, we are only concerned with spring K_{e-1}^a and the moveable rigid body e, so the partial derivatives simplify to the above expression.

To simplify the partial derivatives, several constants are defined:

$$c1 = j_X - i_x * \cos\theta + i_y * \sin\theta,$$

$$c2 = j_Y - i_y * \cos\theta - i_x * \sin\theta,$$

$$c3 = \theta_G + \phi,$$

$$c4 = j_X - P_{1X},$$

$$c5 = j_Y - P_{1Y}.$$

The stiffness matrix is symmetric, so $K_{YX} = K_{XY}$, $K_{ZX} = K_{XZ}$ and $K_{ZY} = K_{YZ}$. The terms in the stiffness matrix are then:

$$K_{XX} = \frac{k\ell_r \left(c2 - P_Y\right)^2}{\left[\left(c1 - P_X\right)^2 + \left(c2 - P_Y\right)^2\right]^{3/2}} - k$$

$$K_{XY} = K_{YX} = -\frac{k\ell_r \left(c1 - P_X\right)\left(c2 - P_Y\right)}{\left[\left(c1 - P_X\right)^2 + \left(c2 - P_Y\right)^2\right]^{3/2}}$$

$$K_{XZ} = k\left(i_y c_{G\phi} + i_x s_{G\phi} + \frac{\ell_r \left(-c5 + i_y c_{G\phi} + i_x s_{G\phi}\right)\left(-i_x^2 - i_y^2 + \left(c4 i_x + c5 i_y\right) c_{G\phi} + \left(c5 i_x - c4 i_y\right) s_{G\phi}\right)}{\left(\left(-c5 + i_y c_{G\phi} + i_x s_{G\phi}\right)^2 + \left(c4 - i_x c_{G\phi} + i_y s_{G\phi}\right)^2\right)^{3/2}} \right)$$

$$K_{YY} = \frac{k\ell_r \left(c1 - P_X\right)^2}{\left[\left(c1 - P_X\right)^2 + \left(c2 - P_Y\right)^2\right]^{3/2}} - k$$

$$K_{YZ} = k\left(-i_x c_{G\phi} + i_y s_{G\phi} + \frac{\ell_r \left(c4 - i_x c_{G\phi} + i_y s_{G\phi}\right)\left(-i_x^2 - i_y^2 + \left(c4 i_x + c5 i_y\right) c_{G\phi} + \left(c5 i_x - c4 i_y\right) s_{G\phi}\right)}{\left(\left(-c5 + i_y c_{G\phi} + i_x s_{G\phi}\right)^2 + \left(c4 - i_x c_{G\phi} + i_y s_{G\phi}\right)^2\right)^{3/2}} \right)$$

$$K_{zz} = -\frac{k\ell_r \left(\left(-c5 i_x + c4 i_y\right) c_{G\phi} + \left(c4 i_x + c5 i_y\right) s_{G\phi}\right)^2}{\left(\left(-c5 + i_y c_{G\phi} + i_x s_{G\phi}\right)^2 + \left(c4 - i_x c_{G\phi} + i_y s_{G\phi}\right)^2\right)^{3/2}}$$

$$+k\left(\left(-c4 i_x - c5 i_y\right) c_{G\phi} + \left(-c5 i_x + c4 i_y\right) s_{G\phi}\right)\left(1 - \frac{\ell_r}{\sqrt{\left(-c5 + i_y c_{G\phi} + i_x s_{G\phi}\right)^2 + \left(c4 - i_x c_{G\phi} + i_y s_{G\phi}\right)^2}} \right)$$

Because only planar translations are considered during the load control loop, a 2x2 stiffness matrix used.

4.2.18 Work Done on Rigid Body by Spring i, Potential Energy in System

The magnitude of the force exerted on the rigid body by spring i is $F_i = -k_i\delta_i$. For a conservative force, such as a spring force, the potential energy is $U(\delta) = U(\delta_0) - \int_{\delta_0}^{\delta} F(\delta')\,d\delta'$. By plugging in the equation for spring force, we can solve for the potential energy of spring i: $U_i(\delta_i) = U_i(\delta_{i0}) - \int_{\delta_{i0}}^{\delta_i} -k_i\delta'\,d\delta = U_{i0} + \frac{1}{2}k_i\delta_i^2 - \frac{1}{2}k_i\delta_{i0}^2$. If we do not make any simplifications to this equation, we must know U_{i0} for any value of δ_{i0}. To simplify the equation for potential in spring i we arbitrarily set $\delta_{i0} = 0$. Then $U_{i0} = 0$ because there is no potential energy when the spring is at its resting length. We are then left with $U_i(\delta_i) = \frac{1}{2}k_i\delta_i^2$. For a spring, work is equal in magnitude and opposite in sign to potential energy:

$W_i(\delta_i) = W_i(\delta_{i0}) + \int_{\delta_{i0}}^{\delta_i} -k_i\delta\,d\delta = W_{i0} - \frac{1}{2}k_i\delta_i^2 + \frac{1}{2}k_i\delta_{i0}^2$, or after simplification: $W_i(\delta_i) = -\frac{1}{2}k_i\delta_i^2$.

For an n spring system, the total potential energy is $U = \sum_{i=1}^{n} U_i(\delta_i)$ and the total work done to the rigid body is $W = \sum_{i=1}^{n} W_i(\delta_i)$. Both forms of $U_i(\delta_i)$ and $W_i(\delta_i)$ (simplified or not) give the same results, so the simplified form should be used because it requires less computation.

4.3 General Closed Form Solution Applied to Rigid Body-Spring Model

The general rigid body-spring model used for simulations is shown in **Figure 23** and **Figure 24**. This model is obviously different than the physical rigid body-spring model shown in **Figure 5**. Point P is at the center of the bar and is the origin of the bar's local coordinate

system, xyz. Two nodes are defined on the bar for each attachment site of each spring. Node a is at the left side of the bar and is the origin of coordinate system $(xyz)_a$. Node b is at the right side of the bar and is the origin of coordinate system $(xyz)_b$. The positions and orientations of $(xyz)_a$ and $(xyz)_b$ are described with respect to xyz. The length of the bar is $2L$. For spring a, the resting length is ℓ_{ar}, the equilibrium length is ℓ_{a0} and the spring constant is k_a. For spring b, the resting length is ℓ_{br}, the equilibrium length is ℓ_{b0} and the spring constant is k_b. The system is conservative.

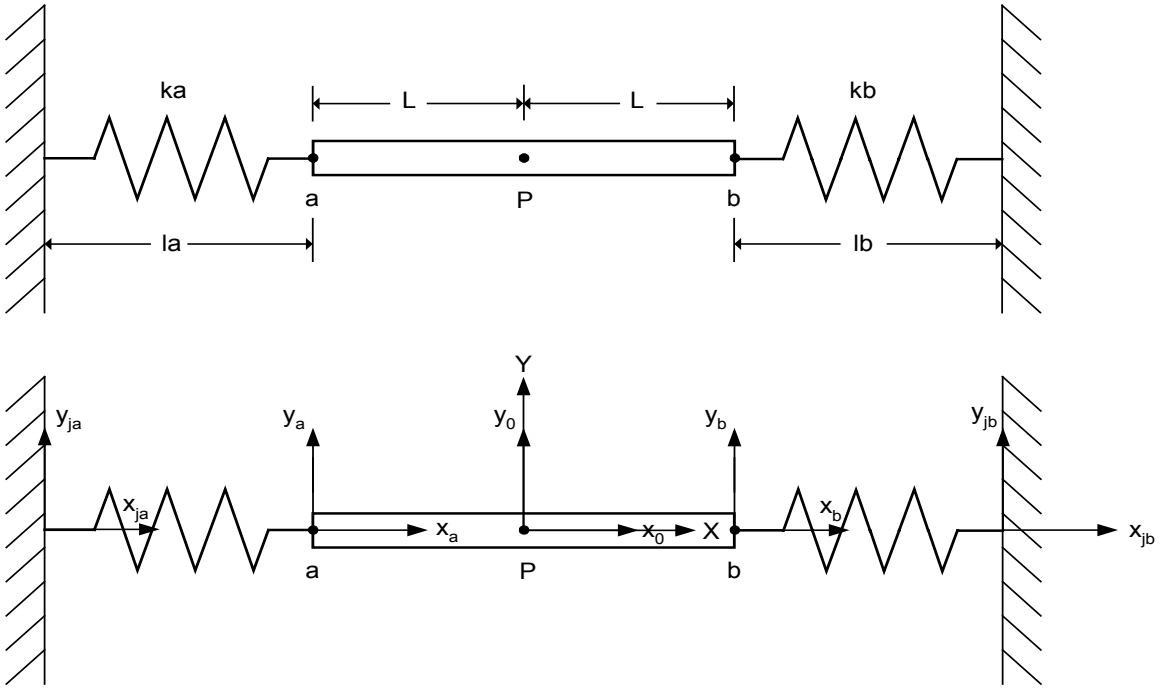


Figure 23 General rigid body-spring model

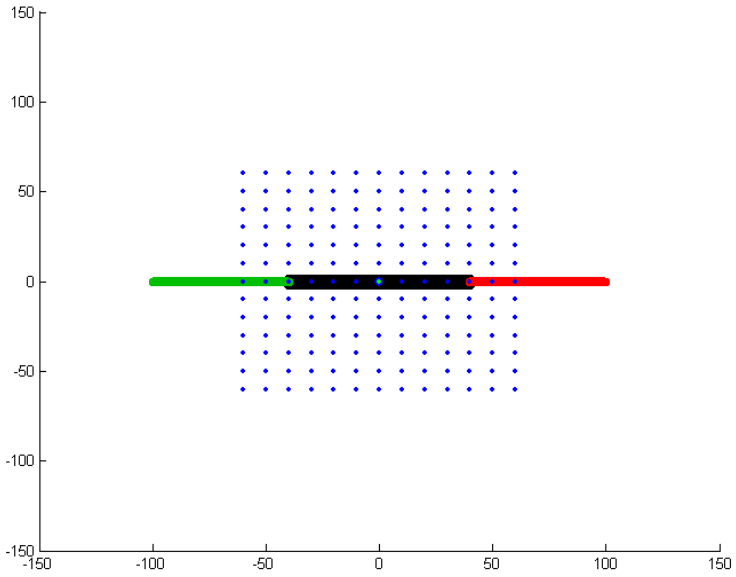


Figure 24 Matlab rigid body-spring model

For convenience, $\ell_{ar} = \ell_{br}$, $\ell_{a0} = \ell_{b0}$, $k_a = k_b$ and all coordinate systems are aligned at the equilibrium position, i.e., $\theta_G = \theta_{COR} = \theta_i = \theta_j = 0$. After one displacement control loop, $\theta_G = \theta_{COR} = \phi$ and $\theta_i = \theta_j = 0$. After n displacement control loops, $\theta_G = \theta_{COR} = n\phi$ and $\theta_i = \theta_j = 0$. In the equilibrium position, the XYZ and $(xyz)_0$ coordinate systems are coincident at point P_0. Therefore, T_G^0 is a $4x4$ identity matrix. This is only true at the equilibrium position. After one incremental rotation, $(xyz)_1$ is rotated by ϕ degrees from XYZ and the origins are offset by an amount due to the rotation. In a test, $\phi = \phi$, $dx = dy = 0$ for the displacement control loop and $\phi = 0$, $dx = dx$ and $dy = dy$ for the subsequent load control loop.

For node a, $T_0^{a0} = T_1^{a1} = \begin{bmatrix} 1 & 0 & 0 & -L \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and $T_G^{ja} = \begin{bmatrix} 1 & 0 & 0 & -(L+\ell_{a0}) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$. For node b,

$T_0^{b0} = T_1^{b1} = \begin{bmatrix} 1 & 0 & 0 & L \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and $T_G^{jb} = \begin{bmatrix} 1 & 0 & 0 & L+\ell_{b0} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

The "physical" rigid body-spring model is shown in **Figure 25**. This model is used to collect simulated data for comparison with experimental data. To make comparisons, we take some measurements of the physical spring model. First, we measure the resting length of each spring using calipers. $\ell_{ar} = 49.53$ mm for spring a (the blue spring). $\ell_{br} = 74.57$ mm for spring b (the red spring). We also use calipers to measure the equilibrium length of each spring when they are in the physical spring model. For spring a, $\ell_{a0} = 59.23$ mm. For spring b, $\ell_{b0} = 84.40$ mm. The radius of the disc was measured with calipers as $L = 28$ mm. We use Adobe Photoshop 6.0 to find θ_G, the angle that the local x_0 axis makes with the global X axis: $\theta_G = 70°$. Now the positions of all nodes can be defined.

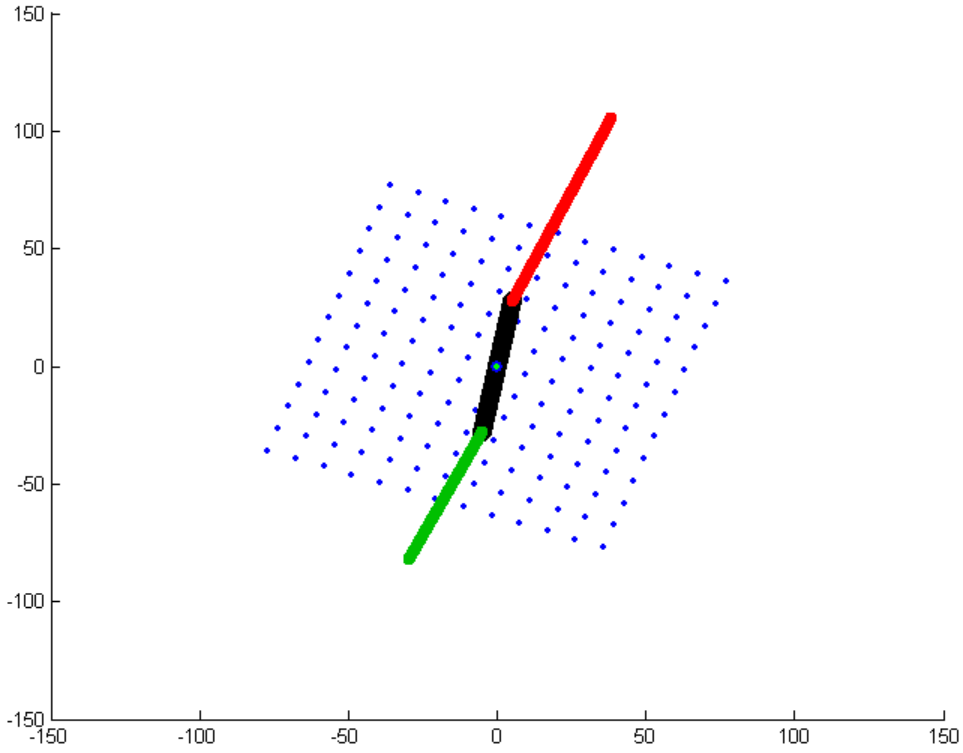


Figure 25 Matlab physical rigid body-spring model

For simplicity, we set $\theta_{COR} = \theta_i = \theta_j = 0$. After one displacement control loop, $\theta_G = 70° + \phi$, $\theta_{COR} = \phi$ and $\theta_i = \theta_j = 0$. After n displacement control loops, $\theta_G = 70° + n\phi$, $\theta_{COR} = n\phi$ and $\theta_i = \theta_j = 0$. In the equilibrium position, the XYZ and $(xyz)_0$ coordinate systems are coincident at point P_0. For node a, $T_0^{a0} = T_1^{a1} = \begin{bmatrix} 1 & 0 & 0 & -28 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and

$T_G^{ja} = \begin{bmatrix} 1 & 0 & 0 & -29.84 \\ 0 & 1 & 0 & -81.97 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$. For node b, $T_0^{b0} = T_1^{b1} = \begin{bmatrix} 1 & 0 & 0 & 28 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and

$T_G^{jb} = \begin{bmatrix} 1 & 0 & 0 & 38.45 \\ 0 & 1 & 0 & 105.63 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

When the springs are inserted into the physical spring model, they are elongated. This means that there is some initial tension in each spring, but the system is in equilibrium because the pretension in one spring negates the pretension in the other spring. No forces or moments should be created when the robot is initially attached to the model. After zeroing out bolt-up loads and loads due to the fixture (stainless steel disc, nuts, bolts, screws, etc.), the UFS will show that no other external loads are acting on it. When the robot applies a rotation/translation, the UFS will show the loads exerted by the model due to the motion; the UFS will not show the initial pretension in the equilibrated system.

5.0 DEVELOPMENT OF EXPERIMENTAL PLATFORM

Use of a robotics-based testing system allows for controlled application of six DOF displacements, facilitating determination of the *in situ* force and moment contribution of musculoskeletal joint structures. After dissecting away extraneous soft tissue, the passive path of the intact joint is found using hybrid control. The *in situ* contribution of a specific structure of interest is found by dissecting it away and replaying the passive path kinematics of the intact joint using pure displacement control, while recording loads. By applying the principle of superposition, the loads of the cut specimen are subtracted from the loads of the intact specimen to find the *in situ* contribution of the dissected structure.

5.1 Description of Robotics-Based Spine Testing System

Low-level control of a robotic system involves input/output of position data to and from the robot and communication with external sensors, whereas high-level control is the processing of that data for robot manipulation. The low-level control of our robotic/UFS testing system is performed using a robotic manipulator (Staubli, RX-90 model; Staubli Inc., Duncan, SC), computerized controller (Staubli, CS7 model, 40 MHz microprocessor, 33 MHz coprocessor, 4 Mb RAM), Adept V+ software (version 11.1) and a six degree of freedom universal force-moment sensor (UFS) (JR3, UFS Model 90M38A-I50 20L100; JR3, Woodland, CA). The Staubli is a servo-controlled, six-joint serial-articulated manipulator with end-effector position repeatability of 0.02 mm translation at constant temperature and maximum payload of 6 kg at nominal speed[48]. The UFS, mounted to a custom machined piece on the end-effector of the Staubli (**Figure 26**) has a full-scale force capacity of 20 lbs for its x and y axes and 50 lbs for its

z axis, and a full scale moment capacity of 100 in-lbs for all axes. Manufacturer-stated force and moment accuracy of the UFS is 2% of full scale for all axes[49]. The high-level computerized control system consists of a real time Staubli CS7 controller serially connected to a personal computer (Dell PC, dual Xeon 1.7 GHz processors, 1 GB RAM). Communication is covered in more detail later. The high-level control programs are performed using Matlab (version 6.1, The Mathworks, Inc., Natick, MA) on the PC. Digital output from the six load channels of the UFS is fed directly to the PC through a DSP-based force sensor/receiver PCI card (JR3). Dr. J. Norberto Pires wrote several Matlab-PCI interface modules for the JR3 PCI card[50]. The control programs written in Matlab and V+ perform a variety of tasks including establishing coordinate systems, reading UFS force-moment data, reading end-effector position data (calculated by the Staubli controller from the robotic joint angles obtained from the encoders of the servomotor of each joint), and issuing commands to the robot to move the end-effector. Depending on the control programs that are executed, the robotic/UFS testing system can be made to operate in either a position (i.e., displacement) control mode, or a hybrid control mode.

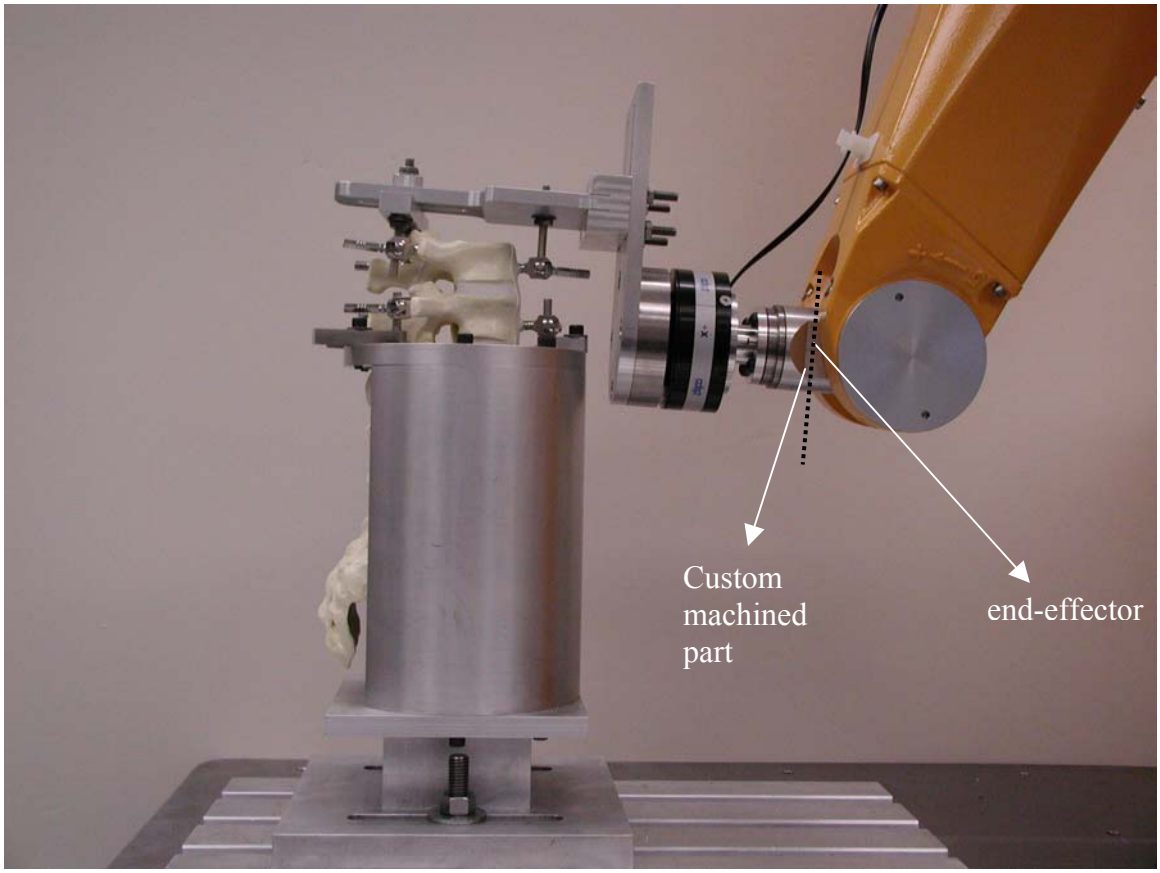


Figure 26 Specimen fixtures in testing system

The manipulator sits on a 30" high stainless steel table that is bolted through ¾" steel runners to the floor (**Figure 27**). A 3/8" thick stainless steel buffer is attached to the UFS. T-slots are attached to the table to provide flexibility of specimen placement in relation to the manipulator. Custom fixtures for specimen mounting are attached to the stainless steel buffer and the T-slots (**Figure 26**).

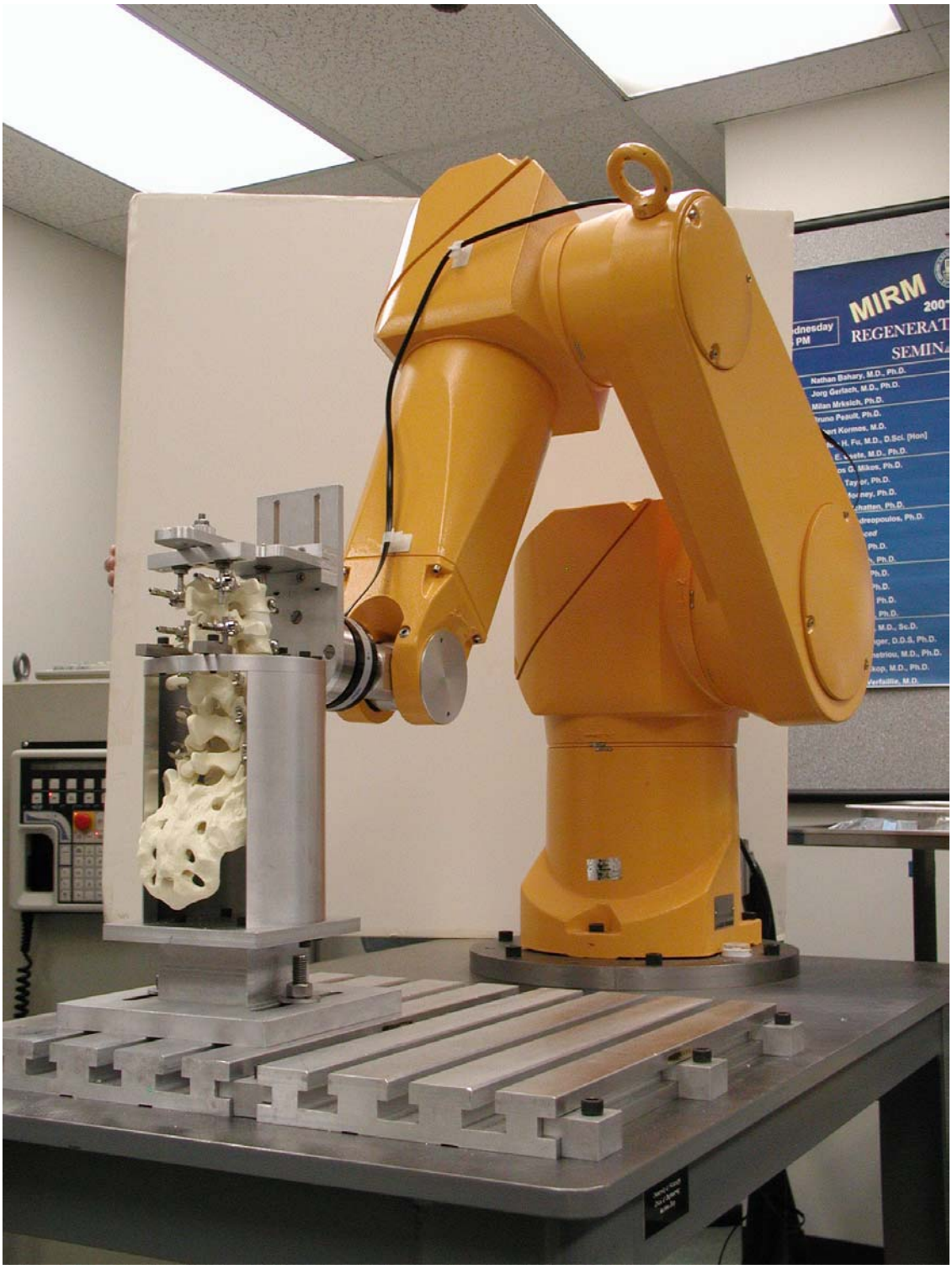


Figure 27 Robotic/UFS testing system

The controller sends the commanded motion to the manipulator in terms of the tool coordinate system (TCS), while returning the global position and orientation of the TCS to the user in response to the "WHERE" or "HERE" commands. The position and orientation of the TCS is measured with respect to the end-effector, which is at the back of the custom machined part (**Figure 26**). If the user does not specify a TCS, the controller sets it at the end-effector so that the transformation describing the relationship between these two coordinate systems is an identity matrix. As mentioned before, the origin of the TCS is set at the specimen COR. The orientation of the TCS is aligned with the specimen's coordinate system. Planar flexion/extension is performed by rotating about the TCS x-axis. When setting a TCS, its position is measured from the UFS face.

If the UFS could be placed at the center of the superior vertebra, the loads would be read at point P as they are during simulations. However, this is impossible so we need a transformation describing the superior vertebra's coordinate system with respect to the UFS coordinate system. Measuring the distance of the superior vertebra's coordinate system from the UFS coordinate system presents an interesting situation for measurements in the z-direction because the position of UFS coordinate system is dependent on the software used to collect load cell data. When using the PCI card to collect load cell data, the Matlab functions put the UFS coordinate system at the center of the UFS. When using the robotic controller to collect load cell data, Adept puts the UFS coordinate system at the back of the UFS.

5.2 Communication

Figure 28 shows the system components, with arrows depicting the data flow loop within the testing system. The controller receives TCS position data from the manipulator and directs this data to the external PC via the serial line. Serial communication is relatively slow, but it is convenient for this purpose since it is available on most commercially available controllers. As mentioned before, load cell data is sent directly to the PC. Directing the robot positions and UFS loads to the PC allows it to act as the high-level controller for the system. For high-level control to occur once the flow of data has been established, a programming language is necessary to implement the desired control algorithm. Matlab was chosen because of its many preprogrammed functions and toolboxes, its data analysis and graphing capabilities, and its readily available serial communication. Once the PC has interpreted the position and load data, the desired robot motion is sent back to the controller via the serial line so low-level control can occur.

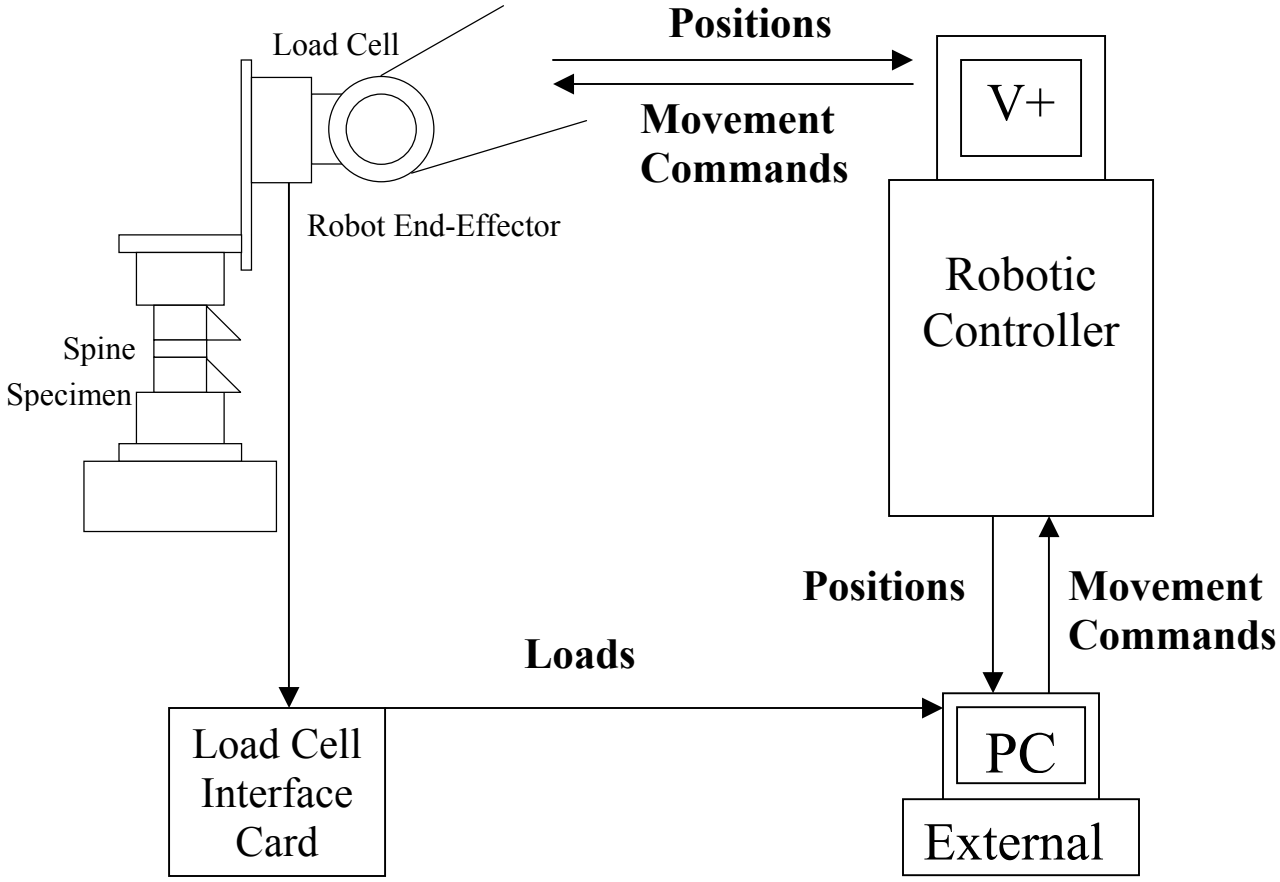


Figure 28 Data flow in testing system

Because the serial line sends and receives data, it is necessary to establish a client/server relationship between the two devices. Since limited use of the robot controller is desired for all high-level operations, it is best to have the external PC (client) request information from the controller (server). This type of relationship necessitates that proper "handshaking" occurs to guarantee that all data is sent and received at the correct time and to the correct device. This is ensured through a system of flags that indicate when the client/server platforms are in a state of proper operational mode, allowing information exchange to occur.

5.3 UFS Calibration

It was shown previously that a large source of error in load cell data may due to "phantom" loads due to change in load cell orientation.[51] The error in load cell data reported in Gilbertson et al. exceeded the manufacturer stated accuracy of 1% of full scale load capacity. For our load cell, the error due to load cell orientation was found within the manufacturer stated accuracy of 2% of full scale, but reproducing the methods in this paper still resulted in a significant improvement in accuracy.

The UFS was rotated about its x-axis without any fixtures attached from $\theta = -25°$ to $25°$. The digital output from the load cell in the y- and z-directions was found to vary linearly with rotation angle from about –0.25 N to about –1 N for F_y and from about 2 N to about 2.5 N for F_z (**Figure 29**). This error was within the manufacturer stated accuracy for both F_y and F_z. However, we proceeded with the protocol to see if the load cell accuracy could be improved further. By following the procedure outlined in Gilbertson et al. it was found that the error could be significantly reduced. The first step was to orient the UFS z-axis down vertically and hang a set of six incremental weights while collecting digital UFS output. Then the UFS was oriented such that the UFS z-axis pointed toward the ceiling and the same incremental weights were stacked while collecting digital UFS output. This procedure was repeated for the UFS y-axis. The F_y and F_z digital output was plotted against the known weights applied in those directions (**Figure 30**) and linear relationships describing the y- and z-axis force calibration were found:

$$F_y = 0.0051733 DO_{F_y} - 0.29147$$

$$F_z = 0.013194 DO_{F_z} - 0.26728,$$

where F_y and F_z are the forces in the UFS y- and z-directions (in Newtons), respectively, and DO_{F_y} and DO_{F_z} are the digital outputs from the UFS in the y- and z-directions, respectively. It is important to note that the above equations do not correct for orientation effects.

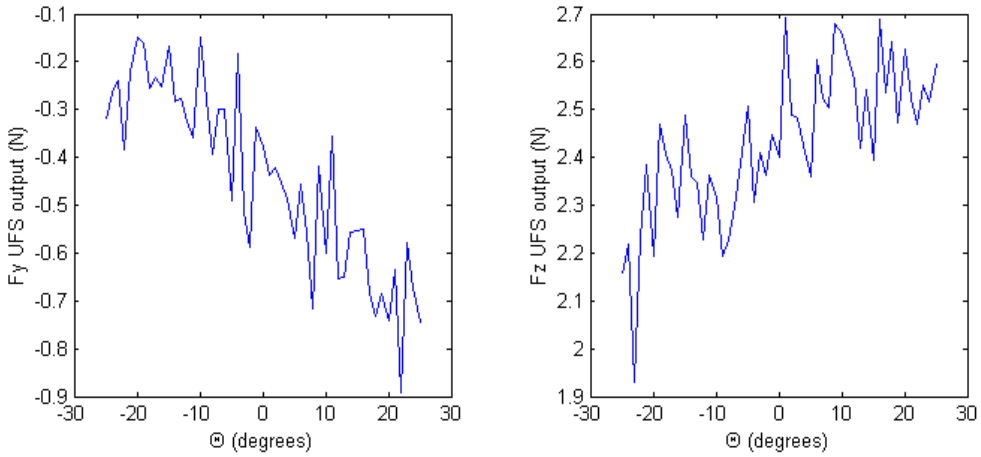


Figure 29 Plot of output from UFS y-axis and z-axis force channel vs. UFS orientation (Θ) when UFS is rotated in 1° increments about its x axis (with nothing attached)

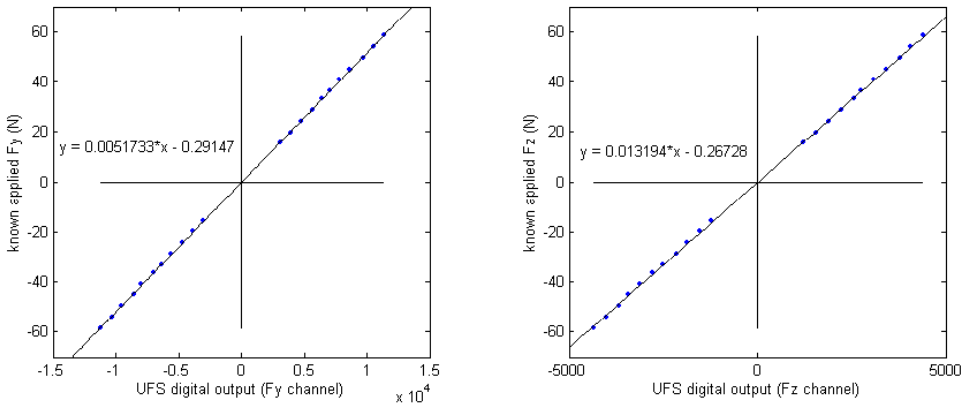


Figure 30 Plot of known applied weight vs. UFS digital output

For the second part of the protocol, the incremental weights used in the first part were attached to the UFS. The UFS was rotated about its x-axis from $\theta = -25°$ to $25°$ while collecting the digital output in the y- and z-directions. DO_{F_y} and DO_{F_z} were inserted into the above set of equations to obtain linearly calibrated F_y and F_z in Newtons. The known applied weights were subtracted from the linearly calibrated F_y and F_z to get the error in y- and z-axis force measurements. The errors for each incremental weight were averaged and plotted against the rotation angle (**Figure 31**). The orientation error was found to be a linear function of the rotation angle:

$$F_y \; error = 0.0085025\theta + 0.14779$$

$$F_z \; error = 0.0012932\theta + 0.19311$$

To correct for orientation effects, the first-order mean error function was subtracted from the linear y- and z-axis calibration:

$$F_y\left(\text{corrected}\right) = \left[0.0051733DO_{F_y} - 0.29147\right] - \left[0.0085025\theta + 0.14779\right]$$

$$F_z\left(\text{corrected}\right) = \left[0.013194DO_{F_z} - 0.26728\right] - \left[0.0012932\theta + 0.19311\right]$$

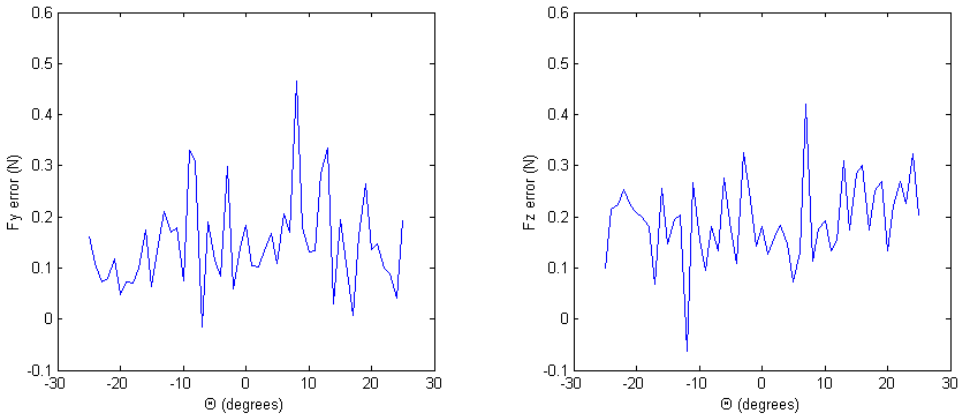


Figure 31 Plots of average Fy and Fz error vs UFS orientation

For the third part of the protocol, the two orientation correction equations were applied to the digital output collected in the second part. Plots of F_y and F_z measured using (1) the linear calibration equations and (2) the orientation corrected equations were plotted against the known applied weights (**Figure 32** and **Figure 33**). Linear regressions showed that both equations resulted in a significant improvement over using raw UFS output for F_y and F_z. However, there was not a significant improvement when using the orientation corrected equation versus the linear calibration equation.

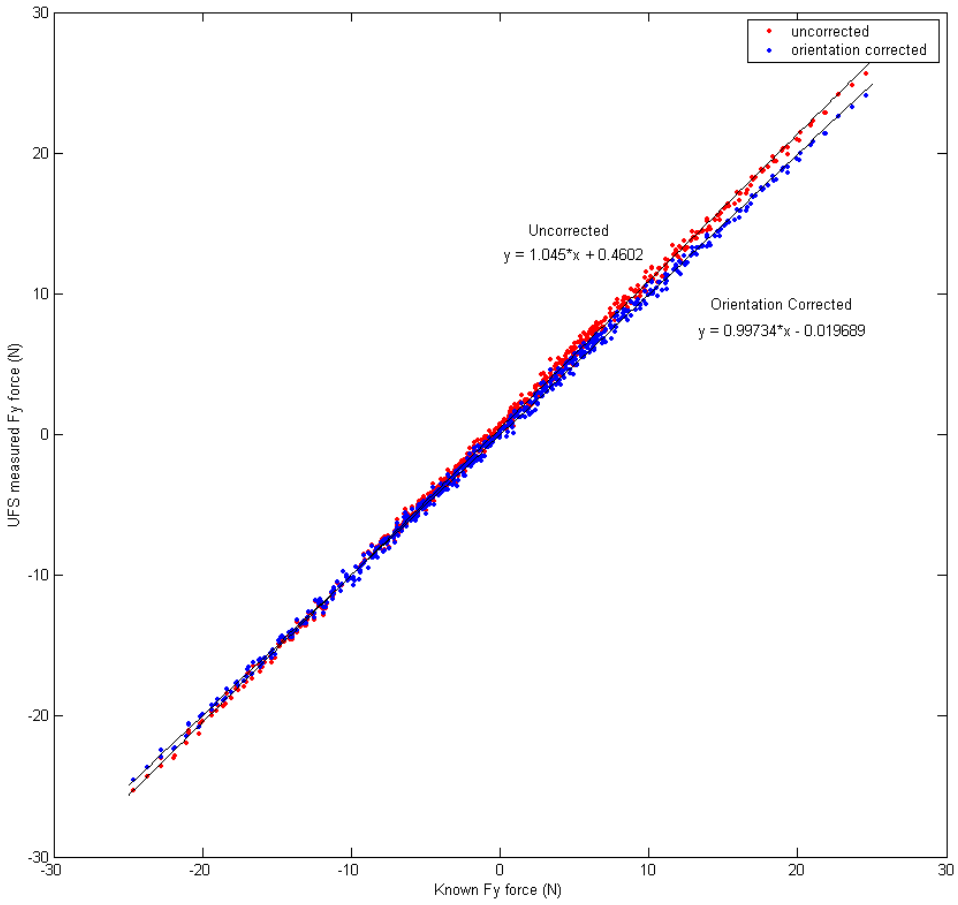


Figure 32 Plot of UFS measured F_y force vs. known F_y force

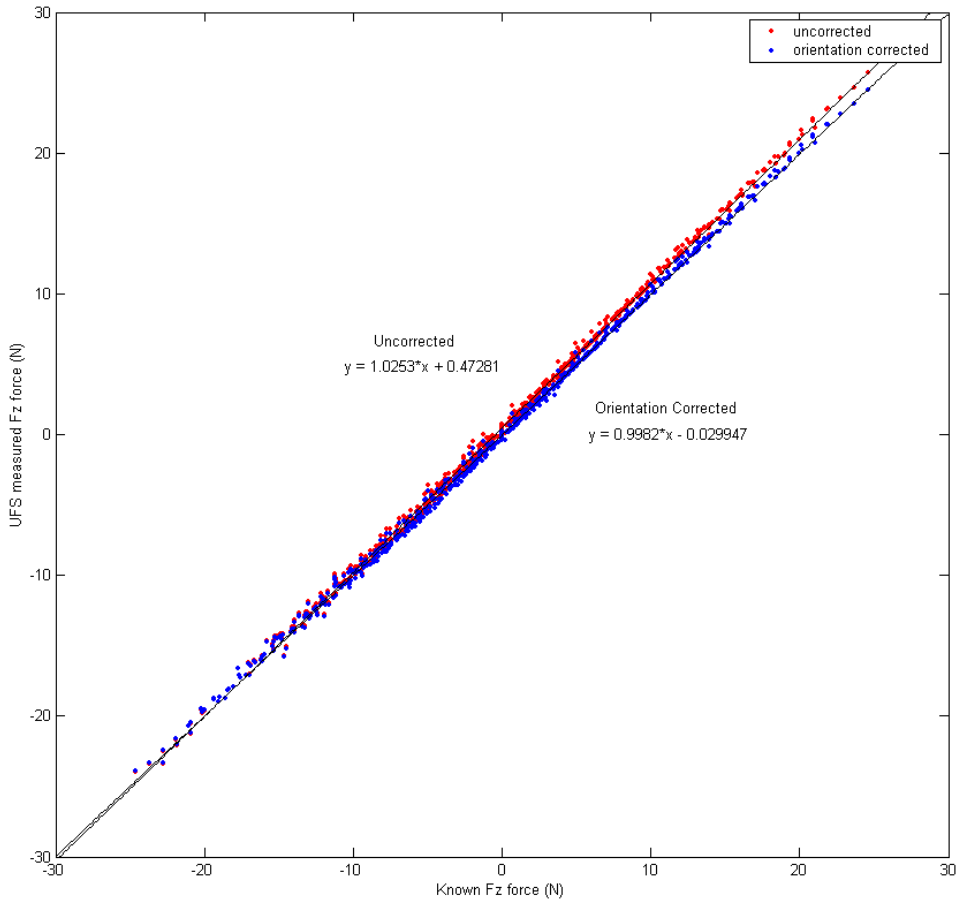


Figure 33 Plot of UFS measured F_z force vs. known F_z force

In conclusion, as long as the error in load cell output is within the manufacturer's stated accuracy, it is not necessary to perform calibration at the beginning of each testing day. If, however, the load cell calibration protocol needs to be performed, a linear calibration equation may be applied to digital output in the y- and z-directions without applying an orientation correction. The entire calibration protocol takes a lot of time to complete, so not applying an orientation correction cuts the time required to finish the protocol by more than half. If the UFS

is mishandled in any way, the entire protocol should be performed to verify orientation correction does not need to be performed.

5.4 Manipulator Accuracy and Precision

Spatial resolution of a robot refers to the smallest change in position that the feedback sensor can detect when a normal distribution of mechanical inaccuracies, such as backlash and joint bending, are considered. Accuracy refers to the ability of the manipulator to get to a commanded point in space and can be considered half of the spatial resolution. Precision (repeatability) is the ability of the manipulator to repeatedly return to a point, regardless of whether or not it is the correct point. It is possible for a robot to have high precision, but poor accuracy. In fact, this is generally the case. Accuracy of robots is generally unreported and assumed to be poor. This is well known, but it hasn't been of too much concern because industrial applications (spot welding, pick-and-place) usually rely on the robot's precision, which is typically very high, to repeatedly move to a taught point. If the commanded points are not taught, but defined in Cartesian space, accuracy becomes an issue. With the integration of robotic technology into biomedical applications, such as in vitro musculoskeletal joint testing, robot assisted surgery and rehabilitation, high accuracy is necessary because the required motion of the end-effector is not known beforehand.

In biomechanical testing of joints, the passive path of the specimen is not known *a priori*. The force minimized points must be stored during pathseek so that they can be returned to repeatedly for multiple replays. Our manipulator has high precision, so if no other factors are considered, the robot would appear to be returning to the same force minimized positions for every replay. However, a precise manipulator is not necessarily an accurate one. This means the

manipulator returns to the same point in space again and again, even if the point is not the stored force minimized one.

A manipulator's compliance describes the degree of displacement of the wrist when a load is applied or removed and is the inverse of stiffness. If a manipulator is very compliant, it is not stiff, and vice versa. Manipulators that are compliant can generally make smaller motions than manipulators that are very stiff, but the wrist can displace more when a static load is applied or removed.

When cutting studies are performed, the load on the end-effector changes, typically within the range of ± 30 N and ± 6 N-m. For an infinitely stiff robot (or least one with a very high payload), this would not be an issue. However, our robot has a relatively low payload (6 kg) and the change in end-effector position with changes in load is visible. When cutting structures on the specimen, and hence remove load from the end-effector, the end-effector visibly springs up. If the robot cannot accurately tell the difference between its position before and after a structure is cut, even though there is an obvious change, then it is unlikely the manipulator will return to the force minimized positions stored for the intact specimen. This is a problem because compliance in the arm may be causing additional loads in the intact structures, which would cause us to underestimate the loads associated with cutting them.

It was hypothesized that significant differences exist in the positional accuracy for varying fractions of payload, that a function exists to describe the relationship between position error and weight for a unique end-effector position, and that this function may be used to correct for position error based on external load cell data. If the manipulator is capable of making the presumably small displacements required to correct for joint laxity/backlash, then an external

measurement system can be used to correct for poor accuracy. To investigate our robot's accuracy and its ability to improve (if needed), three tests were devised.

Test 1: First, the relationship between positional accuracy and precision for varying fractions of maximum load capacity was examined in one degree of freedom (DOF). The end-effector was placed so that the y-axis pointed towards the ceiling. A weight equal to 1/2 of maximum payload was attached to the end-effector. This weight was designated 1/2W. A dial gauge (0.01 mm resolution, 10 mm travel) was rigidly fixed to a rigid table. The manipulator was moved to a position such that the weight attached to the end-effector depressed the dial gauge to 5 mm. This reference position (point A) was saved as a Cartesian coordinate. The manipulator was then moved to a position 40 mm directly above point A. This ensured that there was sufficient clearance between the dial gauge plunger and the weight on the end-effector so that the weight did not touch the plunger when the manipulator was at this point. This position (point B) was also saved as a Cartesian coordinate.

The end-effector moved from point B to point A $n = 30$ times. Each time the end-effector reached point A, the dial gauge reading and the manipulator's own sense of position were recorded. The dial gauge reading was within the manufacturer stated repeatability each time the plunger was depressed. This process was repeated for weights equal to maximum payload (W), 3/4 payload (3/4W), 1/4 payload (1/4W) and no load (0W). The mean dial gauge reading for each fraction of payload was found, with the mean for 1/2W being the reference that all other weights were compared to. The position error (difference between the mean dial gauge reading and the reference mean) was plotted against fraction of payload. This gave the relationship between position error and weight attached to the end-effector.

Test 2: The position error from test 1 was significant, so the possibility of correcting this error in one DOF was inspected. The end-effector moved from point B to point A $n = 30$ times with weight 0W, 1/4W, 3/4W or W attached. An incremental displacement of $n * 0.01$ mm was applied in the positive or negative y-direction (depending on the dial gauge reading) each time the end-effector reached point A. The dial gauge reading and the robot's sense of position were recorded before and after each incremental displacement. The mean difference between the prescribed displacement and the actual displacement for the dial gauge and the robot's sense of position was computed for each weight. The actual displacement was plotted against the prescribed displacement for the dial gauge readings and the robot's sense of position. A function describing the relationship between the prescribed displacement and the actual displacement was then found.

Test 3: Results from test 2 showed that it was possible to correct for poor positional accuracy using prescribed displacements, so a final test was performed to determine if data from an external load cell could be used to calculate the displacement necessary to correct for position error. For all weights (0W, 1/4W, 1/2W, 3/4W, W), the end-effector moved from point B to point A $n = 30$ times. Using the relationship between position error and weight (from test 1), the relationship between prescribed displacement and actual displacement (from test 2) and load cell data, a displacement was applied in the y-direction if needed to move the end-effector to the reference dial gauge position (5 mm). The dial gauge readings and the robot's sense of position were recorded before and after the displacements were applied. The dial gauge readings after the displacements were applied were averaged. The difference between the mean dial gauge readings after displacement and the reference mean were plotted against the fraction of weight, as in test 1.

Figure 34 shows that position error (in mm) is a linear function of the fraction of payload attached to the end-effector:

$$error = 0.0058 * (\%\,\max\,\text{payload}) - 0.28$$

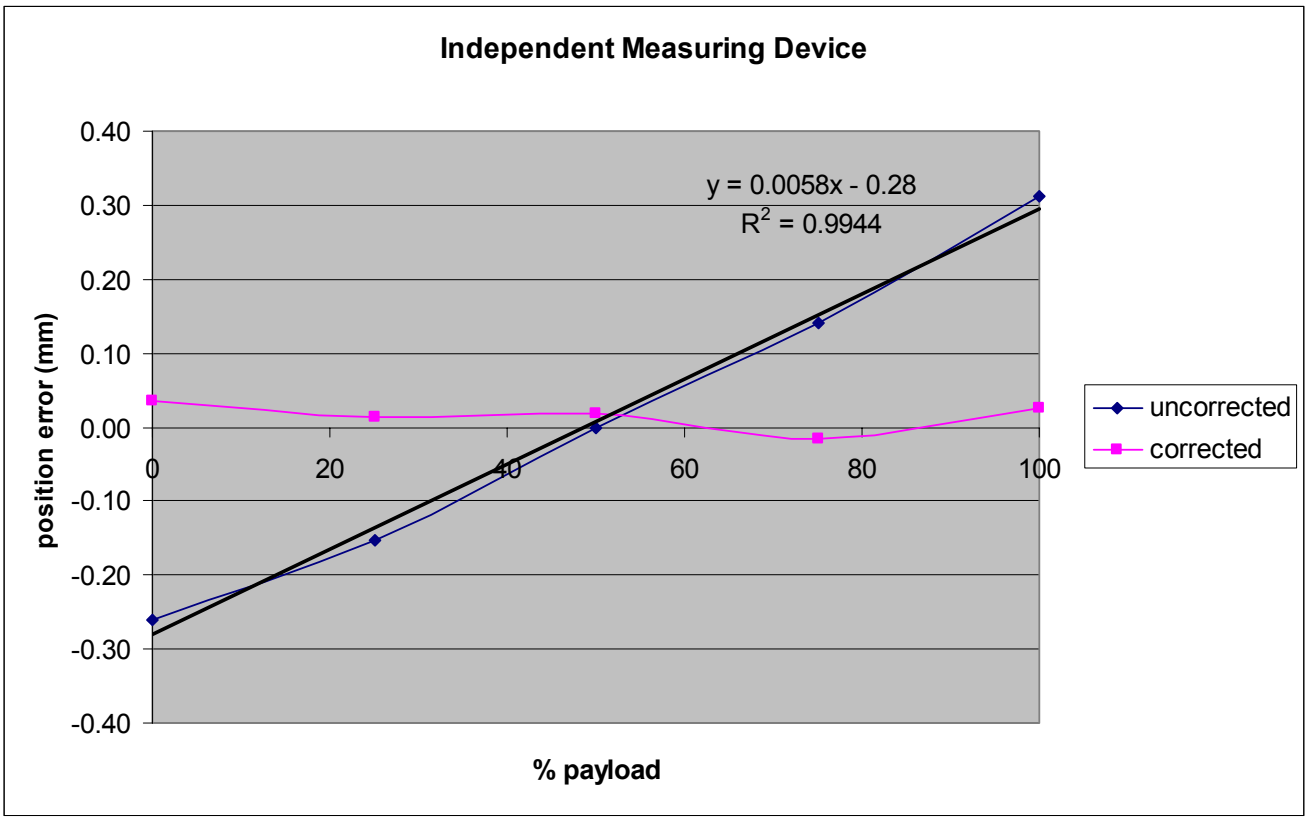


Figure 34 Position error, as measured by an external dial gauge, is a linear function of the weight on the end-effector (blue line). This error may be corrected for (magenta line).

Figure 35 shows that the actual displacement is linearly related to the prescribed displacement for dial gauge measurements. It should have a unit slope with a zero intercept. t-tests were performed to determine whether the slopes and intercepts of the linear regressions of each weight are equal to one and zero, respectively. The slope of each linear regression is not significantly different from one for every weight but 1/4W, and the intercept of each linear regression is not significantly different from zero for every weight but W.

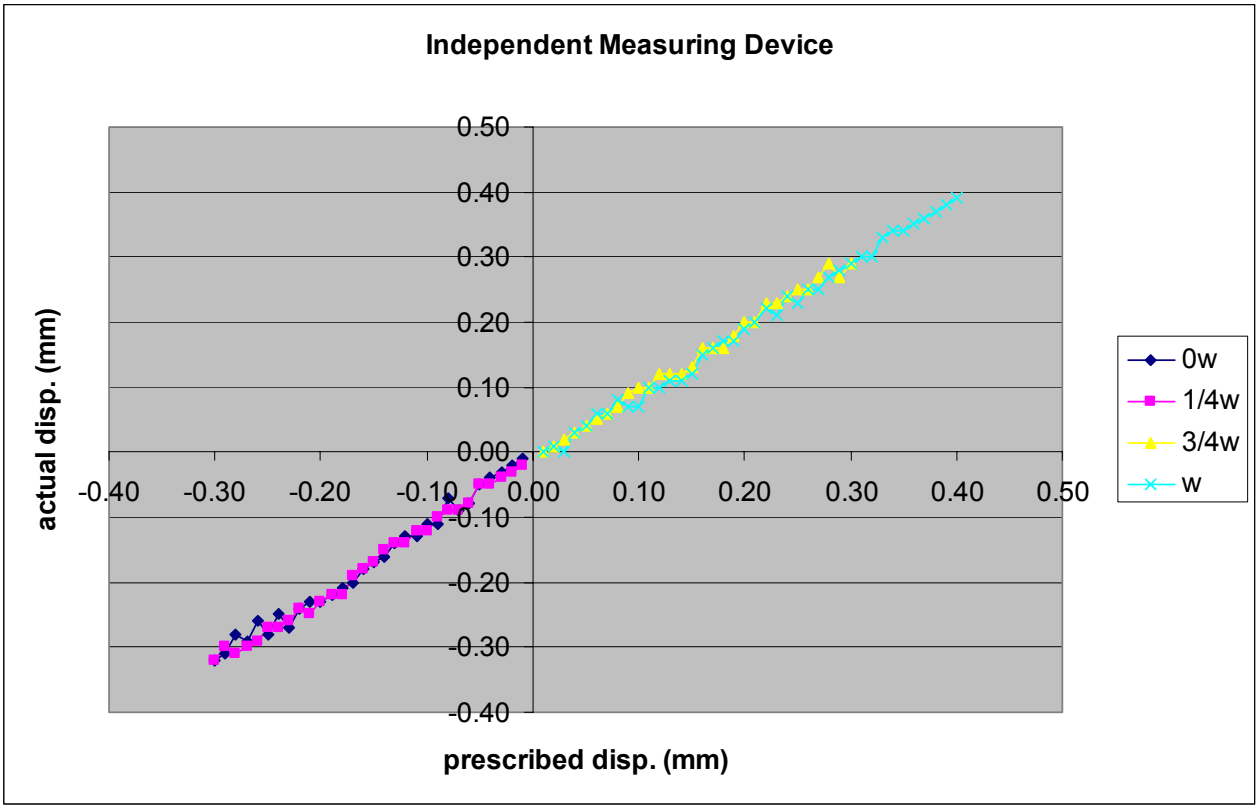


Figure 35 The ratio between the prescribed displacement of the end-effector and the actual displacement is 1:1, as measured using a dial gauge.

Figure 36 shows that the actual displacement is also linearly related to the prescribed displacement for the robot's own sense of position. t-tests were performed to determine whether the slopes and intercepts of the linear regressions of each weight are equal to one and zero, respectively. For every weight, the slope and intercept of each linear regression is not significantly different from one and zero, respectively.

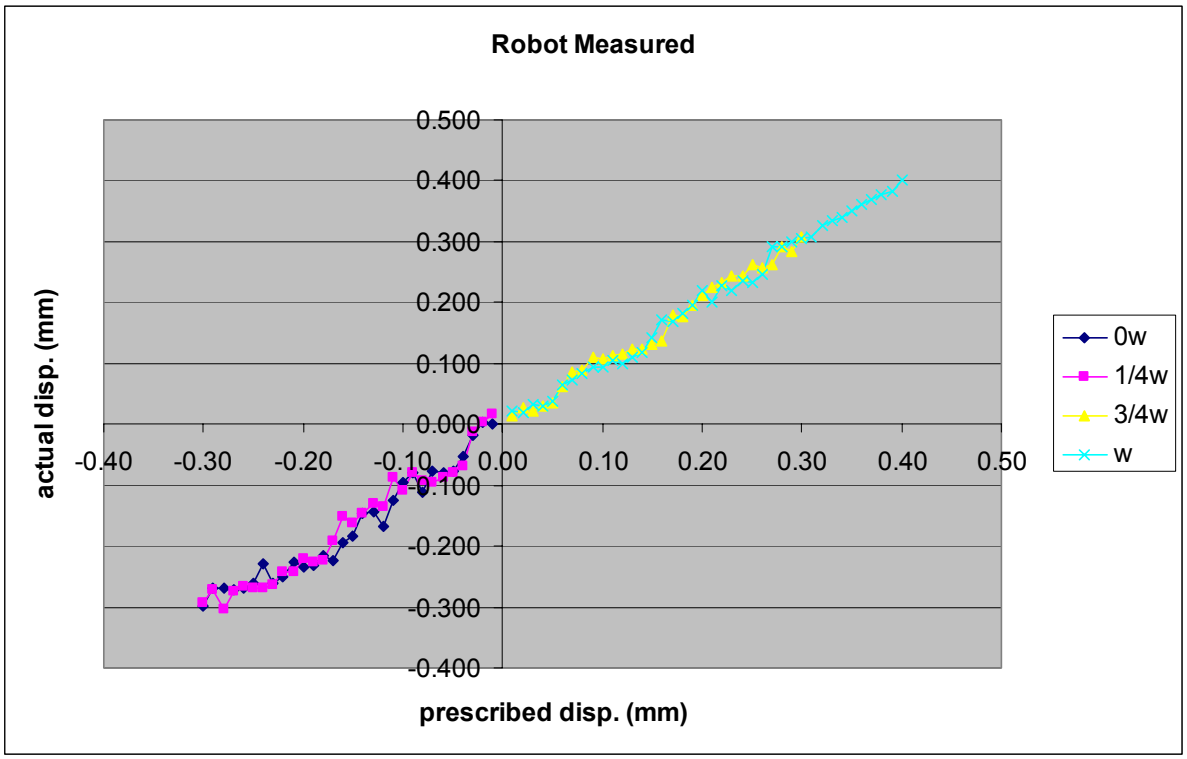


Figure 36 The ratio between the prescribed displacement of the end-effector and the actual displacement is 1:1, as measured using the robotic controller.

Figure 34 shows that an algorithm using external load cell data can be applied to reduce the position error to nearly zero.

5.5 Homogeneous Transformations Defined for Robot Testing System

Homogeneous transformations similar to those developed for the general rigid body-spring model are now developed for the robotic testing system.

5.5.1 Homogeneous Transformation of $(xyz)_{TCS}$ with Respect to $(xyz)_{UFS}$

$$T_{UFS}^{TCS}$$

This transformation is user-defined. See **Figure 37**.

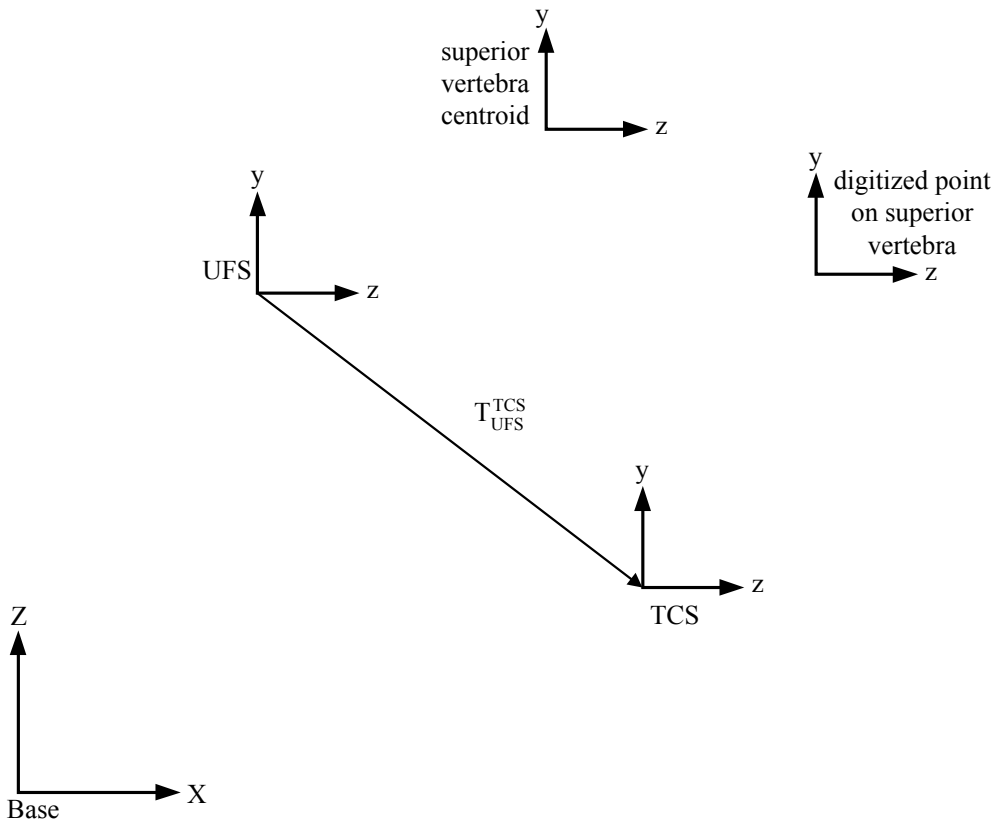


Figure 37 Transformation of $(xyz)_{TCS}$ with respect to $(xyz)_{UFS}$

5.5.2 Homogeneous Transformation of $(xyz)_{TCS}$ with Respect to XYZ

$$T_G^{TCS}$$

This transformation is known through the robot's "WHERE" or "HERE" commands.

See **Figure 38**.

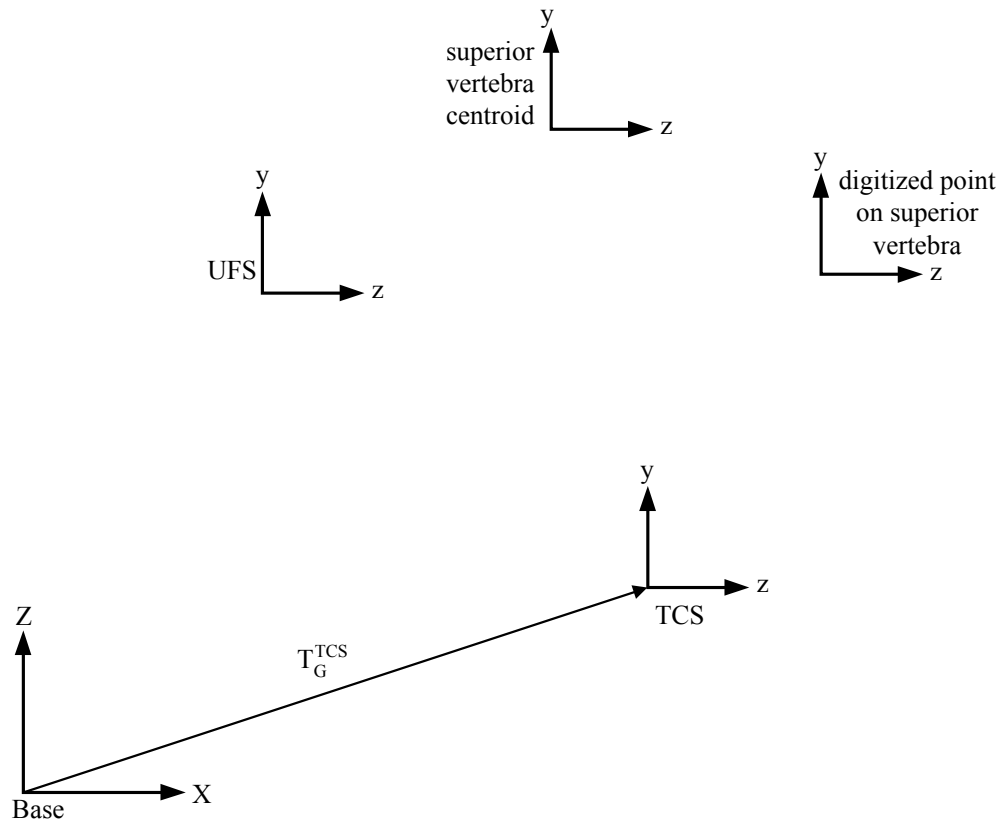


Figure 38 Transformation of $(xyz)_{TCS}$ with respect to XYZ

5.5.3 Homogeneous Transformation of $(xyz)_{TCS}$ with Respect to $(xyz)_{UFS}$

$$T_G^{UFS} = T_G^{TCS}\left(T_{UFS}^{TCS}\right)^{-1}$$

See **Figure 39**.

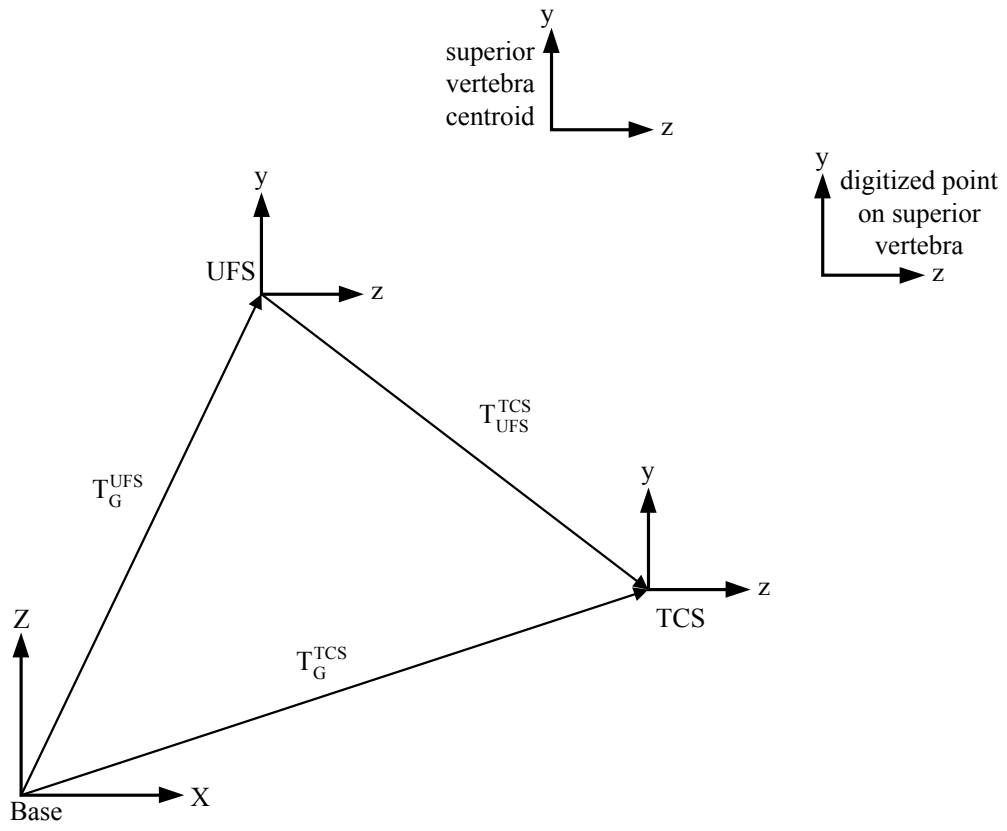


Figure 39 Transformation of $(xyz)_{TCS}$ with respect to $(xyz)_{UFS}$

5.5.4 Homogeneous Transformation of $(xyz)_i$ with Respect to $(xyz)_0$

$$T_0^i$$

This transformation is user-defined (known through X-rays). See **Figure 40**.

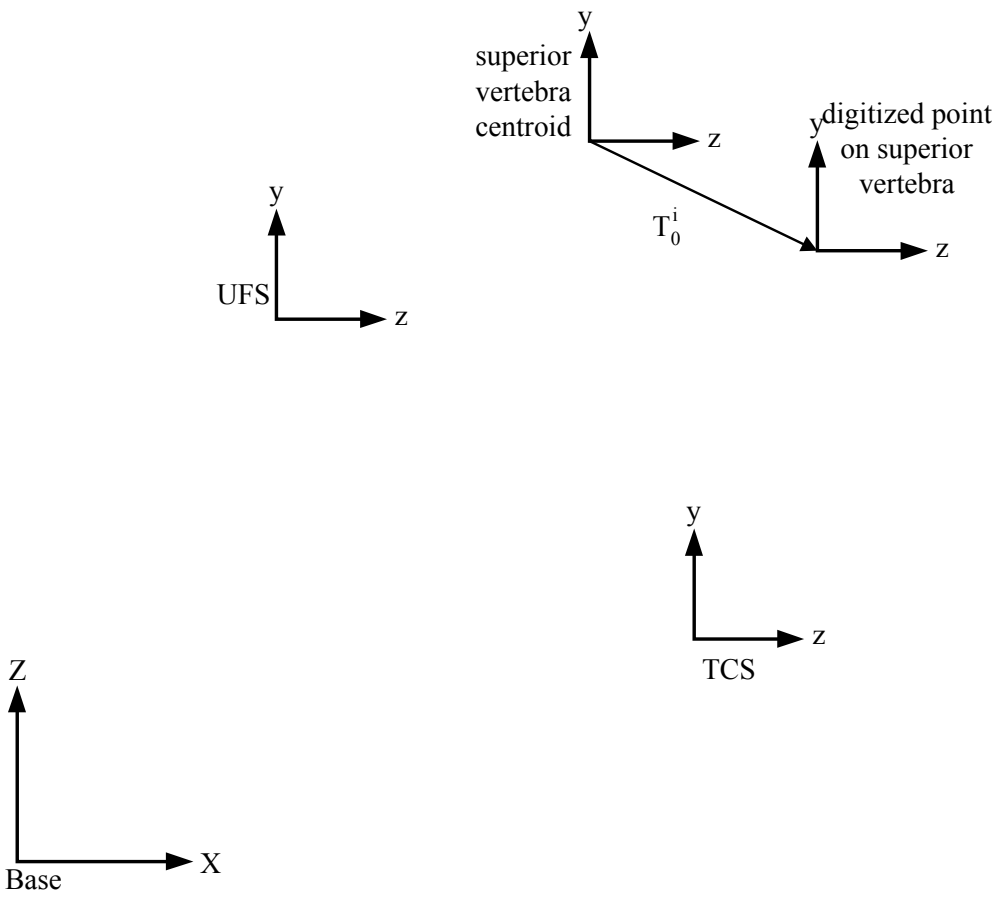


Figure 40 Transformation of $(xyz)_i$ with respect to $(xyz)_0$

5.5.5 Homogeneous Transformation of $(xyz)_i$ with Respect to $(xyz)_{UFS}$

$$T_{UFS}^{i}$$

This transformation is known through digitizing points on the vertebra and UFS (or fixture) with Microscribe. See **Figure 41**.

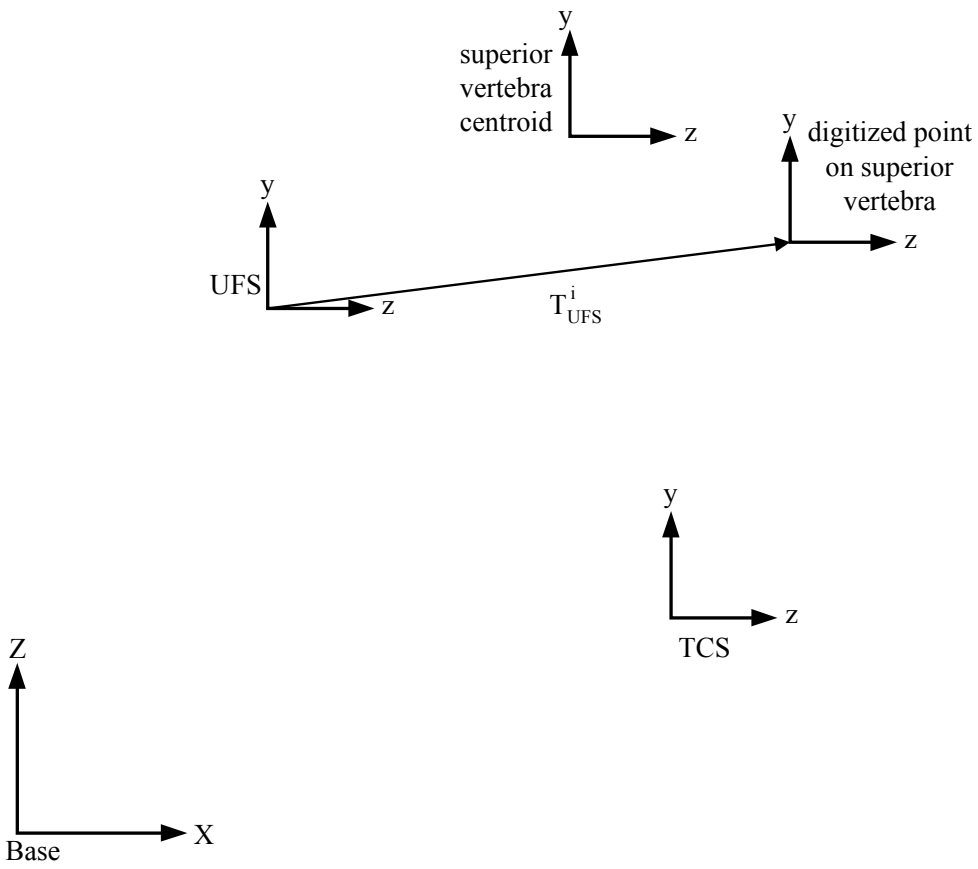


Figure 41 Transformation of $(xyz)_i$ with respect to $(xyz)_{UFS}$

5.5.6 Homogeneous Transformation of $(xyz)_0$ with Respect to $(xyz)_{UFS}$

$$T^0_{UFS} = T^i_{UFS}\left(T^i_0\right)^{-1}$$

See **Figure 42**.

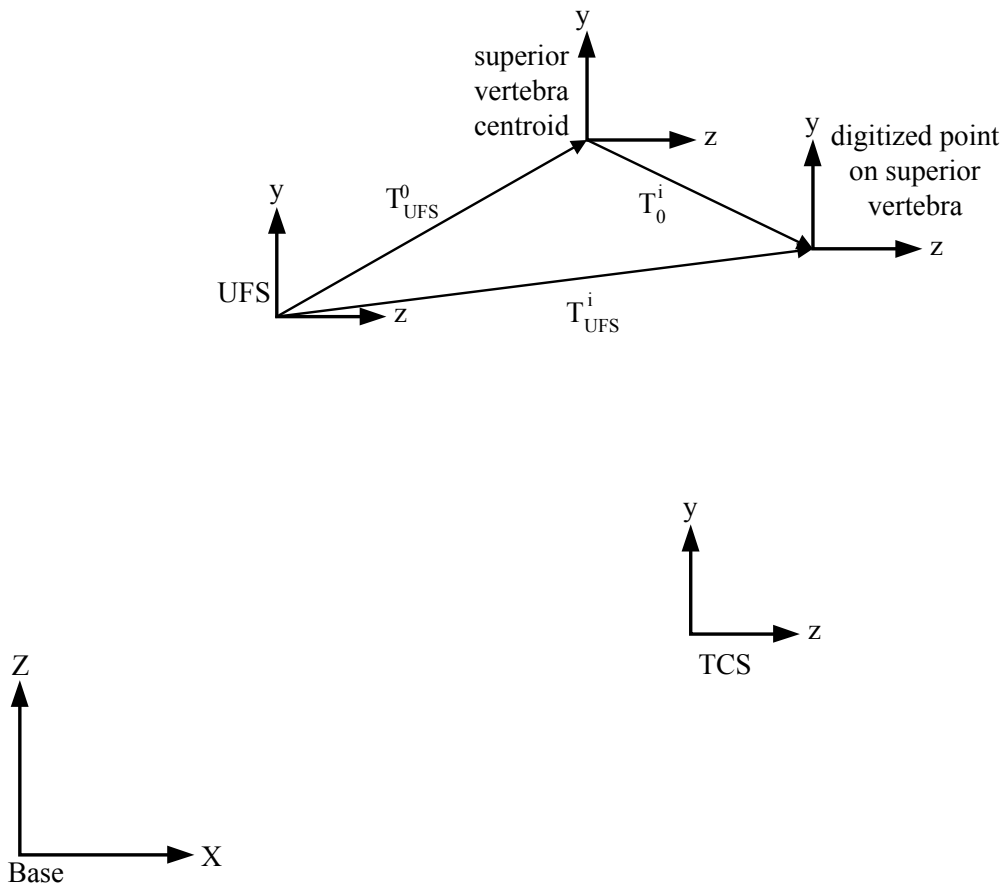


Figure 42 Transformation of $(xyz)_0$ with respect to $(xyz)_{UFS}$

5.5.7 Homogeneous Transformation of $(xyz)_0$ with Respect to XYZ

$$T_G^0 = T_G^{UFS} T_{UFS}^0$$

See **Figure 43**.

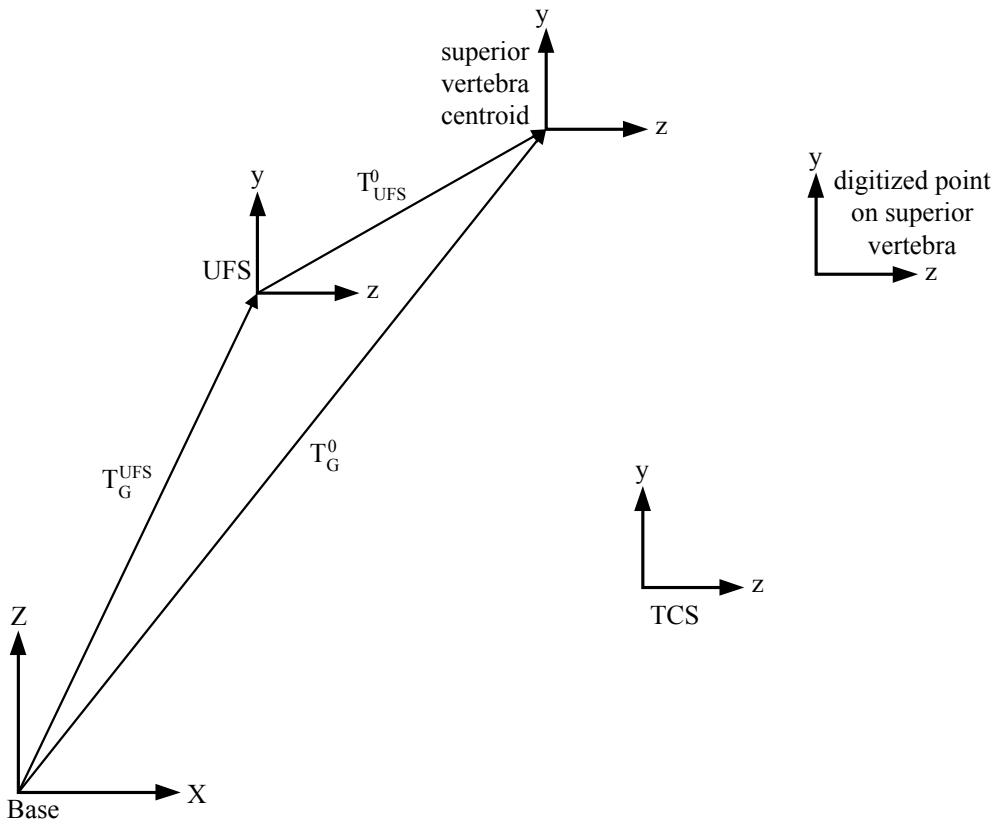


Figure 43 Transformation of $(xyz)_0$ with respect to XYZ

5.5.8 Homogeneous Transformation of $(xyz)_i$ with Respect to XYZ

$$T_G^i = T_G^0 T_0^i$$

See **Figure 44**.

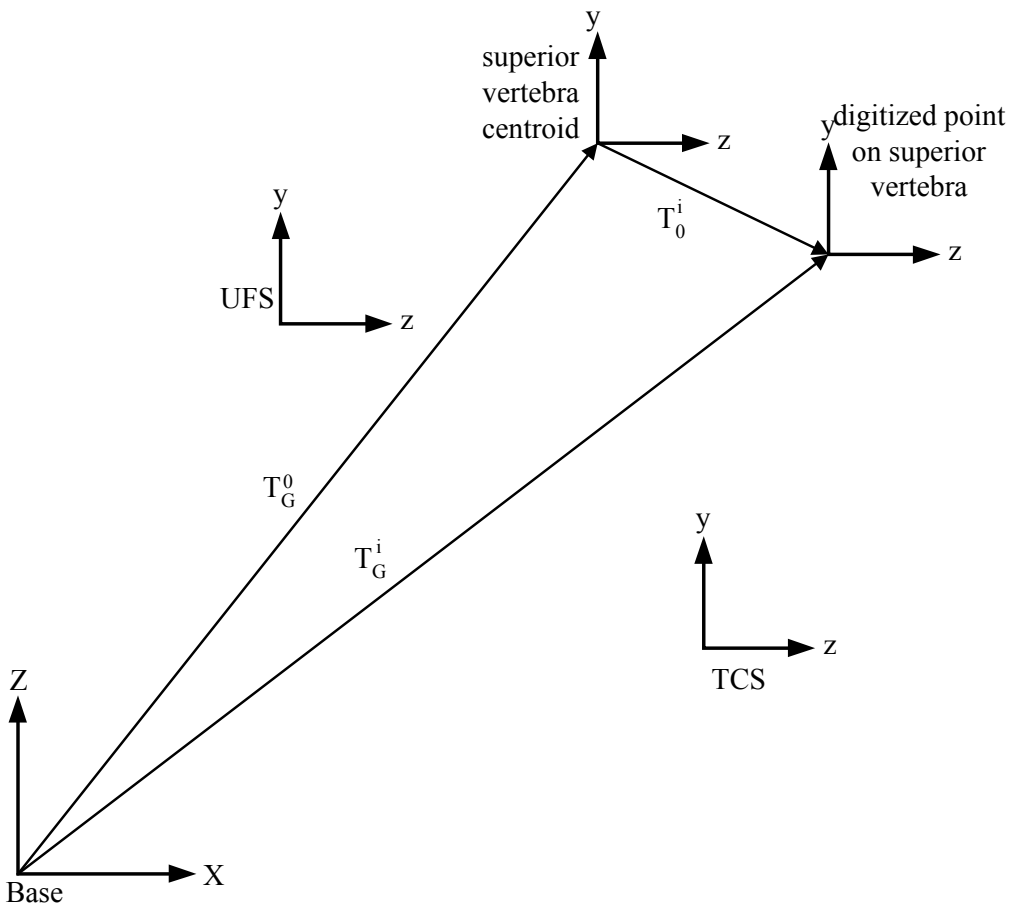


Figure 44 Transformation of $(xyz)_i$ with respect to XYZ

6.0 APPLICATION OF ANALYTICAL PLATFORM TO DEVELOPMENT AND TESTING OF NEW CONTROL METHODS

The robotic/UFS testing system is operated in a hybrid control mode for the determination of the path of passive flexion/extension of a spinal specimen. The hybrid control algorithm used in the current study is shown schematically in **Figure 45**.

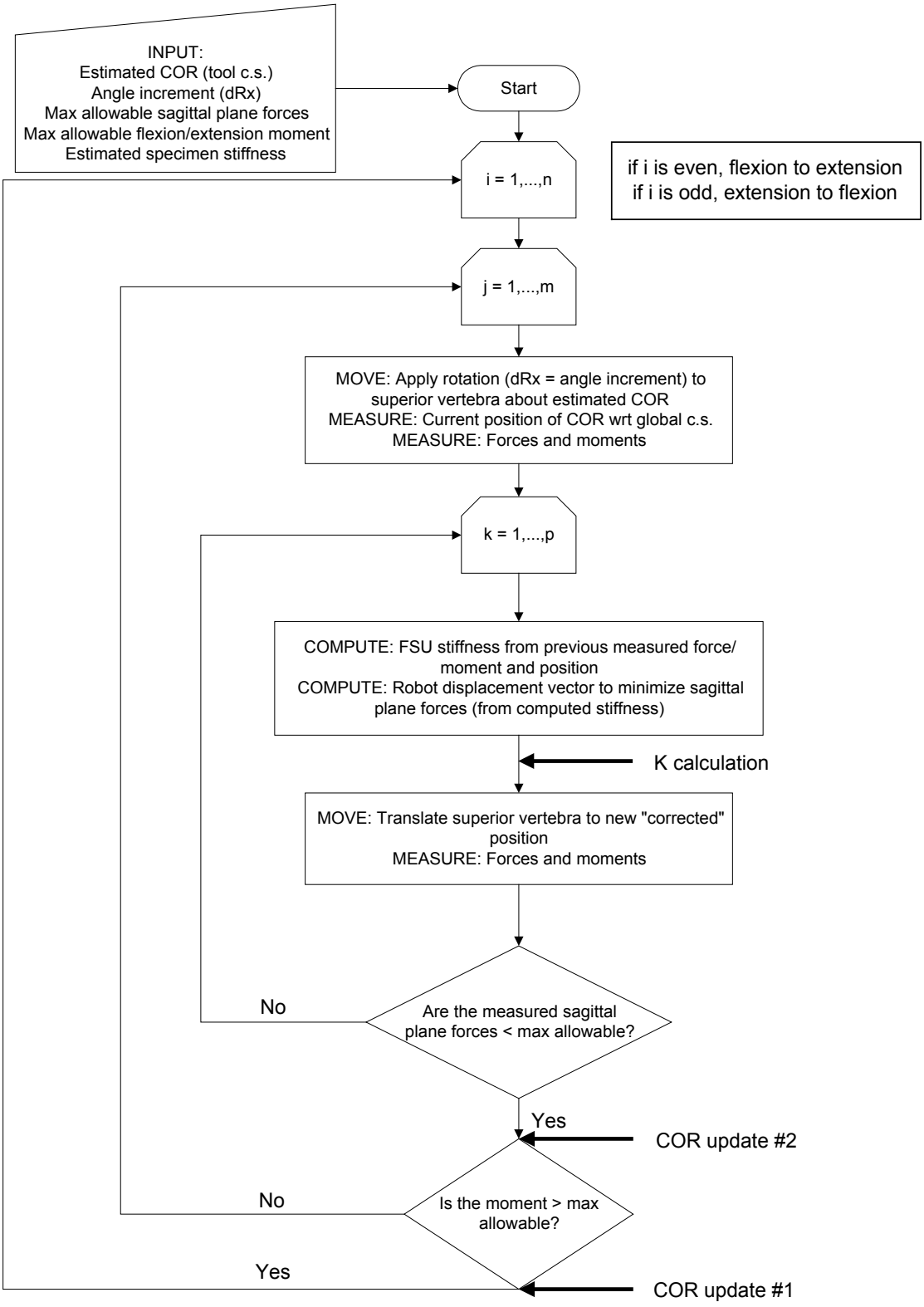


Figure 45 Hybrid control flowchart

As shown in **Figure 45**, the hybrid control testing algorithm consists of an outer loop (displacement control) and an inner loop (load control). There are several inputs to the algorithm: TCS position and orientation, position and orientation of any nodes of interest, rotation increment size, force threshold, maximum number of force minimizing iterations, maximum flexion/extension moment and maximum number of flexion/extension cycles. Once these parameters are input, the hybrid control algorithm begins. During hybrid control, the passive path of the specimen is found and stored for replay. The specimen begins at a neutral zero-load position. An incremental rotation is applied to the superior vertebra about the TCS x-axis to produce planar flexion. If the force created during the rotation is above the user-defined threshold, the superior vertebra translates in the TCS yz-plane until either the force is minimized below the threshold or the maximum number of iterations is reached. When the load control loop finishes, the force minimized position is stored for replay later and the flexion moment is compared to the maximum flexion/extension moment. If the moment at the end of the load control loop has not been greater than the maximum flexion/extension moment three times, the rotation direction remains flexion and incremental rotations continue to be applied until full flexion. If the moment *has* been greater than the maximum three times, the specimen is considered to be at full flexion and the rotation direction changes to extension. The process is the same for full flexion to full extension. One complete flexion/extension cycle is full flexion $\rightarrow$ full extension $\rightarrow$ full flexion. When finding the passive path of the specimen, it undergoes preconditioning because the flexion/extension cycles continue until the maximum number of flexion/extension cycles has been met or the moment and rotation angle at full flexion and full extension from one cycle to the next do not change by more than 4%.

To fully validate and characterize the rigid body-spring model, 3 sets of comprehensive simulations were performed:

Set 1: Use transformation development from section 4.2 to validate model. (**Figure 46**)

Set 2: Use transformation development to characterize model in pure displacement control. (**Figure 47**)

Set 3: Use analytical stiffness matrix from section 4.2 to characterize model in load control. (**Figure 48**)

Test #	**Input parameter varied**	**Testing procedure**	**Output parameter of interest**	**Expected outcome**
1a	Translation of center of bar	Create 13x13 grid of points ($-60 \le X \le 60$ mm and $-60 \le Y \le 60$ mm). Translate center of bar to each point.	X,Y position of center and both ends of bar	X,Y position of bar matches grid of points. X,Y position of left end of bar is –L cm away from the center. X,Y position of right end of bar is L cm away from the center.
			Sign of Fx,Fy,Mz	For xlations along X axis: Fy=0, Fx=neg. for pos. xlations, Fx=pos. for neg. xlations, Mz=0
				For xlations along Y axis: Fx=0, Fy=neg. for pos. xlations, Fy=pos. for neg. xlations, Mz=0
				For xlations in 1st quad: Fx=neg., Fy=neg., Mz=pos.
				For xlations in 2nd quad: Fx=pos., Fy=neg., Mz=neg.
				For xlations in 3rd quad: Fx=pos., Fy=pos., Mz=pos.
				For xlations in 4th quad: Fx=neg., Fy=pos., Mz=neg.
			Magnitude of loads in each spring	Large mag. for points far from fixed end of spring, decreasing mag. for points near fixed end
			Magnitude of loads on bar	Mag. grows for points far from origin
			Potential energy (work)	High energy for points far from origin

			Analytical stiffness matrix (Kxx,Kxy,Kyy)	For xlations along X axis: Kxx=sum of spring constants, Kyy=0
1b	Rotation increment	Use same grid from test 1a for COR locations. Rotate once around random CORs by $\phi = 0°, 30°, 45°, 90°$ $120°, 180°, 210°,$ $270°, 360°$	X,Y position of center and both ends of bar	Check against hand calculations for selected CORs.
			Sign of Fx,Fy,Mz	For COR in 1st quad: Fx=neg.,Fy=pos.,Mz=pos. For COR in 2nd quad: Fx=neg.,Fy=neg.,Mz=neg. For COR in 3rd quad: Fx=pos.,Fy=neg.,Mz=pos. For COR in 4th quad: Fx=pos.,Fy=pos.,Mz=neg.
			Magnitude of loads in each spring	Large mag. for points far from fixed end of spring, decreasing mag. for points near fixed end
			Magnitude of loads on bar	Mag. grows for points far from origin
			Potential energy (work)	High energy for points far from origin
			Analytical stiffness matrix (Kxx,Kxy,Kyy)	Nonlinear stiffness terms

Figure 46 Validate Matlab simulations for rigid body-spring model

Test #	**Input parameter varied**	**Testing procedure**	**Output parameter of interest**	**Expected outcome**
2a	COR location	Create 13x13 grid of CORs ($-60 \le X \le 60$ mm and $-60 \le Y \le 60$ mm). Rotate around each COR once by $30°$, $10°$ and $1°$.	Sign of Fx,Fy,Mz	Results similar to test 1b.
			Magnitude of loads in each spring	
			Magnitude of loads on bar	
			Potential energy (work)	
			Analytical stiffness matrix (Kxx,Kxy,Kyy)	Results similar to test 1b.
2b	Rotation increment	Use CORs from above grid. Rotate about each	Magnitude of loads in each spring	Magnitude of loads (in each spring and on bar) and energy decreases with decreasing

		Rotate about each COR in $\phi = 1°, 0.5°, 0.25°$ increments until reach $\Phi = 30°$. After each increment, translate center of bar to global origin to minimize forces.	Magnitude of loads on bar	decreases with decreasing rotation increment. Stiffness terms do not change.
			Potential energy (work)	
			Analytical stiffness matrix (Kxx,Kxy,Kyy)	
2x	COR location	Use CORs from above grid. Rotate about each COR in $\phi = 1°$ increments until reach $\Phi = 5°$. After each increment, translate center of bar to global origin to minimize forces.	Magnitude of loads in each spring	Use this test to show that force can be minimized by translating bar to global origin.
			Magnitude of loads on bar	
			Potential energy (work)	
			Analytical stiffness matrix (Kxx,Kxy,Kyy)	

Figure 47 Characterize rigid body-spring model in displacement control

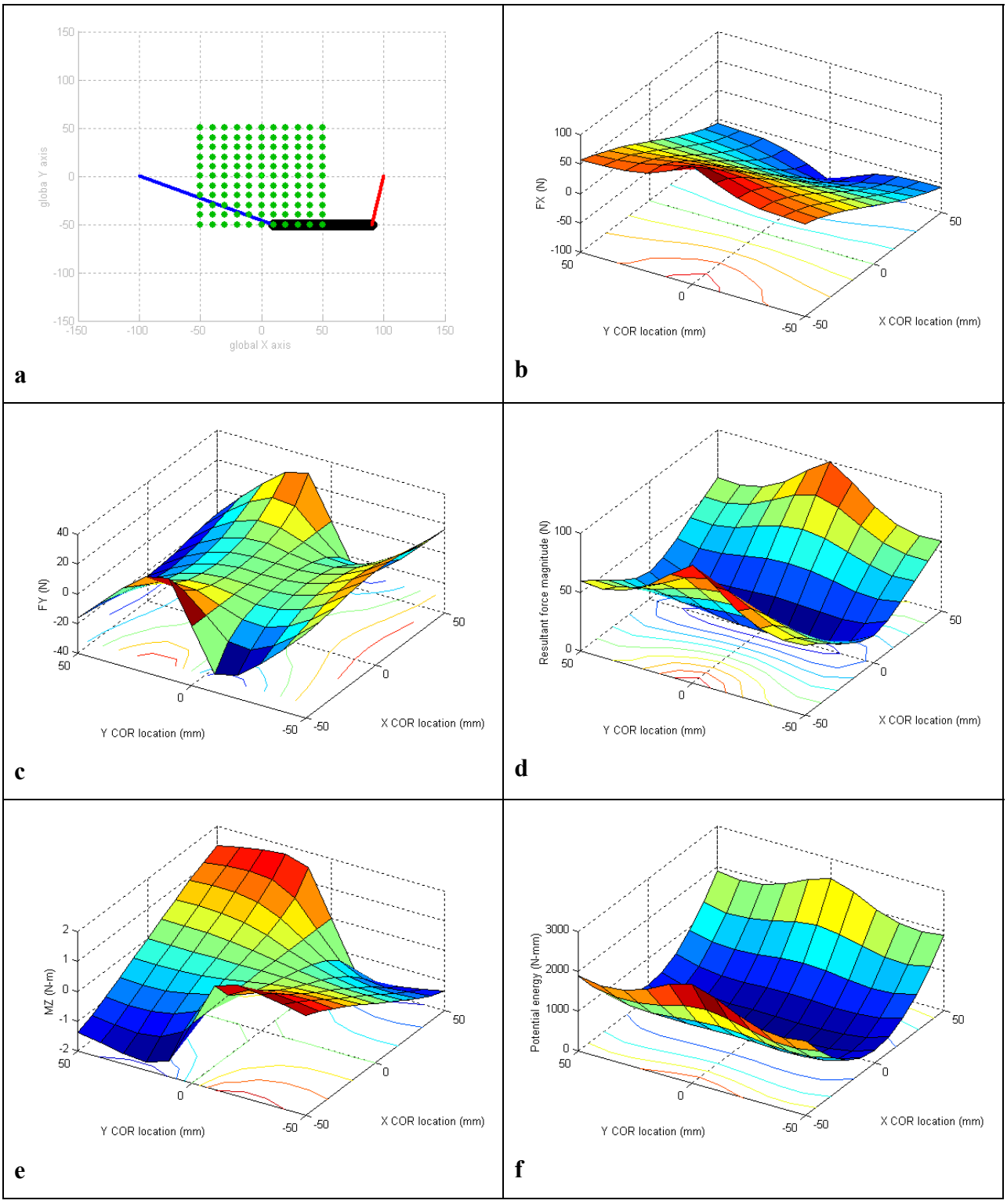
Test #	**Input parameter varied**	**Testing procedure**	**Output parameter of interest**	**Expected outcome**
3	COR location	Create 13x13 grid of CORs ($-60 \le X \le 60$ mm and $-60 \le Y \le 60$ mm). Rotate around each COR by $\phi = 1°$ until reach $\Phi = 30°$. Use analytical stiffness matrix for load control.	No. of iterations to reach minimized force	Two iterations
			Distance of center of bar from true force min. position	Very close to zero
			Potential energy (work)	Very similar to test 1b

Figure 48 Characterize rigid body-spring model in load control

6.1 Displacement Control Loop of Hybrid Control Algorithm

In the following sections, the rigid body-spring model's behavior during pure displacement control will be fully characterized using Matlab simulations. Outcome measures are the moment and peak force created during rotation, potential energy in the system and analytical global stiffness matrix. After this characterization, several potential enhancements to the displacement control loop will be investigated. Three methods of calculating the model's preferred COR are evaluated. Two methods of updating the user-chosen COR to the calculated preferred COR are also examined. One method of calculating the preferred COR and one method of updating the COR will be chosen and incorporated into a new hybrid control algorithm.

Before the rigid body-spring model can be used to test potential enhancements to the displacement or load control loops, it must be validated (**Figure 46**). The transformations developed in section 4.2 to describe general rigid body motion were applied to the general rigid body-spring model shown in **Figure 24**. For model symmetry, the following parameters were set: $2L = 80$ mm, $k_a = k_b = 1$ N/mm, $\ell_{ar} = \ell_{a0} = \ell_{br} = \ell_{b0} = 60$ mm. The center of the bar in the equilibrium position was set at the global origin. Because of symmetry and the equilibrium position, the bar's preferred COR, or the point about which a rotation will result in a pure moment, is at the global origin. **Figure 49** shows data for set 1a and **Figure 50** shows representative data for set 1b. Outcomes are expected. Many hand calculations were performed to validate these results.


a
globa Y axis
global X axis
b
FX (N)
Y COR location (mm)
X COR location (mm)
c
FY (N)
Y COR location (mm)
X COR location (mm)
d
Resultant force magnitude (N)
Y COR location (mm)
X COR location (mm)
e
MZ (N-m)
Y COR location (mm)
X COR location (mm)
f
Potential energy (N-mm)
Y COR location (mm)
X COR location (mm)

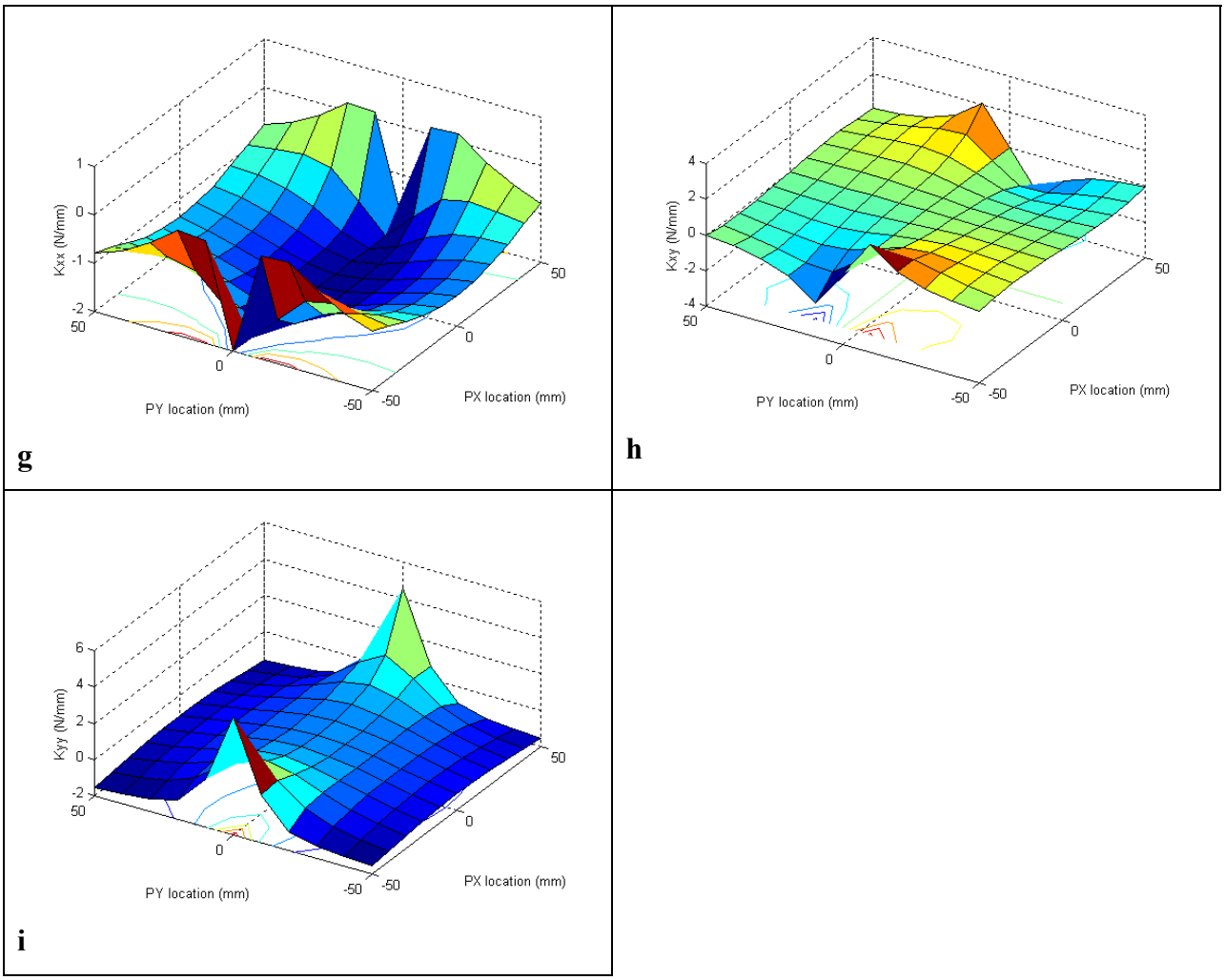
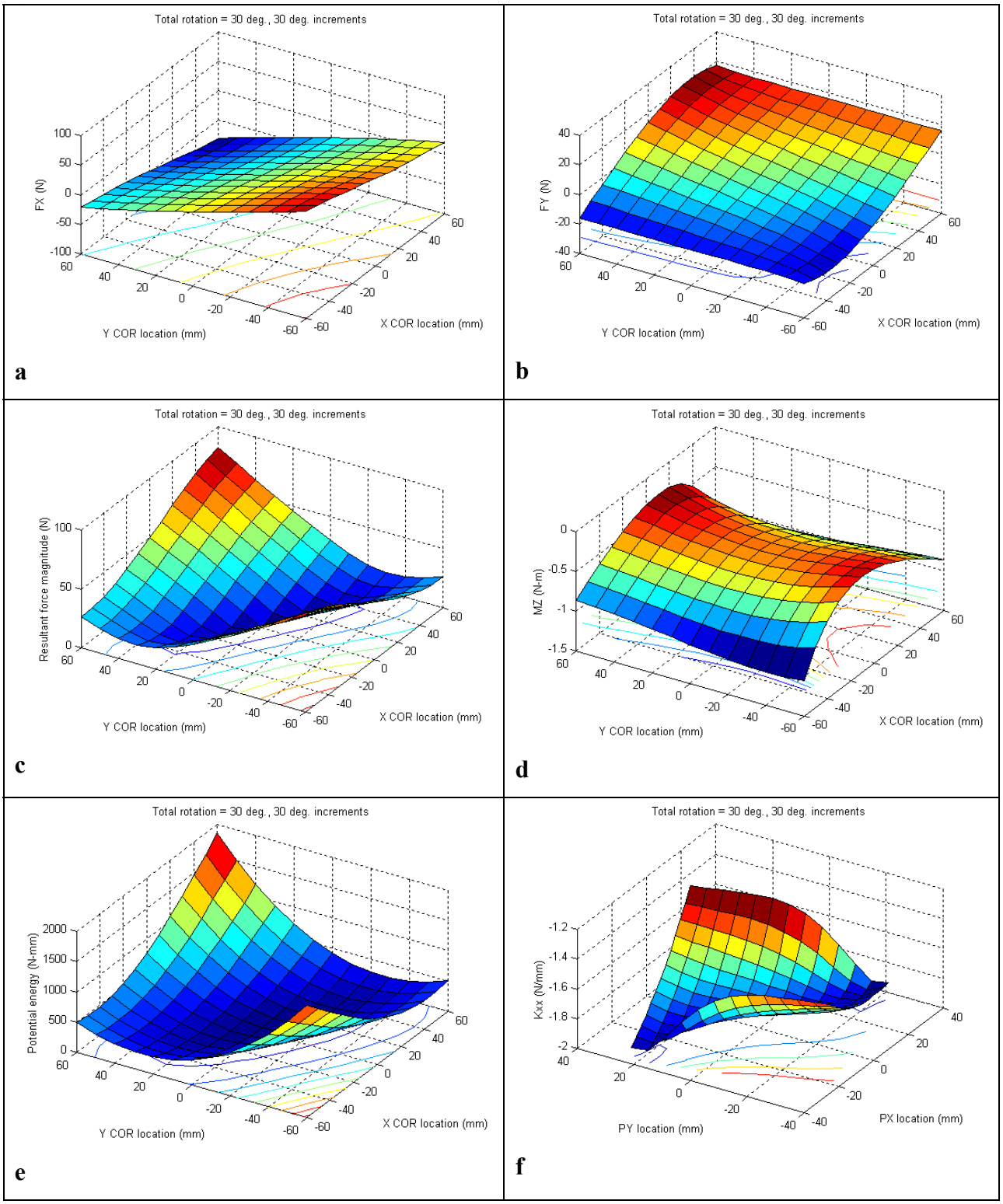


Figure 49 Comprehensive results showing validation of general spring model for translation of center of bar without any rotation (simulation set 1a). (**a**) grid of points in the global XY -plane that the center of the bar was translated to (**b**) force acting on bar in global X direction (outcome 4a). (**c**) force acting on bar in global Y direction (outcome 4b). (**d**) resultant force acting on bar in global XY -plane (outcome 4c). (**e**) moment acting on bar in global Z direction (outcome 4d). (**f**) potential energy in system (outcome 5). (**g**)-(**i**) global stiffness terms (outcomes 6a-6c).


Total rotation = 30 deg., 30 deg. increments
FX (N)
Y COR location (mm)
X COR location (mm)
a
Total rotation = 30 deg., 30 deg. increments
FY (N)
Y COR location (mm)
X COR location (mm)
b
Total rotation = 30 deg., 30 deg. increments
Resultant force magnitude (N)
Y COR location (mm)
X COR location (mm)
c
Total rotation = 30 deg., 30 deg. increments
MZ (N-m)
Y COR location (mm)
X COR location (mm)
d
Total rotation = 30 deg., 30 deg. increments
Potential energy (N-mm)
Y COR location (mm)
X COR location (mm)
e
Total rotation = 30 deg., 30 deg. increments
Kxx (N/mm)
PY location (mm)
PX location (mm)
f

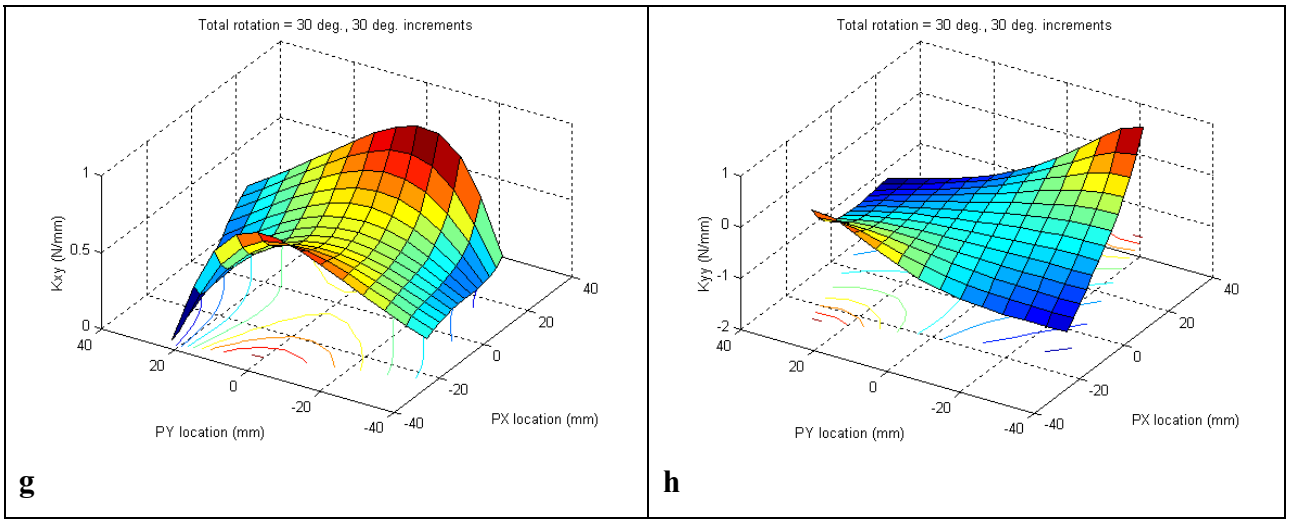
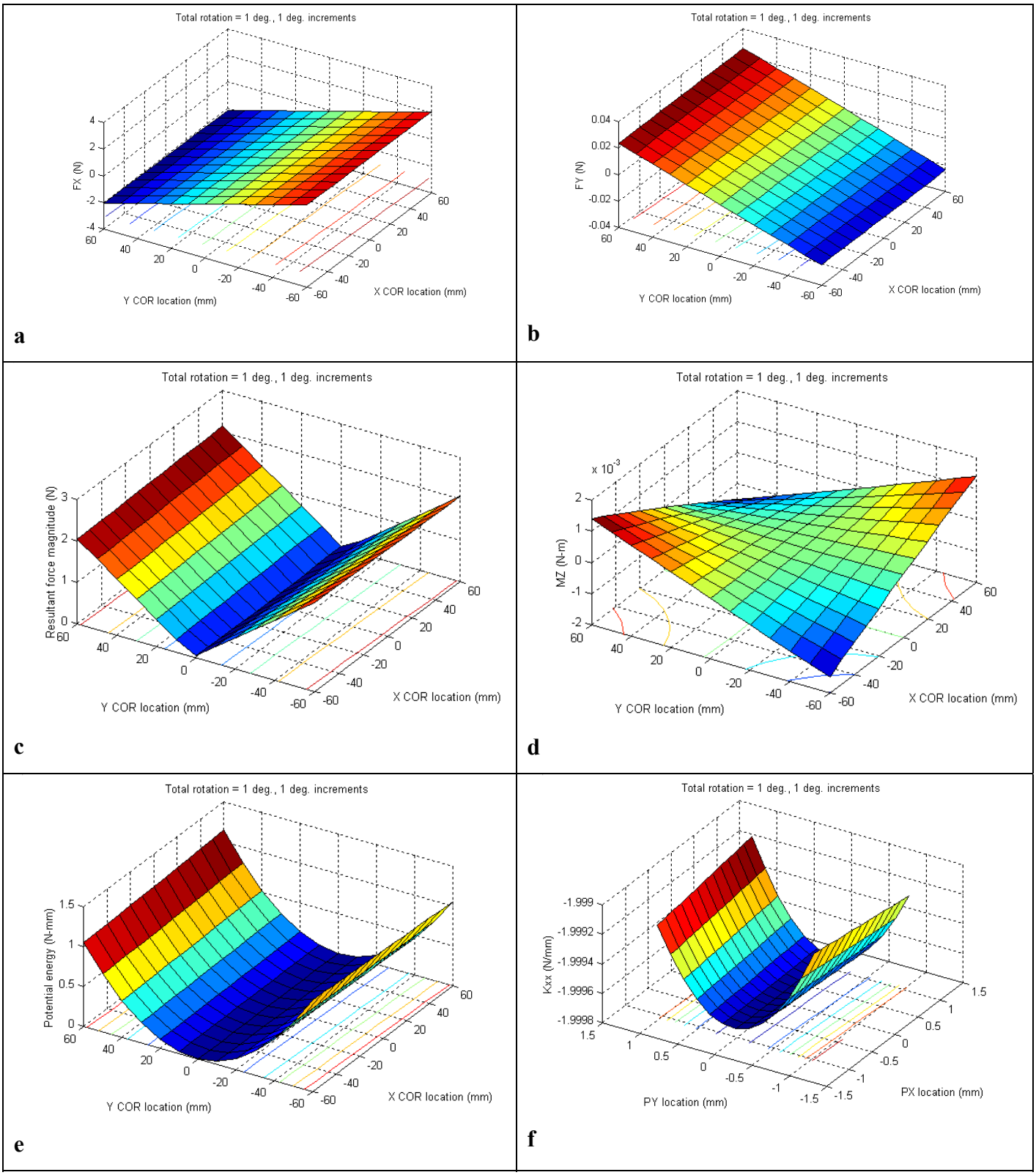


Figure 50 Comprehensive results showing validation of general spring model for rotation of center of bar about same grid of points shown in **Figure 49**, $\Phi = \phi = 30°$ (simulation set 1b). (**a**) force acting on bar in global X direction (outcome 4a). (**b**) force acting on bar in global Y direction (outcome 4b). (**c**) resultant force acting on bar in global XY-plane (outcome 4c). (**d**) moment acting on bar in global Z direction (outcome 4d). (**e**) potential energy in system (outcome 5). (**f**)-(**h**) global stiffness terms (outcomes 6a-6c).

After the model was validated, pure displacement control was applied in various rotation increments about a 13x13 grid of CORs (**Figure 47**). **Figure 51** shows the effect of varying COR location for a given rotation increment (set 2a). These results are similar to those of set 1b in that the farther the COR is from the rigid body's preferred COR, the greater the force created during rotation and the more work is put into the system. The force created during rotation may be relieved by translating the center of the bar to the global origin in one step (**Figure 52**). This is not load control because the force minimized position was known beforehand so the bar could be placed there without regard to the loads acting on it.


Total rotation = 1 deg., 1 deg. increments
FX (N)
Y COR location (mm)
X COR location (mm)
a
Total rotation = 1 deg., 1 deg. increments
FY (N)
Y COR location (mm)
X COR location (mm)
b
Total rotation = 1 deg., 1 deg. increments
Resultant force magnitude (N)
Y COR location (mm)
X COR location (mm)
c
Total rotation = 1 deg., 1 deg. increments
x 10^-3
MZ (N-m)
Y COR location (mm)
X COR location (mm)
d
Total rotation = 1 deg., 1 deg. increments
Potential energy (N-mm)
Y COR location (mm)
X COR location (mm)
e
Total rotation = 1 deg., 1 deg. increments
Kxx (N/mm)
PY location (mm)
PX location (mm)
f

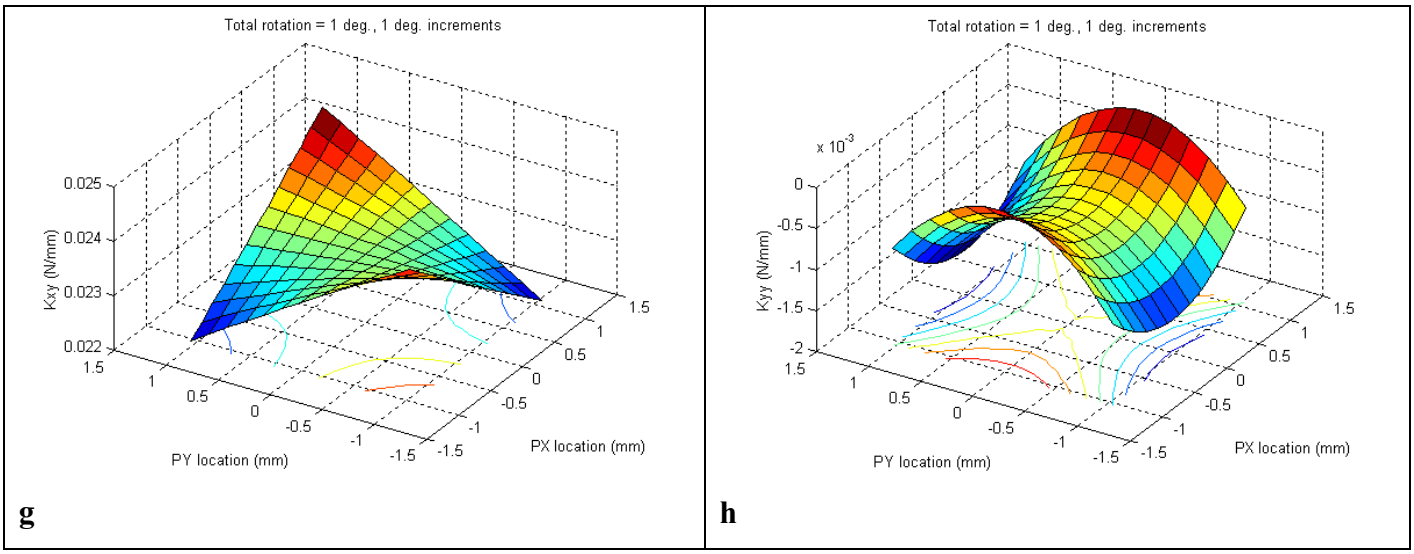


Figure 51 Comprehensive results showing characterization of general spring model in displacement control for $\Phi = \phi = 1°$ (simulation set 2a). (**a**) force acting on bar in global X direction (outcome 3a). (**b**) force acting on bar in global Y direction (outcome 3b). (**c**) resultant force acting on bar in global XY -plane (outcome 3c). (**d**) moment acting on bar in global Z direction (outcome 3d). (**e**) potential energy in system (outcome 4). (**f**)-(**h**) global stiffness terms (outcomes 5a-5c).

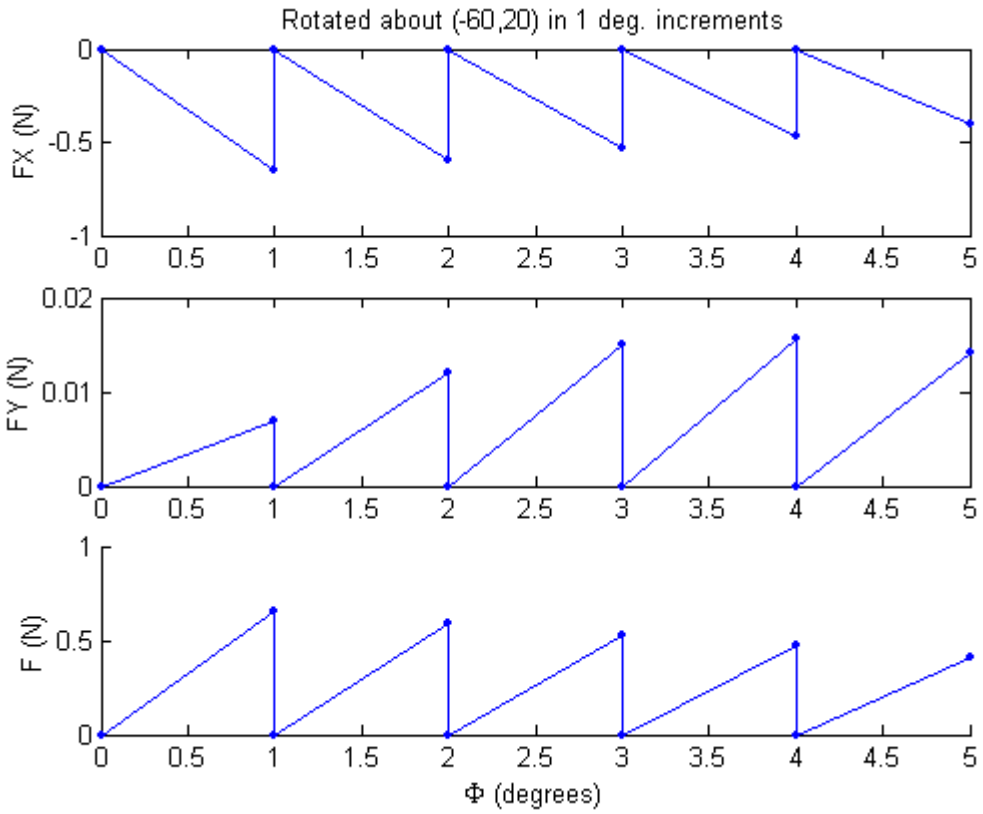


Figure 52 Representative data showing that the force resulting from rotation about a non-preferred COR can be relieved by translating the center of the bar to the origin (simulation set 2x)

Results for set 2b illustrate that the force created during rotation is a function of rotation increment size. The magnitude of the resultant force decreases with decreasing increment size, but the nonlinear trend for rotation about a given COR remains the same. The moment, potential energy and global stiffness terms are not affected by the size of rotation increment. Representative data is shown in **Figure 53** and **Figure 54**. If the only change to displacement control were decreasing the rotation increment ($\leq 1°$), the peak force created during rotation would decrease, as desired. Experimentally, this protects the specimen from potential damage, but increases the time taken to complete a test, possibly introducing stress relaxation to the specimen. Practically, the rotation increment should be kept to around $0.5°$. If the user notices

that the peak force is too high, the increment can be decreased, or if the user notices that the peak force is low, the test can be sped up by increasing the increment size without compromising the specimen's safety. For the remainder of the simulations, a rotation increment of 1° is used to reduce computation time.

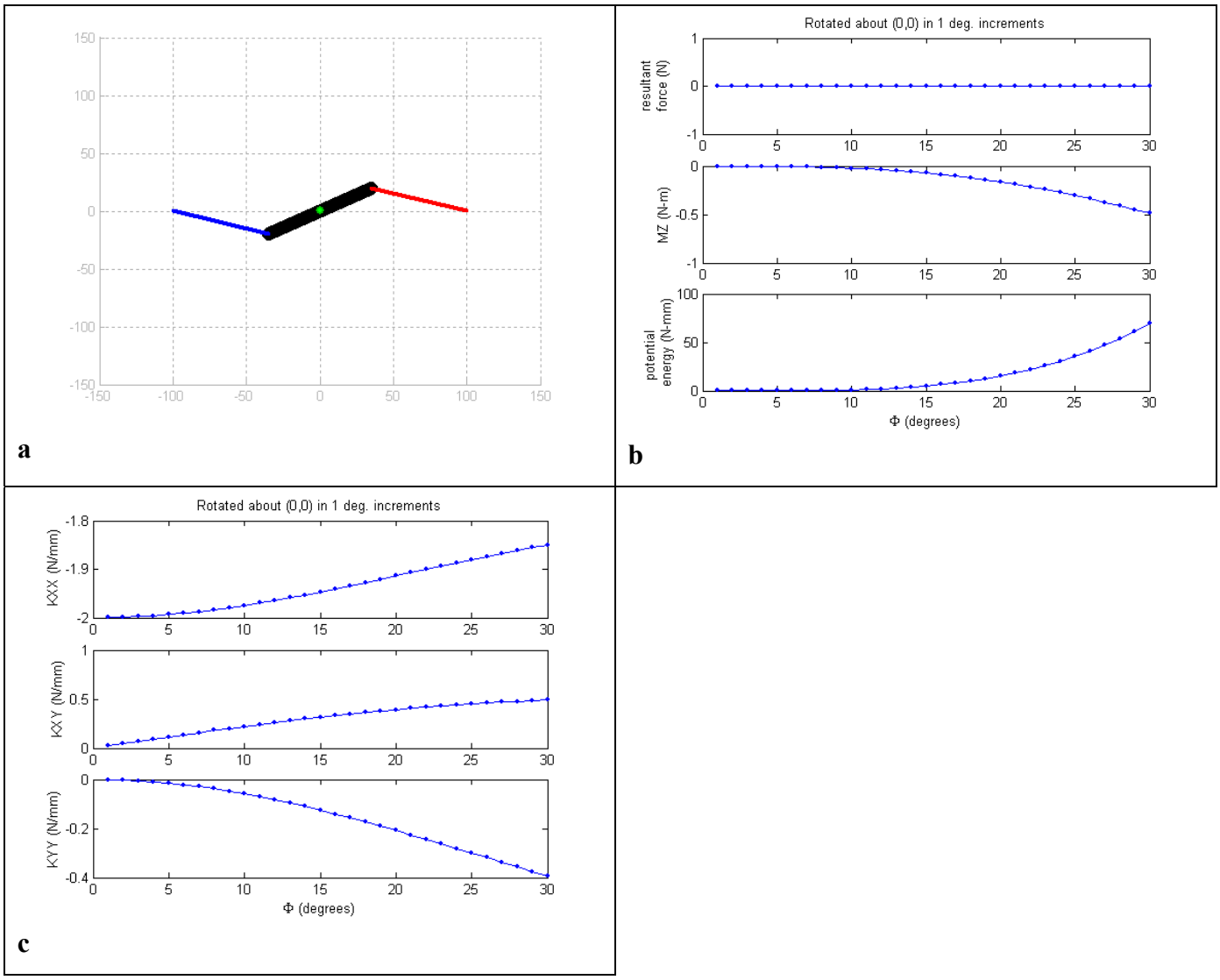


Figure 53 Representative data for full characterization of the general rigid body-spring model during displacement control (simulation set 2b) (**a**) rotated about the true COR located at (0,0) in the global XY-plane in $\phi = 1°$ increments up to $\Phi = 30°$ (**b**) the top row of this plot shows the resultant force acting on the bar after each incremental rotation (outcome 3a), the middle plot shows the moment acting on the bar after each incremental rotation (outcome 3b) and the bottom plot shows the potential energy in the system after each incremental rotation (outcome 4) (**c**) global stiffness terms plotted over total rotation angle (outcome 5)

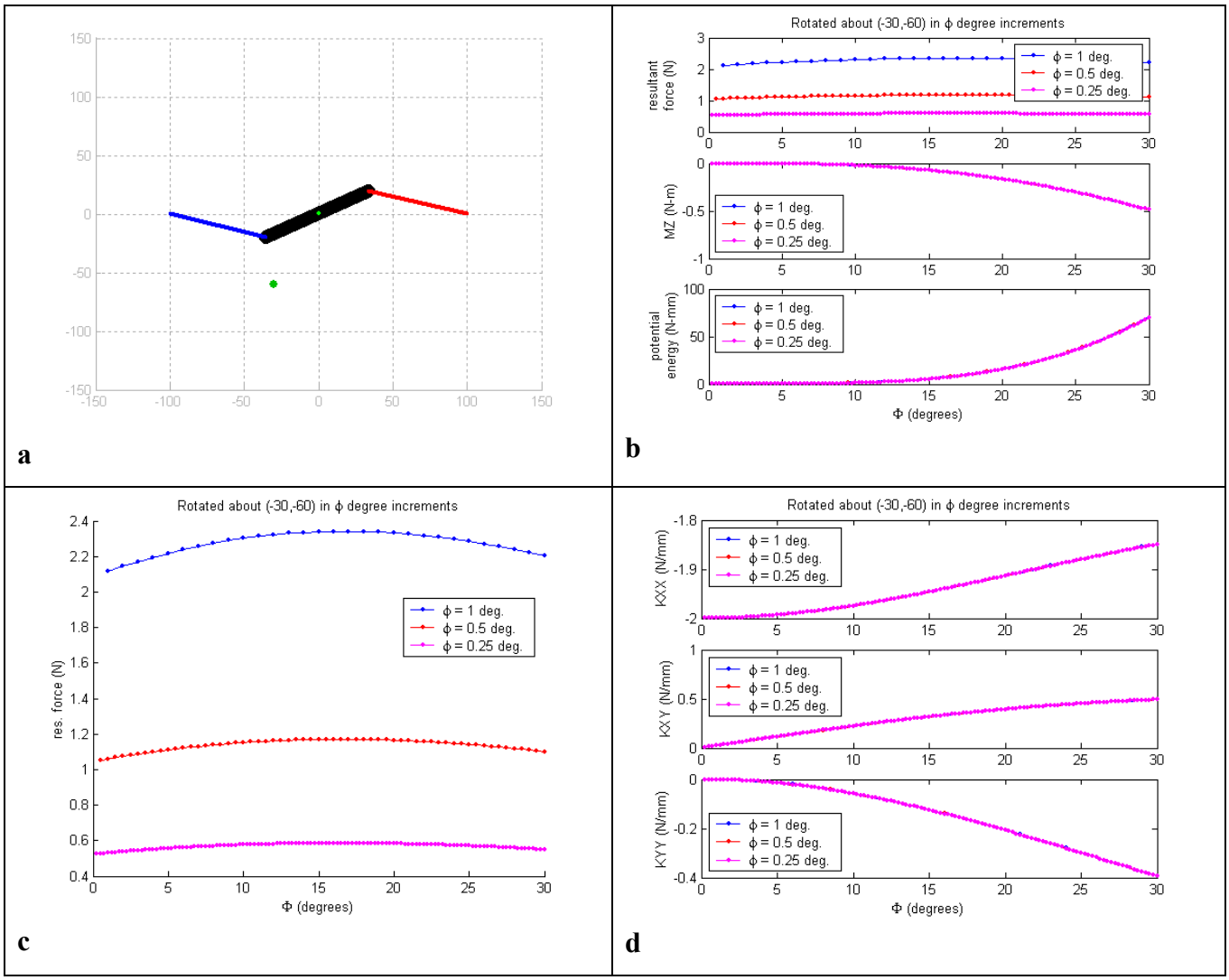


Figure 54 Representative data for full characterization of the general rigid body-spring model during displacement control (simulation set 2b) (**a**) rotated about a COR located at (-30,-60) in the global XY-plane in $\phi = 1°, 0.5°, 0.25°$ increments up to $\Phi = 30°$ (**b**) the top row of this plot shows the resultant force acting on the bar after each incremental rotation (outcome 3a), the middle plot shows the moment acting on the bar after each incremental rotation (outcome 3b) and the bottom plot shows the potential energy in the system after each incremental rotation (outcome 4) (**c**) top plot of (**b**) reproduced, resultant force on bar after each rotation decreases for decreasing rotation increment (**d**) global stiffness terms plotted over total rotation angle (outcome 5)

The penalty of rotating about a COR other than the model's preferred COR while keeping the user-defined COR fixed locally have now been shown. The farther the user-defined COR is from the preferred COR, the more severe the penalty, i.e., the peak force is larger. It is

hypothesized that the displacement control loop can be improved by allowing the COR to move locally. To test this hypothesis, two methods of updating the COR are proposed (**Figure 45**). One updates the COR post hoc, while the other method uses feedback to update the COR. Outcome measures used to evaluate the effectiveness of the proposed changes are the peak force created during rotation, the work put into the system and the number of iterations required to reach the force minimized position. Either one of the proposed changes is deemed an improvement over the current displacement control if the outcome measures decrease.

The first issue to be discussed is how to calculate the preferred COR. Three methods will be considered: Spiegelman and Woo[42], Crisco et al.[43] and Challis[44]. All three methods use the motion of two markers attached to a moving rigid body to calculate the rigid body's COR. The equations reported in literature are reproduced below.

Method #1: Spiegelman and Woo[42]

$$S = X_1 - X_3,\ S' = X_2 - X_4$$

$$T = Y_1 - Y_3,\ T' = Y_2 - Y_4$$

$$\cos\phi = \frac{S'S - T'T}{S^2 + T^2},\ \sin\phi = \frac{S'T - T'S}{S^2 + T^2}$$

$$U = \frac{Y_1 + Y_2}{2} + \frac{\sin\phi\left(X_1 - X_2\right)}{2\left[1 - \cos\phi\right]}$$

$$V = \frac{X_1 + X_2}{2} - \frac{\sin\phi\left(Y_1 - Y_2\right)}{2\left[1 - \cos\phi\right]}$$

$$X_{cor} = X_1 + \frac{Y_2 - U}{\sin\phi} - \frac{\cos\phi\left(Y_1 - U\right)}{\sin\phi}$$

$$Y_{cor} = Y_1 - \frac{X_2 - V}{\sin\phi} + \frac{\cos\phi\left(X_1 - V\right)}{\sin\phi},$$

where (X_1, Y_1) are the initial global coordinates of the first marker, (X_2, Y_2) are the final global coordinates of the first marker, (X_3, Y_3) are the initial global coordinates of the second marker, (X_4, Y_4) are the final global coordinates of the second marker, ϕ is the incremental rotation and (X_{cor}, Y_{cor}) are the global coordinates of the preferred COR (**Figure 55**).

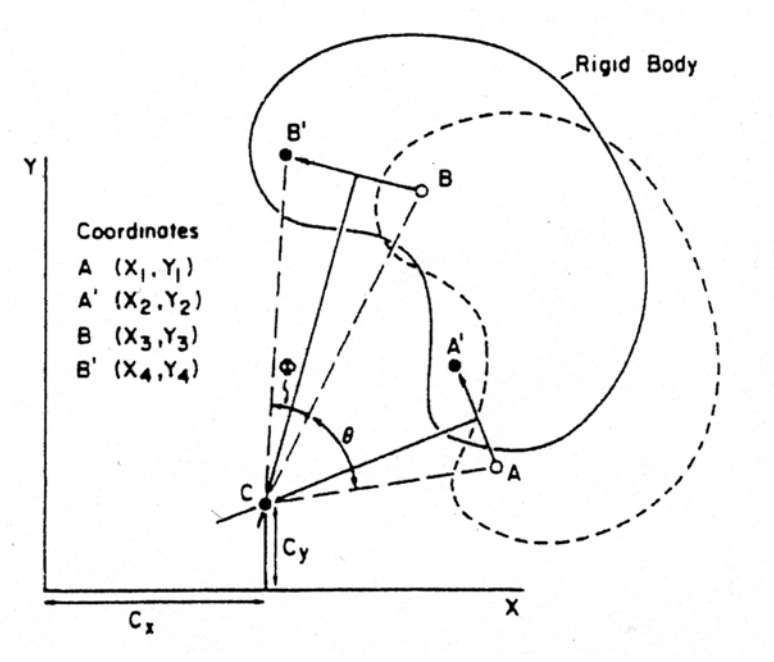


Figure 55 Spiegelman and Woo

Method #2: Crisco et al.[43]

$$A = (x_1, y_1),\ A' = (x_2, y_2)$$

$$B = (x_3, y_3),\ B' = (x_4, y_4)$$

$$\bar{u} = \bar{A} - \bar{B},\ \bar{u}' = \bar{A}' - \bar{B}'$$

$$\cos\phi = \frac{\bar{u} \cdot \bar{u}'}{|\bar{u}||\bar{u}'|},\ \sin\phi = \sqrt{1 - \cos^2\phi}$$

$$X_{cor} = \frac{1}{2}\left(x_1 + x_2\right) + \frac{\left(y_1 - y_2\right)\sin\phi}{2\left(1-\cos\phi\right)}$$

$$Y_{cor} = \frac{1}{2}\left(y_1 + y_2\right) - \frac{\left(x_1 - x_2\right)\sin\phi}{2\left(1-\cos\phi\right)},$$

where $\left(x_1, y_1\right)$ are the initial global coordinates of marker A, $\left(x_2, y_2\right)$ are the final global coordinates of marker A, $\left(x_3, y_3\right)$ are the initial global coordinate of marker B, $\left(x_4, y_4\right)$ are the final global coordinate of marker B, ϕ is the incremental rotation and $\left(X_{cor}, Y_{cor}\right)$ are the global coordinates of the preferred COR.

Method #3: Challis[44]

$$v = \overline{y} - \left[R\right]\overline{x}$$

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \ , \ \overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

where x_i is the global position vector of marker i, y_i is the local position vector of marker i, $i = 2,3,...n$, $\left[R\right]$ is the rotation matrix describing the orientation of the local coordinate system with respect to the global coordinate system and v is the global location of the local coordinate system origin.

$$x_i' = x_i - \overline{x} \ , \ y_i' = y_i - \overline{y}$$

$$P = \sum_{i=1}^{n}\left(y_{xi}' x_{yi}' - y_{yi}' x_{xi}'\right)$$

$$Q = \sum_{i=1}^{n}\left(y_{xi}' x_{xi}' + y_{yi}' x_{yi}'\right)$$

$$\phi = -\tan^{-1}\left(\frac{P}{Q}\right),$$

where ϕ is the incremental rotation. ϕ is inserted into $[R]$ in the first equation to determine v.

$$FCR = p + \left[2\tan(\phi/2)\right]^{-1}\left[R(90°)\right]\Delta v$$

$$X_{cor} = FCR(1),\ Y_{cor} = FCR(2),$$

where $p = \frac{1}{2}\left(v(t_1) + v(t_2)\right)$, $R(90°)$ is a rotation matrix describing a $90°$ rotation, $\Delta v = v(t_2) - v(t_1)$ and (X_{cor}, Y_{cor}) are the global coordinates of the preferred COR (**Figure 56**). For every method, $error = \sqrt{X_{cor}^2 + Y_{cor}^2}$ because the true COR is at the origin of the global coordinate system.

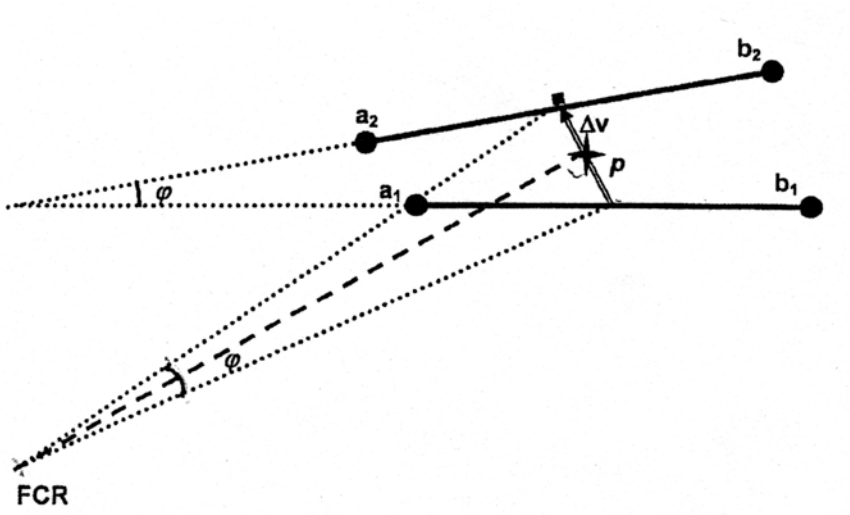


Figure 56 Challis

Several variables must be considered when calculating the preferred COR. The error is a function of the distance of the final force minimized position from the actual force minimized position, noisy marker position data and the size of rotation increment that the COR is calculated over. Any of the methods will calculate the rigid body's preferred COR when the actual force

minimized position of the rigid body is used. If the position of the rigid body at the end of the load control loop is not the true force minimized position or is not close to it, the preferred COR will not be calculated. In addition, all methods are susceptible to error when marker position data is noisy, especially if calculated over small rotation angles, as shown by the authors. If the rigid body reaches its force minimized position and marker position data is known exactly, all methods calculate the rigid body's preferred COR to within a very small error, even at small rotation angles.

Experimentally, marker position data will be noisy. Crisco et al. showed that the error increased exponentially for decreasing rotation angles when normally distributed noise (mean = 0 mm, s.d. = 0.5 mm) is added to marker position for their method and Spiegelman and Woo's method. While Crisco's method performed better, the error for both methods didn't fall into an acceptable range until the rotation angle was $\sim 20°$. Challis showed that his method of calculating the COR results in the least error of the three methods when noise is introduced, but the error still increased exponentially for decreasing rotation angles when the same normally distributed noise is added. Again, the error in this method didn't fall into an acceptable range until $\sim 20°$.

To test the methods of calculating the preferred COR of the analytical rigid body-spring model, noise may be added to marker position, as done in the literature, or it may be added to the loads acting on the rigid body because this will affect the final force minimized position. The experimental system is considered to guide the choice of where to add noise in the analytical simulations. As shown previously, the positional inaccuracy of the manipulator is not random, but is a function of the weight on the end-effector. The robot is told to move by a certain amount to reach the minimum force position. This relies on the robot's precision, ± 0.02 mm, so the

position inaccuracy due to the weight on the end-effector should only be a concern when the "WHERE" command is issued (when we want to know the marker positions). As illustrated by preliminary experiments, the specimen is able to reach its force minimized position even though the load cell data may be quite noisy near the force minimized position. Therefore, it is assumed that the position inaccuracy will be a larger source of noise experimentally and confound COR calculation more than UFS noise. During simulations, noise is added to marker position; noisy load data is not considered.

To add noise to marker position in simulations, the simple accuracy experiment from section 5.4 is used. Recall the linear relationship between percent payload and position error in the UFS y-direction (global Z-direction):

$$error = 0.0058 * (\%\,\max\,\text{payload}) - 0.28$$

If the above equation were also applied to the UFS z-direction, the position error (in mm) would be overestimated because the UFS y-direction had the most slop when performing the experiment. However, extending the above equation to the z-direction is an acceptable approximation. The percent of maximum payload that is acting on the bar (in the Matlab X- and Y-direction) is calculated and inserted into the above equation to obtain position error in the Y- and Z-directions. The calculated errors are then added to the analytically known marker positions.

As mentioned above, all methods result in very large error if noisy markers are used to calculate the preferred COR over small rotation increments. To try to correct this, we can calculate the COR over larger rotation angles ($\sim 5°$) instead of after every increment ($\sim 1°$). We can also limit the amount the COR is allowed to change.

The three methods of calculating the preferred COR need to be evaluated for several cases (**Figure 57**). Set 4a does not simulate the experimental system because the analytical solution to the global stiffness matrix used for load control cannot be known. Set 4b does not simulate the experimental system either because it is highly unlikely that marker position data is not noisy. Even though these evaluations do not simulate the experimental system, they are useful for simulation validation. Sets 4c and 4d more closely simulate the experimental system because noise is added to marker position and the stiffness matrix is calculated numerically (even though the stiffness matrix in simulations is more exact than what would be encountered experimentally because the forces and moment are known analytically during simulations). The numerical calculation of the global stiffness matrix is the one currently used for experiments (the diagonal terms are calculated as $\Delta F/\Delta d$ and the off-diagonal terms are set to zero).

Test #	Input parameter varied	Testing procedure	Output parameter of interest	Expected outcome
4a	COR location	Create 13x13 grid of CORs ($-60 \le X \le 60$ mm and $-60 \le Y \le 60$ mm). Rotate about each COR in $\phi = 1°$ increments until reach $\Phi = 30°$. After each increment, translate center of bar to global origin to minimize forces. Calculate preferred COR after each increment using each proposed method. Do not add noise to marker position.	Distance of calculated preferred COR from true preferred COR.	Because noise is not added to marker position and forces are relieved without using load control, all methods should calculate true preferred COR.

4b	COR location	Use CORs from above grid. Rotate about each COR in $\phi = 1°$ increments until reach $\Phi = 30°$. After each increment, use diagonal stiffness matrix to translate bar to minimize forces. Calculate preferred COR after each increment using each proposed method. Do not add noise to marker position.	Distance of calculated preferred COR from true preferred COR.	Because noise is not added to marker position, all methods should calculate the same preferred COR. As long as the center of the bar reaches the global origin in 20 iterations or less, all methods should calculate true preferred COR.
4c	COR location	Use CORs from above grid. Rotate about each COR in $\phi = 1°$ increments until reach $\Phi = 30°$. After each increment, use diagonal stiffness matrix to translate bar to minimize forces. Calculate preferred COR after each increment using each proposed method. Add noise to marker position.	Distance of calculated preferred COR from true preferred COR.	Because noise is added to marker position, no method will exactly calculate true COR. Same noise is added to marker position, so all methods will calculate the same preferred CORs.

4d	COR location	Use CORs from above grid. Rotate about each COR in $\phi = 1°$ increments until reach $\Phi = 30°$. After each increment, use diagonal stiffness matrix to translate bar to minimize forces. Calculate preferred COR after each $5°$ increment using each proposed method. Add noise to marker position.	Distance of calculated preferred COR from true preferred COR.	Because noise is added to marker position, no method will exactly calculate true COR. Same noise is added to marker position, so all methods will calculate the same preferred CORs. Calculated preferred CORs will be closer to true COR because CORs are calculate over a larger increment.

Figure 57 Evaluation of proposed changes to displacement control (calculate preferred COR)

Figure 58 and **Figure 59** show representative data for sets 4a – 4d. As expected, all three methods calculate the true preferred COR to within a very small error, on the order of 10^{-14} mm, when the bar is translated to the global origin in one step and noise is not added to marker position (set 4a). Also as expected, all three methods calculate the same preferred COR when the currently used numerically calculated diagonal stiffness matrix is used in load control, noise is not added to marker position and the preferred COR is calculated over $1°$ increments (set 4b). As long as the bar reaches the global origin within the allowed number of iterations, the error in calculating the preferred COR is relatively small (**Figure 58**). If the bar does not reach the force minimized position, there is more error in COR calculation (**Figure 59**). When noise is added to marker position (set 4c), all three methods calculate the same preferred COR, again within a relatively small error if the bar reaches the force minimized position. This is not surprising because the same noisy marker positions are used to calculate the COR for all methods. When

the preferred COR is calculated using noisy markers over a larger rotation increment (set 4d), the error remains relatively small, as in **Figure 58**, or it decreases from a large error, as in **Figure 59**. Experimentally, calculating the COR over a larger rotation increment is preferred because there will be additional noise in the system: load cell noise and any end-effector noise that is unaccounted for.

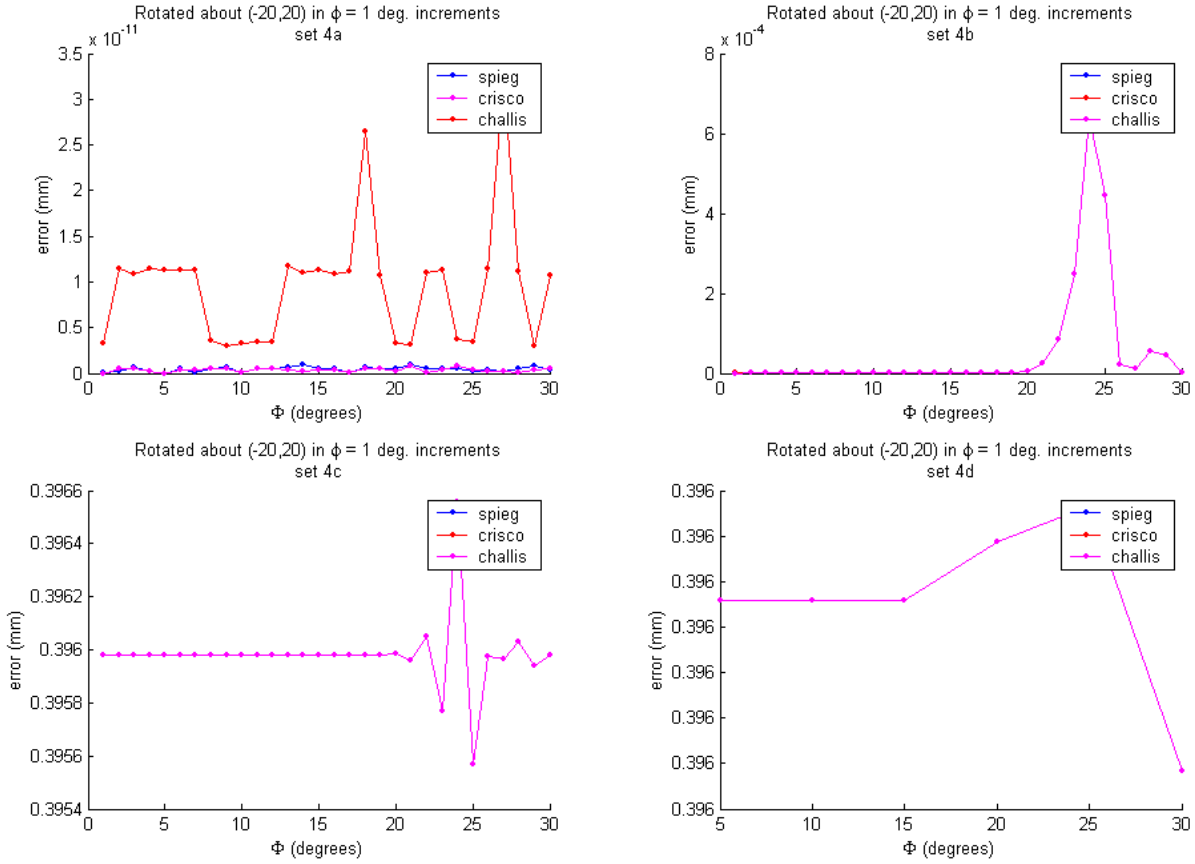


Figure 58 Representative data for characterization of performance of three different methods of calculating the preferred COR, rotated about a COR located at (-20,20) in the global XY-plane in $\phi = 1°$ increments up to $\Phi = 30°$, plots show the error vs. rotation angle for conditions set in simulation set 4a (top left plot), simulation set 4b (top right plot), simulations set 4c (bottom left plot) and simulation set 4d (bottom right plot)

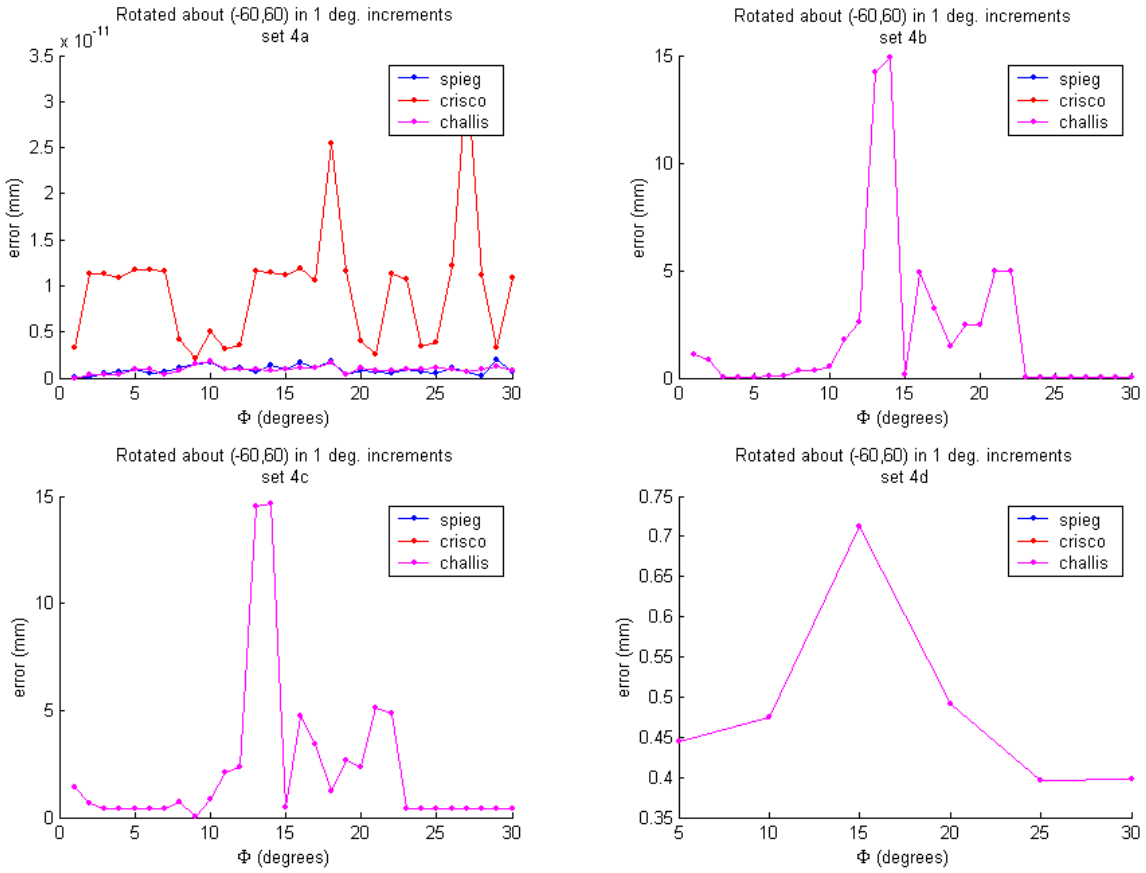


Figure 59 Representative data for characterization of performance of three different methods of calculating the preferred COR, rotated about a COR located at (-60,60) in the global XY -plane in $\phi = 1°$ increments up to $\Phi = 30°$, plots show the error vs. rotation angle for conditions set in simulation set 4a (top left plot), simulation set 4b (top right plot), simulations set 4c (bottom left plot) and simulation set 4d (bottom right plot)

The effect of noise in marker position on the ability of the three methods to calculate the preferred COR has been shown. Because all three methods calculate the same preferred COR, only the Challis method will be considered for further simulations. Even though this method is more computationally intense than the other two, it is hypothesized that it will perform better in the experimental system based on performance reported in the literature. Next, proposed methods of updating the COR are investigated.

To illustrate the difference between post hoc and feedback COR updating, consider the following. Suppose that we are performing the first cycle of pathseek, going from full flexion ($\sim 15°$) to full extension ($\sim -15°$) and the COR is calculated in 5 degree increments, i.e., COR1 is calculated using force minimized positions at $15°$ and $10°$ of flexion, COR2 is calculated using force minimized positions at $10°$ and $5°$ of flexion, and so on. If the COR is updated post hoc, COR1 is stored for use in the second pathseek cycle. The user-chosen COR is not updated to COR1; it is kept the same for $(10+\phi)°$ to $5°$, where COR2 is calculated. Again, COR2 is stored for use in the second pathseek cycle, but the user-chosen COR is not updated to reflect COR2. This algorithm is still stubborn because the initial user-chosen COR is used for the entire first pathseek cycle. Then, for the second pathseek cycle, COR1 is fixed globally for $15°$ to $10°$ of flexion, COR2 is fixed globally for $(10+\phi)°$ to $5°$, etc. If a certain criteria is not met during the second pathseek cycle, new CORs can be calculated again as in the first pathseek cycle. If the COR is updated using feedback, then the user-chosen COR is updated in the first pathseek cycle to COR1 at $10°$ and fixed locally from $(10+\phi)°$ to $5°$. Then COR2 is calculated and used from $(5+\phi)°$ to $0°$. This process is repeated for the entire pathseek test if a certain criteria is not met. Experimentally, the distance the COR is allowed to move will be limited to 5 mm in each direction because the calculated preferred COR may be far away from the true preferred COR.

Test #	**Input parameter varied**	**Testing procedure**	**Output parameter of interest**	**Expected outcome**
5a	COR location	Create 13x13 grid of CORs ($-60 \le X \le 60$	Peak force created during rotation	Peak force will be reduced.

		($-60 \le X \le 60$ mm and $-60 \le Y \le 60$ mm). Rotate about each COR in $\phi = 1°$ increments until reach $\Phi = 30°$. After each increment, use diagonal stiffness matrix to translate bar to minimize forces. Calculate preferred COR after each $5°$ increment using Challis method. Add noise to marker position. Use post hoc method to update COR. Amount COR is allowed to change is limited to 5 mm in each direction.	Number of iterations to minimize force.	Number of iterations will be reduced.
			Work put into system	Work will remain unchanged.
5b	COR location	Use same testing method as test 5a, but use feedback to update COR.	Peak force created during rotation.	Peak force will be reduced.
			Number of iterations to minimize force.	Number of iterations will be reduced.
			Work put into system	Work will remain unchanged.

Figure 60 Evaluation of proposed changes to displacement control (update COR)

Matlab simulations were performed to evaluate the two proposed methods of updating the COR (**Figure 60**). Outcome measures for testing proposed improvements are peak force created during rotation, number of iterations required to minimize force and work put into system (**Figure 61** and **Figure 62**). The work remains unchanged across varying COR location, COR

calculation method and COR update method because the rigid body-spring model ends up at the same force minimized position within the limited number of iterations; the work done to the bar in load control cancels the work done to the bar in displacement control. The results of this test are not entirely expected. It was hypothesized that using feedback to update the COR would perform better (smaller peak force and fewer iterations) than updating the COR post hoc. For most of the CORs in the 13x13 grid, this is true (**Figure 62**), but for some CORs it is not (**Figure 61**). Overall, the feedback method is superior, so it will be used in the new hybrid control algorithm.

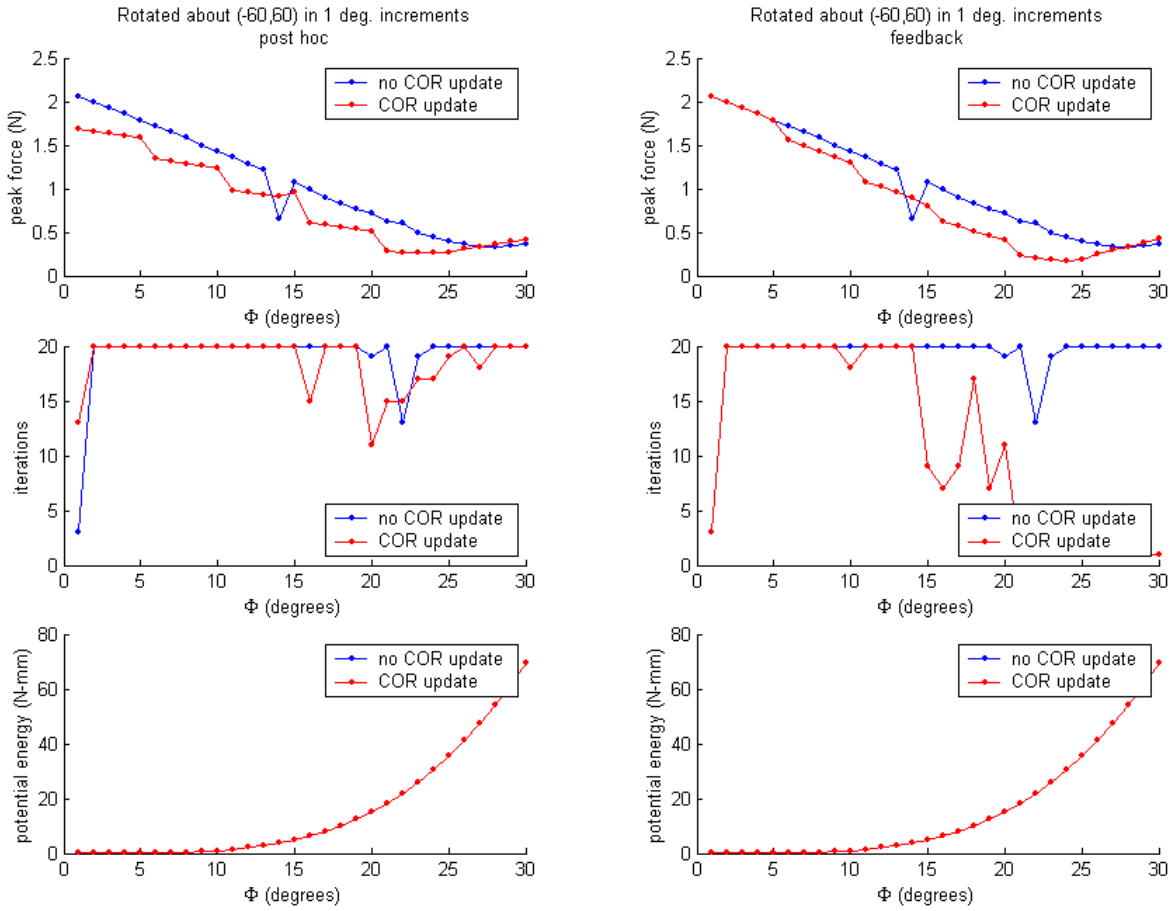


Figure 61 Representative data for characterization of performance of two different methods of updating the user-defined COR as compared with keeping the COR fixed locally (simulation sets 5a and 5b), rotated about a COR located at (-60,60) in the global XY-plane in $\phi = 1°$ increments up to $\Phi = 30°$, the left column shows data using the post hoc method of updating the COR, the right column shows data using feedback to update the COR, the top row of plots show the peak force (in Newtons) created during rotation about the COR vs. rotation angle (outcome 1), the middle row shows the number of iterations required to minimize force vs. rotation angle (outcome 2) and the bottom row shows the potential energy (in Newton-mm) in the system vs. rotation angle (outcome 3)

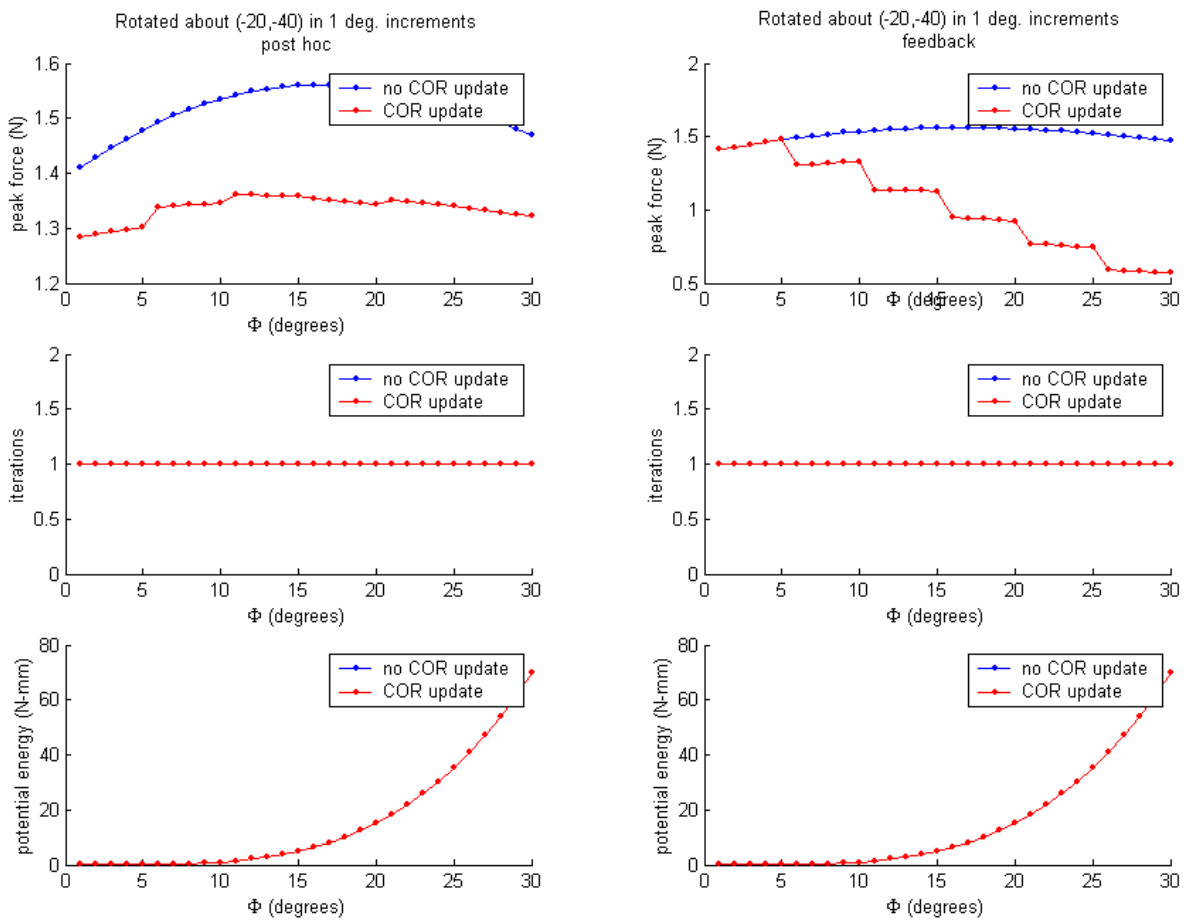


Figure 62 Representative data for characterization of performance of two different methods of updating the user-defined COR as compared with keeping the COR fixed locally (simulation sets 5a and 5b), rotated about a COR located at (-20,-40) in the global XY-plane in $\phi = 1°$ increments up to $\Phi = 30°$, the left column shows data using the post hoc method of updating the COR, the right column shows data using feedback to update the COR, the top row of plots show the peak force (in Newtons) created during rotation about the COR vs. rotation angle (outcome 1), the middle row shows the number of iterations required to minimize force vs. rotation angle (outcome 2) and the bottom row shows the potential energy (in Newton-mm) in the system vs. rotation angle (outcome 3)

Table 1 Tabulated results of simulation sets 5a and 5b showing range of peak force (in Newtons) and average number of force minimizing iterations for the current method (no COR update), post hoc update of COR and feedback update of COR

	Range of peak force (N)	Average iterations
Current method	1.7611 – 2.7476	5.3
Post hoc	0.1386 – 2.7476	3.4
Feedback	0.1563 – 2.3902	2.3

6.2 Load Control Loop of Hybrid Control Algorithm

The general rigid body-spring model is a coupled nonlinear system that can be described by two continuous functions, $f(x,y)$ and $g(x,y)$, where $f(x,y)$ is the analytical solution for F_X developed in section 4.2 and $g(x,y)$ is the analytical solution for F_Y, also developed in section 4.2. The goal of the load control loop is to find the values $x = x^*$ and $y = y^*$ such that $f(x^*, y^*) = 0$ and $g(x^*, y^*) = 0$. Newton's method for minimizing two coupled nonlinear equations is an appropriate method of iteratively calculating the translations Δx_i and Δy_i of the rigid body to minimize the resultant force[52]:

$$\begin{cases} f_x\big|_i \Delta x_i + f_y\big|_i \Delta y_i = -f_i \\ g_x\big|_i \Delta x_i + g_y\big|_i \Delta y_i = -g_i \end{cases},$$

where $\Delta x_i = x^* - x_i$, $\Delta y_i = y^* - y_i$, and the subscript on functions f and g denote the first derivative of the function with respect to the subscript, i.e., f_x is the first derivative of f with respect to x. The above set of equations can be rewritten as

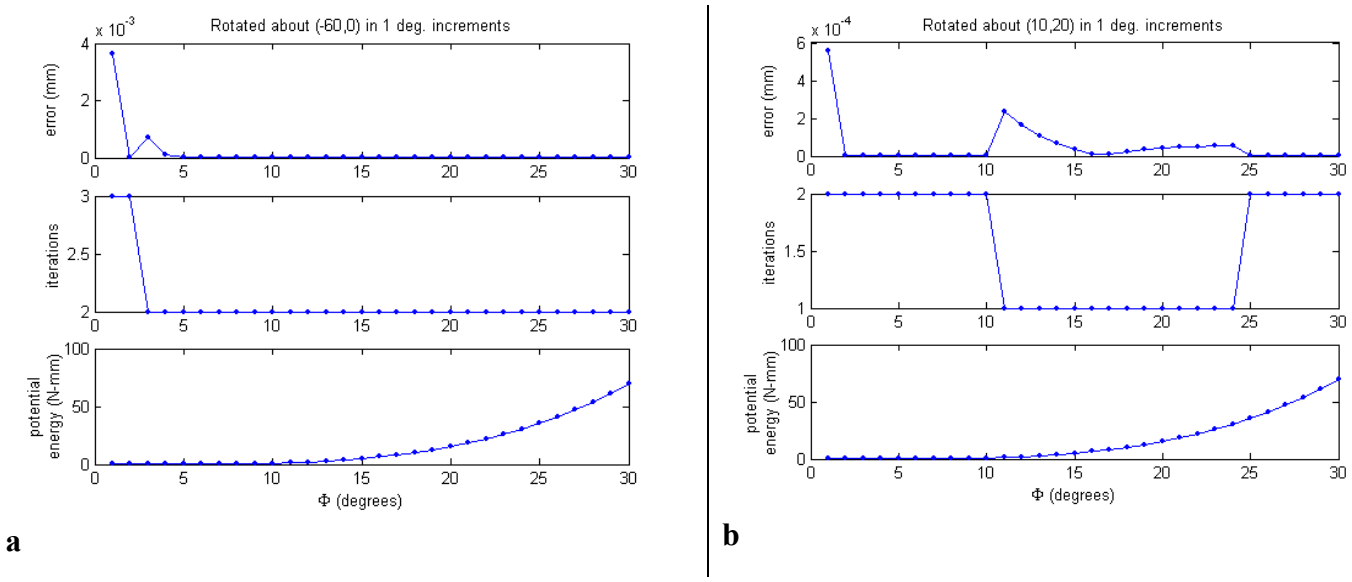
$$\begin{bmatrix} K_{XX} & K_{XY} \\ K_{YX} & K_{YY} \end{bmatrix} \begin{Bmatrix} \Delta x_i \\ \Delta y_i \end{Bmatrix} = \begin{Bmatrix} -F_X|_i \\ -F_Y|_i \end{Bmatrix}.$$

Once Δx_i and Δy_i are known, the new coordinates of the rigid body can be written as

$$\begin{cases} x_{i+1} = x_i + \Delta x_i \\ y_{i+1} = y_i + \Delta y_i \end{cases},$$

This process is repeated iteratively until the rigid body reaches the force minimized position. Because the stiffness matrix $\underline{K}$ is only linear over a small range, the magnitudes of Δx_i and Δy_i are limited. The above equations assume that $\underline{K}$ is known. Because the analytical solution for $\underline{K}$ cannot be known experimentally, it must be calculated numerically. Several methods of finding $\underline{K}$ are covered in more detail later.

Before examining any numerical calculations of the stiffness matrix, Newton's method is applied to the rigid body-spring model in load control to fully characterize the model. To accomplish this, two outcome measures are needed: the number of iterations required to minimize force on the bar and the distance of the final position of the center of the bar from the true force minimized position (**Figure 48**). The fully populated analytical stiffness matrix was used and translations were limited to 1 mm in each direction. When the resultant force on the bar was less than 10^{-5} N, the load control loop ended. **Figure 63** shows the distance of the center of the bar from the global origin, the number of iterations required to minimize force and the potential energy of the system for two randomly chosen CORs. This data is representative of the full 13x13 grid of CORs.



a **b**

Figure 63 Representative data for full characterization of the general rigid body-spring model during load control (simulation set 3), $\phi = 1°$ increments up to $\Phi = 30°$, the top row of the plots show the distance (in mm) of the final force minimized position from the true force minimized position (the global origin) vs. rotation angle (outcome 1), the middle row shows the number of iterations required to minimize force vs. rotation angle (outcome 2) and the bottom row shows the potential energy in the system after each rotation (outcome 3) (**a**) rotated about a COR located at (-60,0) in the global XY -plane (**b**) rotated about a COR located at (10,20) in the global XY -plane

The osteoligamentous spine is a highly nonlinear, coupled system. Traditionally, *in vitro* biomechanical testing has been performed using either the flexibility method or the stiffness method. The flexibility method applies loads, either singly or in combinations, to the FSU and the resulting unconstrained motions are measured. The stiffness method applies displacements and the resulting loads are measured. Flexibility/stiffness coefficients can then be determined. Assembling the flexibility/stiffness matrix is usually simplified by setting coefficients to zero or equating them to one another by assuming specimen symmetry. To examine the importance of coupled flexibility coefficients in modeling cervical spine motion, Winkelstein and Myers[53] fit linear, piecewise nonlinear and logarithmic functions to cervical spine data to assemble the full flexibility matrix. They found that including the coupled terms improved model performance.

For the load control loop of our hybrid control algorithm, a stiffness matrix is calculated numerically and inverted to find the translation necessary to reach the force minimized position (**Figure 45**). Currently, the diagonal terms of the matrix are calculated using one force minimizing translation: $K_{XX} = \Delta F_X / \Delta x$ and $K_{YY} = \Delta F_Y / \Delta y$, where Δx and Δy are the components of a single translation of the rigid body. This attributes all the change in force in a certain direction to the displacement in that direction. However, we know that the specimen is a coupled system. It is hypothesized that including the off-diagonal (coupled) terms in the stiffness matrix will allow the load control loop to converge to the force minimized position faster, but the matrix cannot be fully populated using only one translation. Consider a translation that is some linear combination of x and y: $\Delta d = \{\Delta x, \Delta y\}^T$. This translation results in a change in force in both the x- and y-directions: $\Delta F = \{\Delta F_X, \Delta F_Y\}^T$. We use the linear relationship $\Delta F = K \Delta d$ to calculate $\underline{K}$:

$$\begin{Bmatrix} \Delta F_X \\ \Delta F_Y \end{Bmatrix} = \begin{bmatrix} K_{XX} & K_{XY} \\ K_{YX} & K_{YY} \end{bmatrix} \begin{Bmatrix} \Delta x \\ \Delta y \end{Bmatrix}$$

$$\begin{cases} \Delta F_X = K_{XX} \Delta x + K_{XY} \Delta y \\ \Delta F_Y = K_{YX} \Delta x + K_{YY} \Delta y \end{cases}$$

If we set $K_{XY} = K_{YX} = 0$, then there are two equations and two unknowns: $K_{XX} = \Delta F_X / \Delta x$ and $K_{YY} = \Delta F_Y / \Delta y$. If we do not set $K_{XY} = K_{YX} = 0$, then we have two equations and four unknowns. Thus, the system is underdetermined and we cannot solve for any of the terms in the stiffness matrix without another translation. However, we shouldn't wait to calculate $\underline{K}$ until after every other translation because this would be a poor approximation to $\underline{K}$,

resulting in inaccurate values for the calculated force minimizing translations. To fully populate the stiffness matrix, four methods are proposed.

Method #1: apply two perturbations (~ 1 mm) at each position, with one perturbation being parallel to the global X axis and the other being parallel to the global Y axis. This allows the full stiffness matrix to be calculated at each position:

$$\text{Perturbation \#1: } \Delta x_1 \neq 0, \Delta y_1 = 0 \rightarrow \Delta F_{X1}, \Delta F_{Y1}$$

$$\text{Perturbation \#2: } \Delta x_2 = 0, \Delta y_2 \neq 0 \rightarrow \Delta F_{X2}, \Delta F_{Y2}$$

$$K_{XX} = \frac{\Delta F_{X1}}{\Delta x_1}$$

$$K_{XY} = \frac{\Delta F_{X2}}{\Delta y_2}$$

$$K_{YX} = \frac{\Delta F_{Y1}}{\Delta x_1}$$

$$K_{YY} = \frac{\Delta F_{Y2}}{\Delta y_2}$$

Method #2: apply two perturbations (~ 1 mm) at each position, with one perturbation being a linear combination of global X and Y and the other perturbation being orthogonal to the first one. By using global components of the perturbations, we can calculate the full stiffness matrix at each position.

$$\text{Perturbation \#1: } \Delta x_1 \neq 0, \Delta y_1 \neq 0 \rightarrow \Delta F_{X1}, \Delta F_{Y1}$$

$$\text{Perturbation \#2: } \Delta x_2 \neq 0, \Delta y_2 \neq 0 \rightarrow \Delta F_{X2}, \Delta F_{Y2}$$

$$K_{XX} = -\frac{\Delta F_{X2}\Delta y_1 - \Delta F_{X1}\Delta y_2}{-\Delta x_2 \Delta y_1 + \Delta x_1 \Delta y_2}$$

$$K_{XY} = \frac{-\left(-\Delta F_{X2}\Delta x_1 + \Delta F_{X1}\Delta x_2\right)}{-\Delta x_2 \Delta y_1 + \Delta x_1 \Delta y_2}$$

$$K_{YX} = \frac{-\left(\Delta F_{Y2}\Delta y_1 - \Delta F_{Y1}\Delta y_2\right)}{-\Delta x_2 \Delta y_1 + \Delta x_1 \Delta y_2}$$

$$K_{YY} = \frac{-\left(-\Delta F_{Y2}\Delta x_1 + \Delta F_{Y1}\Delta x_2\right)}{-\Delta x_2 \Delta y_1 + \Delta x_1 \Delta y_2}$$

Method #3: Limit the force minimizing translations in a stepwise fashion:

$$\text{iteration 1: } \Delta x_1 \neq 0, \Delta y_1 = 0 \rightarrow \Delta F_{X1}, \Delta F_{Y1}$$

$$\text{iteration 2: } \Delta x_2 = 0, \Delta y_2 \neq 0 \rightarrow \Delta F_{X2}, \Delta F_{Y2}$$

$$\vdots$$

Three of the four terms in the stiffness matrix may be calculated at each position. Refer to method #1 to see that only two of the terms may be calculated when the displacement in one direction is zero. The third term is known through symmetry:

$$\text{iteration 1: } K_{XX} = \frac{\Delta F_{X1}}{\Delta x_1},\ K_{YX} = \frac{\Delta F_{Y1}}{\Delta x_1},\ K_{XY} = K_{YX}$$

$$\text{iteration 2: } K_{XY} = \frac{\Delta F_{X2}}{\Delta y_2},\ K_{YY} = \frac{\Delta F_{Y2}}{\Delta y_2},\ K_{YX} = K_{XY}$$

$$\vdots$$

The fourth term (K_{YY} for odd numbered iterations, K_{XX} for even numbered iterations) is carried over from the previous calculation. Clearly, an initial guess for $\underline{K}$ is required for this method to work.

Method #4: Limit the translations in a stepwise fashion to calculate three of the four stiffness terms, as in method #3, but apply a perturbation in the orthogonal direction to find the

fourth term. This method is similar to method #1 expect that only one perturbation is applied in this method, whereas two perturbations are applied in method #1.

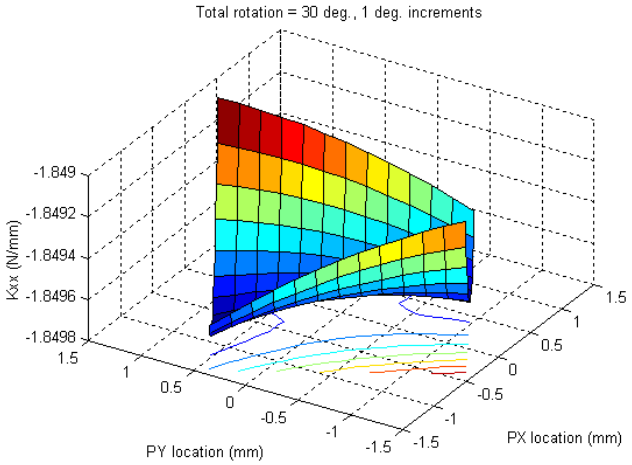
To evaluate the proposed change to stiffness matrix population (**Figure 64**), outcome measures are the calculated global stiffness terms (compared to analytical stiffness), the distance of the final load control position from the true force minimized position (for simulations only), the number of iterations required to minimize the force and the amount of work put into the system. For simulations, the work of each spring can be summed to find the total work in the system. However, this isn't convenient for the experimental system, so it will have to be approximated as $F_X \Delta x + F_Y \Delta y$. The outcome measures are compared to the currently used diagonal stiffness matrix to draw a conclusion about which method to use in the new hybrid control algorithm.

Test #	Input parameter varied	Testing procedure	Output parameter of interest	Expected outcome
6	COR location	Create 13x13 grid of CORs ($-60 \le X \le 60$ mm and $-60 \le Y \le 60$ mm). Rotate around each COR by 1 degree until reach 30 degrees. Use numerical matrices 1-4 for load control.	Stiffness matrix (Kxx,Kxy,Kyy)	Similar to analytical stiffness terms
			No. of iterations to reach minimized force	> 2 iterations
			Distance of center of bar from true force min. position	Very close to zero
			Potential energy (work)	Very similar to test 1b

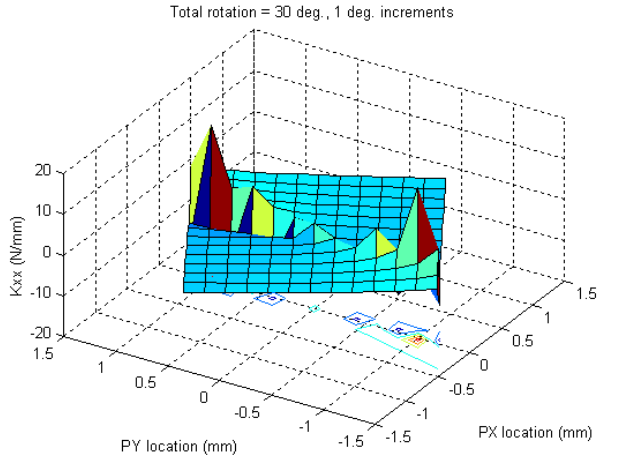
Figure 64 Evaluation of proposed changes to load control

Figure 65 - Figure 69 show representative data for test 6. **Figure 65 - Figure 67** show that methods #1 and #2 calculate the correct stiffness values for K_{XX}, K_{XY} and K_{YY} as compared

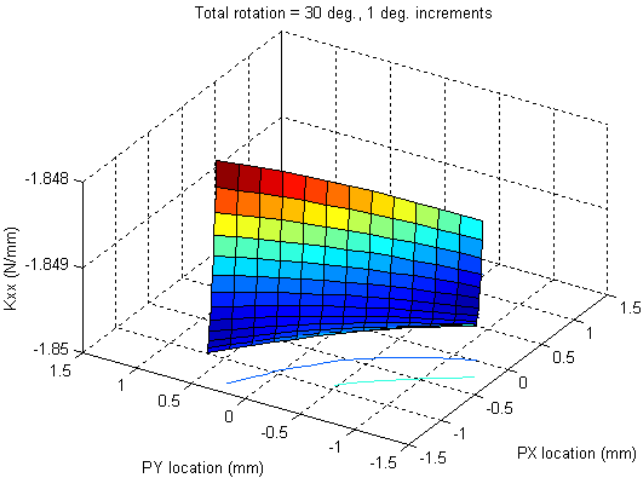
to the analytical values. Methods #3 and #4 also calculate correct values for K_{XX}, K_{XY} and K_{YY}, except for translations of the bar along the global X or Y axes, while the current diagonal stiffness calculation does not calculate the correct stiffness values for K_{XX}, K_{XY} and K_{YY} at any COR location.


Total rotation = 30 deg., 1 deg. increments
Kxx (N/mm)
-1.849
-1.8492
-1.8494
-1.8496
-1.8498
PY location (mm)
PX location (mm)

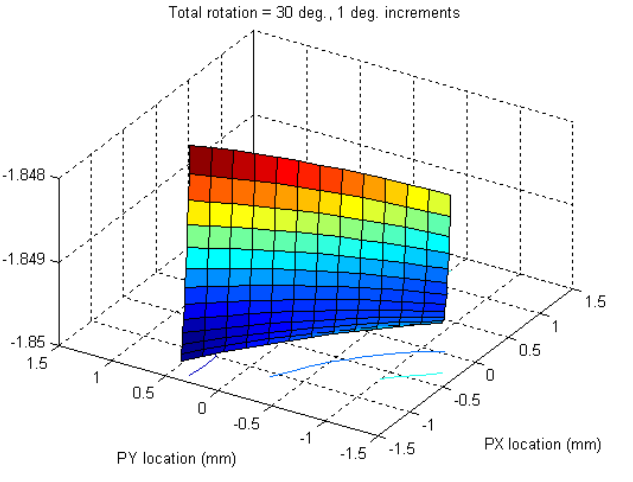
a


Total rotation = 30 deg., 1 deg. increments
Kxx (N/mm)
20
10
0
-10
-20
PY location (mm)
PX location (mm)

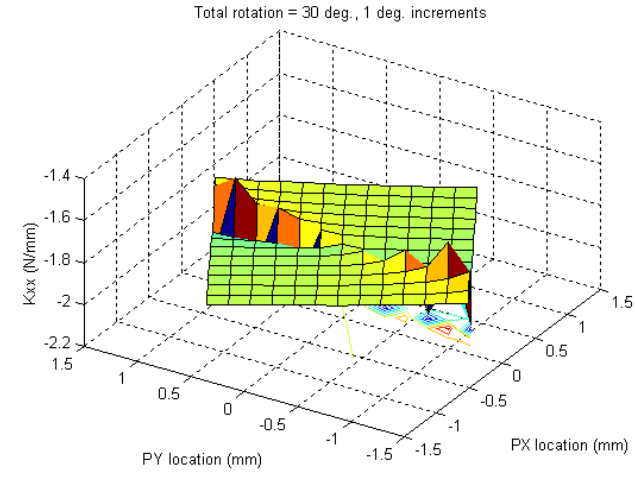
b


Total rotation = 30 deg., 1 deg. increments
Kxx (N/mm)
-1.848
-1.849
-1.85
PY location (mm)
PX location (mm)

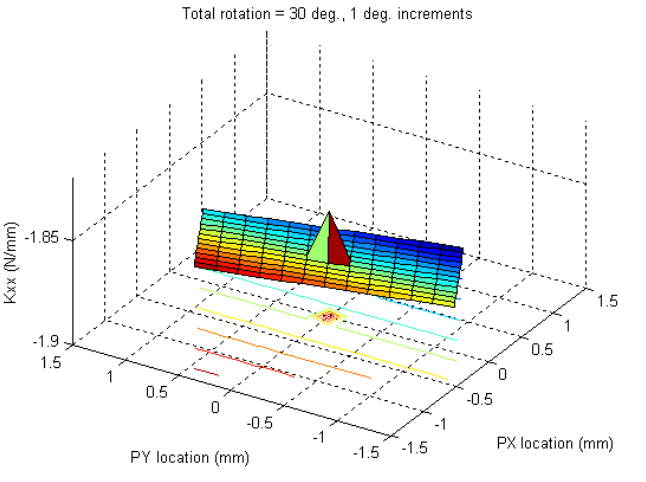
c


Total rotation = 30 deg., 1 deg. increments
Kxx (N/mm)
-1.848
-1.849
-1.85
PY location (mm)
PX location (mm)

d


Total rotation = 30 deg., 1 deg. increments
Kxx (N/mm)
-1.4
-1.6
-1.8
-2
-2.2
PY location (mm)
PX location (mm)

e


Total rotation = 30 deg., 1 deg. increments
Kxx (N/mm)
-1.85
-1.9
PY location (mm)
PX location (mm)

f

Figure 65 Values of K_{XX} for different calculation methods (**a**) analytical solution (**b**) using current method (**c**) using proposed method #1 (**d**) using proposed method #2 (**e**) using proposed method #3 (**f**) using proposed method #4

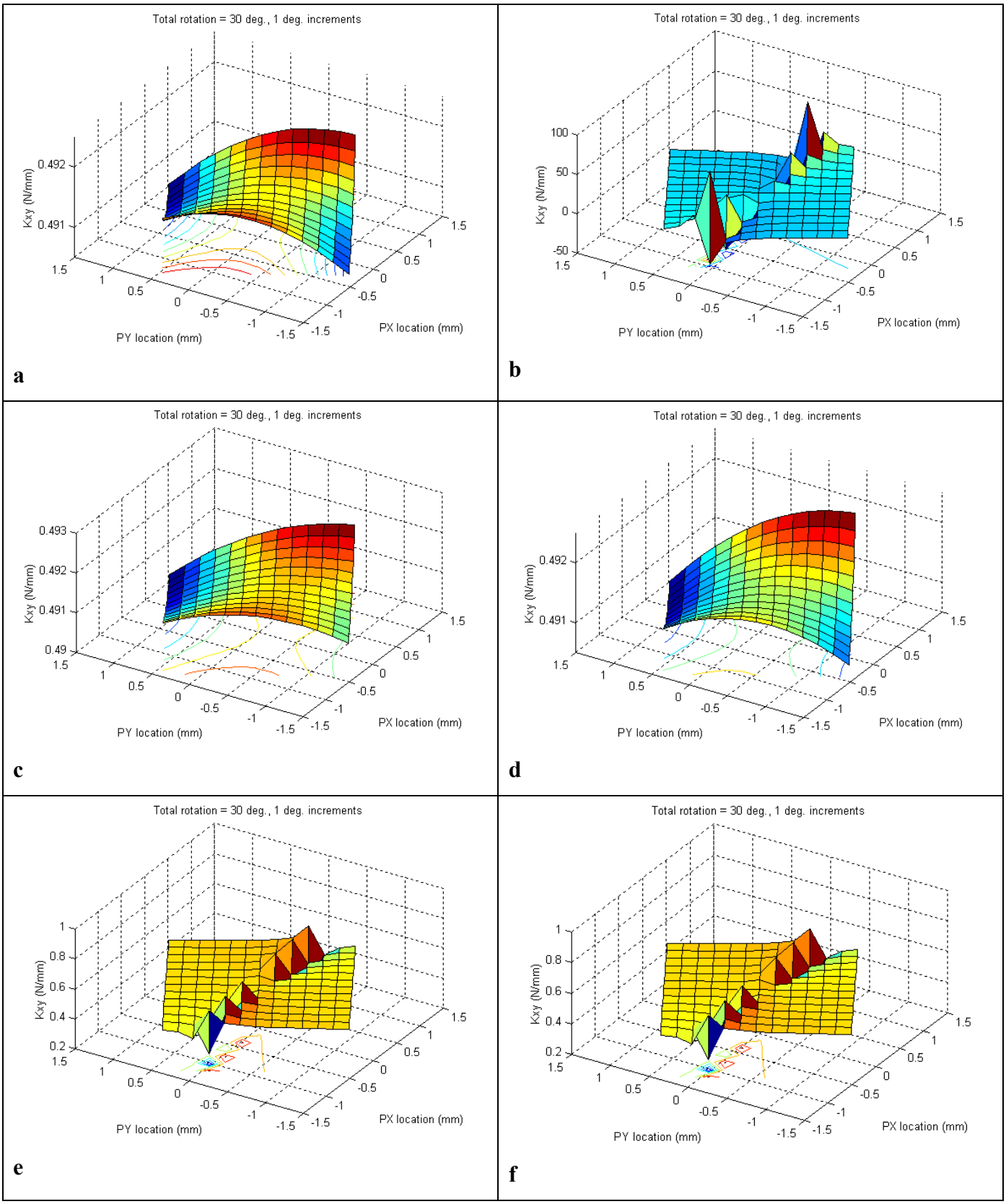
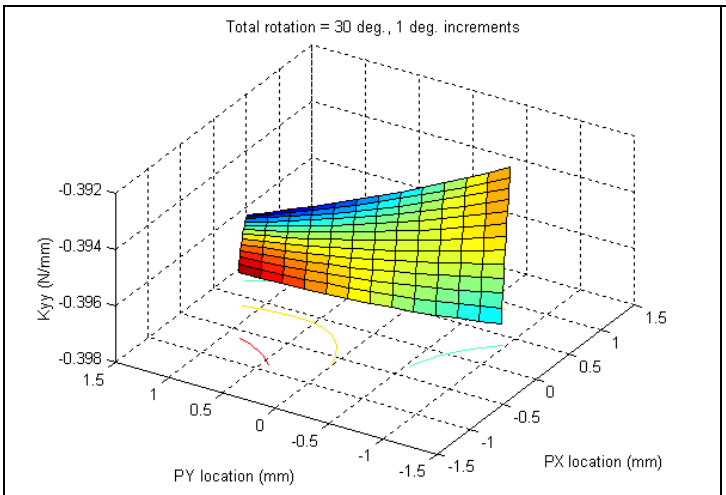
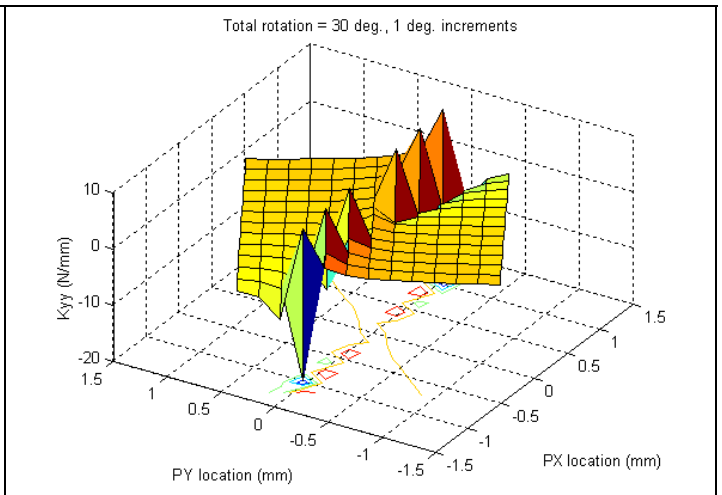

Total rotation = 30 deg., 1 deg. increments
Kxy (N/mm)
PY location (mm)
PX location (mm)
a
b
c
d
e
f

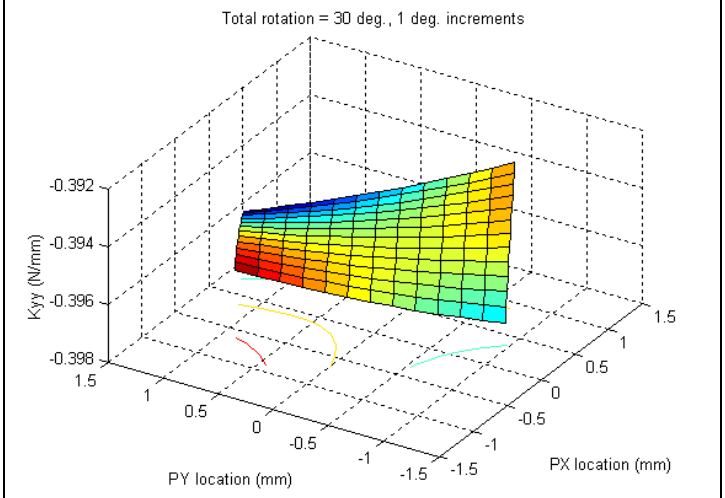
Figure 66 Values of K_{XY} for different calculation methods (**a**) analytical solution (**b**) using current method (**c**) using proposed method #1 (**d**) using proposed method #2 (**e**) using proposed method #3 (**f**) using proposed method #4


Total rotation = 30 deg., 1 deg. increments
Kyy (N/mm)
-0.392
-0.394
-0.396
-0.398
PY location (mm)
PX location (mm)
1.5
1
0.5
0
-0.5
-1
-1.5

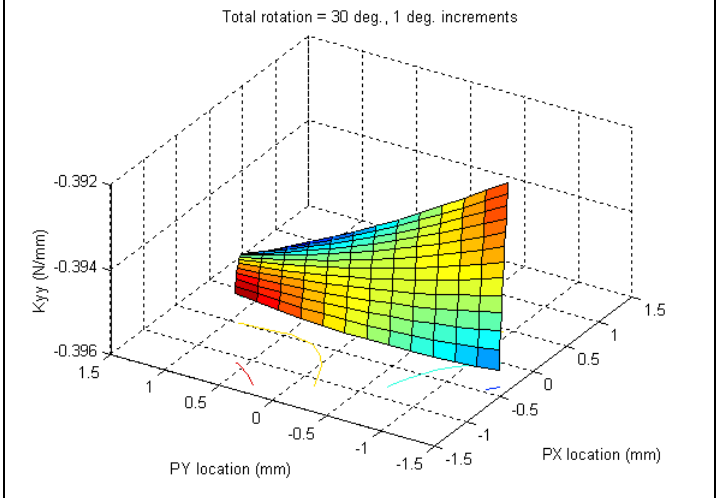
a


Total rotation = 30 deg., 1 deg. increments
Kyy (N/mm)
10
0
-10
-20
PY location (mm)
PX location (mm)
1.5
1
0.5
0
-0.5
-1
-1.5

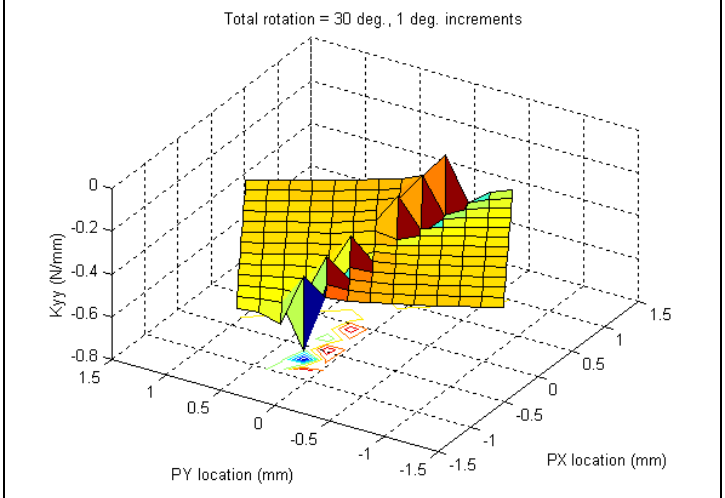
b


Total rotation = 30 deg., 1 deg. increments
Kyy (N/mm)
-0.392
-0.394
-0.396
-0.398
PY location (mm)
PX location (mm)
1.5
1
0.5
0
-0.5
-1
-1.5

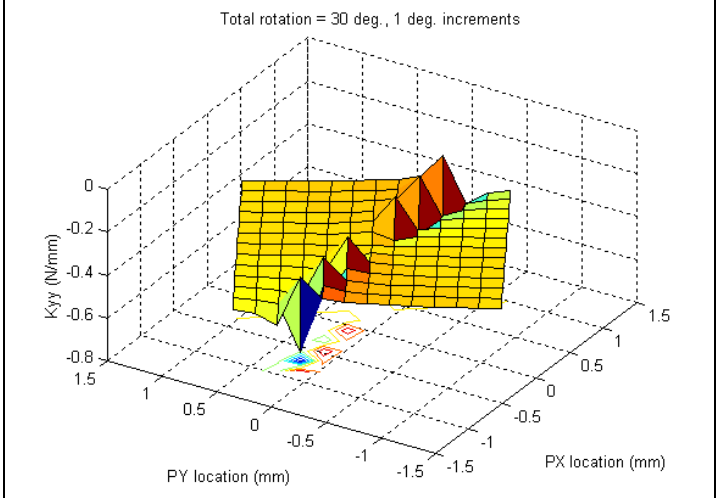
c


Total rotation = 30 deg., 1 deg. increments
Kyy (N/mm)
-0.392
-0.394
-0.396
PY location (mm)
PX location (mm)
1.5
1
0.5
0
-0.5
-1
-1.5

d


Total rotation = 30 deg., 1 deg. increments
Kyy (N/mm)
0
-0.2
-0.4
-0.6
-0.8
PY location (mm)
PX location (mm)
1.5
1
0.5
0
-0.5
-1
-1.5

e


Total rotation = 30 deg., 1 deg. increments
Kyy (N/mm)
0
-0.2
-0.4
-0.6
-0.8
PY location (mm)
PX location (mm)
1.5
1
0.5
0
-0.5
-1
-1.5

f

Figure 67 Values of K_{YY} for different calculation methods (**a**) analytical solution (**b**) using current method (**c**) using proposed method #1 (**d**) using proposed method #2 (**e**) using proposed method #3 (**f**) using proposed method #4

Figure 68 shows that forces created during rotation are reduced using the current diagonal stiffness matrix, even if the correct stiffness values are not calculated. This plot is representative of the full grid of CORs and for each proposed method of stiffness calculation. **Figure 69** shows that method #3 results in the smallest error and few iterations. Methods #1 and #2 take the fewest iterations to minimize force at large rotation angles, but method #3 only takes one or two iterations longer. Method #4 consistently results in a much higher number of iterations even though the error is comparable to the other three methods. It can also be seen in **Figure 69** that when the diagonal stiffness matrix is used during load control, the number of iterations suddenly drops from about 20 iterations at about 2 iterations at 18°, whereas the iterations either decrease predictably or remain low when using one of the full stiffness matrices. There are several CORs in the grid for which this is true. It is reasonable to say that for these CORs the diagonal stiffness terms are either underestimated or overestimated. If the stiffness terms are underestimated, then a large displacement is calculated when the matrix is inverted. The center of the bar is limited to a translation of 1 mm in each direction, so the bar is overshooting the true force minimized position in this case. If the stiffness terms are overestimated, then a small displacement is calculated when the matrix is inverted. The center of the bar then is undershooting the true force minimized position on the first iteration, but is able to minimize force within 2 or 3 iterations because the displacements are not too small.

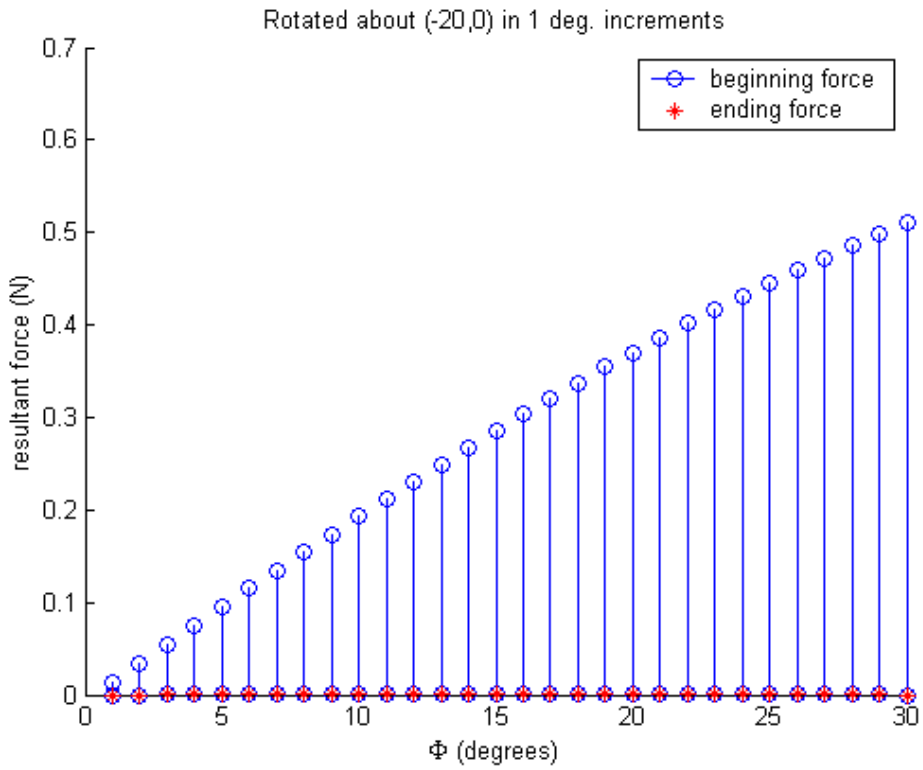


Figure 68 Force created during rotation is minimized by using the current diagonal stiffness matrix

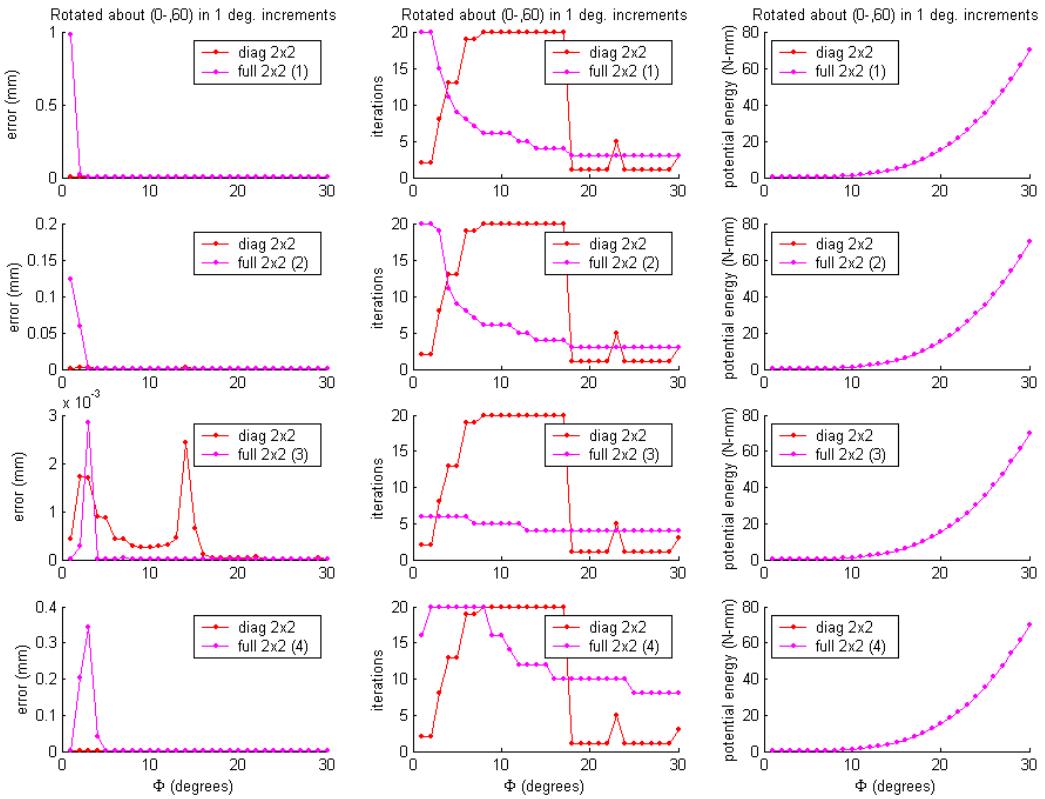


Figure 69 Representative data for characterization of performance of four different methods of calculating the fully populated stiffness matrix as compared with the current diagonal stiffness matrix (simulation set 6), rotated about a COR located at (0,-60) in the global XY -plane in $\phi = 1°$ increments up to $\Phi = 30°$, the left column shows the distance (in mm) of the final force minimized position from the true force minimized position (the global origin) vs. rotation angle (outcome 1), the middle column shows the number of iterations required to minimize force vs. rotation angle (outcome 2) and the right column shows the potential energy (in Newton-mm) in the system vs. rotation angle (outcome 3), the top row of plots shows results for proposed method #1, the next row shows results for proposed method #2, the next row shows results for proposed method #3 and the bottom row shows results for proposed method #4

Choosing which method to use for calculating the stiffness matrix experimentally depends on the desired use of the matrix. If the user simply wishes to use the matrix for load control without concern to the actual stiffness values that are being calculated, method #3 should be used because it results in the least error, takes a small number of iterations to minimize force and results in a faster test because perturbations do not need to be applied at every position.

However, if the user wants to approximate the specimen's stiffness, method #1 or method #2 should be used because these methods not only perform well during load control, but also closely match the analytical stiffness. Method #3 will be used in the new hybrid control algorithm.

Table 2 Tabulated results of simulation set 6 showing average number of force minimizing iterations for the current method (diagonally populated stiffness matrix), proposed method #1 (apply two perturbations parallel to global X and Y axes), proposed method #2 (apply two orthogonal perturbations in global XY-plane), proposed method #3 (constrain force minimizing translations to stairsteps parallel to global X and Y axes) and proposed method #4 (constrain translations as in method #3 and apply one orthogonal perturbation)

	Average iterations
Current method	5.3
Method #1	4.7
Method #2	4.8
Method #3	5.2
Method #4	12.7

6.3 Improved Hybrid Control Algorithm

After identifying the best performing changes to displacement and load control, they were combined into a new hybrid control system and the new algorithm is compared with the old one. (**Figure 70**) A 13x13 grid of CORs ($-60 \le X \le 60$ mm and $-60 \le Y \le 60$ mm) was created. The center of the bar was rotated about each COR by $\phi = 1°$ increments until $\Phi = 30°$. After each incremental rotation, method #3 was used during the load control loop to calculate the global stiffness terms and the force minimizing translations. The preferred COR was calculated using Challis' method by using the force minimized positions at $0°$ and $5°$, $5°$ and $10°$, etc.

The COR was updated using feedback. Outcome measures for evaluating the new hybrid control algorithm are the peak force created during rotation, the number of iterations required to minimize force, the distance of the center of the bar from the global origin at the final load control step and the work done to the system.

Test #	Input parameter varied	Testing procedure	Output parameter of interest	Expected outcome
7	COR location	Create 13x13 grid of CORs ($-60 \le X \le 60$ mm and $-60 \le Y \le 60$ mm). Rotate around each COR by $\phi = 1°$ increments until reach $\Phi = 30°$. Calculate stiffness using method #3. Calculate preferred COR every $5°$ using Challis method. Add noise to marker position. Update COR using feedback method.	Peak loads generated during rotation	Peak loads decreased with new hybrid control algorithm.
			Number of iterations to reach minimum force	Number of iterations to reach minimum force reduced with new hybrid control algorithm.
			Distance of center of bar from global origin.	Because forces and marker positions are known analytically, the error will not change much from old algorithm to new.
			Work done to model	Work done to model unchanged with new hybrid control system

Figure 70 Evaluation of new hybrid control algorithm

Figure 71 shows a representative plot for comparing old and new algorithm outcome measures for simulations. As expected, the work remained unchanged from the old algorithm to the new one. The peak force decreased when using the new algorithm, but the number of iterations increased. However, this increase is still within an acceptable range. The distance of

the center of the bar from the global origin is very small, on the order of 10^{-3} mm. The plots generated for the full grid of CORs show that the new hybrid control algorithm decreases the peak force created during rotation, does not add additional work to the system, results in very little error during load control and takes a relatively small number of iterations to minimize force.

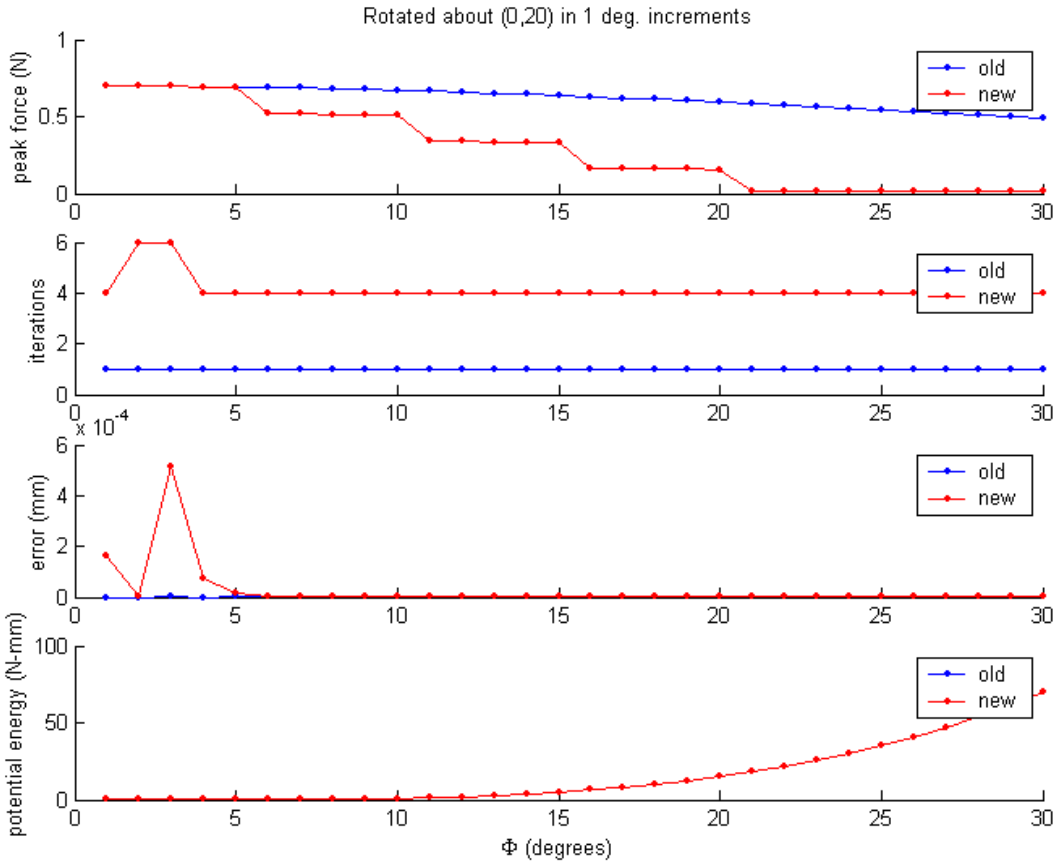


Figure 71 Representative data for characterization of performance of new hybrid control algorithm as compared with the old algorithm (simulation set 7), rotated about a COR located at (0,20) in the global XY-plane in $\phi = 1°$ increments up to $\Phi = 30°$, the top row of the plot shows the peak force (in Newtons) created during rotation vs. rotation angle (outcome 1), the second row shows the number of iterations required to minimize force vs. rotation angle (outcome 2), the third row shows the distance (in mm) of the final force minimized position from the true force minimized position (the global origin) vs. rotation angle (outcome 3) and the bottom row shows the potential energy (in Newton-mm) in the system vs. rotation angle (outcome 4).

Table 3 Tabulated results of simulation set 7 showing range of peak force (in Newtons) and average number of force minimizing iterations for the current hybrid control algorithm (no COR update and diagonally populated stiffness matrix) and the new hybrid control algorithm (feedback COR update and fully populated stiffness matrix calculated using method #3)

	Range of peak force (N)	Average iterations
Old algorithm	1.7611 – 2.7476	5.3
New algorithm	0.1563 – 2.3866	4.9

7.0 DISCUSSION

7.1 Summary

A tabletop robotic/UFS testing system that interacts with Matlab to apply hybrid control to testing of lumbar spines was developed. The experimental system was validated analytically using rigid body transformations simulated in Matlab. Changes to displacement control and load control were tested and an improved hybrid control algorithm was developed that may be used for delineating biomechanical properties of the human lumbar spine. Specific aim 2a was performed to test the hypothesis that allowing the user-defined COR to move locally as well as globally would decrease the peak force created during rotation and decrease the number of iterations required to minimize that force. Results from section 6.1 show that the general rigid body-spring model used during simulations supported this hypothesis. Specific aim 2b was performed to test the hypotheses that fully populating the stiffness matrix would decrease the number of iterations required to minimize force created during rotation and that the fully populated matrix would provide a better approximation of the true stiffness values than the diagonal stiffness matrix. Results from section 6.2 show that the number of iterations was reduced for proposed methods #1 - #3, while the iterations increased for proposed method #4. Results from this section also show that methods #1 and #2 provided the best approximation to the true stiffness values of the general rigid body-spring model for all CORs in the grid, while methods #3 and #4 closely approximated the true stiffness values for most of the CORs in the grid. Results from section 6.3 show that after combining the proposed changes to both the displacement and load control loops, the range of peak force created during rotation about the grid of user-defined CORs and the number of iterations required to minimize that force both

decreased. The analytical and experimental platforms will work in conjunction for future studies of advanced control methods for spine testing. Because Matlab was used both analytically and experimentally, the programs may be executed on any PC.

7.2 Limitations and Future Work

Despite some apparent limitations of using hybrid control with a robotic/UFS testing system, including specimen viscoelasticity and some slop in the manipulator's joints, the testing system described in the above sections provides a controllable testing apparatus with a control algorithm that is hypothesized to be similar to what the body employs in vivo. Control algorithms can always be improved and the research done in this thesis is no exception. For further improvement to the displacement control loop, another method of updating the COR should be considered: feedforward. Feedforward can be used if some pattern is recognized in the path of the calculated CORs. Suppose that the preferred COR is calculated every $5°$ and the path made by $n\phi/5$ calculated CORs looks approximately quadratic. Then a quadratic function can be fit to the path of CORs and the position of the next COR can be predicted. This predicted COR is fed forward and used for the next rotation. Least squares is one possible method of fitting a function to the COR path. To use least squares, a curve is fit to a set of data points: (x_i, Y_i) for $i = 1, 2, ..., n$. The least squares approximation to the data is a function of x_i: $y = f(x_i)$. In order to ideally use least squares with feedforward control, we should know the form of y a priori. However, this is highly unlikely unless many specimens have been tested and a pattern emerges. One option is to plot the COR path during the test. If a pattern emerges from the plot, the user would have to be allowed to stop the test and suggest the form of y to the

Matlab program. Problems with this include longer tests and keeping the specimen at some position other than neutral for extended periods of time. Another option is to fit a linear function between two successive CORs. This may be a poor approximation, but further tests will have to elucidate that.

Another improvement to displacement control involves fully characterizing the manipulator's position inaccuracy. The simple one DOF experiment from section 5.4 shows that the manipulator's position inaccuracy due to varying weight on the end-effector may be corrected using load cell data for a certain position in space. Future work should be done to fully characterize the position inaccuracy in each direction for the workspace encountered during specimen testing. This will improve COR calculation because marker position noise will be reduced to nearly zero.

A further limitation of the current study is the tacit assumption of sagittal plane symmetry of the specimens. The hybrid control algorithm constrains motions to the mid-sagittal plane—thus non-sagittal force (F_x) and moments (M_y and M_z) are not explicitly controlled, even though coupled loads in these non-sagittal DOF have been found to be rather minimal.[13] In the future, we would like to extend the control algorithm to three dimensions to enable minimization of all coupled loads. One approach may be to base the three-dimensional hybrid control algorithm on finite rotations about and translations along a continually updated screw displacement axis (SDA) or helical axis of motion (HAM). In addition to correcting for sagittal plane asymmetry, full three-dimensional motion can elucidate the altered kinematics of clinically unstable specimens. The hybrid control algorithm itself can elucidate the kinematics of either clinically stable or unstable specimens because it finds the passive path of the specimen. However, suppose that a specimen has undergone a unilateral facetectomy. In this case,

confining motion to the sagittal plane may not find the passive path during flexion/extension because it is possible that the passive path does not lie in this plane. Finding the altered passive path moments then requires full three-dimensional motion.

7.3 Conclusion

This thesis has described development of analytical and experimental platforms and refinement of the testing algorithm for delineating spine kinetics. The analytical platform provides the ability to test experimental protocols and elucidate subtle complexities of any given change to the testing algorithm that may be lost in the experimental system. The robotic/UFS testing system provides a system that is totally controlled with the regulated application of six DOF loads and displacements. The refined hybrid control algorithm produces better data by reducing imposition of a COR that the specimen does not prefer and including coupled stiffness terms that had previously been ignored. Utilizing an off the shelf, readily available language such as Matlab introduces uniformity into robotic systems. Built-in functions in Matlab and a PC with a fast processor allow the user to simplify the program and implement complicated control systems. If several different types of controllers are to be used together, each with its own language, a single PC-based language can be used to standardize the system. This allows programs to be written and shared between any number of users with an external PC interface system. In summary, the robotic/UFS testing system with refined hybrid control facilitates improved biomechanical testing of spinal segments, thus leading to a better understanding and treatment of spinal pathologies.

APPENDICES

APPENDIX A

Matlab code for simulations

hand14a.m is a Matlab script that simulates hybrid control of a rigid body-spring model. Parameters for the spring model are input at the beginning of the script. Several functions are called by hand14a.m. They are included in this appendix after hand14a.m in the order in which they appear in the script.

```
% hand14a.m
% analytical rigid body-spring model
% rotate about COR in phi degree increments
% Amy Loveless
% 3/12/2003

clear all

% =======================================================
% General model parameters
% spring constants (N/mm)
ka = 1;
kb = 1;
% resting length of springs (mm)
lar = 60;
lbr = 60;
% length of spring when inserted into system (mm)
la_init = 60;
lb_init = 60;
% length of half of bar (mm)
L = 40;
% local positions of nodes attached to rigid body (mm)
axy = [-L 0];
bxy = [L 0];
% global positions of fixed nodes (mm)
jaXY = [-(L+la_init) 0];
jbXY = [L+lb_init 0];
theta = 0;

%% =======================================================
%% Physical model parameters
%% spring constants (N/mm)
%ka = 12.033; % blue
%kb = 11.55;  % red
%% resting length of springs (mm)
%lar = 1.955*2.54*10; % blue
```

```
%lbr = 2.936*2.54*10; % red
%% length of springs when inserted into system (mm)
%la_init = lar + 0.382*2.54*10; % blue
%lb_init = lbr + 0.387*2.54*10; % red
%% length of half of bar (mm)
%L = 28;
%% local positions of nodes attached to rigid body (mm)
%axy = [-L 0];
%bxy = [L 0];
%% global positions of fixed nodes (mm)
%jaXY = [-cos(70*pi/180)*(L+la_init) -sin(70*pi/180)*(L+la_init)];
%jbXY = [cos(70*pi/180)*(L+lb_init) sin(70*pi/180)*(L+lb_init)];
%theta = 70;
%% ======================================================

% local positions of markers attached to rigid body (mm)
mark1xy = [-L 0];
mark2xy = [L 0];

% amount of rotation from resting position (rad)
phi = deg2rad(1);
PHI = 30;
kk = round(PHI/rad2deg(phi));

% limit magnitude of translations
t_lim = 1; % mm
const_stiff = 10; % N/mm

cor_lim = 5; % mm
ftarget = 10^-5; % N
iterations = 20;
tick = 0;
index = 1;

% define unit vectors
ihat = [1; 0; 0];
jhat = [0; 1; 0];
khat = [0; 0; 1];

% % initialize graph display for forces and moments
% fh = figure('Position', [150 100 600 600], 'Color', 'w', 'doublebuffer',
'on');
% fgraph = axes('Parent', fh, 'Position', [.1 .6 .8 .35], 'XLim', [0
iterations], 'YLim', [-50 50], 'nextplot', 'add');
% forceufs = line('XData', 0, 'YData', 0, 'Color', 'k', 'Marker', '.',
'markersize', 8, 'erasemode', 'none');
%
% % initialize variables for drawing position of bar
% ah = axes('Position', [.1 .05 .5 .5], 'GridLineStyle', ':', 'XLim', [-110
110], 'YLim', [-110 110], 'nextplot', 'add');
% set(ah, 'XColor', [.7 .7 .7], 'YColor', [.7 .7 .7], 'XGrid', 'on', 'YGrid',
'on');
% bar = line('xdata', [0 0], 'ydata', [0 0], 'color', 'k', 'linewidth', 10);
% springa = line('xdata', [0 0], 'ydata', [0 0], 'color', 'b', 'linewidth',
3);
% springb = line('xdata', [0 0], 'ydata', [0 0], 'color', 'r', 'linewidth',
3);
```

```
%
% handles = [fh, fgraph, forceufs, ah, bar, springa, springb];

for var = 1:1:13
for cycle = 1:2
for z = 6:5:11
  for i = 13:13
     for j = var:var

        if (z == 11) & (cycle == 1)
           break
        elseif (z == 6) & (cycle == 2)
           break
        end

        theta = deg2rad(0);
        thetaG(i,j) = theta;
        temp = theta;
        thetaCOR(i,j) = 0;
        thetaa(i,j) = 0;
        thetab(i,j) = 0;
        thetaja(i,j) = 0;
        thetajb(i,j) = 0;

        % make 13x13 grid of points for COR and translation of center of bar
(mm)
        % these points are defined FROM X,Y TO xTCS0,yTCS0
        PXY = [0 0];
        if (cycle == 1) | (cycle == 2)
            corX(i,j) = (-60+(j-1)*10)*cos(theta)+(60-(i-1)*10)*sin(theta);
            corY(i,j) = (60-(i-1)*10)*cos(theta)-(-60+(j-1)*10)*sin(theta);
            corXY = [corX(i,j) corY(i,j)];
        end
%         dxy = [-60+(j-1)*10 60-(i-1)*10];
        dxy = [0 0];

        % initialize variables
        work = 0;
        u = 0;
        f_temp = 0*ones(1,6);
        fmw = f_temp;
        index = 1;
        numdiagK2 = -ones(2,2);, numfullK2 = -ones(2,2);, numdiagK3 = -
ones(3,3);, numfullK3 = -ones(3,3);
        pertK = -ones(2,2);
        fminmzd(j,1:kk) = 0;

        for k = 1:kk

%             % COR update option 1:
%             % store calculated CORs to be replayed in the next cycle
%             if cycle ~= 1
%                 corXY = [cXY(1,index) cXY(2,index)];
%             end

            % COR update option 2:
            % calculate & update COR every 5 degrees
```

```
%             if (isequal(int2str((k+4)/5), num2str((k+4)/5))) & (k > 1) &
(cycle ~= 1)
            if (isequal(int2str((k+4)/5), num2str((k+4)/5))) & (k > 1) &
(cycle ~= 1) & (z == 11)
                corXY = [cXY(1,index-1) cXY(2,index-1)];
            end

            % TRANSFORMATIONS
=====================================================

            % (P1X, P1Y) is the global position of reference point P after
planar motion
            [P1X(i,j), P1Y(i,j), TG0, TG1, corXY] =
refpointtrans(thetaG(i,j), PXY, thetaCOR(i,j), corXY, phi, dxy, 'g');
            PX(k,1) = P1X(i,j);, PY(k,1) = P1Y(i,j);

            % call node 1 "node a" and node 2 "node b"

            % (a1X, a1Y) is the global position of node a after planar motion
(from X,Y to xa1,ya1)
            % La0 and La1 are the lengths of spring a at time t0 and time t1
(mm)
            [a1X(i,j), a1Y(i,j), la0, la1, T0a0, T1a1] =
nodaltrans(thetaa(i,j), axy, thetaja(i,j), jaXY, TG0, TG1);
            if isequal(int2str((k+4)/5), num2str((k+4)/5)), TGa0 = TG0*T0a0;,
a0X = TGa0(1,4);, a0Y = TGa0(2,4);, end
%             TGa0 = TG0*T0a0;, a0X = TGa0(1,4);, a0Y = TGa0(2,4);

            % (b1X, b1Y) is the global position of node b after planar motion
(from X,Y to xb1,yb1)
            % Lb0 and Lb1 are the lengths of spring b at time t0 and time t1
(mm)
            [b1X(i,j), b1Y(i,j), lb0, lb1, T0b0, T1b1] =
nodaltrans(thetab(i,j), bxy, thetajb(i,j), jbXY, TG0, TG1);
            if isequal(int2str((k+4)/5), num2str((k+4)/5)), TGb0 = TG0*T0b0;,
b0X = TGb0(1,4);, b0Y = TGb0(2,4);, end
%             TGb0 = TG0*T0b0;, b0X = TGb0(1,4);, b0Y = TGb0(2,4);

            % calculate change in length of spring a at time t0 and time t1
(mm)
            deltaa0(i,j) = sqrt(la0'*la0) - lar;
            deltaa(i,j) = sqrt(la1'*la1) - lar;

            % calculate change in length of spring b at time t0 and time t1
(mm)
            deltab0(i,j) = sqrt(lb0'*lb0) - lbr;
            deltab(i,j) = sqrt(lb1'*lb1) - lbr;

            % (mark1X, mark1Y) & (mark2X, mark2Y) are the global positions of
markers 1 & 2
            if isequal(int2str((k+4)/5), num2str((k+4)/5))
                T0mark1 = trans(0, mark1xy(1), mark1xy(2), 0);, TGmark1 =
TG0*T0mark1;
                T0mark2 = trans(0, mark2xy(1), mark2xy(2), 0);, TGmark2 =
TG0*T0mark2;
                mark1X(k) = TGmark1(1,4);, mark1Y(k) = TGmark1(2,4);
                mark2X(k) = TGmark2(1,4);, mark2Y(k) = TGmark2(2,4);
```

```
            end
            [mark1X(k+1), mark1Y(k+1)] = nodaltrans(0, mark1xy, thetajb(i,j),
jbXY, TG0, TG1);
            [mark2X(k+1), mark2Y(k+1)] = nodaltrans(0, mark2xy, thetajb(i,j),
jbXY, TG0, TG1);

            % FORCES/MOMENTS
=========================================================

            % calculate global force at time t0 due to spring a
            % this is the force present in the system at time t0, but would
not
            % appear in the UFS when the robot is initially attached to the
model.
            Fa0 = force2(ka, deltaa0(i,j), la0, thetaG(i,j)+thetaa(i,j), 0,
0, 0);
            % calculate global force at time t1 due to spring a
            Fa1 = force2(ka, deltaa(i,j), la1, thetaG(i,j)+thetaa(i,j)+phi,
0, 0, 0);

            % calculate global force at time t0 and time t1 due to spring b
            Fb0 = force2(kb, deltab0(i,j), lb0, thetaG(i,j)+thetab(i,j), 0,
0, 0);
            Fb1 = force2(kb, deltab(i,j), lb1, thetaG(i,j)+thetab(i,j)+phi,
0, 0, 0);

            % calculate global forces and moment at point P at time t0
            F0X(i,j) = dot(Fa0+Fb0,ihat);, F0Y(i,j) = dot(Fa0+Fb0,jhat);
            Ma0 = moment1(thetaG(i,j), 0, 0, 0, T0a0(1:3,4), Fa0, khat);
            Mb0 = moment1(thetaG(i,j), 0, 0, 0, T0b0(1:3,4), Fb0, khat);
            M0Z(i,j) = Ma0 + Mb0;

            % calculate global forces and moment at point P at time t1
            F1X(i,j) = dot(Fa1+Fb1,ihat);, F1Y(i,j) = dot(Fa1+Fb1,jhat);
            Ma1 = moment1(thetaG(i,j)+phi, 0, 0, 0, T1a1(1:3,4), Fa1, khat);
            Mb1 = moment1(thetaG(i,j)+phi, 0, 0, 0, T1b1(1:3,4), Fb1, khat);
            M1Z(i,j) = Ma1 + Mb1;

            fmw = [F1X(i,j) F1Y(i,j) 0 0 0 M1Z(i,j)];
            FX(k,1) = F1X(i,j);, FY(k,1) = F1Y(i,j);, MZ(1) = M1Z(i,j);
%             F(j,k) = sqrt(F1X(i,j)^2 + F1Y(i,j)^2);
            F(k,1) = sqrt(F1X(i,j)^2 + F1Y(i,j)^2);

            % find the magnitude of the peak force for each degree of
rotation
            magf(k,1) = sqrt(F1X(i,j)^2 + F1Y(i,j)^2);
            work = (-0.5)*ka*deltaa(i,j)^2 + (-0.5)*kb*deltab(i,j)^2;
            u = 0.5*ka*deltaa(i,j)^2 + 0.5*kb*deltab(i,j)^2;

            % STIFFNESS
=============================================================

            % use analytical solution to find stiffness
            PXYstiff = [P1X(i,j) P1Y(i,j)];
            if (z == 2) | (z == 3)
                [Kxx1(i,j), Kxy1(i,j), Kyx1(i,j), Kyy1(i,j)] = stiff(z, ka,
lar, jaXY, axy, PXYstiff, thetaG(i,j)+phi, kb, lbr, jbXY, bxy);
```

```
                anadiagK2 = [Kxx1(i,j) 0; 0 Kyy1(i,j)];
                anafullK2 = [Kxx1(i,j) Kxy1(i,j); Kyx1(i,j) Kyy1(i,j)];
            end

            % use numerical method to find stiffness
%             if (z == 6) | (z == 8)
            if ((z == 6) | (z == 8)) & (cycle == 1)
                diffload = [fmw(1)-f_temp(1) fmw(2)-f_temp(2) fmw(6)-
f_temp(6)];
                diffdisp = [P1X(i,j)-PXY(1) P1Y(i,j)-PXY(2) phi];
                [Kxx, Kyy, Kzz] = stiff(z, diffload, diffdisp, fmw,
numdiagK3);
                % 2x2 matrix
                numdiagK2 = [Kxx 0; 0 Kyy];
                % 3x3 matrix
                numdiagK3 = [Kxx 0 0; 0 Kyy 0; 0 0 Kzz];
            end

%             % perturb bar to find stiffness
%             if (z == 10) | (z == 11)
%             [Kxxp(i,j), Kxyp(i,j), Kyxp(i,j), Kyyp(i,j)] = stiff(z, 2,
thetaG(i,j)+thetaa(i,j)+phi, la1, thetaG(i,j)+thetab(i,j)+phi, lb1,...
%                     lar, lbr, ka, kb, [F1X(i,j) F1Y(i,j)], thetaG(i,j)+phi,
T1a1(1:3,4), T1b1(1:3,4), M1Z(i,j), [P1X(i,j) P1Y(i,j)],...
%                     thetaCOR(i,j)+phi, corXY, thetaa(i,j), axy,
thetaja(i,j), jaXY, thetab(i,j), bxy, thetajb(i,j), jbXY);
%             pertK = [Kxxp(i,j), Kxyp(i,j); Kyxp(i,j), Kyyp(i,j)];
% %             pertK = [Kxxp(i,j), Kxyp(i,j) Kzxp(i,j); Kyxp(i,j), Kyyp(i,j)
Kyzp(i,j); Kzxp(i,j) Kzyp(i,j) Kzzp(i,j)];
%             end

            % store current force and position
            f_temp = fmw;
            PXY = [P1X(i,j) P1Y(i,j)];

            % housekeeping variables
            KXX(k,1) = pertK(1,1);, KXY(k,1) = pertK(1,2);
            KYX(k,1) = pertK(2,1);, KYY(k,1) = pertK(2,2);
%             invK = pinv([numfullK2(1,1) numfullK2(1,2); numfullK2(2,1)
numfullK2(2,2)]);
%             invKXX(k,1) = invK(1,1);, invKXY(k,1) = invK(1,2);
%             invKYX(k,1) = invK(2,1);, invKYY(k,1) = invK(2,2);

%             % draw position of bar
%             draw2(handles, [P1X(i,j), P1Y(i,j)], [corX(i,j) corY(i,j)],
[a1X(i,j), a1Y(i,j)], [b1X(i,j), b1Y(i,j)], jaXY, jbXY, temp,...
%                 [mark1X(k+1), mark1Y(k+1)], [mark2X(k+1), mark2Y(k+1)],
[F1X(i,j), F1Y(i,j)], tick);

            % update total rotation angle
            theta = theta + phi;
            thetaG(i,j) = theta;
            thetaCOR(i,j) = theta - thetaG(i,j) + k*phi;

            % FORCE MINIMIZATION
==================================================
            for counter = 1:iterations
```

```
%                 if (z == 13) | (z == 12) | (z == 11) | (z == 10)
                if ((z == 13) | (z == 12) | (z == 11) | (z == 10)) & (cycle >
1)
                    dXY = fmin(z, pertK, [fmw(1:2) fmw(6)]', 'y', t_lim);
                    % if counter is even, set translation in X direction to
zero
                    % if counter is odd, set translation in Y direction to
zero
                    if isequal(num2str(counter/2), int2str(counter/2)) %
counter is even
                        dXY(1) = 0;
                    else % counter is odd
                        dXY(2) = 0;
                    end
                elseif z == 9
                    dXY = fmin(z, numfullK3, [fmw(1:2) fmw(6)]', 'y', t_lim);
                elseif z == 8
                    dXY = fmin(z, numdiagK3, [fmw(1:2) fmw(6)]', 'y', t_lim);
                elseif z == 7
                    dXY = fmin(z, numfullK2, [fmw(1:2) fmw(6)]', 'y', t_lim);
%                 elseif z == 6
                elseif (z == 6) & (cycle == 1)
                    dXY = fmin(z, numdiagK2, [fmw(1:2) fmw(6)]', 'y', t_lim);
                elseif z == 5
                    dXY = fmin(z, anafullK3, [fmw(1:2) fmw(6)]', 'y', t_lim);
                elseif z == 4
                    dXY = fmin(z, anadiagK3, [fmw(1:2) fmw(6)]', 'y', t_lim);
                elseif z == 3
                    dXY = fmin(z, anafullK2, [fmw(1:2) fmw(6)]', 'y', t_lim);
                elseif z == 2
                    dXY = fmin(z, anadiagK2, [fmw(1:2) fmw(6)]', 'y', t_lim);
                elseif z == 1
                    dXY = fmin(z, F1X(i,j), F1Y(i,j), const_stiff);
                end

%                 dXY = [-P1X(i,j) -P1Y(i,j)]';

                ddXY(:,1) = dXY;
                dX(k,counter) = dXY(1);
                dY(k,counter) = dXY(2);

                % find global positions of point P, nodes a & b, and markers
1 & 2 at new force minimized position
                [P1X(i,j), P1Y(i,j), TG0, TG1, corXY] =
refpointtrans(thetaG(i,j), PXY, thetaCOR(i,j), corXY, 0, [dXY(1) dXY(2)],
'g');
                PX(k,counter+1) = P1X(i,j);, PY(k,counter+1) = P1Y(i,j);

                [a1X(i,j), a1Y(i,j), la0, la1, T0a0, T1a1] =
nodaltrans(thetaa(i,j), axy, thetaja(i,j), jaXY, TG0, TG1);
                [b1X(i,j), b1Y(i,j), lb0, lb1, T0b0, T1b1] =
nodaltrans(thetab(i,j), bxy, thetajb(i,j), jbXY, TG0, TG1);
                deltaa(i,j) = sqrt(la1'*la1) - lar;
                deltab(i,j) = sqrt(lb1'*lb1) - lbr;
                [mark1X(k+2), mark1Y(k+2)] = nodaltrans(0, mark1xy,
thetajb(i,j), jbXY, TG0, TG1);
```

```
                [mark2X(k+2), mark2Y(k+2)] = nodaltrans(0, mark2xy,
thetajb(i,j), jbXY, TG0, TG1);
%                 corX(i,j) = corXY(1);, corY(i,j) = corXY(2);

%                 % draw position of bar
%                 tick = tick + 1;
%                 draw2(handles, [P1X(i,j), P1Y(i,j)], [corX(i,j) corY(i,j)],
[a1X(i,j), a1Y(i,j)], [b1X(i,j), b1Y(i,j)], jaXY, jbXY, temp,...
%                     [mark1X(end), mark1Y(end)], [mark2X(end), mark2Y(end)],
[F1X(i,j), F1Y(i,j)], tick);

                % find forces & moment at new position
                Fa1 = force2(ka, deltaa(i,j), la1, thetaG(i,j)+thetaa(i,j),
0, 0, 0);
                Fb1 = force2(kb, deltab(i,j), lb1, thetaG(i,j)+thetab(i,j),
0, 0, 0);
                F1X(i,j) = dot(Fa1+Fb1,ihat);, F1Y(i,j) = dot(Fa1+Fb1,jhat);
                Ma1 = moment1(thetaG(i,j), 0, 0, 0, T1a1(1:3,4), Fa1, khat);
                Mb1 = moment1(thetaG(i,j), 0, 0, 0, T1b1(1:3,4), Fb1, khat);
                M1Z(i,j) = Ma1 + Mb1;

                fmw = [F1X(i,j) F1Y(i,j) 0 0 0 M1Z(i,j)];
                FX(k,counter+1) = F1X(i,j);, FY(k,counter+1) = F1Y(i,j);,
MZ(counter+1) = M1Z(i,j);
                F(k,counter+1) = sqrt(F1X(i,j)^2 + F1Y(i,j)^2);

                if (sqrt(FX(k,counter+1)^2 + FY(k,counter+1)^2) <= ftarget) &
(fminmzd(j,k) == 0)
                    fminmzd(j,k) = counter;
                end
                if (counter == iterations) & (fminmzd(j,k) == 0)
                    fminmzd(j,k) = counter;
                end

                magf(k,1) = sqrt(F1X(i,j)^2 + F1Y(i,j)^2);
                work = (-0.5)*ka*deltaa(i,j)^2 + (-0.5)*kb*deltab(i,j)^2;
                u = 0.5*ka*deltaa(i,j)^2 + 0.5*kb*deltab(i,j)^2;

                % use analytical method to calculate stiffness at new
position
                if (z == 2) | (z == 3)
                    PXYstiff = [P1X(i,j) P1Y(i,j)];
                    [Kxx1(i,j), Kxy1(i,j), Kyx1(i,j), Kyy1(i,j)] = stiff(z,
ka, lar, jaXY, axy, PXYstiff, thetaG(i,j), kb, lbr, jbXY, bxy);
                    anadiagK2 = [Kxx1(i,j) 0; 0 Kyy1(i,j)];
                    anafullK2 = [Kxx1(i,j) Kxy1(i,j); Kyx1(i,j) Kyy1(i,j)];
                end

                % use numerical method to find stiffness
%                 if (z == 6) | (z == 8)
                if ((z == 6) | (z == 8)) & (cycle == 1)
                    diffload = [fmw(1)-f_temp(1) fmw(2)-f_temp(2) fmw(6)-
f_temp(6)];
                    diffdisp = [P1X(i,j)-PXY(1) P1Y(i,j)-PXY(2) phi];
                    [Kxx, Kyy, Kzz] = stiff(z, diffload, diffdisp, fmw,
numdiagK3);
                    % 2x2 matrix
```

```
                    numdiagK2 = [Kxx 0; 0 Kyy];
                    % 3x3 matrix
                    numdiagK3 = [Kxx 0 0; 0 Kyy 0; 0 0 Kzz];
                end

%                 % perturb bar to find stiffness
%                 % use one translation (in either X or Y) to find 3 of 4
stiffness terms OR
%                 % use one translation (in either X or Y) and one applied
perturbation to find full stiffness matrix OR
%                 % apply two perturbations to find full stiffness matrix
%                 if (z == 10) | (z == 11)
                if ((z == 10) | (z == 11)) & (cycle > 1)
                    [Kxxp(i,j), Kxyp(i,j), Kyxp(i,j), Kyyp(i,j)] = stiff(z,
3, fmw, f_temp, dXY, pertK);
                    pertK = [Kxxp(i,j) Kxyp(i,j); Kyxp(i,j) Kyyp(i,j)];
%                   pertK = [Kxxp(i,j), Kxyp(i,j) Kzxp(i,j); Kyxp(i,j),
Kyyp(i,j) Kyzp(i,j); Kzxp(i,j) Kzyp(i,j) Kzzp(i,j)];
                end

%                 if (z == 10) | (z == 11)
%%                    [Kxxp(i,j), Kxyp(i,j), Kyxp(i,j), Kyyp(i,j)] =
stiff(z, 2, thetaG(i,j)+thetaa(i,j)+phi, la1, thetaG(i,j)+thetab(i,j)+phi,
lb1,...
%%                          lar, lbr, ka, kb, [F1X(i,j) F1Y(i,j)],
thetaG(i,j)+phi, T1a1(1:3,4), T1b1(1:3,4), M1Z(i,j), [P1X(i,j) P1Y(i,j)],...
%%                          thetaCOR(i,j)+phi, corXY, thetaa(i,j), axy,
thetaja(i,j), jaXY, thetab(i,j), bxy, thetajb(i,j), jbXY);
%                     [Kxxp(i,j), Kxyp(i,j), Kyxp(i,j), Kyyp(i,j)] = stiff(z,
4, thetaG(i,j)+thetaa(i,j)+phi, la1, thetaG(i,j)+thetab(i,j)+phi, lb1,...
%                         lar, lbr, ka, kb, [F1X(i,j) F1Y(i,j)],
thetaG(i,j)+phi, T1a1(1:3,4), T1b1(1:3,4), M1Z(i,j), [P1X(i,j) P1Y(i,j)],...
%                         thetaCOR(i,j)+phi, corXY, thetaa(i,j), axy,
thetaja(i,j), jaXY, thetab(i,j), bxy, thetajb(i,j), jbXY,...
%                         fmw, f_temp, dXY, pertK);
%                       pertK = [Kxxp(i,j), Kxyp(i,j); Kyxp(i,j), Kyyp(i,j)];
% %                     pertK = [Kxxp(i,j), Kxyp(i,j) Kzxp(i,j); Kyxp(i,j),
Kyyp(i,j) Kyzp(i,j); Kzxp(i,j) Kzyp(i,j) Kzzp(i,j)];
%                 end

                % store current force and position
                f_temp = fmw;
                PXY = [P1X(i,j) P1Y(i,j)];

                % housekeeping variables
                KXX(k,counter+1) = pertK(1,1);, KXY(k,counter+1) =
pertK(1,2);
                KYX(k,counter+1) = pertK(2,1);, KYY(k,counter+1) =
pertK(2,2);
%                 invK = pinv([numfullK2(1,1) numfullK2(1,2); numfullK2(2,1)
numfullK2(2,2)]);
%                 invKXX(k,counter+1) = invK(1,1);, invKXY(k,counter+1) =
invK(1,2);
%                 invKYX(k,counter+1) = invK(2,1);, invKYY(k,counter+1) =
invK(2,2);
```

```
                if (sqrt(FX(k,counter+1)^2 + FY(k,counter+1)^2) <= ftarget),
break, end
            end

            loadctlerror(j,k) = sqrt(P1X(i,j)^2+P1Y(i,j)^2);
            avgiter(j,z) = sum(fminmzd(j,:))/k;
            Utotal(j,k) = u;
            peakF(j,z) = max(max(F));
            maxF(j,z) = max(max(F));
            minF(j,z) = min(min(F(k,counter+1)));
%             avgiter(i,j) = sum(fminmzd(j,:))/k;

            % COR CALCULATION
=======================================================

%             if (round(k*rad2deg(phi)/5) == k*rad2deg(phi)/5)
%             if (round(k*rad2deg(phi)/5) == k*rad2deg(phi)/5) & (cycle ~= 1)
            if (round(k*rad2deg(phi)/5) == k*rad2deg(phi)/5) & (cycle > 1) &
(z == 11)
%                 % find the true COR using Spiegelman and Woo
%                 for n = 1:1
%                 [tcorX(n), tcorY(n)] = spieg(a0X, a0Y, a1X(i,j), a1Y(i,j),
b0X, b0Y, b1X(i,j), b1Y(i,j), fmw(1), fmw(2));
%                 end
%                 tcorX = sum(tcorX)/n;, tcorY = sum(tcorY)/n;
%                 tcorXspieg(j,k) = tcorX;, tcorYspieg(j,k) = tcorY;
%                 errorspieg(j,k) = sqrt(tcorXspieg(j,k)^2 +
tcorYspieg(j,k)^2);
%                 Xsignspieg(j,k) = isequal(sign(-corX(i,j)),sign(-
tcorXspieg(j,k)));
%                 Ysignspieg(j,k) = isequal(sign(-corY(i,j)),sign(-
tcorYspieg(j,k)));
%
%                 % find the true COR using Crisco et al.
%                 for n = 1:1
%                 [tcorX(n), tcorY(n)] = crisco(a0X, a0Y, b0X, b0Y, a1X(i,j),
a1Y(i,j), b1X(i,j), b1Y(i,j), fmw(1), fmw(2));
%                 end
%                 tcorX = sum(tcorX)/n;, tcorY = sum(tcorY)/n;
%                 tcorXcrisco(j,k) = tcorX;, tcorYcrisco(j,k) = tcorY;
%                 errorcrisco(j,k) = sqrt(tcorXcrisco(j,k)^2 +
tcorYcrisco(j,k)^2);
%                 Xsigncrisco(j,k) = isequal(sign(-corX(i,j)),sign(-
tcorXcrisco(j,k)));
%                 Ysigncrisco(j,k) = isequal(sign(-corY(i,j)),sign(-
tcorYcrisco(j,k)));
%
                % find the true COR using Challis
                for n = 1:1
                [tcorX(n), tcorY(n)] = challis(axy, a0X, a0Y, a1X(i,j),
a1Y(i,j), bxy, b0X, b0Y, b1X(i,j), b1Y(i,j), fmw(1), fmw(2));
                end
                tcorX = sum(tcorX)/n;, tcorY = sum(tcorY)/n;
                tcorXchallis(j,k) = tcorX;, tcorYchallis(j,k) = tcorY;
                errorchallis(j,k) = sqrt(tcorXchallis(j,k)^2 +
tcorYchallis(j,k)^2);
```

```
                Xsignchallis(j,k) = isequal(sign(-corX(i,j)),sign(-
tcorXchallis(j,k)));
                Ysignchallis(j,k) = isequal(sign(-corY(i,j)),sign(-
tcorYchallis(j,k)));
            end

            tick = 0;

%             if k ~= kk
%                 delete(fgraph)
%                 fgraph = axes('Parent', fh, 'Position', [.1 .6 .8 .35],
'YLim', [-50 50], 'XLim', [0 iterations]);
%                 forceufs = line('XData', 0, 'YData', 0, 'Color', 'k',
'Marker', '.', 'markersize', 8, 'erasemode', 'none');
%             end

            % COR update option 1: calculate COR every 5 degrees & store to
be replayed in next cycle
            % COR update option 2: calculate & update COR every 5 degrees
%             if (round(k*rad2deg(phi)/5) == k*rad2deg(phi)/5)
%             if (round(k*rad2deg(phi)/5) == k*rad2deg(phi)/5) & (cycle ~= 1)
            if (round(k*rad2deg(phi)/5) == k*rad2deg(phi)/5) & (cycle ~= 1) &
(z == 11)
                cXY(:,index) = corupdate(corXY(1), corXY(2),
tcorXchallis(j,k), tcorYchallis(j,k), 'y', cor_lim);
                index = index + 1;
            end
        end

        % find the average magnitude of the peak force for each COR
        peakw(i,j) = work;
        peaku(i,j) = u;
        poteng(j,z) = u;
    end
end

% % PLOTS FOR TEST 3 USING PHYSICAL MODEL
% figure
% subplot(2,1,1), plot((1:length(fminmzd(j,:)))*rad2deg(phi),fminmzd(j,:),'.-
b'), ylabel('iterations'),...
%     title(['Rotated about (', num2str(corX(i,j)), ',', num2str(corY(i,j)),
') in ', num2str(phi*180/pi), ' deg. increments'])
% subplot(2,1,2), plot((1:length(Utotal(j,:)))*rad2deg(phi),Utotal(j,:),'.-
b'), ylabel('potential energy (N-mm)'), xlabel('\Phi (degrees)')

% % % PLOTS FOR TEST 4A & 4B & 4C
% figure, hold on
%  plot((1:length(errorspieg(j,:)))*rad2deg(phi),errorspieg(j,:),'.-b');
%  plot((1:length(errorcrisco(j,:)))*rad2deg(phi),errorcrisco(j,:),'.-r');
%  plot((1:length(errorchallis(j,:)))*rad2deg(phi),errorchallis(j,:),'.-m');
% % % PLOTS FOR TEST 4D
% % plot((5:5:30)*rad2deg(phi), errorspieg(j,5:5:end), '.-b')
% % plot((5:5:30)*rad2deg(phi), errorcrisco(j,5:5:end), '.-r')
% % plot((5:5:30)*rad2deg(phi), errorchallis(j,5:5:end), '.-m')
% xlabel('\Phi (degrees)'), ylabel('error (mm)'), title(['Rotated about (',
num2str(corX(i,j)), ',', num2str(corY(i,j)), ') in ',...
%         num2str(phi*180/pi), ' deg. increments'])
```

```
% legend('spieg', 'crisco', 'challis')

% % PLOTS FOR TESTS 5A & 5B
% if cycle == 1
%  figure
%     subplot(3,1,1), hold on, plot((1:kk)*rad2deg(phi), F(:,1), '.-b'),
ylabel('peak force (N)'), title(['Rotated about (', num2str(corX(i,j)), ',',
num2str(corY(i,j)), ') in ',...
%             num2str(phi*180/pi), ' deg. increments'])
%     subplot(3,1,2), hold on,
plot((1:length(fminmzd(j,:)))*rad2deg(phi),fminmzd(j,:),'.-b'),
ylabel('iterations')
%     subplot(3,1,3), hold on,
plot((1:length(Utotal(j,:)))*rad2deg(phi),Utotal(j,:),'.-b'),
ylabel('potential energy (N-mm)'), xlabel('\Phi (degrees)')
% elseif cycle == 2
%     subplot(3,1,1), plot((1:kk)*rad2deg(phi), F(:,1), '.-r'), ylabel('peak
force (N)'), title(['Rotated about (', num2str(corX(i,j)), ',',
num2str(corY(i,j)), ') in ',...
%             num2str(phi*180/pi), ' deg. increments'])
% %     legend('cycle 1', 'cycle 2')
%     legend('no COR update', 'COR update')
%     subplot(3,1,2),
plot((1:length(fminmzd(j,:)))*rad2deg(phi),fminmzd(j,:),'.-r'),
ylabel('iterations')
% %     legend('cycle 1', 'cycle 2')
%     legend('no COR update', 'COR update')
%     subplot(3,1,3),
plot((1:length(Utotal(j,:)))*rad2deg(phi),Utotal(j,:),'.-r'),
ylabel('potential energy (N-mm)'), xlabel('\Phi (degrees)')
% %     legend('cycle 1', 'cycle 2')
%     legend('no COR update', 'COR update')
% end

% % PLOTS FOR TEST 7 USING GENERAL MODEL
% if (cycle == 1) & (z == 6)
%     figure('position', [149   359   771   575])
%     subplot(4,1,1), hold on, plot((1:kk)*rad2deg(phi), F(:,1), '.-b'),
ylabel('peak force (N)'), title(['Rotated about (', num2str(corX(i,j)), ',',
num2str(corY(i,j)), ') in ',...
%             num2str(phi*180/pi), ' deg. increments'])
%     subplot(4,1,2), hold on, plot((1:length(fminmzd(j,:)))*rad2deg(phi),
fminmzd(j,:), '.-b'), ylabel('iterations')
%     subplot(4,1,3), hold on,
plot((1:length(loadctlerror(j,:)))*rad2deg(phi), loadctlerror(j,:), '.-b'),
ylabel('error (mm)')
%     subplot(4,1,4), hold on, plot((1:length(Utotal(j,:)))*rad2deg(phi),
Utotal(j,:), '.-b'), ylabel('potential energy (N-mm)'), xlabel('\Phi
(degrees)')
% elseif (cycle == 2) & (z == 11)
% %     figure
%     subplot(4,1,1), hold on, plot((1:kk)*rad2deg(phi), F(:,1), '.-r'),
ylabel('peak force (N)')
%     legend('old', 'new')
%     subplot(4,1,2), hold on, plot((1:length(fminmzd(j,:)))*rad2deg(phi),
fminmzd(j,:), '.-r'), ylabel('iterations')
%     legend('old', 'new')
```

```
%     subplot(4,1,3), hold on,
plot((1:length(loadctlerror(j,:)))*rad2deg(phi), loadctlerror(j,:), '.-r'),
ylabel('error (mm)')
%     legend('old', 'new')
%     subplot(4,1,4), hold on, plot((1:length(Utotal(j,:)))*rad2deg(phi),
Utotal(j,:), '.-r'), ylabel('potential energy (N-mm)')
%     legend('old', 'new')
% end

%% PLOTS FOR TEST 7 USING PHYSICAL MODEL
%if (cycle == 1) & (z == 6)
%    figure('position', [149   359   771   575])
%    subplot(3,1,1), hold on, plot((1:kk)*rad2deg(phi), F(:,1), '.-b'),
ylabel('peak force (N)'), title(['Rotated about (', num2str(corX(i,j)), ',',
num2str(corY(i,j)), ') in ',...
%            num2str(phi*180/pi), ' deg. increments'])
%    subplot(3,1,2), hold on, plot((1:length(fminmzd(j,:)))*rad2deg(phi),
fminmzd(j,:), '.-b'), ylabel('iterations')
%    subplot(3,1,3), hold on, plot((1:length(Utotal(j,:)))*rad2deg(phi),
Utotal(j,:), '.-b'), ylabel('potential energy (N-mm)'), xlabel('\Phi
(degrees)')
%elseif (cycle == 2) & (z == 11)
%%     figure
%    subplot(3,1,1), hold on, plot((1:kk)*rad2deg(phi), F(:,1), '.-r'),
ylabel('peak force (N)')
%    legend('old', 'new')
%    subplot(3,1,2), hold on, plot((1:length(fminmzd(j,:)))*rad2deg(phi),
fminmzd(j,:), '.-r'), ylabel('iterations')
%    legend('old', 'new')
%    subplot(3,1,3), hold on, plot((1:length(Utotal(j,:)))*rad2deg(phi),
Utotal(j,:), '.-r'), ylabel('potential energy (N-mm)')
%    legend('old', 'new')
%end

if z == 1
plot((1:length(loadctlerror(j,:)))*rad2deg(phi),loadctlerror(j,:),'.-b');
elseif z == 2
figure
% plot((1:length(loadctlerror(j,:)))*rad2deg(phi),loadctlerror(j,:),'.-r');
subplot(3,1,1), hold on,
plot((1:length(loadctlerror(j,:)))*rad2deg(phi),loadctlerror(j,:),'.-r'),
ylabel('error (mm)'),...
    title(['Rotated about (', num2str(corX(i,j)), ',', num2str(corY(i,j)), ')
in ', num2str(phi*180/pi), ' deg. increments'])
subplot(3,1,2), hold on,
plot((1:length(fminmzd(j,:)))*rad2deg(phi),fminmzd(j,:),'.-r'),
ylabel('iterations')
subplot(3,1,3), hold on,
plot((1:length(Utotal(j,:)))*rad2deg(phi),Utotal(j,:),'.-r'),
ylabel('potential energy (N-mm)'), xlabel('\Phi (degrees)')
elseif z == 3
% plot((1:length(loadctlerror(j,:)))*rad2deg(phi),loadctlerror(j,:),'.-m');
% subplot(3,1,1),
plot((1:length(loadctlerror(j,:)))*rad2deg(phi),loadctlerror(j,:),'.-m'),
legend('diag 2x2 (a)', 'full 2x2 (a)')
% subplot(3,1,2), plot((1:length(fminmzd(j,:)))*rad2deg(phi),fminmzd(j,:),'.-
m'), legend('diag 2x2 (a)', 'full 2x2 (a)')
```

```
% subplot(3,1,3), plot((1:length(Utotal(j,:)))*rad2deg(phi),Utotal(j,:),'.-
m'), legend('diag 2x2 (a)', 'full 2x2 (a)')
elseif z == 4
plot((1:length(loadctlerror(j,:)))*rad2deg(phi),loadctlerror(j,:),'.-g');
elseif z == 5
plot((1:length(loadctlerror(j,:)))*rad2deg(phi),loadctlerror(j,:),'.-r');
elseif z == 6
% % PLOTS FOR TEST 6 USING GENERAL MODEL
% % plot((1:length(loadctlerror(j,:)))*rad2deg(phi),loadctlerror(j,:),'.-r');
% figure
% subplot(3,1,1), hold on,
plot((1:length(loadctlerror(j,:)))*rad2deg(phi),loadctlerror(j,:),'.-r'),
ylabel('error (mm)'),...
%     title(['Rotated about (', num2str(corX(i,j)), ',', num2str(corY(i,j)),
') in ', num2str(phi*180/pi), ' deg. increments'])
% subplot(3,1,2), hold on,
plot((1:length(fminmzd(j,:)))*rad2deg(phi),fminmzd(j,:),'.-r'),
ylabel('iterations')
% subplot(3,1,3), hold on,
plot((1:length(Utotal(j,:)))*rad2deg(phi),Utotal(j,:),'.-r'),
ylabel('potential energy (N-mm)'), xlabel('\Phi (degrees)')

% % PLOTS FOR TEST 6 USING PHYSICAL MODEL
% figure
% subplot(2,1,1), hold on,
plot((1:length(fminmzd(j,:)))*rad2deg(phi),fminmzd(j,:),'.-r'),
ylabel('iterations'),...
%     title(['Rotated about (', num2str(corX(i,j)), ',', num2str(corY(i,j)),
') in ', num2str(phi*180/pi), ' deg. increments'])
% subplot(2,1,2), hold on,
plot((1:length(Utotal(j,:)))*rad2deg(phi),Utotal(j,:),'.-r'),
ylabel('potential energy (N-mm)'), xlabel('\Phi (degrees)')
elseif z == 7
plot((1:length(loadctlerror(j,:)))*rad2deg(phi),loadctlerror(j,:),'.-r');
elseif z == 8
plot((1:length(loadctlerror(j,:)))*rad2deg(phi),loadctlerror(j,:),'.-',
'markerfacecolor', [0 .75 0]);
elseif z == 9
plot((1:length(loadctlerror(j,:)))*rad2deg(phi),loadctlerror(j,:),'.-m');
elseif z == 10
figure, hold on
% plot((1:length(loadctlerror(j,:)))*rad2deg(phi),loadctlerror(j,:),'.-r');
elseif z == 11
% % PLOTS FOR TEST 6 USING GENERAL MODEL
% plot((1:length(loadctlerror(j,:)))*rad2deg(phi),loadctlerror(j,:),'.-m');
% subplot(3,1,1), hold on,
plot((1:length(loadctlerror(j,:)))*rad2deg(phi),loadctlerror(j,:),'.-m'),
ylabel('error (mm)'),...
%     title(['Rotated about (', num2str(corX(i,j)), ',', num2str(corY(i,j)),
') in ', num2str(phi*180/pi), ' deg. increments'])
% legend('diag 2x2 (n)', 'full 2x2 (p4)')
% subplot(3,1,2), hold on,
plot((1:length(fminmzd(j,:)))*rad2deg(phi),fminmzd(j,:),'.-m'),
ylabel('iterations')
% legend('diag 2x2 (n)', 'full 2x2 (p4)')
```

```
% subplot(3,1,3), hold on,
plot((1:length(Utotal(j,:)))*rad2deg(phi),Utotal(j,:),'.-m'),
ylabel('potential energy (N-mm)'), xlabel('\Phi (degrees)')
% legend('diag 2x2 (n)', 'full 2x2 (p4)')

% % PLOTS FOR TEST 6 USING PHYSICAL MODEL
% subplot(2,1,1), plot((1:length(fminmzd(j,:)))*rad2deg(phi),fminmzd(j,:),'.-
m'), ylabel('iterations'),...
%     title(['Rotated about (', num2str(corX(i,j)), ',', num2str(corY(i,j)),
') in ', num2str(phi*180/pi), ' deg. increments'])
% legend('diag 2x2 (n)', 'full 2x2 (p4)')
% subplot(2,1,2), plot((1:length(Utotal(j,:)))*rad2deg(phi),Utotal(j,:),'.-
m'), ylabel('potential energy (N-mm)'), xlabel('\Phi (degrees)')
% legend('diag 2x2 (n)', 'full 2x2 (p4)')
elseif z == 12
plot((1:length(loadctlerror(j,:)))*rad2deg(phi),loadctlerror(j,:),'.-r');
elseif z == 13
plot((1:length(loadctlerror(j,:)))*rad2deg(phi),loadctlerror(j,:),'.-m');
end

% % PLOTS FOR TEST 6
% figure, hold on
% for m = 1:kk
%     phb = plot([m m], [F(m,1) F(m,fminmzd(j,m)+1)], '-ob');
%     phe = plot([m m], [F(m,fminmzd(j,m)+1) F(m,fminmzd(j,m)+1)], '*r');
% end
% title(['Rotated about (', num2str(corX(i,j)), ',', num2str(corY(i,j)), ')
in ', num2str(phi*180/pi), ' deg. increments'])
% xlabel('\Phi (degrees)'), ylabel('resultant force (N)')
% legend_handles = [phb; phe];
% legend(legend_handles, 'beginning force', 'ending force');

end
end
end

% figure
% subplot(3,1,1), plot(1:length(FX), FX), ylabel('FX (N)');
% subplot(3,1,2), plot(1:length(FY), FY), ylabel('FY (N)');
% subplot(3,1,3), plot(1:length(MZ), MZ), ylabel('MZ (N-m)');,
xlabel('iterations');
%
% figure
% subplot(2,1,1), plot(1:length(KXX), KXX), ylabel('KXX (N/mm)')
% subplot(2,1,2), plot(1:length(KYY), KYY), ylabel('KYY (N/mm)');,
xlabel('iterations');

% [xx, yy]=meshgrid(-60:10:60,60:-10:-60);
% draw(xx, yy, F1X, F1Y, M1Z, F, peaku, P1X, P1Y, Kxxp, Kxyp, Kyyp, PHI, phi,
1);
% draw(xx, yy, F1X, F1Y, M1Z, F, peaku, P1X, P1Y, Kxx1, Kxy1, Kyy1, PHI, phi,
2);
```

refpointtrans.m is a function called by hand14a.m. It uses the rigid body transformations developed in sections 4.2.1 - 4.2.7 to track the movement of the center of the bar.

```
function [P1X, P1Y, TG0, TG1, corXY] = refpointtrans(thetaG, PXY, thetaCOR,
corXY, phi, dxy, cs)

% define xformation from X,Y to x0,y0
TG0 = trans(thetaG, PXY(1), PXY(2), 0);

% define xformation from X,Y to xTCS0,yTCS0
TGTCS0 = trans(thetaCOR, corXY(1), corXY(2), 0);

% calculate xformation from xTCS0,yTCS0 to x0,y0
TTCS00 = inv(TGTCS0)*TG0;

% define xformation from xTCS0,yTCS0 to xTCS1,yTCS1
% here dxy is defined in the TCS0 c.s.
% if the translation (dxy) is defined in the global c.s.,
% it must first be transformed to the TCS0 c.s.
if cs == 'g'
    dxy = (TGTCS0(1:3,1:3))'*[dxy 0]';
    dxy(3) = [];
end
TTCS0TCS1 = trans(phi, dxy(1), dxy(2), 0);

% calculate xformation from X,Y to xTCS1,yTCS1
TGTCS1 = TGTCS0*TTCS0TCS1;
corXY = [TGTCS1(1,4) TGTCS1(2,4)];

% define xformation from xTCS1,yTCS1 to x1,y1
TTCS11 = TTCS00;

% calculate xformation from X,Y to x1,y1
TG1 = TGTCS1*TTCS11;
P1X = TG1(1,4);
P1Y = TG1(2,4);
```

nodaltrans.m is a function called by hand14a.m. It uses the rigid body transformations developed in sections 4.2.8 - 4.2.14 to track the movement of the ends of the bar.

```
function [i1X, i1Y, l0, l1, T0i0, T1i1] = nodaltrans(thetai, ixy, thetaj,
jXY, TG0, TG1);

% define xformation from x0,y0 to xi0,yi0
T0i0 = trans(thetai, ixy(1), ixy(2), 0);

% calculate xformation from X,Y to xi0,yi0
TGi0 = TG0*T0i0;
```

```
% define xformation from x1,y1 to xi1,yi1
T1i1 = T0i0;

% calculate xformation from X,Y to xi1,yi1
TGi1 = TG1*T1i1;
i1X = TGi1(1,4);
i1Y = TGi1(2,4);

% define xformation from X,Y to xj,yj
TGj = trans(thetaj, jXY(1), jXY(2), 0);

% calculate xformation from xi0,yi0 to xj,yj
Ti0j = inv(TGi0)*TGj;
% calculate length of spring 1 at time t0 (mm)
l0 = Ti0j(1:3,4);
% L0 is the vector pointing from node i to node j in local coordinates
% later, this is the force acting on the bar.  if we want the force
% that the spring exerts, we would use -L0
% calculate xformation from xi1,yi1, to xj,yj
Ti1j = inv(TGi1)*TGj;
% calculate length of spring 1 at time t1 (mm)
l1 = Ti1j(1:3,4);
```

force2.m is a short function called by hand14a.m that calculates the force in coordinates.

```
function F = force2(k, delta, l, theta, X, Y, Z)

f = k*delta*l/(sqrt(l'*l));
F = trans(theta, X, Y, Z)*[f; 1];
F(4,:) = [];
```

moment1.m is a short function called by hand14a.m that calculates the moment in global coordinates.

```
function M = moment1(theta, X, Y, Z, L, F, unit);

R = trans(theta, X, Y, Z)*[L; 1];
R(4) = [];
M = dot(cross(R,F),unit)/1000;
```

stiff.m is a function called by hand14a.m that calculates the global stiffness terms using the method of choice (defined by the input variable “flag”).

```
function varargout = stiff(flag, varargin)

switch flag
case {2, 3, 4, 5}
    % calculate analytical stiffness matrix
```

```
    ka = varargin{1};
    lar = varargin{2};
    jaXY = varargin{3};
    axy = varargin{4};
    PXYstiff = varargin{5};
    angle = varargin{6};
    kb = varargin{7};
    lbr = varargin{8};
    jbXY = varargin{9};
    bxy = varargin{10};
    [kxxa, kxya, kxza, kyxa, kyya, kyza, kzxa, kzya, kzza] = anastiff(ka,
lar, jaXY, axy, PXYstiff, angle);
    [kxxb, kxyb, kxzb, kyxb, kyyb, kyzb, kzxb, kzyb, kzzb] = anastiff(kb,
lbr, jbXY, bxy, PXYstiff, angle);
    Kxx = kxxa + kxxb;, Kxy = kxya + kxyb;, Kxz = kxza + kxzb;
    Kyx = kyxa + kyxb;, Kyy = kyya + kyyb;, Kyz = kyza + kyzb;
    Kzx = kzxa + kzxb;, Kzy = kzya + kzyb;, Kzz = kzza + kzzb;
    if (flag == 2) | (flag == 3)
        varargout = {Kxx, Kxy, Kyx, Kyy};
    elseif (flag == 4) | (flag == 5)
        varargout = {Kxx, Kxy, Kxz, Kyx, Kyy, Kyz, Kzx, Kzy, Kzz};
    end

case {6, 7, 8, 9}
    % calculate numerical stiffness matrix
    diffload = varargin{1};
    diffdisp = varargin{2};
    fmw = varargin{3};
    if (diffload ~= [0 0 0]) & (diffdisp ~= [0 0 0]) & (fmw(1)~=0 | fmw(2)~=0
| fmw(6)~=0)
        Kxx = diffload(1)/diffdisp(1);
        Kyy = diffload(2)/diffdisp(2);
        Kzz = diffload(3)/diffdisp(3);
    else
        K = varargin{4};
        Kxx = K(1,1);
        Kyy = K(2,2);
        Kzz = K(3,3);
    end
    varargout = {Kxx, Kyy, Kzz};

case {10, 11}
    method = varargin{1};

    if (method == 1) | (method == 2) | (method == 4)
        angle = varargin{2};
        la1 = varargin{3};
        La1 = [cos(angle) -sin(angle) 0; sin(angle) cos(angle) 0; 0 0 1] *
la1;

        angle = varargin{4};
        lb1 = varargin{5};
        Lb1 = [cos(angle) -sin(angle) 0; sin(angle) cos(angle) 0; 0 0 1] *
lb1;
    end

    if (method == 1) | (method == 2)
```

```
        lar = varargin{6};
        lbr = varargin{7};
        ka = varargin{8};
        kb = varargin{9};
        FXY = varargin{10};
        thetaGphi = varargin{11};
        T1a1 = varargin{12};
        T1b1 = varargin{13};
        MZ = varargin{14};
        PXY = varargin{15};
        thetaCORphi = varargin{16};
        corXY = varargin{17};
        thetaa = varargin{18};
        axy = varargin{19};
        thetaja = varargin{20};
        jaXY = varargin{21};
        thetab = varargin{22};
        bxy = varargin{23};
        thetajb = varargin{24};
        jbXY = varargin{25};

        if method == 1
            % calculate full 2x2 perturbed matrix w/ method #1 (two global
pert., one parallel to X, other parallel to Y)
            [Kxx, Kxy, Kyx, Kyy, Kzx, Kzy, Kxz, Kyz, Kzz] = pertstiff(La1,
Lb1, lar, lbr, ka, kb,...
                FXY, thetaGphi, T1a1, T1b1, MZ, PXY, thetaCORphi, corXY,
thetaa, axy, thetaja, jaXY, thetab, bxy, thetajb, jbXY);
        elseif method == 2
            % calculate full 2x2 perturbed matrix w/ method #2 (two global
pert., one in XY-plane, other orthogonal)
            [Kxx, Kxy, Kyx, Kyy, Kzx, Kzy, Kxz, Kyz, Kzz] = pertstiff2(La1,
Lb1, lar, lbr, ka, kb,...
                FXY, thetaGphi, T1a1, T1b1, MZ, PXY, thetaCORphi, corXY,
thetaa, axy, thetaja, jaXY, thetab, bxy, thetajb, jbXY);
        end
    end

    if method == 3
       % calculate full 2x2 perturbed matrix w/ method #3 (one global
translation, parallel to either X or Y)
       fmw = varargin{2};
       f_temp = varargin{3};
       dXY = varargin{4};
       K = varargin{5};

       dFXY = [fmw(1)-f_temp(1) fmw(2)-f_temp(2)];
       if (dXY(1) == 0) & (dFXY ~= [0 0]) & (dXY(2) ~= 0) & (fmw(1) ~= 0 |
fmw(2)~= 0)
            Kxy = dFXY(1)/dXY(2);
            Kyy = dFXY(2)/dXY(2);
            Kyx = Kxy;
            Kxx = K(1,1);
        elseif (dXY(2) == 0) & (dFXY ~= [0 0]) & (dXY(1) ~= 0) & (fmw(1) ~= 0
| fmw(2)~= 0)
            Kxx = dFXY(1)/dXY(1);
            Kyx = dFXY(2)/dXY(1);
```

```
            Kxy = Kyx;
            Kyy = K(2,2);
        else
            Kxx = K(1,1);
            Kxy = K(1,2);
            Kyx = K(2,1);
            Kyy = K(2,2);
        end
    elseif method == 4
        % calculate full 2x2 perturbed matrix w/ method #4 (one global
translation, one global pert. orthogonal to translation)
        lar = varargin{6};
        lbr = varargin{7};
        ka = varargin{8};
        kb = varargin{9};
        FXY = varargin{10};
        thetaGphi = varargin{11};
        T1a1 = varargin{12};
        T1b1 = varargin{13};
        MZ = varargin{14};
        PXY = varargin{15};
        thetaCORphi = varargin{16};
        corXY = varargin{17};
        thetaa = varargin{18};
        axy = varargin{19};
        thetaja = varargin{20};
        jaXY = varargin{21};
        thetab = varargin{22};
        bxy = varargin{23};
        thetajb = varargin{24};
        jbXY = varargin{25};

        fmw = varargin{26};
        f_temp = varargin{27};
        dXY = varargin{28};
        K = varargin{29};

       dFXY = [fmw(1)-f_temp(1) fmw(2)-f_temp(2)];
       if (dXY(1) == 0) & (dFXY ~= [0 0]) & (dXY(2) ~= 0) & (fmw(1) ~= 0 |
fmw(2)~= 0)
            Kxy = dFXY(1)/dXY(2);
            Kyy = dFXY(2)/dXY(2);
            Kyx = Kxy;
            Kxx = pertstiff3(La1, Lb1, lar, lbr, ka, kb, FXY, thetaGphi,
T1a1, T1b1, MZ, PXY, thetaCORphi, corXY,...
                thetaa, axy, thetaja, jaXY, thetab, bxy, thetajb, jbXY,
dXY(2), 'even');
        elseif (dXY(2) == 0) & (dFXY ~= [0 0]) & (dXY(1) ~= 0) & (fmw(1) ~= 0
| fmw(2)~= 0)
            Kxx = dFXY(1)/dXY(1);
            Kyx = dFXY(2)/dXY(1);
            Kxy = Kyx;
            Kyy = pertstiff3(La1, Lb1, lar, lbr, ka, kb, FXY, thetaGphi,
T1a1, T1b1, MZ, PXY, thetaCORphi, corXY,...
                thetaa, axy, thetaja, jaXY, thetab, bxy, thetajb, jbXY,
dXY(1), 'odd');
        else
```

```
            Kxx = K(1,1);
            Kxy = K(1,2);
            Kyx = K(2,1);
            Kyy = K(2,2);
        end
    end

    varargout = {Kxx, Kxy, Kyx, Kyy};

end
```

anastiff.m is a function called by stiff.m to calculate the global stiffness terms analytically.

```
function [kxx, kxy, kxz, kyx, kyy, kyz, kzx, kzy, kzz] = anastiff(k, lr, jXY,
ixy, PXY, angle)

c1 = jXY(1) - ixy(1)*cos(angle) + ixy(2)*sin(angle);
c2 = jXY(2) - ixy(2)*cos(angle) - ixy(1)*sin(angle);
c4 = jXY(1) - PXY(1);
c5 = jXY(2) - PXY(2);

kxx = k*(-1 + (lr*(c2-PXY(2))^2)/(((c1-PXY(1))^2+(c2-PXY(2))^2)^(3/2)));
kxy = -k*lr*(c1-PXY(1))*(c2-PXY(2))/(((c1-PXY(1))^2+(c2-PXY(2))^2)^(3/2));
kxz = k*(ixy(2)*cos(angle) + ixy(1)*sin(angle) + lr*(-
c5+ixy(2)*cos(angle)+ixy(1)*sin(angle))*...
    (-ixy(1)^2-ixy(2)^2+(c4*ixy(1)+c5*ixy(2))*cos(angle)+(c5*ixy(1)-
c4*ixy(2))*sin(angle))/...
    (((-c5+ixy(2)*cos(angle)+ixy(1)*sin(angle))^2 + (c4-
ixy(1)*cos(angle)+ixy(2)*sin(angle))^2)^(3/2)));

kyx = kxy;
kyy = k*(-1 + (lr*(c1-PXY(1))^2)/(((c1-PXY(1))^2+(c2-PXY(2))^2)^(3/2)));
kyz = k*(-ixy(1)*cos(angle) + ixy(2)*sin(angle) + lr*(c4-
ixy(1)*cos(angle)+ixy(2)*sin(angle))*...
    (-ixy(1)^2-ixy(2)^2+(c4*ixy(1)+c5*ixy(2))*cos(angle)+(c5*ixy(1)-
c4*ixy(2))*sin(angle))/...
    (((-c5+ixy(2)*cos(angle)+ixy(1)*sin(angle))^2 + (c4-
ixy(1)*cos(angle)+ixy(2)*sin(angle))^2)^(3/2)));

kzx = kxz;
kzy = kyz;
kzz = -k*lr*(((-c5*ixy(1)+c4*ixy(2))*cos(angle) +
(c4*ixy(1)+c5*ixy(2))*sin(angle))^2)/...
    (((-c5+ixy(2)*cos(angle)+ixy(1)*sin(angle))^2 + (c4-
ixy(1)*cos(angle)+ixy(2)*sin(angle))^2)^(3/2)) + ...
    k*((-c4*ixy(1)-c5*ixy(2))*cos(angle) + (-
c5*ixy(1)+c4*ixy(2))*sin(angle))*...
    (1 - lr/sqrt((-c5+ixy(2)*cos(angle)+ixy(1)*sin(angle))^2 + (c4-
ixy(1)*cos(angle)+ixy(2)*sin(angle))^2));
kzz = kzz/1000;
```

pertstiff.m is a function called by stiff.m that calculates the global stiffness terms using proposed method #1.

```
function [Kxx, Kxy, Kyx, Kyy, Kzx, Kzy, Kxz, Kyz, Kzz] = pertstiff(La1, Lb1,
lar, lbr, ka, kb, FXY,...
    thetaGphi, T1a1, T1b1, MZ, PXY, thetaCORphi, corXY, thetaa, axy, thetaja,
jaXY, thetab, bxy, thetajb, jbXY)

ihat = [1 0 0]';
jhat = [0 1 0]';
khat = [0 0 1]';

% perturb the bar in each direction to find full stiffness matrix
pert = 0.5; % mm

% Only consider the perturbation along the global X axis
% La is a vector pointing from node a to node ja in X,Y coordinates
La = La1 - [pert 0 0]';
delta = sqrt(La'*La) - lar;
Fa = ka*delta*La/sqrt(La'*La);
Ma = moment1(thetaGphi, 0, 0, 0, T1a1, Fa, khat);
% Lb is a vector pointing from node b to node jb in X,Y coordinates
Lb = Lb1 - [pert 0 0]';
delta = sqrt(Lb'*Lb) - lbr;
Fb = kb*delta*Lb/sqrt(Lb'*Lb);
Mb = moment1(thetaGphi, 0, 0, 0, T1b1, Fb, khat);
% find Kxx and Kyx
FXY(2,1) = dot(Fa,ihat) + dot(Fb,ihat);, FXY(2,2) = dot(Fa,jhat) +
dot(Fb,jhat);, MZ(2,1) = Ma + Mb;
dFX = FXY(2,1) - FXY(1,1);, dFY = FXY(2,2) - FXY(1,2);, dMZ = MZ(2,1) -
MZ(1,1);
Kxx = dFX/pert; % N/mm
Kyx = dFY/pert; % N/mm
Kzx = dMZ*1000/pert; % N-mm/mm

% Now consider the perturbation along the global Y axis
% La is a vector pointing from node a to node ja in X,Y coordinates
La = La1 - [0 pert 0]';
delta = sqrt(La'*La) - lar;
Fa = ka*delta*La/sqrt(La'*La);
Ma = moment1(thetaGphi, 0, 0, 0, T1a1, Fa, khat);
% Lb is a vector pointing from node b to node jb in X,Y coordinates
Lb = Lb1 - [0 pert 0]';
delta = sqrt(Lb'*Lb) - lbr;
Fb = kb*delta*Lb/sqrt(Lb'*Lb);
Mb = moment1(thetaGphi, 0, 0, 0, T1b1, Fb, khat);
% find Kxy and Kyy
FXY(2,1) = dot(Fa,ihat) + dot(Fb,ihat);, FXY(2,2) = dot(Fa,jhat) +
dot(Fb,jhat);, MZ(2,1) = Ma + Mb;
dFX = FXY(2,1) - FXY(1,1);, dFY = FXY(2,2) - FXY(1,2);, dMZ = MZ(2,1) -
MZ(1,1);
Kxy = dFX/pert; % N/mm
Kyy = dFY/pert; % N/mm
Kzy = dMZ*1000/pert; % N-mm/mm
```

```
% Consider a perturbation about the global Z axis
%pert = deg2rad(0.1); % radians
pert = 0.1*pi/180;
[P1X, P1Y, TG0, TG1] = refpointtrans(thetaGphi, PXY, thetaCORphi, corXY,
pert, [0 0], 'g');
[a1X, a1Y, la0, la1, T0a0, T1a1] = nodaltrans(thetaa, axy, thetaja, jaXY,
TG0, TG1);
[b1X, b1Y, lb0, lb1, T0b0, T1b1] = nodaltrans(thetab, bxy, thetajb, jbXY,
TG0, TG1);
deltaa = sqrt(la1'*la1) - lar;
deltab = sqrt(lb1'*lb1) - lbr;
% find forces & moment at new position
Fa = force2(ka, deltaa, la1, thetaGphi+thetaa+pert, 0, 0, 0);
Fb = force2(kb, deltab, lb1, thetaGphi+thetab+pert, 0, 0, 0);
Ma = moment1(thetaGphi+pert, 0, 0, 0, T1a1(1:3,4), Fa, khat);
Mb = moment1(thetaGphi+pert, 0, 0, 0, T1b1(1:3,4), Fb, khat);
FXY(2,1) = dot(Fa,ihat) + dot(Fb,ihat);, FXY(2,2) = dot(Fa,jhat) +
dot(Fb,jhat);, MZ(2,1) = Ma + Mb;
dFX = FXY(2,1) - FXY(1,1);, dFY = FXY(2,2) - FXY(1,2);, dMZ = MZ(2,1) -
MZ(1,1);
Kxz = dFX/pert; % N/rad
Kyz = dFY/pert; % N/rad
Kzz = dMZ/pert; % N-m/rad
```

pertstiff2.m is a function called by stiff.m that calculates the global stiffness terms using proposed method #2.

```
function [Kxx, Kxy, Kyx, Kyy, Kzx, Kzy, Kxz, Kyz, Kzz] = pertstiff2(La1, Lb1,
lar, lbr, ka, kb, FXY,...
    thetaGphi, T1a1, T1b1, MZ, PXY, thetaCORphi, corXY, thetaa, axy, thetaja,
jaXY, thetab, bxy, thetajb, jbXY)

ihat = [1 0 0]';
jhat = [0 1 0]';
khat = [0 0 1]';

% perturb the bar in each direction to find full stiffness matrix
% perturbations are orthogonal linear combinations of X and Y
pert = 0.5; % mm

% Consider a perturbation in the global XY plane
% La is a vector pointing from node a to node ja in X,Y coordinates
La = La1 - [pert pert 0]';
delta = sqrt(La'*La) - lar;
Fa = ka*delta*La/sqrt(La'*La);
Ma = moment1(thetaGphi, 0, 0, 0, T1a1, Fa, khat);
% Lb is a vector pointing from node b to node jb in X,Y coordinates
Lb = Lb1 - [pert pert 0]';
delta = sqrt(Lb'*Lb) - lbr;
Fb = kb*delta*Lb/sqrt(Lb'*Lb);
Mb = moment1(thetaGphi, 0, 0, 0, T1b1, Fb, khat);
% find dF and dM
```

```
FXY(2,1) = dot(Fa,ihat) + dot(Fb,ihat);, FXY(2,2) = dot(Fa,jhat) +
dot(Fb,jhat);, MZ(2,1) = Ma + Mb;
dFX1 = FXY(2,1) - FXY(1,1);, dFY1 = FXY(2,2) - FXY(1,2);, dMZ1 = MZ(2,1) -
MZ(1,1);

% Consider another perturbation in the global XY axis, perpendicular to the
first pert.
% La is a vector pointing from node a to node ja in X,Y coordinates
La = La1 - [-pert pert 0]';
delta = sqrt(La'*La) - lar;
Fa = ka*delta*La/sqrt(La'*La);
Ma = moment1(thetaGphi, 0, 0, 0, T1a1, Fa, khat);
% Lb is a vector pointing from node b to node jb in X,Y coordinates
Lb = Lb1 - [-pert pert 0]';
delta = sqrt(Lb'*Lb) - lbr;
Fb = kb*delta*Lb/sqrt(Lb'*Lb);
Mb = moment1(thetaGphi, 0, 0, 0, T1b1, Fb, khat);
% find dF and dM
FXY(2,1) = dot(Fa,ihat) + dot(Fb,ihat);, FXY(2,2) = dot(Fa,jhat) +
dot(Fb,jhat);, MZ(2,1) = Ma + Mb;
dFX2 = FXY(2,1) - FXY(1,1);, dFY2 = FXY(2,2) - FXY(1,2);, dMZ2 = MZ(2,1) -
MZ(1,1);

dX1 = pert;,  dY1 = pert;
dX2 = -pert;, dY2 = pert;

Kxx = -(dFX2*dY1 - dFX1*dY2) / (-dX2*dY1 + dX1*dY2);  % N/mm
Kxy = -(-dFX2*dX1 + dFX1*dX2) / (-dX2*dY1 + dX1*dY2); % N/mm
Kyx = -(dFY2*dY1 - dFY1*dY2) / (-dX2*dY1 + dX1*dY2);  % N/mm
Kyy = -(-dFY2*dX1 + dFY1*dX2) / (-dX2*dY1 + dX1*dY2); % N/mm

Kzx = 1;
Kzy = 1;

% Consider a perturbation about the global Z axis
pert = deg2rad(0.1); % radians
[P1X, P1Y, TG0, TG1] = refpointtrans(thetaGphi, PXY, thetaCORphi, corXY,
pert, [0 0], 'g');
[a1X, a1Y, la0, la1, T0a0, T1a1] = nodaltrans(thetaa, axy, thetaja, jaXY,
TG0, TG1);
[b1X, b1Y, lb0, lb1, T0b0, T1b1] = nodaltrans(thetab, bxy, thetajb, jbXY,
TG0, TG1);
deltaa = sqrt(la1'*la1) - lar;
deltab = sqrt(lb1'*lb1) - lbr;
% find forces & moment at new position
Fa = force2(ka, deltaa, la1, thetaGphi+thetaa+pert, 0, 0, 0);
Fb = force2(kb, deltab, lb1, thetaGphi+thetab+pert, 0, 0, 0);
Ma = moment1(thetaGphi+pert, 0, 0, 0, T1a1(1:3,4), Fa, khat);
Mb = moment1(thetaGphi+pert, 0, 0, 0, T1b1(1:3,4), Fb, khat);
FXY(2,1) = dot(Fa,ihat) + dot(Fb,ihat);, FXY(2,2) = dot(Fa,jhat) +
dot(Fb,jhat);, MZ(2,1) = Ma + Mb;
dFX = FXY(2,1) - FXY(1,1);, dFY = FXY(2,2) - FXY(1,2);, dMZ = MZ(2,1) -
MZ(1,1);
Kxz = dFX/pert; % N/rad
Kyz = dFY/pert; % N/rad
Kzz = dMZ/pert; % N-m/rad
```

pertstiff3.m is a function called by stiff.m that calculates the global stiffness terms using proposed method #3.

```
function [varargout] = pertstiff3(La1, Lb1, lar, lbr, ka, kb, FXY,...
    thetaGphi, T1a1, T1b1, MZ, PXY, thetaCORphi, corXY, thetaa, axy, thetaja,
jaXY, thetab, bxy, thetajb, jbXY, pert, flag)

ihat = [1 0 0]';
jhat = [0 1 0]';
khat = [0 0 1]';

% use translation AND applied perturbation to find full stiffness matrix at a
given position
% perturb the bar in each direction to find full stiffness matrix
pert = 0.5; % mm

% if counter is odd, translation in Y is set to zero, only solve for Kxx,
Kxy, Kyx
% if counter is even, translation in X is set to zero, only solve for Kxy,
Kyx, Kyy
switch flag
case 'even'
    % Only consider the perturbation along the global X axis
    % La is a vector pointing from node a to node ja in X,Y coordinates
    La = La1 - [pert 0 0]';
    delta = sqrt(La'*La) - lar;
    Fa = ka*delta*La/sqrt(La'*La);
    Ma = moment1(thetaGphi, 0, 0, 0, T1a1, Fa, khat);
    % Lb is a vector pointing from node b to node jb in X,Y coordinates
    Lb = Lb1 - [pert 0 0]';
    delta = sqrt(Lb'*Lb) - lbr;
    Fb = kb*delta*Lb/sqrt(Lb'*Lb);
    Mb = moment1(thetaGphi, 0, 0, 0, T1b1, Fb, khat);
    % find Kxx and Kyx
    FXY(2,1) = dot(Fa,ihat) + dot(Fb,ihat);, FXY(2,2) = dot(Fa,jhat) +
dot(Fb,jhat);, MZ(2,1) = Ma + Mb;
    dFX = FXY(2,1) - FXY(1,1);, dFY = FXY(2,2) - FXY(1,2);, dMZ = MZ(2,1) -
MZ(1,1);
    Kxx = dFX/pert; % N/mm
    Kyx = dFY/pert; % N/mm
    Kxy = Kyx;
    Kzx = dMZ*1000/pert; % N-mm/mm
    varargout = {Kxx};
case 'odd'
    % Only consider the perturbation along the global Y axis
    % La is a vector pointing from node a to node ja in X,Y coordinates
    La = La1 - [0 pert 0]';
    delta = sqrt(La'*La) - lar;
    Fa = ka*delta*La/sqrt(La'*La);
    Ma = moment1(thetaGphi, 0, 0, 0, T1a1, Fa, khat);
    % Lb is a vector pointing from node b to node jb in X,Y coordinates
    Lb = Lb1 - [0 pert 0]';
```

```
    delta = sqrt(Lb'*Lb) - lbr;
    Fb = kb*delta*Lb/sqrt(Lb'*Lb);
    Mb = moment1(thetaGphi, 0, 0, 0, T1b1, Fb, khat);
    % find Kxy and Kyy
    FXY(2,1) = dot(Fa,ihat) + dot(Fb,ihat);, FXY(2,2) = dot(Fa,jhat) +
dot(Fb,jhat);, MZ(2,1) = Ma + Mb;
    dFX = FXY(2,1) - FXY(1,1);, dFY = FXY(2,2) - FXY(1,2);, dMZ = MZ(2,1) -
MZ(1,1);
    Kxy = dFX/pert; % N/mm
    Kyy = dFY/pert; % N/mm
    Kyx = Kxy;
    Kzy = dMZ*1000/pert; % N-mm/mm
    varargout = {Kyy};
end

% Consider a perturbation about the global Z axis
pert = deg2rad(0.1); % radians
[P1X, P1Y, TG0, TG1] = refpointtrans(thetaGphi, PXY, thetaCORphi, corXY,
pert, [0 0], 'g');
[a1X, a1Y, la0, la1, T0a0, T1a1] = nodaltrans(thetaa, axy, thetaja, jaXY,
TG0, TG1);
[b1X, b1Y, lb0, lb1, T0b0, T1b1] = nodaltrans(thetab, bxy, thetajb, jbXY,
TG0, TG1);
deltaa = sqrt(la1'*la1) - lar;
deltab = sqrt(lb1'*lb1) - lbr;
% find forces & moment at new position
Fa = force2(ka, deltaa, la1, thetaGphi+thetaa+pert, 0, 0, 0);
Fb = force2(kb, deltab, lb1, thetaGphi+thetab+pert, 0, 0, 0);
Ma = moment1(thetaGphi+pert, 0, 0, 0, T1a1(1:3,4), Fa, khat);
Mb = moment1(thetaGphi+pert, 0, 0, 0, T1b1(1:3,4), Fb, khat);
FXY(2,1) = dot(Fa,ihat) + dot(Fb,ihat);, FXY(2,2) = dot(Fa,jhat) +
dot(Fb,jhat);, MZ(2,1) = Ma + Mb;
dFX = FXY(2,1) - FXY(1,1);, dFY = FXY(2,2) - FXY(1,2);, dMZ = MZ(2,1) -
MZ(1,1);
Kxz = dFX/pert; % N/rad
Kyz = dFY/pert; % N/rad
Kzz = dMZ/pert; % N-m/rad
```

fmin.m is a function called by hand14a.m that calculates the translation required to minimize force using the method of choice (defined by the input variable “flag”).

```
function dXY = fmin(flag, varargin)

switch flag
case 1
    % let bar follow force without using constant user-defined stiffness
    FX = varargin{1};
    FY = varargin{2};
    const_stiff = varargin{3};
    dXY = [FX/const_stiff FY/const_stiff]';
case {2, 4}
    % calculate global displacement to force minimized position using
analytical diagonal K matrix (either 2x2 or 3x3)
```

```
    K = varargin{1};
    fmw = varargin{2};
    limit = varargin{3};
    if size(K,1) == 2, fmw(3) = [];, end
    dXY = -K\fmw;
    if limit == 'y'
        t_lim = varargin{4};
        if abs(dXY(1)) > t_lim, dXY(1) = sign(dXY(1))*t_lim; end
        if abs(dXY(2)) > t_lim, dXY(2) = sign(dXY(2))*t_lim; end
    end
    if size(dXY,1) == 3, dXY(3) = [];, end
case {3, 5}
    % calculate global displacement to force minimized position using
analytical full K matrix (either 2x2 or 3x3)
    K = varargin{1};
    fmw = varargin{2};
    limit = varargin{3};
    if size(K,1) == 2, fmw(3) = [];, end
    dXY = -K\fmw;
    if limit == 'y'
        t_lim = varargin{4};
        if abs(dXY(1)) > t_lim, dXY(1) = sign(dXY(1))*t_lim; end
        if abs(dXY(2)) > t_lim, dXY(2) = sign(dXY(2))*t_lim; end
    end
    if size(dXY,1) == 3, dXY(3) = [];, end
case {6, 8}
    % calculate global displacement to force minimized position using
numerical diagonal K matrix method (either 2x2 or 3x3)
    K = varargin{1};
    fmw = varargin{2};
    limit = varargin{3};
    if size(K,1) == 2, fmw(3) = [];, end
    dXY = -K\fmw;
    if limit == 'y'
        t_lim = varargin{4};
        if abs(dXY(1)) > t_lim, dXY(1) = sign(dXY(1))*t_lim; end
        if abs(dXY(2)) > t_lim, dXY(2) = sign(dXY(2))*t_lim; end
    end
    if size(dXY,1) == 3, dXY(3) = [];, end
case {7, 9}
    % calculate global displacement to force minimized position using
numerical full K matrix method (either 2x2 or 3x3)
    K = varargin{1};
    fmw = varargin{2};
    limit = varargin{3};
    if size(K,1) == 2, fmw(3) = [];, end
    dXY = -pinv(K)*fmw;
    if limit == 'y'
        t_lim = varargin{4};
        if abs(dXY(1)) > t_lim, dXY(1) = sign(dXY(1))*t_lim; end
        if abs(dXY(2)) > t_lim, dXY(2) = sign(dXY(2))*t_lim; end
    end
    if size(dXY,1) == 3, dXY(3) = [];, end
case 10
    % calculate global displacement to force minimized position using
perturbations (diagonal 2x2)
    pertk = varargin{1};
```

```
    fmw = varargin{2};
    limit = varargin{3};
    if size(pertk,1) == 2, fmw(3) = [];, end
    dXY = -[pertk(1,1) 0; 0 pertk(2,2)]\fmw;
    if limit == 'y'
        t_lim = varargin{4};
        if abs(dXY(1)) > t_lim, dXY(1) = sign(dXY(1))*t_lim; end
        if abs(dXY(2)) > t_lim, dXY(2) = sign(dXY(2))*t_lim; end
    end
    if size(dXY,1) == 3, dXY(3) = [];, end
case {11, 13}
    % calculate global displacement to force minimized position using
perturbations (full 2x2 or full 3x3)
    pertk = varargin{1};
    fmw = varargin{2};
    limit = varargin{3};
    if size(pertk,1) == 2, fmw(3) = [];, end
    dXY = -pinv(pertk)*fmw;
    if limit == 'y'
        t_lim = varargin{4};
        if abs(dXY(1)) > t_lim, dXY(1) = sign(dXY(1))*t_lim; end
        if abs(dXY(2)) > t_lim, dXY(2) = sign(dXY(2))*t_lim; end
    end
    if size(dXY,1) == 3, dXY(3) = [];, end
case 12
    % calculate global displacement to force minimized position using
perturbations (diagonal 3x3)
    pertk = varargin{1};
    fmw = varargin{2};
    limit = varargin{3};
    if size(pertk,1) == 2, fmw(3) = [];, end
    dXY = -[pertk(1,1) 0 0; 0 pertk(2,2) 0; 0 0 pertk(3,3)]\fmw;
    if limit == 'y'
        t_lim = varargin{4};
        if abs(dXY(1)) > t_lim, dXY(1) = sign(dXY(1))*t_lim; end
        if abs(dXY(2)) > t_lim, dXY(2) = sign(dXY(2))*t_lim; end
    end
    if size(dXY,1) == 3, dXY(3) = [];, end
end
```

spieg.m is a function called by hand14a.m that calculates the preferred COR using the method described by Spiegelman and Woo.

```
function [corX, corY] = spieg(mark1X, mark1Y, mark1Xp, mark1Yp, mark2X,
mark2Y, mark2Xp, mark2Yp, fx, fy)

% find the true COR using Spiegelman and Woo

% (X1,Y1) & (X2,Y2) are the initial and final global coordinates of marker 1
% (X3,Y3) & (X4,Y4) are the initial and final global coordinates of marker 2
% node a = first marker, node b = second marker

% % noise is normally distributed with mean = 0 mm and std = 0.5 mm
```

```
% X1 = mark1X+normrnd(0,0.5);, Y1 = mark1Y+normrnd(0,0.5);
% X2 = mark1Xp+normrnd(0,0.5);, Y2 = mark1Yp+normrnd(0,0.5);
% X3 = mark2X+normrnd(0,0.5);, Y3 = mark2Y+normrnd(0,0.5);
% X4 = mark2Xp+normrnd(0,0.5);, Y4 = mark2Yp+normrnd(0,0.5);

% noise is assumed to be due to weight on end-effector
% calculate how much noise should be added
pctpay = fx/(6*9.81)*100;, xnoise = 0.0058*pctpay - 0.28;
pctpay = fy/(6*9.81)*100;, ynoise = 0.0058*pctpay - 0.28;
X1 = mark1X+xnoise;, Y1 = mark1Y+ynoise;
X2 = mark1Xp+xnoise;, Y2 = mark1Yp+ynoise;
X3 = mark2X+xnoise;, Y3 = mark2Y+ynoise;
X4 = mark2Xp+xnoise;, Y4 = mark2Yp+ynoise;

% % no noise added
% X1 = mark1X;, Y1 = mark1Y;
% X2 = mark1Xp;, Y2 = mark1Yp;
% X3 = mark2X;, Y3 = mark2Y;
% X4 = mark2Xp;, Y4 = mark2Yp;

S = X1-X3;, Sp = X2-X4;
T = Y1-Y3;, Tp = Y2-Y4;
cosphi = (Sp*S + Tp*T)/(S^2 + T^2);
sinphi = (Sp*T - Tp*S)/(S^2 + T^2);
U = (Y1+Y2)/2 + sinphi*(X1-X2)/(2*(1-cosphi));
V = (X1+X2)/2 - sinphi*(Y1-Y2)/(2*(1-cosphi));
corX = X1 + (Y2-U)/sinphi - cosphi*(Y1-U)/sinphi;
corY = Y1 - (X2-V)/sinphi + cosphi*(X1-V)/sinphi;
```

crisco.m is a function called by hand14a.m that calculates the preferred COR using the method described by Crisco et al.

```
function [corX, corY] = crisco(mark1X, mark1Y, mark2X, mark2Y, mark1Xp,
mark1Yp, mark2Xp, mark2Yp, fx, fy)

% find the true COR using Crisco et al.

% (x1,y1) & (x2,y2) are the initial & final global coordinates of marker 1
% (x3,y3) & (x4,y4) are the initial & final global coordinates of marker 2

% % noise is normally distributed with mean = 0 mm and std = 0.5 mm
% x1 = mark1X+normrnd(0,0.5);,  y1 = mark1Y+normrnd(0,0.5);,  A  = [x1; y1];
% x2 = mark1Xp+normrnd(0,0.5);, y2 = mark1Yp+normrnd(0,0.5);, Ap = [x2; y2];
% x3 = mark2X+normrnd(0,0.5);,  y3 = mark2Y+normrnd(0,0.5);,  B  = [x3; y3];
% x4 = mark2Xp+normrnd(0,0.5);, y4 = mark2Yp+normrnd(0,0.5);, Bp = [x4; y4];

% noise is assumed to be due to weight on end-effector
% calculate how much noise should be added
pctpay = fx/(6*9.81)*100;, xnoise = 0.0058*pctpay - 0.28;
pctpay = fy/(6*9.81)*100;, ynoise = 0.0058*pctpay - 0.28;
x1 = mark1X+xnoise;,  y1 = mark1Y+ynoise;,  A  = [x1; y1];
```

```
x2 = mark1Xp+xnoise;, y2 = mark1Yp+ynoise;, Ap = [x2; y2];
x3 = mark2X+xnoise;,  y3 = mark2Y+ynoise;,  B  = [x3; y3];
x4 = mark2Xp+xnoise;, y4 = mark2Yp+ynoise;, Bp = [x4; y4];

% % no noise added
% x1 = mark1X;,  y1 = mark1Y;,  A  = [x1; y1];
% x2 = mark1Xp;, y2 = mark1Yp;, Ap = [x2; y2];
% x3 = mark2X;,  y3 = mark2Y;,  B  = [x3; y3];
% x4 = mark2Xp;, y4 = mark2Yp;, Bp = [x4; y4];

u = A-B;
up = Ap-Bp;
cosphi = dot(u,up)/(sqrt(u'*u)*sqrt(up'*up));
sinphi = sqrt(1-(cosphi)^2);
cp = cross([u;0],[up;0]);
if sign(cp(3)) > 0
    sinphi = sinphi;
elseif sign(cp(3)) < 0
    sinphi = -sinphi;
end
corX = (1/2)*(x1+x2) + (y1-y2)*sinphi/(2*(1-cosphi));
corY = (1/2)*(y1+y2) - (x1-x2)*sinphi/(2*(1-cosphi));
```

challis.m is a function called by hand14a.m that calculates the preferred COR using the method described by Challis.

```
function [corX, corY, xnoise, ynoise] = challis(axy, a0X, a0Y, a1X, a1Y, bxy,
b0X, b0Y, b1X, b1Y, fx, fy);

% find the true COR using Challis

% x(t)i is the position of point i on the rigid body measured in the rigid
body ref. frame
% y(t)i is the position of point i on the rigid body measured in the inertial
ref. frame
% x(t)i and y(t)i are vectors, not single points
% node a: i = 1 (initial) & 3 (final)
% node b: i = 2 (initial) & 4 (final)

% % noise is normally distributed with mean = 0 mm and std = 0.5 mm
% x1 = [axy(1); axy(2)] + normrnd(0,0.5,2,1);, y1 = [a0X; a0Y] +
normrnd(0,0.5,2,1);
% x2 = [bxy(1); bxy(2)] + normrnd(0,0.5,2,1);, y2 = [b0X; b0Y] +
normrnd(0,0.5,2,1);
% x3 = [axy(1); axy(2)] + normrnd(0,0.5,2,1);, y3 = [a1X; a1Y] +
normrnd(0,0.5,2,1);
% x4 = [bxy(1); bxy(2)] + normrnd(0,0.5,2,1);, y4 = [b1X; b1Y] +
normrnd(0,0.5,2,1);

% noise is assumed to be due to weight on end-effector
% calculate how much noise should be added
pctpay = fx/(6*9.81)*100;, xnoise = 0.0058*pctpay - 0.28;
pctpay = fy/(6*9.81)*100;, ynoise = 0.0058*pctpay - 0.28;
```

```
x1 = [axy(1); axy(2)] + [xnoise; ynoise];, y1 = [a0X; a0Y] + [xnoise;
ynoise];
x2 = [bxy(1); bxy(2)] + [xnoise; ynoise];, y2 = [b0X; b0Y] + [xnoise;
ynoise];
x3 = [axy(1); axy(2)] + [xnoise; ynoise];, y3 = [a1X; a1Y] + [xnoise;
ynoise];
x4 = [bxy(1); bxy(2)] + [xnoise; ynoise];, y4 = [b1X; b1Y] + [xnoise;
ynoise];

% % no noise added
% x1 = [axy(1); axy(2)];, y1 = [a0X; a0Y];
% x2 = [bxy(1); bxy(2)];, y2 = [b0X; b0Y];
% x3 = [axy(1); axy(2)];, y3 = [a1X; a1Y];
% x4 = [bxy(1); bxy(2)];, y4 = [b1X; b1Y];

xbar = (x1+x2)/2;, ybar = (y1+y2)/2;
x1p = x1-xbar;, y1p = y1-ybar;
x2p = x2-xbar;, y2p = y2-ybar;
P = y1p(1)*x1p(2)-y1p(2)*x1p(1) + y2p(1)*x2p(2)-y2p(2)*x2p(1);
Q = y1p(1)*x1p(1)+y1p(2)*x1p(2) + y2p(1)*x2p(1)+y2p(2)*x2p(2);
phi = -atan(P/Q);, phi0 = phi;
v0 = (y1+y2)/2 - [cos(phi) -sin(phi); sin(phi) cos(phi)]*(x1+x2)/2;

xbar = (x3+x4)/2;, ybar = (y3+y4)/2;
x3p = x3-xbar;, y3p = y3-ybar;
x4p = x4-xbar;, y4p = y4-ybar;
P = y3p(1)*x3p(2)-y3p(2)*x3p(1) + y4p(1)*x4p(2)-y4p(2)*x4p(1);
Q = y3p(1)*x3p(1)+y3p(2)*x3p(2) + y4p(1)*x4p(1)+y4p(2)*x4p(2);
phi = -atan(P/Q);, phi1 = phi;
v1 = (y3+y4)/2 - [cos(phi) -sin(phi); sin(phi) cos(phi)]*(x3+x4)/2;

dv = v1-v0;
p = (v0+v1)/2;
phi = phi1 - phi0;
FCR = p + 1/(2*tan(phi/2))*[cos(pi/2) -sin(pi/2); sin(pi/2) cos(pi/2)]*dv;
corX = FCR(1);
corY = FCR(2);
```

corupdate.m is a short function called by hand14a.m that updates the user-defined COR to the calculated preferred COR.

```
function corXY = corupdate(corX, corY, corXtemp, corYtemp, limit, cor_lim)

dcorX = corXtemp-corX;
dcorY = corYtemp-corY;
if limit == 'y'
    if abs(dcorX) > cor_lim, dcorX = sign(dcorX)*cor_lim;, end
    if abs(dcorY) > cor_lim, dcorY = sign(dcorY)*cor_lim;, end
end
corX = corX+dcorX;, corY = corY+dcorY;, corXY = [corX corY]';
```

draw.m is a function called by hand14a.m that makes 3D plots or contour plots, depending on the input variable "flag".

```
function draw(xx, yy, FX, FY, MZ, F, u, PX, PY, K1, K2, K3, PHI, phi, flag);

switch flag
   case 1
      fh = figure;
      surfc(xx, yy, FX);
      xlabel('X COR location (mm)');
      ylabel('Y COR location (mm)');
      zlabel('FX (N)');
      title(['Total rotation = ',num2str(PHI),' deg., ', num2str(phi*180/pi),
' deg. increments']);
      view(-58.50, 46);

      fh = figure;
      surfc(xx, yy, FY);
      xlabel('X COR location (mm)');
      ylabel('Y COR location (mm)');
      zlabel('FY (N)');
      title(['Total rotation = ',num2str(PHI),' deg., ', num2str(phi*180/pi),
' deg. increments']);
      view(-58.50, 46);

      fh = figure;
      surfc(xx, yy, MZ);
      xlabel('X COR location (mm)');
      ylabel('Y COR location (mm)');
      zlabel('MZ (N-m)');
      title(['Total rotation = ',num2str(PHI),' deg., ', num2str(phi*180/pi),
' deg. increments']);
      view(-58.50, 46);

      fh = figure;
      surfc(xx, yy, F);
      xlabel('X COR location (mm)');
      ylabel('Y COR location (mm)');
      zlabel('Resultant force magnitude (N)');
      title(['Total rotation = ',num2str(PHI),' deg., ', num2str(phi*180/pi),
' deg. increments']);
      view(-58.50, 46);

        fh = figure;
        surfc(xx, yy, u);
        xlabel('X COR location (mm)');
      ylabel('Y COR location (mm)');
      zlabel('Potential energy (N-mm)');
      title(['Total rotation = ',num2str(PHI),' deg., ', num2str(phi*180/pi),
' deg. increments']);
      view(-58.50, 46);

      fh = figure;
      surfc(PX, PY, K1);
      xlabel('PX location (mm)');
```

```
      ylabel('PY location (mm)');
      zlabel('Kxx (N/mm)');
      title(['Total rotation = ',num2str(PHI),' deg., ', num2str(phi*180/pi),
' deg. increments']);
      view(-58.50, 46);

        fh = figure;
        surfc(PX, PY, K2);
        xlabel('PX location (mm)');
        ylabel('PY location (mm)');
        zlabel('Kxy (N/mm)');
        title(['Total rotation = ',num2str(PHI),' deg.,
',num2str(phi*180/pi),' deg. increments']);
        view(-58.50, 46);

      fh = figure;
      surfc(PX, PY, K3);
      xlabel('PX location (mm)');
      ylabel('PY location (mm)');
      zlabel('Kyy (N/mm)');
      title(['Total rotation = ',num2str(PHI),' deg., ', num2str(phi*180/pi),
' deg. increments']);
      view(-58.50, 46);

   case 2
      fh = figure;
      [C,h] = contour(xx, yy, FX);
      xline = line('xdata', [xx(1) xx(end)], 'ydata', [0 0], 'color', 'k');
      yline = line('xdata', [0 0], 'ydata', [yy(1) yy(end)], 'color', 'k');
      xlabel('X COR location (mm)');
      ylabel('Y COR location (mm)');
      title(['FX (N), Total rotation = ',num2str(PHI),' deg., ',
num2str(phi*180/pi), ' deg. increments']);
      clabel(C,h);

      fh = figure;
      [C,h] = contour(xx, yy, FY);
        xline = line('xdata', [xx(1) xx(end)], 'ydata', [0 0], 'color', 'k');
      yline = line('xdata', [0 0], 'ydata', [yy(1) yy(end)], 'color', 'k');
      xlabel('X COR location (mm)');
      ylabel('Y COR location (mm)');
      title(['FY (N), Total rotation = ',num2str(PHI),' deg., ',
num2str(phi*180/pi), ' deg. increments']);
      clabel(C,h);

      fh = figure;
      [C,h] = contour(xx, yy, MZ);
        xline = line('xdata', [xx(1) xx(end)], 'ydata', [0 0], 'color', 'k');
      yline = line('xdata', [0 0], 'ydata', [yy(1) yy(end)], 'color', 'k');
      xlabel('X COR location (mm)');
      ylabel('Y COR location (mm)');
      title(['MZ (N-m), Total rotation = ',num2str(PHI),' deg., ',
num2str(phi*180/pi), ' deg. increments']);
      clabel(C,h);

      fh = figure;
      [C,h] = contour(xx, yy, avgf);
```

```
        xline = line('xdata', [xx(1) xx(end)], 'ydata', [0 0], 'color', 'k');
      yline = line('xdata', [0 0], 'ydata', [yy(1) yy(end)], 'color', 'k');
      xlabel('X COR location (mm)');
      ylabel('Y COR location (mm)');
      title(['Resultant force magnitude (N), Total rotation = ', ...
                num2str(PHI),' deg., ', num2str(phi*180/pi), ' deg.
increments']);
      clabel(C,h);

        fh = figure;
        [C,h] = contour(xx, yy, u);
        xline = line('xdata', [xx(1) xx(end)], 'ydata', [0 0], 'color', 'k');
      yline = line('xdata', [0 0], 'ydata', [yy(1) yy(end)], 'color', 'k');
      xlabel('X COR location (mm)');
      ylabel('Y COR location (mm)');
      title(['Potential energy (N-mm), Total rotation = ',num2str(PHI),'
deg., ', num2str(phi*180/pi), ' deg. increments']);
      clabel(C,h);

      fh = figure;
      [C,h] = contour(PX, PY, Kxx);
        xline = line('xdata', [xx(1) xx(end)], 'ydata', [0 0], 'color', 'k');
      yline = line('xdata', [0 0], 'ydata', [yy(1) yy(end)], 'color', 'k');
      xlabel('PX location (mm)');
      ylabel('PY location (mm)');
      title(['Kxx (N/mm), Total rotation = ',num2str(PHI),' deg., ',
num2str(phi*180/pi), ' deg. increments']);
      clabel(C,h);

      fh = figure;
      [C,h] = contour(PX, PY, Kyy);
        xline = line('xdata', [xx(1) xx(end)], 'ydata', [0 0], 'color', 'k');
      yline = line('xdata', [0 0], 'ydata', [yy(1) yy(end)], 'color', 'k');
      xlabel('PX location (mm)');
      ylabel('PY location (mm)');
      title(['Kyy (N/mm), Total rotation = ',num2str(PHI),' deg., ',
num2str(phi*180/pi), ' deg. increments']);
      clabel(C,h);
end
```

draw2.m is a function called by hand14a.m that plots the current position of the bar and the resultant force acting at the center of the bar.

```
function draw2(handles, PXY, corXY, aXY, bXY, jaXY, jbXY, angle, mark1XY,
mark2XY, FXY, tick)

fh = handles(1);
fgraph = handles(2);
forceufs = handles(3);
ah = handles(4);
bar = handles(5);
springa = handles(6);
springb = handles(7);
```

```
xbar = [aXY(1) bXY(1)];
ybar = [aXY(2) bXY(2)];
set(bar, 'xdata', xbar, 'ydata', ybar);
set(springa, 'xdata', [aXY(1) jaXY(1)], 'ydata', [aXY(2) jaXY(2)]);
set(springb, 'xdata', [jbXY(1) bXY(1)], 'ydata', [jbXY(2) bXY(2)]);
drawnow
plot(corXY(1)*cos(2*angle)-corXY(2)*sin(2*angle),
corXY(1)*sin(2*angle)+corXY(2)*cos(2*angle), '.',...
    'markeredgecolor', [0 .75 0], 'markersize', 20);
plot(PXY(1), PXY(2), '.', 'markeredgecolor', [.827 .122 .592]);
plot(0, 0, '.g', mark1XY(1), mark1XY(2), '.m', mark2XY(1), mark2XY(2), '.m');
set(fh, 'currentaxes', fgraph)
hold on
plot(tick, sqrt(FXY(1)^2+FXY(2)^2), '.k', 'markersize', 8);
set(fh, 'currentaxes', ah)
for m = 1:100000, ; end
```

APPENDIX B

Matlab code for experimental tests

spine_display.m is a function called by the Matlab GUI developed to allow any user to control the experimental tests.

```
   function varargout = spine_display2(varargin)
% SPINE_DISPLAY2 Application M-file for spine_display2.fig
%    FIG = SPINE_DISPLAY2 launch spine_display2 GUI.
%    SPINE_DISPLAY2('callback_name', ...) invoke the named callback.

% Amy Loveless
% Last Modified by GUIDE v2.0 05-Jun-2003 14:06:37

if nargin == 0  % LAUNCH GUI

   fig = openfig(mfilename,'reuse');

   % Use system color scheme for figure:
   set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));

   % Generate a structure of handles to pass to callbacks, and store it.
   handles = guihandles(fig);
   guidata(fig, handles);

    assignin('base', 'guihandles', handles)

%     assignin('base', 'hok', handles.ok_push_button);
%     assignin('base', 'hbolt', handles.boltup_push_button);
%     assignin('base', 'hbefore', handles.fm_before_push_button);
%     assignin('base', 'hafter', handles.load_control_push_button);
%     assignin('base', 'hpath', handles.pathseek_push_button);
%     assignin('base', 'hval', handles.val_path_push_button);
%     assignin('base', 'hreplay', handles.replay_push_button);
%     assignin('base', 'hend', handles.end_push_button);

    global ok_flag
    ok_flag = 0;

   if nargout > 0
      varargout{1} = fig;
   end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

   try
      if (nargout)
```

```
            [varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
        else
            feval(varargin{:}); % FEVAL switchyard
        end
    catch
        disp(lasterr);
    end

end

%| ABOUT CALLBACKS:
%| GUIDE automatically appends subfunction prototypes to this file, and
%| sets objects' callback properties to call them through the FEVAL
%| switchyard above. This comment describes that mechanism.
%|
%| Each callback subfunction declaration has the following form:
%| <SUBFUNCTION_NAME>(H, EVENTDATA, HANDLES, VARARGIN)
%|
%| The subfunction name is composed using the object's Tag and the
%| callback type separated by '_', e.g. 'slider2_Callback',
%| 'figure1_CloseRequestFcn', 'axis1_ButtondownFcn'.
%|
%| H is the callback object's handle (obtained using GCBO).
%|
%| EVENTDATA is empty, but reserved for future use.
%|
%| HANDLES is a structure containing handles of components in GUI using
%| tags as fieldnames, e.g. handles.figure1, handles.slider2. This
%| structure is created at GUI startup using GUIHANDLES and stored in
%| the figure's application data using GUIDATA. A copy of the structure
%| is passed to each callback.  You can store additional information in
%| this structure at GUI startup, and you can change the structure
%| during callbacks.  Call guidata(h, handles) after changing your
%| copy to replace the stored original so that subsequent callbacks see
%| the updates. Type "help guihandles" and "help guidata" for more
%| information.
%|
%| VARARGIN contains any extra arguments you have passed to the
%| callback. Specify the extra arguments by editing the callback
%| property in the inspector. By default, GUIDE sets the property to:
%| <MFILENAME>('<SUBFUNCTION_NAME>', gcbo, [], guidata(gcbo))
%| Add any extra arguments after the last argument, before the final
%| closing parenthesis.

% --------------------------------------------------------------------
function varargout = default_push_button_Callback(h, eventdata, handles,
varargin)
set(handles.corx_edit,'String','0')
set(handles.cory_edit,'String','15')
set(handles.corz_edit,'String','110')
set(handles.corrx_edit,'String','0')
set(handles.corry_edit,'String','0')
set(handles.corrz_edit,'String','0')
set(handles.sup_vert_x,'String','0')
set(handles.sup_vert_y,'String','15')
set(handles.sup_vert_z,'String','110')
set(handles.sup_vert_rx,'String','0')
```

```
set(handles.sup_vert_ry,'String','0')
set(handles.sup_vert_rz,'String','0')
set(handles.start_edit,'String','0')
set(handles.inc_edit,'String','0.5')
set(handles.fxtarget_edit,'String','0.5')
set(handles.fytarget_edit,'String','0.5')
set(handles.fztarget_edit,'String','0.5')
set(handles.mxtarget_edit,'String','0.25')
set(handles.mytarget_edit,'String','0.25')
set(handles.mztarget_edit,'String','0.25')

set(handles.default_push_button,'Enable','off')

% --------------------------------------------------------------------
function varargout = ok_push_button_Callback(h, eventdata, handles, varargin)
corx = str2num(get(handles.corx_edit,'String'));
cory = str2num(get(handles.cory_edit,'String'));
corz = str2num(get(handles.corz_edit,'String'));
corrx = str2num(get(handles.corrx_edit,'String'));
corry = str2num(get(handles.corry_edit,'String'));
corrz = str2num(get(handles.corrz_edit,'String'));
supvertx = str2num(get(handles.sup_vert_x,'String'));
supverty = str2num(get(handles.sup_vert_y,'String'));
supvertz = str2num(get(handles.sup_vert_z,'String'));
supvertrx = str2num(get(handles.sup_vert_rx,'String'));
supvertry = str2num(get(handles.sup_vert_ry,'String'));
supvertrz = str2num(get(handles.sup_vert_rz,'String'));
start = str2num(get(handles.start_edit,'String'));
inc = str2num(get(handles.inc_edit,'String'));
fxtarget = str2num(get(handles.fxtarget_edit,'String'));
fytarget = str2num(get(handles.fytarget_edit,'String'));
fztarget = str2num(get(handles.fztarget_edit,'String'));
mxtarget = str2num(get(handles.mxtarget_edit,'String'));
mytarget = str2num(get(handles.mytarget_edit,'String'));
mztarget = str2num(get(handles.mztarget_edit,'String'));

assignin('base','x1',corx/1000)
assignin('base','y1',cory/1000)
assignin('base','z1',(corz+64)/1000)
if corrx == 0, assignin('base','rx1',0.0000001), else,
assignin('base','rx1',corrx), end
if corry == 0, assignin('base','ry1',0.0000001), else,
assignin('base','ry1',corry), end
if corrz == 0, assignin('base','rz1',0.0000001), else,
assignin('base','rz1',corrz), end
assignin('base','x2',supvertx/1000)
assignin('base','y2',supverty/1000)
assignin('base','z2',(supvertz+64)/1000)
if supvertrx == 0, assignin('base','rx2',0.0000001), else,
assignin('base','rx2',supvertrx), end
if supvertry == 0, assignin('base','ry2',0.0000001), else,
assignin('base','ry2',supvertry), end
if supvertrz == 0, assignin('base','rz2',0.0000001), else,
assignin('base','rz2',supvertrz), end
assignin('base','w_start',start)
assignin('base','w_ang',inc)
assignin('base','w_neg',-inc)
```

```
assignin('base','z_target',[fxtarget fytarget fztarget mxtarget mytarget
mztarget])
assignin('base','cuts',0)

global ok_flag
if ok_flag == 0
    % Initiate communication with the UFS
    a = matjr3pci('init_jr3',0,0,0,0,0);

    % Create, configure and open serial port object
    port1 = Serial('COM1');
    set(port1, 'BaudRate',19200, 'Terminator','CR/LF', 'Timeout', 900);
    fopen(port1);
    assignin('base','port1',port1)
end

set(handles.boltup_push_button,'Enable','on')
set(handles.fm_before_push_button,'Enable','on')
set(handles.load_control_push_button,'Enable','on')
set(handles.pathseek_push_button,'Enable','on')
set(handles.val_path_push_button,'Enable','on')
set(handles.replay_push_button,'Enable','on')
set(handles.end_push_button,'Enable','on')

ok_flag = ok_flag + 1;

% --------------------------------------------------------------------
function varargout = end_push_button_Callback(h, eventdata, handles,
varargin)
% launch dialog box to confirm close
pos_size = get(handles.figure1,'Position');
pos_size = [55 15 pos_size(3) pos_size(4)];
user_response = modaldlg([pos_size(1)+pos_size(3)/5
pos_size(2)+pos_size(4)/5]);
switch user_response
case {'no','cancel'}
   % take no action
case 'yes'
   % Prepare to close GUI application window
    % Halt communication with the UFS
    matjr3pci('close_jr3');

    % Close the serial port
    port1 = evalin('base', 'port1');
    fclose(port1);

    delete(handles.figure1)
end

% --------------------------------------------------------------------
function varargout = file_menu_Callback(h, eventdata, handles, varargin)


% --------------------------------------------------------------------
function varargout = print_file_sub_menu_Callback(h, eventdata, handles,
varargin)
% List dialog box to select figure to print
```

```
str = {'Figure 1'; 'Figure 2'};
[selection, ok] = listdlg('ListString',str, 'Name','Print Figure',...
    'PromptString','Select a figure to print')

% Set current figure to selected figure
% set(gcf,handles.figure2)

% Print figure
% print

% Set current figure back to GUI
% set(gcf,handles.figure1)

% --------------------------------------------------------------------
function varargout = tool_menu_Callback(h, eventdata, handles, varargin)

% --------------------------------------------------------------------
function varargout = stop_tools_sub_menu_Callback(h, eventdata, handles,
varargin)
monitor_flag = 1;
assignin('base', 'monitor_flag', monitor_flag)

% --------------------------------------------------------------------
function varargout = help_menu_Callback(h, eventdata, handles, varargin)



% --------------------------------------------------------------------
function varargout = protocol_help_sub_menu_Callback(h, eventdata, handles,
varargin)
```

The user must follow several steps in the GUI before getting to the hybrid control algorithm: remove bolt-up loads (boltup_flex_ext3.m) and minimize any loads arising from the attachement of the end-effector to the superior fixture (initial_loads2.m and boltup_leash2.m).

boltup_flex_ext3.m is a script called by the GUI to calculate the loads on the UFS due to bolt-up, the weight of the attachments on the UFS and the center of gravity of the attachments.

```
%function [avg, x0, y0, z0, w_mg] = boltup_flex_ext3;

% boltup_flex_ext3
%controller moves robot into #pp1-6
%function to read forces/moments at each #pp

% % Disable buttons on GUI until boltup_flxn_ext.m is done running
```

```
% set(hok, 'Enable', 'off');
% set(hbolt, 'Enable', 'off');
% set(hbefore, 'Enable', 'off');
% set(hafter, 'Enable', 'off');
% set(hpath, 'Enable', 'off');
% set(hval, 'Enable', 'off');
% set(hreplay, 'Enable', 'off');
% set(hend, 'Enable', 'off');

% Disable buttons on GUI until spine3h_pathseek7.m is done running
buttons(guihandles, 'off');

pp(1,1:6) = [0,-45.005,135.001,0,-.05,-180.145];
pp(2,1:6) = [0,-45.005,135.001,0,-.05,-.142];
pp(3,1:6) = [0,-45.005,135.001,0,-.05,89.855];
pp(4,1:6) = [0,-45.005,135.001,0,-.05,-90.147];
pp(5,1:6) = [0,-45.005,135.001,0,-90.05,-90.15];
pp(6,1:6) = [0,-45.005,135.001,0,89.95,-90.15];

% % set transformation for COR from UFS face (remember that the UFS has a
left-hand rule, so positive z axis points toward the robot)
% trans_ufst = [1,round(x1*1000/0.0254), 2,round(y1*1000/0.0254), 3,round(-
(z1-0.045)*1000/0.0254), 4,round(rx1*32768/180), 5,round(ry1*32768/180),
6,round(rz1*32768/180),0];
% b = matjr3pci('set_transforms', 0, 'trans_ufst', 13, 0);
%
% % use transformation
% b = matjr3pci('use_transforms', 0, 0);
%
% % only use pause if updating COR
% pause(1);

for p = 1:6
    fprintf(port1, pp(p,1:6));
    flag = 0;
    flag = fscanf(port1);
    newflag = sscanf(flag, '%f');
    if newflag == 1
        get_loads;
        % fm_ufs = get_loads;
        pp_fin(1:3,p)=fm_ufs(1:3)';
        pp_min(1:3,p)=fm_ufs(4:6)';
        cg_fin(1:3,p)=fm_ufs(1:3)';
        cg_min(1:3,p)=fm_ufs(4:6)';
    else
        var = 1
    end
end

fprintf(port1, pp(3,1:6));

% FSU forces/moments=UFS forces/moments[]-avg[]-fixture wt[]
favgx = (pp_fin(1,3)+pp_fin(1,4)+pp_fin(1,5)+pp_fin(1,6))/4;
favgy = (pp_fin(2,1)+pp_fin(2,2)+pp_fin(2,5)+pp_fin(2,6))/4;
favgz = (pp_fin(3,1)+pp_fin(3,2)+pp_fin(3,3)+pp_fin(3,4))/4;
mavgx = (pp_min(1,1)+pp_min(1,2))/2;
mavgy = (pp_min(2,3)+pp_min(2,4)+pp_min(2,5)+pp_min(2,6))/4;
```

```
mavgz = (pp_min(3,3)+pp_min(3,4)+pp_min(3,5)+pp_min(3,6))/4;

avg = -[favgx favgy favgz mavgx mavgy mavgz];
avg_dig(2) = avg(2)*16384/20/4.44;
avg_dig(3) = avg(3)*16384/50/4.44;

% FSU forces/moments=UFS forces/moments[]-avg[]-fixture wt[]
cg_favgx = (cg_fin(1,3)+cg_fin(1,4)+cg_fin(1,5)+cg_fin(1,6))/4;
cg_favgy = (cg_fin(2,1)+cg_fin(2,2)+cg_fin(2,5)+cg_fin(2,6))/4;
cg_favgz = (cg_fin(3,1)+cg_fin(3,2)+cg_fin(3,3)+cg_fin(3,4))/4;
cg_mavgx = (cg_min(1,1)+cg_min(1,2))/2;
cg_mavgy = (cg_min(2,3)+cg_min(2,4)+cg_min(2,5)+cg_min(2,6))/4;
cg_mavgz = (cg_min(3,3)+cg_min(3,4)+cg_min(3,5)+cg_min(3,6))/4;

% Calculate the center of gravity and mass of top fixture.

% 3 and 4 : d = z
% 3 : dz = -mx/fy
% 4 : dz = -mx/fy
fy_cg3 = -cg_fin(2,3) + cg_favgy;
fy_cg4 = -cg_fin(2,4) + cg_favgy;
mx_cg3 = -cg_min(1,3) + cg_mavgx;
mx_cg4 = -cg_min(1,4) + cg_mavgx;
momarm_z1 = -(mx_cg3/fy_cg3)*1000;
momarm_z2 = -(mx_cg4/fy_cg4)*1000;
momarm_z = (momarm_z1 + momarm_z2)/2;
z0 = momarm_z/1000;

% 1 and 2 : d = y
% 1 : dy = -mz/fx
% 2 : dy = -mz/fxfy_cg3 = cg_fin(2,3);
fx_cg1 = -cg_fin(1,1) + cg_favgx;
fx_cg2 = -cg_fin(1,2) + cg_favgx;
mz_cg1 = -cg_min(3,1) + cg_mavgz;
mz_cg2 = -cg_min(3,2) + cg_mavgz;
momarm_y1 = -(mz_cg1/fx_cg1)*1000;
momarm_y2 = -(mz_cg2/fx_cg2)*1000;
momarm_y = (momarm_y1 + momarm_y2)/2;
y0 = momarm_y/1000;

% 5 and 6 : d = x
% 5 : dx = -my/fz
% 6 : dx = -my/fzfy_cg3 = cg_fin(2,3);
fz_cg5 = -cg_fin(3,5) + cg_favgz;
fz_cg6 = -cg_fin(3,6) + cg_favgz;
my_cg5 = -cg_min(2,5) + cg_mavgy;
my_cg6 = -cg_min(2,6) + cg_mavgy;
momarm_x1 = -(my_cg5/fz_cg5*1000);
momarm_x2 = -(my_cg6/fz_cg6*1000);
momarm_x = (momarm_x1 + momarm_x2)/2;
x0 = momarm_x/1000;

% mass = 3(-fy), 4(fy), 1(-fx), 2(fx), 5(-fz), 6(fz)
mass_calc = ((-fy_cg3) + (fy_cg4) + (-fx_cg1) + (fx_cg2) + (-fz_cg5) +
(fz_cg6))/6;
mass_calc = -mass_calc;
w_mg = [0 0 mass_calc]';
```

```
filename = ['c:\robot\temp\temp ', date];
save(filename);

% % Enable buttons on GUI when boltup_flex_ext3.m is done running
% set(hok, 'Enable', 'on');
% set(hbolt, 'Enable', 'on');
% set(hbefore, 'Enable', 'on');
% set(hafter, 'Enable', 'on');
% set(hpath, 'Enable', 'on');
% set(hval, 'Enable', 'on');
% set(hreplay, 'Enable', 'on');
% set(hend, 'Enable', 'on');

% Enable buttons on GUI when spine3h_pathseek7.m is done running
buttons(guihandles, 'on');
```

get_loads.m is a script called by several other scripts to read the loads from the JR3 PCI card.

```
% function fm_ufs = get_loads;

% get_loads
% Kevin M. Bell
% 03/18/02

% % Commented out on 09-04-02.
% % We are having a problem inplementing the set and use transformation
functions.  It appears that the transformation is randomly used and not used,
% % meaning that sometimes the loads are read at the c.s. we set (the
specimen COR) and sometimes they are read at the center of the UFS.
% % There does not seem to be any kind of pattern to this behavior, so we
just took the transformation functions out.  Now we read all loads at the
center
% % of the UFS and transform them later to the COR.  If we can get the
transformation functions to work later, we may go back to using them.

% % set transformation for COR from center of UFS (remember that the UFS has
a left-hand rule, so positive z axis points toward the robot)
% trans_ufst = [1,round(x1*1000/0.0254), 2,round(y1*1000/0.0254), 3,round(-
(z1-0.045)*1000/0.0254), 4,round(rx1*32768/180), 5,round(ry1*32768/180),
6,round(rz1*32768/180),0];
% b = matjr3pci('set_transforms', 0, 'trans_ufst', 13, 0);
%
% % use transformation
% b = matjr3pci('use_transforms', 0, 0);
%
% pause(1);

araw = 0;
%read in full scales
full = matjr3pci('get_full_scales',0);
```

```
% read in raw data from UFS at center of UFS
for i = 1:30
    raw(i,:) = matjr3pci('read_ftdata',3,0);
    % flip y to make right hand c.s.
    raw(i,2) = -raw(i,2);
    raw(i,5) = -raw(i,5);
    araw = araw + raw(i,:);
    % pause added so that data from pci card is not read too quickly,
    % otherwise, all forces and moments in raw() are the same
%     pause(0.01);
end

% average 30 readings
araw = araw/30;

% Calculate forces/moments in pounds/inch-pounds
% Negative sign to show f/m in robot point of view
fm_ufs = -araw.*full/16384;

% Remember that fm_ufs[] are loads at the center of the UFS.  They are
transformed to the COR later.
% seperate and convert forces and moments
fm_ufs(1:3) = fm_ufs(1:3)*4.44;
fm_ufs(4:6) = fm_ufs(4:6)*4.44*.0254;
fm_ufs(7:8) = [];
```

initial_loads2.m is a script called by the GUI to find the loads on the UFS before attaching the superior fixture to the end-effector.

```
% Disable buttons on GUI until initial_loads2.m is done running
buttons(guihandles, 'off');

fm_before = [0 0 0 0 0 0];

for j = 1:100

    get_loads;
    fm_before = fm_before + fm_ufs;

end

fm_before = fm_before'/100;

filename = ['c:\robot\temp\temp ', date];
save(filename);

% Enable buttons on GUI when spine3h_pathseek7.m is done running
buttons(guihandles, 'on');
```

boltup_leash2.m is a script called by the GUI that relieves the loads created during attachment of the superior fixture to the end-effector.

```
% boltup_leash2.m
% use load control to make sure that no f/m are added when fixture is
attached to UFS
% modified from trpy.m by Lianfang Tian
% July 28, 2002

% Disable buttons on GUI until boltup_leash2.m is done running
buttons(guihandles, 'off');

% setup figure to graphically monitor loads
[fx, fy, fz, mx, my, mz, fh] = attach_display1;

% Define the threshold value for force and moment
f_min = 0.5; % N
m_min = 0.25; % N-m

% Limit for displacements
lim_dis = 0.1; % mm
lim_mdis = 0.1; % degrees

% Define stiffness
for n=1:3
    stiff(n) = 10.00001; % N/mm
end

for n=4:6
    stiff(n) = 10.00001; % N-m/degrees
end

% convert rotations about tool x,y,z axes to Euler angles
eul = rad2deg(tr2eul(rpy2tr(deg2rad(rz1), deg2rad(ry1), deg2rad(rx1))));

% send x1, y1, z1, rx1, ry1, rz1 to V+ to make tool transformation
ok = 0;
flag = 0.1;
fprintf(port1, [ok, flag]);
fprintf(port1, [(x1*1000)+.1, (y1*1000)+.1, (z1*1000)+.1, eul(1)+.1,
eul(2)+.1, eul(3)+.1]);

done_moving = fscanf(port1);
done_moving = sscanf(done_moving, '%f');

zero_flag = 0;
kk = 0;

while zero_flag == 0
    kk = kk + 1;

    fm_after = [0 0 0 0 0 0];

    pause(2);
```

```
    % Read forces/moments
    for j = 1:100
        get_loads;
        fm_after = fm_after + fm_ufs;
    end

    % Average forces/moments
    fm_after = fm_after'/100;
    total_fm_after(1:6,kk) = fm_after;

    % Remove forces/moments present before fixture attachment
    fm_diff = fm_after - fm_before;

    % Display forces and moments
    attach_display2([fm_diff', fx, fy, fz], [mx, my, mz], [f_min, m_min]);
    %=========================================================

    % Find translations in UFS c.s.
    for k1=1:3
        if (abs(fm_diff(k1)) < abs(f_min))
            dis(k1) = 0;
        else
            dis(k1) = fm_diff(k1)/stiff(k1);
        end

        if abs(dis(k1)) > lim_dis
            dis(k1) = sign(dis(k1))*lim_dis;
        end
    end

    % out_dis1, out_dis2, out_dis3 are translations in UFS c.s.
    out_dis1(kk)=dis(1);
    out_dis2(kk)=dis(2);
    out_dis3(kk)=dis(3);
    %=========================================================

    % Find rotations in UFS c.s.
    for k1=4:6
        if (abs(fm_diff(k1))<abs(m_min))
            dis(k1) = 0.0000001;
        else
            dis(k1) = fm_diff(k1)/stiff(k1);
        end

        if abs(dis(k1)) > lim_mdis
            dis(k1) = sign(dis(k1))*lim_mdis;
        end
    end

    % out_mdis1, out_mdis2, out_mdis3 are rotations about tool c.s. axes
    % need to convert to Euler angles
    out_mdis = rad2deg(tr2eul(rpy2tr(deg2rad(dis(6)), deg2rad(dis(5)),
deg2rad(dis(4)))));
    out_mdis1(kk)=out_mdis(1);
    out_mdis2(kk)=out_mdis(2);
    out_mdis3(kk)=out_mdis(3);
    %=========================================================
```

```
   % Send position data to robot
   % motions need to be negative to account for forces
   send = -[out_dis1(kk), out_dis2(kk), out_dis3(kk), out_mdis1(kk),
out_mdis2(kk), out_mdis3(kk)];
    ok = 0;
    flag = 2.1;
    fprintf(port1, [ok, flag]);
   fprintf(port1, send);

    done_moving = fscanf(port1);
    done_moving = sscanf(done_moving, '%f');

    % end while loop if done minimizing forces or if reach too many
iterations (kk)
    if send == -[0 0 0 rad2deg(tr2eul(rpy2tr(deg2rad(0.0000001),
deg2rad(0.0000001), deg2rad(0.0000001))))] | kk == 50;
        zero_flag = 1;
    end
end

% remove monitor loads figure from screen
delete(fh)

% Enable buttons on GUI when boltup_leash2.m is done running
buttons(guihandles, 'on');
```

buttons.m is a function called by several scripts to disable and enable the buttons on the GUI.

```
function buttons(handles, flag)

hok = handles.ok_push_button;
hbolt = handles.boltup_push_button;
hbefore = handles.fm_before_push_button;
hafter = handles.load_control_push_button;
hpath = handles.pathseek_push_button;
hval = handles.val_path_push_button;
hreplay = handles.replay_push_button;
hend = handles.end_push_button;

switch flag
case 'on'
    % Enable buttons on GUI when spine3h_pathseek6.m is done running
   set(hok, 'Enable', 'on');
   set(hbolt, 'Enable', 'on');
   set(hbefore, 'Enable', 'on');
   set(hafter, 'Enable', 'on');
   set(hpath, 'Enable', 'on');
   set(hval, 'Enable', 'on');
   set(hreplay, 'Enable', 'on');
   set(hend, 'Enable', 'on');
case 'off'
```

```
    % Disable buttons on GUI until spine3h_pathseek4.m is done running
   set(hok, 'Enable', 'off');
   set(hbolt, 'Enable', 'off');
   set(hbefore, 'Enable', 'off');
   set(hafter, 'Enable', 'off');
   set(hpath, 'Enable', 'off');
   set(hval, 'Enable', 'off');
   set(hreplay, 'Enable', 'off');
   set(hend, 'Enable', 'off');
end
```

spine3h_pathseek4.m is a script called by the GUI to perform pathseek. Several other scripts are called during execution of spine3h_pathseek4.m and follow in this appendix in the order in which they appear in spine3h_pathseek4.m

```
% spine3h_pathseek4.m
% perform flexion/extension with all position and load data stored
% converted from spine3h.v2
% Amy Loveless
% 7/4/2002

% Disable buttons on GUI until spine3h_pathseek4.m is done running
buttons(guihandles, 'off');

% Input dialog box to get the filename for data storage
prompt = {'Enter Filename'};
title = 'Filename';
lines = 1;
def = {'c:\robot'};
answer = inputdlg(prompt,title,lines,def);
if isequal(answer,{}) == 1
    % Enable buttons on GUI
    buttons(guihandles, 'on');
else
    filename = answer{1};
end

% Clear variables created for inputdlg
clear prompt title lines def answer;

% initialize stiffness, target f/m, temp. f/m, temp positions
z_stiff = [100 100 100 10 10 10];
z_flag = [0 0 0 0 0 0];
z_stop = [30 30 30 9 9 9];
f_temp = [0 0 0 0 0 0];
p_temp = [0 0 0 0 0 0 0 0 0 0 0 0]';

% initialize iterations
z_ct = 1;       % keeps track of no. of iterations to reach min. force
z_count = 10;    % limit to z_count iterations
z_ct_temp = z_count;  % keeps track of no. of iterations to reach min. force
```

```
z_step = 1;     % index to keep track of what direction and angle data
gathered was at
z_xform = 1;    % index to keep track of global c.s. to tool c.s. xform info
sent to Matlab
z_mom_flag = 1;     % how many rotation angles the moment > max.mom
z_index = 1;    % index to keep track of number of iterations per angle

% initialize direction
dir_flag = 0;   % change direction if dir_flag <> 0
dir = 0; % begin with start -> flxn

% initialize stablity check
stable_flag = 0;
stable_flag_flxn = 0;
stable_flag_extn = 0;
start_counter = 0;
flxn_counter = 0;
extn_counter = 0;

% define the limits for displacement, rotation, f/e moment and pathseek limit
lim_dis = 1; % mm
lim_mdis = 3; % degrees
max_mom = 2.40; % N-m
path_limit = 4;

% initialize work
work = 0;

% initialize timer
tic;

% setup figure to graphically monitor loads
[fx, fy, fz, mx, my, mz, handles, fh] = pathseek_display1;

% send x1, y1, z1, rx1, ry1, rz1 to V+ to make tool transformation
ok = 0;
flag = 0.1;
fprintf(port1, [ok, flag]);
fprintf(port1, [(x1*1000)+.1, (y1*1000)+.1, (z1*1000)+.1, rx1+.1, ry1+.1,
rz1+.1]);

done_moving = fscanf(port1);
done_moving = sscanf(done_moving, '%f');
%=========================================================================
===================================

while stable_flag ~= 100

    if dir == 0
        w_begin = w_start;
        w_inc = w_neg;
        start_counter = start_counter + 1;
   end
   if dir == 400
        w_begin = w_current;
        w_inc = w_ang;
        flxn_counter = flxn_counter + 1;
```

```
        stable_flag_flxn = 0;
        stable_flag_extn = 0;
    end
    if dir == 800
        w_begin = w_current;
        w_inc = w_neg;
        extn_counter = extn_counter + 1;
    end
     if dir == 900
        w_begin = w_current;
        w_end = w_start;
        w_inc = w_ang;
        stable_flag = 100;
        start_counter = start_counter + 1;
     end

     w_now = w_begin;

     while dir_flag == 0
        ok = 0;
        flag = 1.1;
        fprintf(port1, [ok, flag]);
        gt_jt_angles = fscanf(port1);
        gt_jt_angles = sscanf(gt_jt_angles, '%f');
        if dir == 0 | dir == 900
            z_gt0(1:6,z_xform,start_counter) = gt_jt_angles(1:6);
            z_jt_angles0(1:6,z_xform,start_counter) = gt_jt_angles(7:12);
        elseif dir == 400
            z_gt400(1:6,z_xform,flxn_counter) = gt_jt_angles(1:6);
            z_jt_angles400(1:6,z_xform,flxn_counter) = gt_jt_angles(7:12);
        elseif dir == 800
            z_gt800(1:6,z_xform,extn_counter) = gt_jt_angles(1:6);
            z_jt_angles800(1:6,z_xform,extn_counter) = gt_jt_angles(7:12);
        end
        z_xform = z_xform + 1;
        for n = 1:6
            z_sign(n) = 0;
            z_flag(n) = 0;
        end

        ct = 1;
      %=============================================
      load_control_first3; % load control (inner) loop
      %=============================================

        % are the measured sagittal plane forces < max allowable?
      % if no, begin load control loop again
      % if yes, put data in matrices
      % limit to 8 iterations (will want to change to time limit)
        while z_ct < z_count
            if sqrt(fa(2)^2 + fa(3)^2) > z_target(2)
%             if (abs(fa(2)) > z_target(2)) | (abs(fa(3)) > z_target(3))
                z_ct = z_ct + 1;
            z_step = z_step + 1;
            z_xform = z_xform + 1;
                %=========================================
            load_control3; % load control (inner) loop
```

```
            %=========================================
        else
            z_ct_temp = z_ct;
            z_ct = z_count;
        end
    end

    z_ct = z_ct_temp;

    if dir == 0
        % Build array of start position data that could be for replay
        start_replay1(1:6,z_index) =
z_gt0(1:6,z_xform,start_counter)+0.000001;
        % Build array of rotation angles at last iteration
        rot_angle0_end_pts(1,z_index,start_counter) = w_now;
        % Build array of loads at last iteration
        start_load_end_pts(1:6,z_index,start_counter) =
load0(1:6,z_step,start_counter);
        % Build array of work at last iteration
        works0end(1,z_index,start_counter) = work;
    elseif dir == 400
        % Build array of flxn position data to be written to V+ for
replay
        flxn_replay(1:6,z_index) =
z_gt400(1:6,z_xform,flxn_counter)+0.000001;
        % Build array of rotation angles at last iterations
        rot_angle400_end_pts(1,z_index,flxn_counter) = w_now;
        % Build array of loads at last iterations
        flxn_load_end_pts(1:6,z_index,flxn_counter) =
load400(1:6,z_step,flxn_counter);
    elseif dir == 800
        % Build array of extn position data to be written to V+ for
replay
        extn_replay(1:6,z_index) =
z_gt800(1:6,z_xform,extn_counter)+0.000001;
        % Build array of rotation angles at last iterations
        rot_angle800_end_pts(1,z_index,extn_counter) = w_now;
        % Build array of loads at last iterations
        extn_load_end_pts(1:6,z_index,extn_counter) =
load800(1:6,z_step,extn_counter);
    elseif dir == 900
        % Build array of start position data that could be for replay (it
will not)
        start_replay2(1:6,z_index) =
z_gt0(1:6,z_xform,start_counter)+0.000001;
        % Build array of rotation angles at last iterations
        rot_angle0_end_pts(1,z_index,start_counter) = w_now;
        % Build array of loads at last iterations
        start_load_end_pts(1:6,z_index,start_counter) =
load0(1:6,z_step,start_counter);
    end

    if dir == 0 | dir == 900
        z_ct0_total(1,z_index,start_counter) = z_ct;
    elseif dir == 400
        z_ct400_total(1,z_index,flxn_counter) = z_ct;
    elseif dir == 800
```

```
            z_ct800_total(1,z_index,extn_counter) = z_ct;
        end
        z_ct = 1;
        z_step = z_step + 1;
        z_xform = z_xform + 1;
        z_index = z_index + 1;
        z_ct_temp = z_count;

        %=========================================
        max_moment2; % max moment loop
        %=========================================

        % ----------------------------------------
        % this part added for testing
        if w_now < -0.9
            dir_flag = 1;
        end
        % ----------------------------------------

        if dir_flag == 0  % continue with current direction
            % for planar f/e program, displacement control should be a pure
rotation about the x axis, but tr2eul does not give us correct
            % yaw,pitch,roll for a pure rotation about the x axis, therefore,
we have to have a very small rotation about the y and z axes, too.
            % (see the m file for tr2eul.m to see how the Euler angles are
calculated.)
%             rot_inc_x = rotx(deg2rad(w_inc));
%             rot_inc_y = roty(deg2rad(0.0000001));
%             rot_inc_z = rotz(deg2rad(0.0000001));
%             rot_inc = rot_inc_x*rot_inc_y*rot_inc_z;
%             rotate_inc = tr2eul(rot_inc);
%             rotate_inc = rad2deg(rotate_inc);
            rotate_inc = rad2deg(tr2eul(rpy2tr(deg2rad([0.0000001, 0.0000001,
w_inc])))) + 0.0000001;
            ok = 0;
            flag = 2.1;
            fprintf(port1, [ok, flag]);
            fprintf(port1, [0 0 0 rotate_inc(1) rotate_inc(2)
rotate_inc(3)]);
            done_moving = fscanf(port1);
            done_moving = sscanf(done_moving, '%f');
            w_now = w_now + w_inc;
        end

        if dir_flag == 1  % change direction
            w_current = w_now;
            break
        end
    end

    % stability check
    if dir == 800 & extn_counter > 1
        % use flxn_load_end_pts & rot_angle400_end_pts
        flxn_mx_percent = 100*abs((flxn_load_end_pts(4,1,flxn_counter-1)-
flxn_load_end_pts(4,1,flxn_counter))/flxn_load_end_pts(4,1,flxn_counter));
        flxn_rot_angle_percent =
100*abs((rot_angle400_end_pts(1,1,flxn_counter-1)-
```

```
rot_angle400_end_pts(1,1,flxn_counter))/rot_angle400_end_pts(1,1,flxn_counter
));
        if flxn_mx_percent < 4 & flxn_rot_angle_percent < 4
            stable_flag_flxn = 25;
        end
        % use extn_load_end_pts & rot_angle800_end_pts
        extn_mx_percent = 100*abs((extn_load_end_pts(4,1,extn_counter-1)-
extn_load_end_pts(4,1,extn_counter))/extn_load_end_pts(4,1,extn_counter));
        extn_rot_angle_percent =
100*abs((rot_angle800_end_pts(1,1,extn_counter-1)-
rot_angle800_end_pts(1,1,extn_counter))/rot_angle800_end_pts(1,1,extn_counter
));
        if extn_mx_percent < 4 & extn_rot_angle_percent < 4
            stable_flag_extn = 25;
        end
    end

    % if stable_flag == 100, then the while loop will end
    if stable_flag ~= 100
        stable_flag = stable_flag_flxn + stable_flag_extn;
    end

    % added to test program with only one pathseek
    if dir == 800
        dir = 900;
    end

    % commented so that we can test program with only pathseek
%     if stable_flag == 50 | extn_counter > path_limit
%         dir = 900;
%     end
%
    w_current = w_now;

    %-------------------------------------------------------
    % this part added for testing
    if dir == 0
        stable_flag = 100;
    end
    %-------------------------------------------------------

    % ----------------------------
    % commented out for testing
%     if dir == 0 | dir == 400
%         dir = dir + 400;
%     elseif dir == 800
%         dir = 400;
%     end
    % ----------------------------

    dir_flag = 0;
    z_xform = 1;
    z_step = 1;
    z_index = 1;

end
```

```
% remove monitor loads figure from screen
delete(fh);

% Save workspace
save(filename)
disp('Data has been saved.')

%==========================================
% data_display_pathseek4; % display data
%==========================================

% Enable buttons on GUI when spine3h_pathseek4.m is done running
buttons(guihandles, 'on');
```

pathseek_display1.m is a function called by spine3h_pathseek4.m that sets up the plot for UFS loads.

```
function [fx, fy, fz, mx, my, mz, handles, fh] = pathseek_display1

% setup figure to graphically monitor loads
fh = figure('Position',[400 300 600 600],'Color','w');
subplot(2,1,1)
set(gca,'XLim', [-30 30], 'YLim', [0 4], 'YTick', [1 2 3], 'YTickLabel', 'Fz
(N)|Fy (N)|Fx (N)')
title('Forces')
fx = line('XData', [0 0], 'YData', [3 3], 'LineWidth', 24, 'Color', [0 0.75
0]);
fy = line('XData', [0 0], 'YData', [2 2], 'LineWidth', 24, 'Color', [0 0.75
0]);
fz = line('XData', [0 0], 'YData', [1 1], 'LineWidth', 24, 'Color', [0 0.75
0]);
origin = line('XData', [0 0], 'YData', [0 4]);

subplot(2,1,2)
set(gca,'XLim', [-10 10], 'YLim', [0 4], 'YTick', [1 2 3], 'YTickLabel', 'Mz
(Nm)|My (Nm)|Mx (Nm)')
title('Moments')
mx = line('XData', [0 0], 'YData', [3 3], 'LineWidth', 24, 'Color', [0 0.75
0]);
my = line('XData', [0 0], 'YData', [2 2], 'LineWidth', 24, 'Color', [0 0.75
0]);
mz = line('XData', [0 0], 'YData', [1 1], 'LineWidth', 24, 'Color', [0 0.75
0]);
origin = line('XData', [0 0], 'YData', [0 4]);

uicontrol('Style', 'text', 'Tag', 'current_text',...
    'Position', [20 0 60 20], 'BackgroundColor', [1 1 1], 'FontSize', 12,
'String', 'Current:');
uicontrol('Style', 'edit', 'Tag', 'w_now_edit',...
    'Position', [135 20 60 20], 'BackgroundColor', [1 1 1], 'FontSize', 12);
uicontrol('Style', 'text', 'Tag', 'w_now_text',...
    'Position', [135 0 60 20], 'BackgroundColor', [1 1 1], 'FontSize', 12,
'String', 'Angle');
```

```
uicontrol('Style', 'edit', 'Tag', 'iterations_edit',...
    'Position', [235 20 60 20], 'BackgroundColor', [1 1 1], 'FontSize', 12);
uicontrol('Style', 'text', 'Tag', 'iteration_text',...
    'Position', [235 0 60 20], 'BackgroundColor', [1 1 1], 'FontSize', 12,
'String', 'Iteration');
uicontrol('Style', 'edit', 'Tag', 'pathseek_edit',...
    'Position', [335 20 60 20], 'BackgroundColor', [1 1 1], 'FontSize', 12);
uicontrol('Style', 'text', 'Tag', 'pathseek_text',...
    'Position', [335 0 70 20], 'BackgroundColor', [1 1 1], 'FontSize', 12,
'String', 'Pathseek #');
uicontrol('Style', 'edit', 'Tag', 'stable_edit',...
    'Position', [435 20 60 20], 'BackgroundColor', [1 1 1], 'FontSize', 12);
uicontrol('Style', 'text', 'Tag', 'stable_text',...
    'Position', [435 0 65 20], 'BackgroundColor', [1 1 1], 'FontSize', 12,
'String', 'Stability %');
handles = guihandles(fh);
guidata(fh, handles);

% any of these changes should make simple animations smooth
% zbuffer can be very slow and on my computer none of these are necessary to
stop flashing
set(fh,'doublebuffer','on');
% set(fh,'renderer','zbuffer');
% set(hfig,'renderer','opengl');
```

load_control_first3.m is a script called by spine3h_pathseek4.m.

```
% load_control_first3.m
% load control (inner) loop
% Amy Loveless
% converted to Matlab 7/10/02

%=========================================
get_loads; % measure: forces and moments
% fm_ufs = get_loads;
%=========================================

%=========================================
fm_tare5; % tare out bolt-up and fixture wt
% [x, fa, fmw, rGT] = fm_tare5(w_mg, x0, y0, z0, x1, y1, z1, rx1, ry1, rz1,
fm_ufs, avg)
%=========================================

time = toc;
tic;

% store current position
for i = 1:6
    p_temp(i) = x(i);
end

% compute: FSU stiffness from previous measured force and position
if z_flag(1) == 0
   % compute: robot displacement vector to minimize sagittal forces and
moments (from computed stiffness)
```

```
   for i = 1:6
        z_flag(i) = 1;
        f_temp(i) = fmw(i); % keep previous f/m
        dis(i) = fmw(i)/z_stiff(i)/(1+1*z_sign(i));
   end
else
   for i = 1:6
        if (fmw(i) ~= f_temp(i)) & (ds(i) ~= 0) & (fmw(i) ~= 0)
            % STIFFNESS = old*1/3    +ABS(df/ds)*2/3
            z_stiff(i) = z_stiff(i)/3+abs((fmw(i)-f_temp(i))/ds(i))*2/3;
            % we changed to ds(i) from dis_tool_actual(i) on 07-29-02
        end

      if z_stiff(i) > 99999
            z_stiff(i) = 100000; % maximum z_stiff
      end

      if sign(f_temp(i)*fmw(i)) < 1
         z_sign(i) = 1;
      end

        % compute: robot displacement vector to minimize sagittal forces and
moments (from computed stiffness)
        z_flag(i) = 1;
        f_temp(i) = fmw(i); % keep previous f/m
        dis(i) = fmw(i)/z_stiff(i)/(1+1*z_sign(i));
   end
end

% determine translations based on forces
for i = 1:3
    if abs(dis(i)) > lim_dis
        dis(i) = sign(dis(i))*lim_dis;
    end
end

% transform from global c.s. to tool c.s.
dis_tool_calc(1:3) = rGT'*dis(1:3)';

% determine rotations based on moments
for i = 4:6
    if abs(dis(i)) > lim_mdis
        dis(i) = sign(dis(i))*lim_mdis;
        dis(i) = deg2rad(dis(i));
    end
end

% transform from global c.s. to tool c.s.
dis_tool_calc(4:6) = rGT'*dis(4:6)';

% disa[4]-[6] are rotations about x,y,z, not y,p,r, so need to make them
y,p,r
rot_x = rotx(dis_tool_calc(4));
rot_y = roty(dis_tool_calc(5));
rot_z = rotz(dis_tool_calc(6));
rot_xyz = rot_z*rot_y*rot_x;
rotate = tr2eul(rot_xyz);
```

```
rotate = rad2deg(rotate);

% ask for current position
ok = 0;
flag = 1.1;
fprintf(port1, [ok, flag]);
gt_jt_angles = fscanf(port1);
gt_jt_angles = sscanf(gt_jt_angles, '%f');

% display f/m after taring out bolt-up and fixture wt
pathseek_display2([fa, fx, fy, fz], [mx, my, mz], handles, [w_now, z_ct,
flxn_counter, stable_flag, z_target]);

% find actual translations and rotations in global c.s., transform to tool
c.s.
for i = 1:6
    ds(i) = x(i)-p_temp(i);
    p_temp(i) = x(i);
end
dis_tool_actual(1:3) = rGT'*ds(1:3)';
dis_tool_actual(4:6) = rGT'*ds(4:6)';

%work done by the bar
work=work+abs(0.5*(fmw(1)+f_temp(1))*ds(1)) ...
    +abs(0.5*(fmw(3)+f_temp(3))*ds(3)) ...
    +abs(0.5*(fmw(5)+f_temp(5))*deg2rad(w_inc));
works0(z_index, z_ct)=work;

% peak force
peak(z_index,z_ct) = sqrt(fmw(1)^2+fmw(3)^2);
peakX(z_index,z_ct) = fmw(1);
peakZ(z_index,z_ct) = fmw(3);

% put data in matrices
if dir == 0 | dir == 900
   eval(['dis_calc',int2str(dir),'(1:6,z_step,start_counter) = [0; 0; 0; 0;
0; 0];'])
   eval(['dis_actual_tool',int2str(dir),'(1:6,z_step,start_counter) =
transpose(dis_tool_actual);'])
   eval(['dis_actual_global',int2str(dir),'(1:6,z_step,start_counter) =
transpose(ds);'])
   eval(['load',int2str(dir),'(1:6,z_step,start_counter) = transpose(fa);'])
   eval(['stiff',int2str(dir),'(1:6,z_step,start_counter) =
transpose(z_stiff);'])
   eval(['time_total',int2str(dir),'(1,z_step,start_counter) = time;'])
   eval(['rot_angle',int2str(dir),'(1,z_step,start_counter) = w_now;'])
   eval(['z_gt',int2str(dir),'(1:6,z_xform,start_counter) =
gt_jt_angles(1:6);'])
   eval(['z_jt_angles',int2str(dir),'(1:6,z_xform,start_counter) =
gt_jt_angles(7:12);'])
elseif dir == 400 | dir == 800
    eval(['dis_calc',int2str(dir),'(1:6,z_step,flxn_counter) = [0; 0; 0; 0;
0; 0];'])
   eval(['dis_actual_tool',int2str(dir),'(1:6,z_step,flxn_counter) =
transpose(dis_tool_actual);'])
   eval(['dis_actual_global',int2str(dir),'(1:6,z_step,flxn_counter) =
transpose(ds);'])
```

```
   eval(['load',int2str(dir),'(1:6,z_step,flxn_counter) = transpose(fa);'])
   eval(['stiff',int2str(dir),'(1:6,z_step,flxn_counter) =
transpose(z_stiff);'])
   eval(['time_total',int2str(dir),'(1,z_step,flxn_counter) = time;'])
   eval(['rot_angle',int2str(dir),'(1,z_step,flxn_counter) = w_now;'])
   eval(['z_gt',int2str(dir),'(1:6,z_xform,flxn_counter) =
gt_jt_angles(1:6);'])
   eval(['z_jt_angles',int2str(dir),'(1:6,z_xform,flxn_counter) =
gt_jt_angles(7:12);'])
end

% % put data in matrices
% if dir == 0 | dir == 900
%     dis_calc0(1:6,z_step,start_counter) = [0 0 0 0 0 0]';
%  dis_actual_tool0(1:6,z_step,start_counter) = dis_tool_actual';
%  dis_actual_global0(1:6,z_step,start_counter) = ds';
%  load0(1:6,z_step,start_counter) = fa';
%  stiff0(1:6,z_step,start_counter) = z_stiff';
%  time_total0(1,z_step,start_counter) = time;
%  rot_angle0(1,z_step,start_counter) = w_now;
%  z_gt0(1:6,z_xform,start_counter) = gt_jt_angles(1:6);
%  z_jt_angles0(1:6,z_xform,start_counter) = gt_jt_angles(7:12);
% elseif dir == 400
%     dis_calc400(1:6,z_step,flxn_counter) = [0 0 0 0 0 0]';
%  dis_actual_tool400(1:6,z_step,flxn_counter) = dis_tool_actual';
%  dis_actual_global400(1:6,z_step,flxn_counter) = ds';
%  load400(1:6,z_step,flxn_counter) = fa';
%  stiff400(1:6,z_step,flxn_counter) = z_stiff';
%  time_total400(1,z_step,flxn_counter) = time;
%  rot_angle400(1,z_step,flxn_counter) = w_now;
%  z_gt400(1:6,z_xform,flxn_counter) = gt_jt_angles(1:6);
%  z_jt_angles400(1:6,z_xform,flxn_counter) = gt_jt_angles(7:12);
% elseif dir == 800
%     dis_calc800(1:6,z_step,flxn_counter) = [0 0 0 0 0 0]';
%  dis_actual_tool800(1:6,z_step,flxn_counter) = dis_tool_actual';
%  dis_actual_global800(1:6,z_step,flxn_counter) = ds';
%  load800(1:6,z_step,flxn_counter) = fa';
%  stiff800(1:6,z_step,flxn_counter) = z_stiff';
%  time_total800(1,z_step,flxn_counter) = time;
%  rot_angle800(1,z_step,flxn_counter) = w_now;
%  z_gt800(1:6,z_xform,flxn_counter) = gt_jt_angles(1:6);
%  z_jt_angles800(1:6,z_xform,flxn_counter) = gt_jt_angles(7:12);
% end
```

fm_tare5.m is a script called by load_control_first3.m and load_control3.m to tare out the weight of the attachments on the UFS. This is done to know what loads on the UFS are due to the specimen.

```
% function [x, fa, fmw, rGTCS] = fm_tare5(w_mg, x0, y0, z0, x1, y1, z1, rx1,
ry1, rz1, fm_ufs, avg)

% fm_tare5.m
```

```
% tare out bolt-up f/m and fixture wt
% this program can be used if yaw, pitch, roll <> 0 from UFS to tool
% Amy Loveless
% 3/3/2003
% the f/m are read in UFS c.s.
% bolt-up and weight are subtracted from f/m
% the positions are read in tool c.s. wrt global c.s.
% resulting forces transformed to global c.s.
% resulting moments transformed to global c.s.

% yaw, pitch, roll store position and orientation of end-effector
ok = 0;
flag = 1.1;
fprintf(port1, [ok, flag]);
x = fscanf(port1);
x = sscanf(x, '%f');
yaw = deg2rad(x(4));
pitch = deg2rad(x(5));
roll = deg2rad(x(6));

% tGTCS[] is the transformation matrix of tool c.s. wrt global c.s.
TGTCS = eul2tr(yaw, pitch, roll);
TGTCS(1:3,4) = [x(1) x(2) x(3)]';

% rGTCS[] is the rotation matrix of tool c.s. wrt global c.s.
rGTCS = TGTCS;
rGTCS(:,4) = [];
rGTCS(4,:) = [];
rGT = rGTCS;

% tUFSTCS is the transformation matix of UFS face c.s. to tool c.s. (this is
a constant transformation)
rot_rx1 = rotx(deg2rad(rx1));
rot_ry1 = roty(deg2rad(ry1));
rot_rz1 = rotz(deg2rad(rz1));
TUFSTCS = rot_rz1*rot_ry1*rot_rx1;
% Need to subtract (64-19) back off of z1 because values are compared from
face 08-12-02
TUFSTCS(1:3,4) = [x1 y1 (z1 - 45/1000)]';

% rUFSTCS is the rotation matrix of UFS face c.s. to tool c.s. (this is a
constant rotation)
rUFSTCS = TUFSTCS;
rUFSTCS(:,4) = [];
rUFSTCS(4,:) = [];

% tGUFS is the transformation matrix of UFS c.s. wrt global c.s.
TGUFS = TGTCS*inv(TUFSTCS);
% rGUFS is the rotation matrix of global c.s. to UFS face c.s.
% if rUFSTCS is an identity matrix, rGUFS = rGTCS
rGUFS = TGUFS;
rGUFS(:,4) = [];
rGUFS(4,:) = [];

% If we use the loads at the COR to calculate x0, y0, z0, then x0, y0, z0 is
the vector from the COR to c.g. (cg_rot).
```

```
% cg_rot is only used if we can get the transformation functions for the pci
card to work.
% cg_rot is only calculated if the loads are read at the COR.
% transform c.g. coordinates from UFS face c.s. to tool c.s.
% cg_rot = pinv(tUFST)*[x0 y0 z0 1]';

% w_mg[] are the loads from the fixture (c.g.) defined in the global c.s.
% w_mg_rot[] are the loads from the fixture (c.g.) defined in the UFS c.s.
% If the loads are found at the center of the UFS, x0,y0,z0 is the coordinate
of the c.g. measured in the UFS c.s.,
% which is centered in the UFS.
% Use [x0 y0 z0]' to find the moment of the c.g. about the center of the UFS
(where all the loads are found).
w_mg_rot(1:3) = rGUFS'*w_mg;
w_mg_rot(4:6) = cross([x0; y0; z0],w_mg_rot(1:3));
% Commented out on 09-04-02 (see notes above).
% w_mg_rot(4:6) = cross(cg_rot(1:3),w_mg_rot(1:3));

% convert fm_ufs[] to digital
% fm_ufs_dig(2) = fm_ufs(2)*16384/20/4.44;
% fm_ufs_dig(3) = fm_ufs(3)*16384/50/4.44;

% fa_unrot[1]-fa_unrot[3] are forces after bolt-up and fixture wt removed
from forces (in the UFS c.s.).
fa_unrot(1) = -(fm_ufs(1))-(avg(1))-(w_mg_rot(1));
fa_unrot(2) = -(fm_ufs(2))-(avg(2))-(w_mg_rot(2));
fa_unrot(3) = -(fm_ufs(3))-(avg(3))-(w_mg_rot(3));
% fa_unrot(2) = y_eq(1)*(fm_ufs_dig(2)-avg_dig(2))+y_eq(2)-w_mg_rot(2);
% fa_unrot(3) = z_eq(1)*(fm_ufs_dig(3)-avg_dig(3))+z_eq(2)-w_mg_rot(3);

% fa[1]-fa[3] are forces rotated to the tool c.s.
fa(1:3) = rUFSTCS'*fa_unrot(1:3)';

% fmw[1]-fmw[3] are forces in global c.s., rotated because calculations are
made in global c.s.
% fmw(1:3) = rotGT*fa(1:3)';
fmw(1:3) = rGUFS*fa_unrot(1:3)';

% fa_unrot[4]-fa_unrot[6] are moments after bolt-up and fixture wt removed
from moments (in the UFS c.s.).
fa_unrot(4) = -(fm_ufs(4))-(avg(4))-(w_mg_rot(4));
fa_unrot(5) = -(fm_ufs(5))-(avg(5))-(w_mg_rot(5));
fa_unrot(6) = -(fm_ufs(6))-(avg(6))-(w_mg_rot(6));

% fa[4]-fa[6] are moments rotated to tool c.s.
fa(4:6) = cross(-TUFSTCS(1:3,4),fa(1:3))' + rUFSTCS'*fa_unrot(4:6)';

% fmw[4]-fmw[6] are moments in global c.s., rotate because calculations are
made in global c.s.
fmw(4:6) = rGTCS*fa(4:6)';

% ================================================
% added for updating COR
TUFSa = rpy2tr(deg2rad([rza, rya, rxa]));
TUFSa(1:3,4) = [xa, ya, za]';
% TGa = TGUFS*TUFSa;
```

```
TUFSb = rpy2tr(deg2rad([rzb, ryb, rxb]));
TUFSb(1:3,4) = [xb, yb, zb]';
% TGb = TGUFS*TUFSb;
```

pathseek_display2.m is a function called by load_control_first3.m and load_control3.m that plots the load on the UFS.

```
function pathseek_display2(forces, moments, handles, misc)

fa = forces(1:6);
fx = forces(7);
fy = forces(8);
fz = forces(9);

mx = moments(1);
my = moments(2);
mz = moments(3);

w_now = misc(1);
z_ct = misc(2);
flxn_counter = misc(3);
stable_flag = misc(4);
limit = misc(5:10);

for i = 1:3
    if abs(fa(i)) > limit(i)
        line_color(i,1:3) = [1 0 0];
    else
        line_color(i,1:3) = [0 0.75 0];
    end
end

for i = 4:6
    if abs(fa(i)) > limit(i)
        line_color(i,1:3) = [1 0 0];
    else
        line_color(i,1:3) = [0 0.75 0];
    end
end

subplot(2,1,1), set(fx, 'XData', [0 fa(1)], 'Color', line_color(1,:));
subplot(2,1,1), set(fy, 'XData', [0 fa(2)], 'Color', line_color(2,:));
subplot(2,1,1), set(fz, 'XData', [0 fa(3)], 'Color', line_color(3,:));
subplot(2,1,2), set(mx, 'XData', [0 fa(4)], 'Color', line_color(4,:));
subplot(2,1,2), set(my, 'XData', [0 fa(5)], 'Color', line_color(5,:));
subplot(2,1,2), set(mz, 'XData', [0 fa(6)], 'Color', line_color(6,:));
set(handles.w_now_edit, 'String', w_now);
set(handles.iterations_edit, 'String', z_ct);
if flxn_counter == 0
    set(handles.pathseek_edit, 'String', 1);
else
    set(handles.pathseek_edit, 'String', flxn_counter);
end
```

```
set(handles.stable_edit, 'String', stable_flag);

drawnow
```

load_control3.m is a script called by spine3h_pathseek4.m that calculates the translation required to minimize force and commands the manipulator to displace by the calculated amount.

```
% load_control3.m
% load control (inner) loop
% Amy Loveless
% converted to Matlab 7/10/02

time = toc;
tic;

% store current position
for i = 1:6
    p_temp(i) = x(i);
end

% compute: FSU stiffness from previous measured force and position
if z_flag(1) == 0
   % compute: robot displacement vector to minimize sagittal forces and
moments (from computed stiffness)
   for i = 1:6
        z_flag(i) = 1;
        f_temp(i) = fmw(i); % keep previous f/m
        dis(i) = fmw(i)/z_stiff(i)/(1+1*z_sign(i));
   end
else
   for i = 1:6
        if (fmw(i) ~= f_temp(i)) & (ds(i) ~= 0) & (fmw(i) ~= 0)
            % STIFFNESS = old*1/3    +ABS(df/ds)*2/3
            z_stiff(i) = z_stiff(i)/3+abs((fmw(i)-f_temp(i))/ds(i))*2/3;
            % we changed to ds(i) from dis_tool_actual(i) on 07-29-02
        end

      if z_stiff(i) > 99999
            z_stiff(i) = 100000; % maximum z_stiff
      end

      if sign(f_temp(i)*fmw(i)) < 1
         z_sign(i) = 1;
      end

        % compute: robot displacement vector to minimize sagittal forces and
moments (from computed stiffness)
        z_flag(i) = 1;
        f_temp(i) = fmw(i); % keep previous f/m
        dis(i) = fmw(i)/z_stiff(i)/(1+1*z_sign(i));
   end
end
```

```
% determine translations based on forces
for i = 1:3
    if abs(dis(i)) > lim_dis
        dis(i) = sign(dis(i))*lim_dis;
    end
end

% transform from global c.s. to tool c.s.
dis_tool_calc(1:3) = rGT'*dis(1:3)';

% determine rotations based on moments
for i = 4:6
    if abs(dis(i)) > lim_mdis
        dis(i) = sign(dis(i))*lim_mdis;
        dis(i) = deg2rad(dis(i));
    end
end

% transform from global c.s. to tool c.s.
dis_tool_calc(4:6) = rGT'*dis(4:6)';

% disa[4]-[6] are rotations about x,y,z, not y,p,r, so need to make them
y,p,r
rot_x = rotx(dis_tool_calc(4));
rot_y = roty(dis_tool_calc(5));
rot_z = rotz(dis_tool_calc(6));
rot_xyz = rot_z*rot_y*rot_x;
rotate = tr2eul(rot_xyz);
rotate = rad2deg(rotate);

% move: translate superior vertebra to new "corrected" position
ok = 0;
flag = 2.1;
fprintf(port1, [ok, flag]);
fprintf(port1, [0 dis_tool_calc(2) dis_tool_calc(3) 0 0 0]);

done_moving = fscanf(port1);
done_moving = sscanf(done_moving, '%f');

% ask for current position
ok = 0;
flag = 1.1;
fprintf(port1, [ok, flag]);
gt_jt_angles = fscanf(port1);
gt_jt_angles = sscanf(gt_jt_angles, '%f');

%=========================================
get_loads; % measure: forces and moments
% fm_ufs = get_loads;
%=========================================

%=========================================
fm_tare5; % tare out bolt-up and fixture wt
% [x, fa, fmw, rGT] = fm_tare5(w_mg, x0, y0, z0, x1, y1, z1, rx1, ry1, rz1,
fm_ufs, avg)
%=========================================
```

```
% display f/m after taring out bolt-up and fixture wt
pathseek_display2([fa, fx, fy, fz], [mx, my, mz], handles, [w_now, z_ct,
flxn_counter, stable_flag, z_target]);

% find actual translations and rotations in global c.s., transform to tool
c.s.
for i = 1:6
    ds(i) = x(i)-p_temp(i);
    p_temp(i) = x(i);
end
dis_tool_actual(1:3) = rGT'*ds(1:3)';
dis_tool_actual(4:6) = rGT'*ds(4:6)';

%work done by the bar
work=work+abs(0.5*(fmw(1)+f_temp(1))*ds(1)) ...
    +abs(0.5*(fmw(3)+f_temp(3))*ds(3)) ...
    +abs(0.5*(fmw(5)+f_temp(5))*deg2rad(0));
works0(z_index, z_ct)=work;

% peak force
peak(z_index,z_ct) = sqrt(fmw(1)^2+fmw(3)^2);
peakX(z_index,z_ct) = fmw(1);
peakZ(z_index,z_ct) = fmw(3);

% put data in matrices
if dir == 0 | dir == 900
   eval(['dis_calc',int2str(dir),'(1:6,z_step,start_counter) = [0; 0; 0; 0;
0; 0];'])
   eval(['dis_actual_tool',int2str(dir),'(1:6,z_step,start_counter) =
transpose(dis_tool_actual);'])
   eval(['dis_actual_global',int2str(dir),'(1:6,z_step,start_counter) =
transpose(ds);'])
   eval(['load',int2str(dir),'(1:6,z_step,start_counter) = transpose(fa);'])
   eval(['stiff',int2str(dir),'(1:6,z_step,start_counter) =
transpose(z_stiff);'])
   eval(['time_total',int2str(dir),'(1,z_step,start_counter) = time;'])
   eval(['rot_angle',int2str(dir),'(1,z_step,start_counter) = w_now;'])
   eval(['z_gt',int2str(dir),'(1:6,z_xform,start_counter) =
gt_jt_angles(1:6);'])
   eval(['z_jt_angles',int2str(dir),'(1:6,z_xform,start_counter) =
gt_jt_angles(7:12);'])
elseif dir == 400 | dir == 800
    eval(['dis_calc',int2str(dir),'(1:6,z_step,flxn_counter) = [0; 0; 0; 0;
0; 0];'])
   eval(['dis_actual_tool',int2str(dir),'(1:6,z_step,flxn_counter) =
transpose(dis_tool_actual);'])
   eval(['dis_actual_global',int2str(dir),'(1:6,z_step,flxn_counter) =
transpose(ds);'])
   eval(['load',int2str(dir),'(1:6,z_step,flxn_counter) = transpose(fa);'])
   eval(['stiff',int2str(dir),'(1:6,z_step,flxn_counter) =
transpose(z_stiff);'])
   eval(['time_total',int2str(dir),'(1,z_step,flxn_counter) = time;'])
   eval(['rot_angle',int2str(dir),'(1,z_step,flxn_counter) = w_now;'])
   eval(['z_gt',int2str(dir),'(1:6,z_xform,flxn_counter) =
gt_jt_angles(1:6);'])
   eval(['z_jt_angles',int2str(dir),'(1:6,z_xform,flxn_counter) =
gt_jt_angles(7:12);'])
```

```
end

% % put data in matrices
% if dir == 0 | dir == 900
%     dis_calc0(1:6,z_step,start_counter) = [dis_tool_calc(1:3)
rotate(1:3)]';
%  dis_actual_tool0(1:6,z_step,start_counter) = dis_tool_actual';
%  dis_actual_global0(1:6,z_step,start_counter) = ds';
%  load0(1:6,z_step,start_counter) = fa';
%  stiff0(1:6,z_step,start_counter) = z_stiff';
%  time_total0(1,z_step,start_counter) = time;
%  rot_angle0(1,z_step,start_counter) = w_now;
%  z_gt0(1:6,z_xform,start_counter) = gt_jt_angles(1:6);
%  z_jt_angles0(1:6,z_xform,start_counter) = gt_jt_angles(7:12);
% elseif dir == 400
%     dis_calc400(1:6,z_step,flxn_counter) = [dis_tool_calc(1:3)
rotate(1:3)]';
%  dis_actual_tool400(1:6,z_step,flxn_counter) = dis_tool_actual';
%  dis_actual_global400(1:6,z_step,flxn_counter) = ds';
%  load400(1:6,z_step,flxn_counter) = fa';
%  stiff400(1:6,z_step,flxn_counter) = z_stiff';
%  time_total400(1,z_step,flxn_counter) = time;
%  rot_angle400(1,z_step,flxn_counter) = w_now;
%  z_gt400(1:6,z_xform,flxn_counter) = gt_jt_angles(1:6);
%  z_jt_angles400(1:6,z_xform,flxn_counter) = gt_jt_angles(7:12);
% elseif dir == 800
%     dis_calc800(1:6,z_step,flxn_counter) = [dis_tool_calc(1:3)
rotate(1:3)]';
%  dis_actual_tool800(1:6,z_step,flxn_counter) = dis_tool_actual';
%  dis_actual_global800(1:6,z_step,flxn_counter) = ds';
%  load800(1:6,z_step,flxn_counter) = fa';
%  stiff800(1:6,z_step,flxn_counter) = z_stiff';
%  time_total800(1,z_step,flxn_counter) = time;
%  rot_angle800(1,z_step,flxn_counter) = w_now;
%  z_gt800(1:6,z_xform,flxn_counter) = gt_jt_angles(1:6);
%  z_jt_angles800(1:6,z_xform,flxn_counter) = gt_jt_angles(7:12);
% end
```

max_moment2.m is a script called by spine3h_pathseek4.m that compares the current moment to the user-defined maximum moment. If the maximum moment has been greater than the user-defined maximum three times, the test changes direction.

```
% max_moment2.m
% max moment loop
% Amy Loveless
% converted to Matlab 7/10/02

% is the measured f/e moment < max allowable?
% if yes, correct COR
% if no, go on to next direction
if abs(fa(4)) < max_mom
    % compute: corrected COR
```

```
    % when chgs made to COR, x1, y1, z1, yaw1, pitch1, roll1 will be chged
    % disp('This is where I would correct the COR')
elseif (abs(fa(1)) > z_stop(1)) | (abs(fa(2)) > z_stop(2)) | (abs(fa(3)) >
z_stop(3)) | (abs(fa(4)) > z_stop(4)) | (abs(fa(5)) > z_stop(5)) |
(abs(fa(6)) > z_stop(6))
    % if f/m are > max allowable, change direction
    disp('Forces/moments are too high.')
    disp('********** CHANGING DIRECTION **********')
    z_mom_flag = 1;
    dir_flag = 1;
    continue % change direction
else
    z_mom_flag = z_mom_flag + 1;
    if z_mom_flag == 3
        z_mom_flag = 1;
        dir_flag = 1;
        disp('********** CHANGING DIRECTION **********')
        continue % change direction
    else
        % compute: corrected COR
        % when chgs made to COR, x1, y1, z1, yaw1, pitch1, roll1 will be
chged
        % disp('This is where I would correct the COR')
    end
end

if (w_now >= w_start & dir == 900)
    dir_flag = 1;
    disp('********** CHANGING DIRECTION **********')
    continue % change direction
end
```

data_display_pathseek4.m is a script called by spine3h_pathseek4.m that makes almost every conceivable plot from the gathered data.

```
% data_display_pathseek4.m
% display data
% Amy Loveless
% from data_sto_flxn (7/10/02)

% BUILD TRANSFORMATIONS
% Build transformation for UFS to tool c.s. (this is a constant
transformation)
rot_rx1 = rotx(deg2rad(rx1));
rot_ry1 = roty(deg2rad(ry1));
rot_rz1 = rotz(deg2rad(rz1));
tUFST = rot_rx1*rot_ry1*rot_rz1;
tUFST(1:3,4) = [x1*1000 y1*1000 z1*1000]';

% Build transformation for UFS to pt. of interest (this is a constant
transformation)
rot_rx2 = rotx(deg2rad(rx2));
rot_ry2 = roty(deg2rad(ry2));
```

```
rot_rz2 = rotz(deg2rad(rz2));
tUFSPOI = rot_rx2*rot_ry2*rot_rz2;
tUFSPOI(1:3,4) = [x2*1000 y2*1000 z2*1000]';

% BUILD TRANSFORMATIONS OF TOOL C.S. WRT GLOBAL C.S.
% Find where to truncate matrices that have been padded with zeros at the end
for flxn -> start
test = [0 0 0 0 0 0]';
size_z_gt0 = 0;
for i = 1:size(z_gt0,2)
    tf = isequal(z_gt0(1:6,i,end),test);
    if tf == 1
        size_z_gt0 = i-1;
        break
    end
end
if size_z_gt0 == 0
    size_z_gt0 = size(z_gt0,2);
end
% Build transformations of tool c.s. wrt global c.s. each location for flxn -
> start
for i = 1:size_z_gt0
    tGT0(1:4,i*4-3:4*i) = eul2tr([deg2rad(z_gt0(4,i,end)),
deg2rad(z_gt0(5,i,end)), deg2rad(z_gt0(6,i,end))]);
    tGT0(1:3,4*i) = z_gt0(1:3,i,end);
end

% Find where to truncate matrices that have been padded with zeros at the end
for flxn -> extn
size_z_gt400 = 0;
for i = 1:size(z_gt400,2)
    tf = isequal(z_gt400(1:6,i,end),test);
    if tf == 1
        size_z_gt400 = i-1;
        break
    end
end
if size_z_gt400 == 0
    size_z_gt400 = size(z_gt400,2);
end
% Build transformations of tool c.s. wrt global c.s. for each location for
flxn -> extn
for i = 1:size_z_gt400
    tGT400(1:4,i*4-3:4*i) = eul2tr([deg2rad(z_gt400(4,i,end)),
deg2rad(z_gt400(5,i,end)), deg2rad(z_gt400(6,i,end))]);
    tGT400(1:3,4*i) = z_gt400(1:3,i,end);
end

% Find where to truncate matrices that have been padded with zeros at the end
for extn -> flxn
size_z_gt800 = 0;
for i = 1:size(z_gt800,2)
    tf = isequal(z_gt800(1:6,i,end),test);
    if tf == 1
        size_z_gt800 = i-1;
        break
    end
```

```
end
if size_z_gt800 == 0
    size_z_gt800 = size(z_gt800,2);
end
% Build transformations of tool c.s. wrt global c.s. for each location for
extn -> flxn
for i = 1:size_z_gt800
    tGT800(1:4,i*4-3:4*i) = eul2tr([deg2rad(z_gt800(4,i,end)),
deg2rad(z_gt800(5,i,end)), deg2rad(z_gt800(6,i,end))]);
    tGT800(1:3,4*i) = z_gt800(1:3,i,end);
end

% Build array of position vectors of tool c.s. from tGL for flxn -> start
for i = 1:size(tGT0,2)/4
    tGT0_posn(1:4,i) = tGT0(1:4,i*4);
end

% Build array of position vectors of tool c.s. from tGL for flxn -> extn
for i = 1:size(tGT400,2)/4
    tGT400_posn(1:4,i) = tGT400(1:4,i*4);
end

% Build array of position vectors of tool c.s. from tGL for extn -> flxn
for i = 1:size(tGT800,2)/4
    tGT800_posn(1:4,i) = tGT800(1:4,i*4);
end

% BUILD TRANSFORMATIONS OF UFS WRT GLOBAL C.S.
% Build transformations of UFS wrt global c.s. for each location for start ->
flxn & flxn -> start
for i = 1:size(tGT0,2)/4
    tGUFS0(1:4,i*4-3:4*i) = tGT0(1:4,i*4-3:i*4)*pinv(tUFST);
end

% Build transformations of UFS wrt global c.s. for each location for flxn ->
extn
for i = 1:size(tGT400,2)/4
    tGUFS400(1:4,i*4-3:4*i) = tGT400(1:4,i*4-3:i*4)*pinv(tUFST);
end

% Build transformations of UFS wrt global c.s. for each location for extn ->
flxn
for i = 1:size(tGT800,2)/4
    tGUFS800(1:4,i*4-3:4*i) = tGT800(1:4,i*4-3:i*4)*pinv(tUFST);
end

% Build array of position vectors of UFS from tGUFS for start -> flxn & flxn
-> start
for i = 1:size(tGUFS0,2)/4
    tGUFS0_posn(1:4,i) = tGUFS0(1:4,i*4);
end

% Build array of position vectors of UFS from tGUFS for flxn -> extn
for i = 1:size(tGUFS400,2)/4
    tGUFS400_posn(1:4,i) = tGUFS400(1:4,i*4);
end
```

```
% Build array of position vectors of UFS from tGUFS for extn -> flxn
for i = 1:size(tGUFS800,2)/4
   tGUFS800_posn(1:4,i) = tGUFS800(1:4,i*4);
end

% BUILD TRANSFORMATIONS OF PT. OF INTEREST WRT GLOBAL C.S.
% Build transformations of pt. of interest wrt global c.s. for each location
for flxn -> start
for i = 1:size(tGT0,2)/4
    tGPOI0(1:4,i*4-3:4*i) = tGUFS0(1:4,i*4-3:i*4)*tUFSPOI;
end

% Build transformations of pt. of interest wrt global c.s. for each location
for flxn -> extn
for i = 1:size(tGT400,2)/4
    tGPOI400(1:4,i*4-3:4*i) = tGUFS400(1:4,i*4-3:i*4)*tUFSPOI;
end

% Build transformations of pt. of interest wrt global c.s. for each location
for extn -> flxn
for i = 1:size(tGT800,2)/4
    tGPOI800(1:4,i*4-3:4*i) = tGUFS800(1:4,i*4-3:i*4)*tUFSPOI;
end

% Build array of position vectors of pt. of interest from tGUFS for flxn ->
start
for i = 1:size(tGPOI0,2)/4
    tGPOI0_posn(1:4,i) = tGPOI0(1:4,i*4);
end

% Build array of position vectors of pt. of interest from tGUFS for flxn ->
extn
for i = 1:size(tGPOI400,2)/4
    tGPOI400_posn(1:4,i) = tGPOI400(1:4,i*4);
end

% Build array of position vectors of pt. of interest from tGUFS for extn ->
flxn
for i = 1:size(tGPOI800,2)/4
    tGPOI800_posn(1:4,i) = tGPOI800(1:4,i*4);
end
%=========================================================================
====================================

% BUILD ARRAYS OF DATA TO BE USED FOR PLOTTING
% Find where to truncate matrices that have been padded with zeros at the end
for start -> flxn & flxn -> start
test = [0 0 0 0 0 0]';
for i = 1:start_counter
    size_start(1,i) = 0;
    size_start_end_pts(1,i) = 0;
end
for j = 1:start_counter
   for i = 1:size(load0,2)
        tf = isequal(load0(1:6,i,j),test);
        if tf == 1
            size_start(1,j) = i-1;
```

```
            break
        end
    end
     if size_start(1,j) == 0
        size_start(1,j) = size(load0,2);
     end
end
for j = 1:start_counter
     for i = 1:size(start_load_end_pts,2)
        tf = isequal(start_load_end_pts(1:6,i,j),test);
        if tf == 1
            size_start_end_pts(1,j) = i-1;
            break
        end
     end
     if size_start_end_pts(1,j) == 0
        size_start_end_pts(1,j) = size(start_load_end_pts,2);
     end
end

% Find where to truncate matrices that have been padded with zeros at the end
for flxn -> extn
for i = 1:flxn_counter
     size_flxn(1,i) = 0;
     size_flxn_end_pts(1,i) = 0;
end
for j = 1:flxn_counter
    for i = 1:size(load400,2)
        tf = isequal(load400(1:6,i,j),test);
        if tf == 1
            size_flxn(1,j) = i-1;
            break
        end
    end
     if size_flxn(1,j) == 0
        size_flxn(1,j) = size(load400,2);
    end
end
for j = 1:flxn_counter
    for i = 1:size(flxn_load_end_pts,2)
        tf = isequal(flxn_load_end_pts(1:6,i,j),test);
        if tf == 1
            size_flxn_end_pts(1,j) = i-1;
            break
        end
    end
     if size_flxn_end_pts(1,j) == 0
        size_flxn_end_pts(1,j) = size(flxn_load_end_pts,2);
    end
end

% Find where to truncate matrices that have been padded with zeros at the end
for extn -> flxn
for i = 1:extn_counter
     size_extn(1,i) = 0;
     size_extn_end_pts(1,i) = 0;
end
```

```
for j = 1:extn_counter
   for i = 1:size(load800,2)
        tf = isequal(load800(1:6,i,j),test);
        if tf == 1
            size_extn(1,j) = i-1;
            break
        end
   end
    if size_extn(1,j) == 0
        size_extn(1,j) = size(load800,2);
   end
end
for j = 1:flxn_counter
   for i = 1:size(extn_load_end_pts,2)
        tf = isequal(extn_load_end_pts(1:6,i,j),test);
        if tf == 1
            size_extn_end_pts(1,j) = i-1;
            break
        end
   end
    if size_extn_end_pts(1,j) == 0
        size_extn_end_pts(1,j) = size(extn_load_end_pts,2);
   end
end

% Arrays of fy, fz & mx (all data points of last pathseek)
start_fy = load0(2,1:size_start(end),end);
start_fz = load0(3,1:size_start(end),end);
start_mx_1 = load0(4,1:size_start(1),1);
start_mx_2 = load0(4,1:size_start(end),end);

flxn_fy = load400(2,1:size_flxn(end),end);
flxn_fz = load400(3,1:size_flxn(end),end);
flxn_mx = load400(4,1:size_flxn(end),end);

extn_fy = load800(2,1:size_extn(end),end);
extn_fz = load800(3,1:size_extn(end),end);
extn_mx = load800(4,1:size_extn(end),end);

fy = [flxn_fy extn_fy start_fy];
fz = [flxn_fz extn_fz start_fz];
mx = [flxn_mx extn_mx start_mx_2];

% Array of mx of all data points of all pathseeks
flxn_extn_mx = [];
for i = 1:flxn_counter
    flxn_extn_mx = [flxn_extn_mx load400(4,1:size_flxn(i),i)
load800(4,1:size_extn(i),i)];
end
all_mx = [start_mx_1, flxn_extn_mx, start_mx_2];

% Arrays of fy, fz, mx (only at end of iterations for last pathseek)
start_fy_end_pts = start_load_end_pts(2,1:size_start_end_pts(end),end);
start_fz_end_pts = start_load_end_pts(3,1:size_start_end_pts(end),end);
start_mx_end_pts_1 = start_load_end_pts(4,1:size_start_end_pts(1),1);
start_mx_end_pts_2 = start_load_end_pts(4,1:size_start_end_pts(end),end);
```

```
flxn_fy_end_pts = flxn_load_end_pts(2,1:size_flxn_end_pts(end),end);
flxn_fz_end_pts = flxn_load_end_pts(3,1:size_flxn_end_pts(end),end);
flxn_mx_end_pts = flxn_load_end_pts(4,1:size_flxn_end_pts(end),end);

extn_fy_end_pts = extn_load_end_pts(2,1:size_extn_end_pts(end),end);
extn_fz_end_pts = extn_load_end_pts(3,1:size_extn_end_pts(end),end);
extn_mx_end_pts = extn_load_end_pts(4,1:size_extn_end_pts(end),end);

fy_end_pts = [flxn_fy_end_pts extn_fy_end_pts start_fy_end_pts];
fz_end_pts = [flxn_fz_end_pts extn_fz_end_pts start_fz_end_pts];
mx_end_pts = [flxn_mx_end_pts extn_mx_end_pts start_mx_end_pts_2];

% Array of mx of end data points of all pathseeks
% flxn_extn_mx_end_pts = [];
% for i = 1:flxn_counter
%     flxn_extn_mx_end_pts = [flxn_extn_mx_end_pts
flxn_load_end_pts(4,1:size_flxn_end_pts(i),i)
extn_load_end_pts(4,1:size_extn_end_pts(i),i)];
% end
% all_mx_end_pts = [start_mx_end_pts_1, flxn_extn_mx_end_pts,
start_mx_end_pts_2];
begin_mx_end_pts = [start_mx_end_pts_1,...
        flxn_load_end_pts(4,1:size_flxn_end_pts(1),1),...
        extn_load_end_pts(4,1:size_extn_end_pts(1),1)];
end_mx_end_pts = [flxn_load_end_pts(4,1:size_flxn_end_pts(end),end),...
        extn_load_end_pts(4,1:size_extn_end_pts(end),end),...
        start_mx_end_pts_2];

% Arrays of calculated and actual displacements in local y and z dir.
start_dy_calc = dis_calc0(2,1:size_start(end),end);
start_dy_actual = dis_actual_tool0(2,1:size_start(end),end);
start_dz_calc = dis_calc0(3,1:size_start(end),end);
start_dz_actual = dis_actual_tool0(3,1:size_start(end),end);

flxn_dy_calc = dis_calc400(2,1:size_flxn(end),end);
flxn_dy_actual = dis_actual_tool400(2,1:size_flxn(end),end);
flxn_dz_calc = dis_calc400(3,1:size_flxn(end),end);
flxn_dz_actual = dis_actual_tool400(3,1:size_flxn(end),end);

extn_dy_calc = dis_calc800(2,1:size_extn(end),end);
extn_dy_actual = dis_actual_tool800(2,1:size_extn(end),end);
extn_dz_calc = dis_calc800(3,1:size_extn(end),end);
extn_dz_actual = dis_actual_tool800(3,1:size_extn(end),end);

dy_calc = [flxn_dy_calc extn_dy_calc start_dy_calc];
dz_calc = [flxn_dz_calc extn_dz_calc start_dz_calc];
dy_actual = [flxn_dy_actual extn_dy_actual start_dy_actual];
dz_actual = [flxn_dz_actual extn_dz_actual start_dz_actual];

% Array of time for all iterations of last pathseek
time0 = time_total0(1,1:size_start(end),end);
time400 = time_total400(1,1:size_flxn(end),end);
time800 = time_total800(1,1:size_extn(end),end);
last_time = [time400 time800 time0];
last_time = cumsum(last_time);

% Array of rotation angles for all data points of last pathseek
```

```
last_rot_angle0 = rot_angle0(1,1:size_start(end),end);
last_rot_angle400 = rot_angle400(1,1:size_flxn(end),end);
last_rot_angle800 = rot_angle800(1,1:size_extn(end),end);
last_rot_angle = [last_rot_angle400 last_rot_angle800 last_rot_angle0];

% Array of rotation angles for end points of last pathseek
last_rot_angle_end_pts =
[rot_angle400_end_pts(:,1:size_flxn_end_pts(end),end),
rot_angle800_end_pts(:,1:size_extn_end_pts(end),end),
rot_angle0_end_pts(:,1:size_start_end_pts(end),end)];

% Array of rotation angles for end points of all pathseeks
% rot_angle_400_800_end_pts = [];
% for i = 1:flxn_counter
%     rot_angle_400_800_end_pts = [rot_angle_400_800_end_pts
rot_angle400_end_pts(:,1:size_flxn_end_pts(i),i)
rot_angle800_end_pts(:,1:size_extn_end_pts(i),i)];
% end
% all_rot_angle_end_pts = [rot_angle0_end_pts(:,1:size_start_end_pts(1),1),
rot_angle_400_800_end_pts,
rot_angle0_end_pts(:,1:size_start_end_pts(end),end)];
begin_rot_angle_end_pts =
[rot_angle0_end_pts(:,1:size_start_end_pts(1),1),...
        rot_angle400_end_pts(:,1:size_flxn_end_pts(1),1),...
        rot_angle800_end_pts(:,1:size_extn_end_pts(1),1)];
end_rot_angle_end_pts =
[rot_angle400_end_pts(:,1:size_flxn_end_pts(end),end),...
        rot_angle800_end_pts(:,1:size_extn_end_pts(end),end),...
        rot_angle0_end_pts(:,1:size_start_end_pts(end),end)];

% Arrays of constants
x = 1:length(last_rot_angle);
for i = 1:length(last_rot_angle)
    y(i) = 0;
end
for i = 1:length(fy_end_pts)
    y_end_pts(i) = 0;
end
% for i = 1:length(all_mx_end_pts)
%     all_y_end_pts(i) = 0;
% end

% Cumulative sum of iterations for each direction of last pathseek
test = 0;
size_z_ct0 = 0;
for i = 1:size(z_ct0_total,2)
    tf = isequal(z_ct0_total(1,i,end),test);
    if tf == 1
        size_z_ct0 = i-1;
        break
    end
end
if size_z_ct0 == 0
    size_z_ct0 = size(z_ct0_total,2);
end

size_z_ct400 = 0;
```

```
for i = 1:size(z_ct400_total,2)
    tf = isequal(z_ct400_total(1,i,end),test);
    if tf == 1
        size_z_ct400 = i-1;
        break
    end
    if size_z_ct400 == 0
        size_z_ct400 = size(z_ct400_total,2);
   end
end

size_z_ct800 = 0;
for i = 1:size(z_ct800_total,2)
    tf = isequal(z_ct800_total(1,i,end),test);
    if tf == 1
        size_z_ct800 = i-1;
        break
    end
end
if size_z_ct800 == 0
    size_z_ct800 = size(z_ct800_total,2);
end

z_ct0_sum = cumsum(z_ct0_total(1,1:size_z_ct0(end),end));
z_ct400_sum = cumsum(z_ct400_total(1,1:size_z_ct400(end),end));
z_ct800_sum = cumsum(z_ct800_total(1,1:size_z_ct800(end),end));

% Save workspace
save(filename)
disp('Data has been saved.')
%=========================================================================
====================================

% F/M PLOTS
% Plots of f/m vs. time for last pathseek
fh=figure('Position',[150 100 1000 900],'Color','w');
subplot(3,1,1), plot(last_time, fy, last_time, y, '-k'), title('Fy vs. time
for last pathseek'), xlabel('time (sec)'), ylabel('Fy (N)');
set(gca,'XLim',[0 length(last_time)]);
subplot(3,1,2), plot(last_time, fz, last_time, y, '-k'), title('Fz vs. time
for last pathseek'), xlabel('time (sec)'), ylabel('Fz (N)');
set(gca,'XLim',[0 length(last_time)]);
subplot(3,1,3), plot(last_time, mx, last_time, y, '-k'), title('Mx vs. time
for last pathseek'), xlabel('time (sec)'), ylabel('Mx (Nm)');
set(gca,'XLim',[0 length(last_time)]);

% Plots of rotation angle and fy vs. length(rot_angle) for last pathseek
fh=figure('Position',[150 100 1000 900],'Color','w');
subplot(3,1,1), plot(x, -last_rot_angle), title('rotation angle vs.
length(rotation angle) for last pathseek');
set(gca,'XLim',[0 x(end)]);
subplot(3,1,2), plot(x, fy, x, y, '-k'), title('Fy vs. length(Fy) for last
pathseek'), ylabel('Fy (N)');
set(gca,'XLim',[0 x(end)]);

% Plots of rotation angle and fz vs. length(rot_angle) for last pathseek
fh=figure('Position',[150 100 1000 900],'Color','w');
```

```
subplot(3,1,1), plot(x, -last_rot_angle), title('rotation angle vs.
length(rotation angle) for last pathseek');
set(gca,'XLim',[0 x(end)]);
subplot(3,1,2), plot(x, fz, x, y, '-k'), title('Fz vs. length(Fz) for last
pathseek'), ylabel('Fz (N)');
set(gca,'XLim',[0 x(end)]);

% Plots of rotation angle and mx vs. length(rot_angle) for last pathseek
fh=figure('Position',[150 100 1000 900],'Color','w');
subplot(3,1,1), plot(x, -last_rot_angle), title('rotation angle vs.
length(rotation angle) for last pathseek');
set(gca,'XLim',[0 x(end)]);
subplot(3,1,2), plot(x, mx, x, y, '-k'), title('Mx vs. length(Mx) for last
pathseek'), ylabel('Mx (Nm)');
set(gca,'XLim',[0 x(end)]);

% Plots of rotation angle vs. mx (end points from every pathseek)
fh=figure('Position',[150 100 1000 900],'Color','w');
set(gca, 'NextPlot', 'add');
plot(-begin_rot_angle_end_pts, begin_mx_end_pts, 's-r', 'MarkerEdgeColor',
'r', 'MarkerFaceColor', 'r', 'MarkerSize', 4);
for i = 2:flxn_counter-1
    plot(-rot_angle400_end_pts(:,1:size_flxn_end_pts(i),i),
flxn_load_end_pts(4,1:size_flxn_end_pts(i),i), '.-');
    plot(-rot_angle800_end_pts(:,1:size_extn_end_pts(i),i),
extn_load_end_pts(4,1:size_extn_end_pts(i),i), '*-');
end
plot(-end_rot_angle_end_pts, end_mx_end_pts, 'o-', 'Color', [0 0.75 0],
'MarkerEdgeColor', [0 0.75 0], 'MarkerFaceColor', [0 0.75 0], 'MarkerSize',
5);
line('XData', get(gca, 'XLim'), 'YData', [0 0]);
line('XData', [0 0], 'YData', get(gca, 'YLim'));
title('Mx vs. rotation angle for every pathseek'), xlabel('rotation angle
(deg)'), ylabel('Mx (Nm)');
legend_handles = get(gca, 'Children');
for i = 2:flxn_counter-1
    legend_string_flxn(i-1,1:15) = ['pathseek ', int2str(i), ' flxn'];
    legend_string_extn(i-1,1:15) = ['pathseek ', int2str(i), ' extn'];
end
legend_string_flxn_extn = [];
for i = 1:flxn_counter-2
    legend_string_flxn_extn = [legend_string_flxn_extn;
legend_string_flxn(i,:); legend_string_extn(i,:)];
end
legend_string = ['pathseek 1     '; legend_string_flxn_extn; ['pathseek ',
int2str(flxn_counter), '     ']];
legend(flipdim(legend_handles(3:end),1), legend_string, 2);
% text(1, -2, 'flexion', 'Rotation', 30, 'FontSize', 14);
% text(1, 2, 'extension', 'Rotation', 30, 'FontSize', 14);

% Plots of rotation angle vs. mx, fy & fz (end points of last pathseek)
fh=figure('Position',[150 100 1000 900],'Color','w');
subplot(3,1,1), plot(-last_rot_angle_end_pts, fy_end_pts, '.-b',
last_rot_angle_end_pts, y_end_pts, '-k'), title('Fy vs. rot angle for last
pathseek'), xlabel('rotation angle (deg)'), ylabel('Fy (N)');
```

```
subplot(3,1,2), plot(-last_rot_angle_end_pts, fz_end_pts, '.-b',
last_rot_angle_end_pts, y_end_pts, '-k'), title('Fz vs. rot angle for last
pathseek'), xlabel('rotation angle (deg)'), ylabel('Fz (N)');
subplot(3,1,3), plot(-last_rot_angle_end_pts, mx_end_pts, '.-b',
last_rot_angle_end_pts, y_end_pts, '-k'), title('Mx vs. rot angle for last
pathseek'), xlabel('rotation angle (deg)'), ylabel('Mx (Nm)');

% Plots of first and last points of each rotation angle for fy for last
pathseek
fh=figure('Position',[150 100 1000 900],'Color','w');
for i = 2:length(z_ct400_sum)
    hold on
    subplot(3,1,1), plot_handles_begin = plot(-
[rot_angle400(1,z_ct400_sum(1),end), rot_angle400(1,z_ct400_sum(1),end)],
[flxn_fy(1,1,end), flxn_fy(1,z_ct400_sum(1),end)], '-ob');
    subplot(3,1,1), plot(-[rot_angle400(1,z_ct400_sum(i),end),
rot_angle400(1,z_ct400_sum(i),end)], [flxn_fy(1,z_ct400_sum(i-1)+1,end),
flxn_fy(1,z_ct400_sum(i),end)], '-ob');
    subplot(3,1,1), plot_handles_end = plot(-
rot_angle400(1,z_ct400_sum(1),end), flxn_fy(1,z_ct400_sum(1),end),
'*r','MarkerSize',10);
    subplot(3,1,1), plot(-rot_angle400(1,z_ct400_sum(i),end),
flxn_fy(1,z_ct400_sum(i),end), '*r','MarkerSize',10);
    subplot(3,1,1), line('XData', get(gca, 'XLim'), 'YData', [0.5 0.5],
'LineWidth', 2);
    subplot(3,1,1), line('XData', get(gca, 'XLim'), 'YData', [-0.5 -0.5],
'LineWidth', 2);
    hold off
end
title('fy vs. rotation angle for last pathseek'), xlabel('rotation angle
(deg)'), ylabel('Fy (N)');
legend_handles = [plot_handles_begin; plot_handles_end];
legend(legend_handles, 'beginning force', 'ending force');
for i = 2:length(z_ct800_sum)
    hold on
    subplot(3,1,2), plot_handles_begin = plot(-
[rot_angle800(1,z_ct800_sum(1),end), rot_angle800(1,z_ct800_sum(1),end)],
[extn_fy(1,1,end), extn_fy(1,z_ct800_sum(1),end)], '-ob');
    subplot(3,1,2), plot(-[rot_angle800(1,z_ct800_sum(i),end),
rot_angle800(1,z_ct800_sum(i),end)], [extn_fy(1,z_ct800_sum(i-1)+1,end),
extn_fy(1,z_ct800_sum(i),end)], '-ob');
    subplot(3,1,2), plot_handles_end = plot(-
rot_angle800(1,z_ct800_sum(1),end), extn_fy(1,z_ct800_sum(1),end),
'*r','MarkerSize',10);
    subplot(3,1,2), plot(-rot_angle800(1,z_ct800_sum(i),end),
extn_fy(1,z_ct800_sum(i),end), '*r','MarkerSize',10);
    subplot(3,1,2), line('XData', get(gca, 'XLim'), 'YData', [0.5 0.5],
'LineWidth', 2);
    subplot(3,1,2), line('XData', get(gca, 'XLim'), 'YData', [-0.5 -0.5],
'LineWidth', 2);
    hold off
end
title('fy vs. rotation angle for last pathseek'), xlabel('rotation angle
(deg)'), ylabel('Fy (N)');
legend_handles = [plot_handles_begin; plot_handles_end];
legend(legend_handles, 'beginning force', 'ending force');
for i = 2:length(z_ct0_sum)
```

```
    hold on
    subplot(3,1,3), plot_handles_begin = plot(-
[rot_angle0(1,z_ct0_sum(1),end), rot_angle0(1,z_ct0_sum(1),end)],
[start_fy(1,1,end), start_fy(1,z_ct0_sum(1),end)], '-ob');
    subplot(3,1,3), plot(-[rot_angle0(1,z_ct0_sum(i),end),
rot_angle0(1,z_ct0_sum(i),end)], [start_fy(1,z_ct0_sum(i-1)+1,end),
start_fy(1,z_ct0_sum(i),end)], '-ob');
    subplot(3,1,3), plot_handles_end = plot(-rot_angle0(1,z_ct0_sum(1),end),
start_fy(1,z_ct0_sum(1),end), '*r', 'MarkerSize', 10);
    subplot(3,1,3), plot(-rot_angle0(1,z_ct0_sum(i),end),
start_fy(z_ct0_sum(i)), '*r', 'MarkerSize', 10);
    subplot(3,1,3), line('XData', get(gca, 'XLim'), 'YData', [0.5 0.5],
'LineWidth', 2);
    subplot(3,1,3), line('XData', get(gca, 'XLim'), 'YData', [-0.5 -0.5],
'LineWidth', 2);
    hold off
end
title('fy vs. rotation angle for last pathseek'), xlabel('rotation angle
(deg)'), ylabel('Fy (N)');
legend_handles = [plot_handles_begin; plot_handles_end];
legend(legend_handles, 'beginning force', 'ending force');

% Plots of first and last points of each rotation angle for fz for last
pathseek
fh=figure('Position',[150 100 1000 900],'Color','w');
for i = 2:length(z_ct400_sum)
    hold on
    subplot(3,1,1), plot_handles_begin = plot(-
[rot_angle400(1,z_ct400_sum(1),end), rot_angle400(1,z_ct400_sum(1),end)],
[flxn_fz(1,1,end), flxn_fz(1,z_ct400_sum(1),end)], '-ob');
    subplot(3,1,1), plot(-[rot_angle400(1,z_ct400_sum(i),end),
rot_angle400(1,z_ct400_sum(i),end)], [flxn_fz(1,z_ct400_sum(i-1)+1,end),
flxn_fz(1,z_ct400_sum(i),end)], '-ob');
    subplot(3,1,1), plot_handles_end = plot(-
rot_angle400(1,z_ct400_sum(1),end), flxn_fz(1,z_ct400_sum(1),end),
'*r','MarkerSize',10);
    subplot(3,1,1), plot(-rot_angle400(1,z_ct400_sum(i),end),
flxn_fz(1,z_ct400_sum(i),end), '*r','MarkerSize',10);
    subplot(3,1,1), line('XData', get(gca, 'XLim'), 'YData', [0.5 0.5],
'LineWidth', 2);
    subplot(3,1,1), line('XData', get(gca, 'XLim'), 'YData', [-0.5 -0.5],
'LineWidth', 2);
    hold off
end
title('fz vs. rotation angle for last pathseek'), xlabel('rotation angle
(deg)'), ylabel('Fz (N)');
legend_handles = [plot_handles_begin; plot_handles_end];
legend(legend_handles, 'beginning force', 'ending force');
for i = 2:length(z_ct800_sum)
    hold on
    subplot(3,1,2), plot_handles_begin = plot(-
[rot_angle800(1,z_ct800_sum(1),end), rot_angle800(1,z_ct800_sum(1),end)],
[extn_fz(1,1,end), extn_fz(1,z_ct800_sum(1),end)], '-ob');
    subplot(3,1,2), plot(-[rot_angle800(1,z_ct800_sum(i),end),
rot_angle800(1,z_ct800_sum(i),end)], [extn_fz(1,z_ct800_sum(i-1)+1,end),
extn_fz(1,z_ct800_sum(i),end)], '-ob');
```

```
    subplot(3,1,2), plot_handles_end = plot(-
rot_angle800(1,z_ct800_sum(1),end), extn_fz(1,z_ct800_sum(1),end),
'*r','MarkerSize',10);
    subplot(3,1,2), plot(-rot_angle800(1,z_ct800_sum(i),end),
extn_fz(1,z_ct800_sum(i),end), '*r','MarkerSize',10);
    subplot(3,1,2), line('XData', get(gca, 'XLim'), 'YData', [0.5 0.5],
'LineWidth', 2);
    subplot(3,1,2), line('XData', get(gca, 'XLim'), 'YData', [-0.5 -0.5],
'LineWidth', 2);
    hold off
end
title('fz vs. rotation angle for last pathseek'), xlabel('rotation angle
(deg)'), ylabel('Fz (N)');
legend_handles = [plot_handles_begin; plot_handles_end];
legend(legend_handles, 'beginning force', 'ending force');
for i = 2:length(z_ct0_sum)
    hold on
    subplot(3,1,3), plot_handles_begin = plot(-
[rot_angle0(1,z_ct0_sum(1),end), rot_angle0(1,z_ct0_sum(1),end)],
[start_fz(1,1,end), start_fz(1,z_ct0_sum(1),end)], '-ob');
    subplot(3,1,3), plot(-[rot_angle0(1,z_ct0_sum(i),end),
rot_angle0(1,z_ct0_sum(i),end)], [start_fz(1,z_ct0_sum(i-1)+1,end),
start_fz(1,z_ct0_sum(i),end)], '-ob');
    subplot(3,1,3), plot_handles_end = plot(-rot_angle0(1,z_ct0_sum(1),end),
start_fz(1,z_ct0_sum(1),end), '*r', 'MarkerSize', 10);
    subplot(3,1,3), plot(-rot_angle0(1,z_ct0_sum(i),end),
start_fz(z_ct0_sum(i)), '*r', 'MarkerSize', 10);
    subplot(3,1,3), line('XData', get(gca, 'XLim'), 'YData', [0.5 0.5],
'LineWidth', 2);
    subplot(3,1,3), line('XData', get(gca, 'XLim'), 'YData', [-0.5 -0.5],
'LineWidth', 2);
    hold off
end
title('fz vs. rotation angle for last pathseek'), xlabel('rotation angle
(deg)'), ylabel('Fz (N)');
legend_handles = [plot_handles_begin; plot_handles_end];
legend(legend_handles, 'beginning force', 'ending force');

% Plots of first and last points of each rotation angle for mx for last
pathseek
fh=figure('Position',[150 100 1000 900],'Color','w');
for i = 2:length(z_ct400_sum)
    hold on
    subplot(3,1,1), plot_handles_begin = plot(-
[rot_angle400(1,z_ct400_sum(1),end), rot_angle400(1,z_ct400_sum(1),end)],
[flxn_mx(1,1,end), flxn_mx(1,z_ct400_sum(1),end)], '-ob');
    subplot(3,1,1), plot(-[rot_angle400(1,z_ct400_sum(i),end),
rot_angle400(1,z_ct400_sum(i),end)], [flxn_mx(1,z_ct400_sum(i-1)+1,end),
flxn_mx(1,z_ct400_sum(i),end)], '-ob');
    subplot(3,1,1), plot_handles_end = plot(-
rot_angle400(1,z_ct400_sum(1),end), flxn_mx(1,z_ct400_sum(1),end),
'*r','MarkerSize',10);
    subplot(3,1,1), plot(-rot_angle400(1,z_ct400_sum(i),end),
flxn_mx(1,z_ct400_sum(i),end), '*r','MarkerSize',10);
    hold off
end
```

```
title('mx vs. rotation angle for last pathseek'), xlabel('rotation angle
(deg)'), ylabel('Mx (Nm)');
legend_handles = [plot_handles_begin; plot_handles_end];
legend(legend_handles, 'beginning moment', 'ending moment');
for i = 2:length(z_ct800_sum)
    hold on
    subplot(3,1,2), plot_handles_begin = plot(-
[rot_angle800(1,z_ct800_sum(1),end), rot_angle800(1,z_ct800_sum(1),end)],
[extn_mx(1,1,end), extn_mx(1,z_ct800_sum(1),end)], '-ob');
    subplot(3,1,2), plot(-[rot_angle800(1,z_ct800_sum(i),end),
rot_angle800(1,z_ct800_sum(i),end)], [extn_mx(1,z_ct800_sum(i-1)+1,end),
extn_mx(1,z_ct800_sum(i),end)], '-ob');
    subplot(3,1,2), plot_handles_end = plot(-
rot_angle800(1,z_ct800_sum(1),end), extn_mx(1,z_ct800_sum(1),end),
'*r','MarkerSize',10);
    subplot(3,1,2), plot(-rot_angle800(1,z_ct800_sum(i),end),
extn_mx(1,z_ct800_sum(i),end), '*r','MarkerSize',10);
    hold off
end
title('mx vs. rotation angle for last pathseek'), xlabel('rotation angle
(deg)'), ylabel('Mx (Nm)');
legend_handles = [plot_handles_begin; plot_handles_end];
legend(legend_handles, 'beginning moment', 'ending moment');
for i = 2:length(z_ct0_sum)
    hold on
    subplot(3,1,3), plot_handles_begin = plot(-
[rot_angle0(1,z_ct0_sum(1),end), rot_angle0(1,z_ct0_sum(1),end)],
[start_mx_2(1,1,end), start_mx_2(1,z_ct0_sum(1),end)], '-ob');
    subplot(3,1,3), plot(-[rot_angle0(1,z_ct0_sum(i),end),
rot_angle0(1,z_ct0_sum(i),end)], [start_mx_2(1,z_ct0_sum(i-1)+1,end),
start_mx_2(1,z_ct0_sum(i),end)], '-ob');
    subplot(3,1,3), plot_handles_end = plot(-rot_angle0(1,z_ct0_sum(1),end),
start_mx_2(1,z_ct0_sum(1),end), '*r', 'MarkerSize', 10);
    subplot(3,1,3), plot(-rot_angle0(1,z_ct0_sum(i),end),
start_mx_2(z_ct0_sum(i)), '*r', 'MarkerSize', 10);
    hold off
end
title('mx vs. rotation angle for last pathseek'), xlabel('rotation angle
(deg)'), ylabel('Mx (Nm)');
legend_handles = [plot_handles_begin; plot_handles_end];
legend(legend_handles, 'beginning moment', 'ending moment');
%=========================================================================
====================================

% PLOTS OF DISPLACEMENTS IN LOCAL Y AND Z DIR.
% Plots of dy/dz calc/actual vs. time for last pathseek
fh=figure('Position',[150 100 1000 900],'Color','w');
subplot(2,2,1), plot(last_time, dy_calc), title('dy calc vs. time for last
pathseek'), xlabel('time (sec)'), ylabel('dy (mm)');
subplot(2,2,3), plot(last_time, dz_calc), title('dz calc vs. time for last
pathseek'), xlabel('time (sec)'), ylabel('dz (mm)');
subplot(2,2,2), plot(last_time, dy_actual), title('dy actual vs. time for
last pathseek'), xlabel('time (sec)'), ylabel('dy (mm)');
subplot(2,2,4), plot(last_time, dz_actual), title('dz actual vs. time for
last pathseek'), xlabel('time (sec)'), ylabel('dz (mm)');
```

```
% Plots of rotation angle and dy calc/actual vs. length(rot_angle) for last
pathseek
fh=figure('Position',[150 100 1000 900],'Color','w');
subplot(3,1,1), plot(x, -last_rot_angle), title('rotation angle vs length(rot
angle) for last pathseek'), ylabel('rotation angle (deg)');
set(gca,'XLim',[0 length(last_rot_angle)]);
subplot(3,1,2), plot(x, dy_calc, 'b', x, y, 'k'), title('dy calc vs
length(dy) for last pathseek'), ylabel('dy (mm)');
set(gca,'XLim',[0 length(last_rot_angle)]);
subplot(3,1,3), plot(x, dy_actual, 'b', x, y, 'k'), title('dy actual vs
length(dy) for last pathseek'), ylabel('dy (mm)');
set(gca,'XLim',[0 length(last_rot_angle)]);

% Plots of rotation angle and dz calc/actual vs. length(rot_angle) for last
pathseek
fh=figure('Position',[150 100 1000 900],'Color','w');
subplot(3,1,1), plot(x, -last_rot_angle), title('rotation angle vs length(rot
angle) for last pathseek'), ylabel('rotation angle (deg)');
set(gca,'XLim',[0 length(last_rot_angle)]);
subplot(3,1,2), plot(x, dz_calc, 'b', x, y, 'k'), title('dz calc vs
length(dz) for last pathseek'), ylabel('dz (mm)');
set(gca,'XLim',[0 length(last_rot_angle)]);
subplot(3,1,3), plot(x, dz_actual, 'b', x, y, 'k'), title('dz actual vs
length(dz) for last pathseek'), ylabel('dz (mm)');
set(gca,'XLim',[0 length(last_rot_angle)]);

% Plots of dy/dz calc vs. dy/dz actual for last pathseek
fh=figure('Position',[150 100 1000 900],'Color','w');
subplot(2,1,1), plot(dy_calc, dy_actual), title('dy actual vs. dy calc for
last pathseek'), xlabel('dy (mm)'), ylabel('dy (mm)');
subplot(2,1,2), plot(dz_calc, dz_actual), title('dz actual vs. dz calc for
last pathseek'), xlabel('dz (mm)'), ylabel('dz (mm)');
%=================================================================================
====================================

% PLOTS OF RX, TY, TZ VS. MX (END POINTS FROM LAST PATHSEEK)
% Plots of Rx, Ty, Tz vs. mx (end points from last pathseek)
fh=figure('Position',[150 100 1000 900],'Color','w');
hold on
for i = 1:size(z_ct0_sum,2)
    plot(-last_rot_angle_end_pts, mx_end_pts, '.-b',...
        -last_rot_angle_end_pts(i), start_dy_actual(z_ct0_sum(i)),'.b',...
        -last_rot_angle_end_pts(i), start_dz_actual(z_ct0_sum(i)),'*r');
end
for i = 1:size(z_ct400_sum,2)
    plot(-last_rot_angle_end_pts, mx_end_pts, '.-b',...
        -last_rot_angle_end_pts(i), flxn_dy_actual(z_ct400_sum(i)),'.b',...
        -last_rot_angle_end_pts(i), flxn_dz_actual(z_ct400_sum(i)),'*r');
end
for i = 1:size(z_ct800_sum,2)
    plot_handles = plot(-last_rot_angle_end_pts, mx_end_pts, '.-b',...
        -last_rot_angle_end_pts(i), extn_dy_actual(z_ct800_sum(i)),'.b',...
        -last_rot_angle_end_pts(i), extn_dz_actual(z_ct800_sum(i)),'*r');
end
line('XData', get(gca, 'XLim'), 'YData', [0 0]);
line('XData', [0 0], 'YData', get(gca, 'YLim'));
hold off
```

```
title('Rx, Ty, Tz vs. Mx for last pathseek'), ylabel('displacement (deg or
mm)'), xlabel('Mx (Nm)');
legend(plot_handles, 'Mx', 'Ty', 'Tz');
%=========================================================================
====================================

% PLOT OF MOVEMENT OF POINT OF INTEREST IN GLOBAL C.S.
% Z vs. X in global c.s. (start, extn & flxn together, starting & ending
points only) for last pathseek
fh=figure('Position',[150 100 1000 900],'Color','w');
temp = 0;
hold on
for i = 1:size(z_ct0_sum,2)
    plot_handles_3 = plot([tGPOI0_posn(1,temp+1),
tGPOI0_posn(1,temp+1+z_ct0_total(1,i,end))],...
        [tGPOI0_posn(3,temp+1), tGPOI0_posn(3,temp+1+z_ct0_total(1,i,end))],
'-ob');
    temp = temp + 1 + z_ct0_total(1,i,end);
end
temp = 0;
for i = 1:size(z_ct400_sum,2)
    plot_handles_1 = plot([tGPOI400_posn(1,temp+1),
tGPOI400_posn(1,temp+1+z_ct400_total(1,i,end))],...
        [tGPOI400_posn(3,temp+1),
tGPOI400_posn(3,temp+1+z_ct400_total(1,i,end))], '-or');
    temp = temp + 1 + z_ct400_total(1,i,end);
end
temp = 0;
for i = 1:size(z_ct800_sum,2)
    plot_handles_2 = plot([tGPOI800_posn(1,temp+1),
tGPOI800_posn(1,temp+1+z_ct800_total(1,i,end))],...
        [tGPOI800_posn(3,temp+1),
tGPOI800_posn(3,temp+1+z_ct800_total(1,i,end))], '-ok');
    temp = temp + 1 + z_ct800_total(1,i,end);
end
hold off
title('Z vs. X for point of interest for last pathseek'), xlabel('X (mm)'),
ylabel('Z (mm)');
legend_handles = [plot_handles_1; plot_handles_2; plot_handles_3];
legend(legend_handles, 'flxn \rightarrow extn', 'extn \rightarrow flxn',
'flxn \rightarrow start');
%=========================================================================
====================================

% PLOT OF MOVEMENT OF UFS IN GLOBAL C.S.
% Z vs. X in global c.s. (start, extn & flxn together, starting & ending
points only) for last pathseek
fh=figure('Position',[150 100 1000 900],'Color','w');
temp = 0;
hold on
for i = 1:size(z_ct0_sum,2)
    plot_handles_3 = plot([tGUFS0_posn(1,temp+1),
tGUFS0_posn(1,temp+1+z_ct0_total(1,i,end))],...
        [tGUFS0_posn(3,temp+1), tGUFS0_posn(3,temp+1+z_ct0_total(1,i,end))],
'-ob');
    temp = temp + 1 + z_ct0_total(1,i,end);
end
```

```
temp = 0;
for i = 1:size(z_ct400_sum,2)
    plot_handles_1 = plot([tGUFS400_posn(1,temp+1),
tGUFS400_posn(1,temp+1+z_ct400_total(1,i,end))],...
        [tGUFS400_posn(3,temp+1),
tGUFS400_posn(3,temp+1+z_ct400_total(1,i,end))], '-or');
    temp = temp + 1 + z_ct400_total(1,i,end);
end
temp = 0;
for i = 1:size(z_ct800_sum,2)
    plot_handles_2 = plot([tGUFS800_posn(1,temp+1),
tGUFS800_posn(1,temp+1+z_ct800_total(1,i,end))],...
        [tGUFS800_posn(3,temp+1),
tGUFS800_posn(3,temp+1+z_ct800_total(1,i,end))], '-ok');
    temp = temp + 1 + z_ct800_total(1,i,end);
end
hold off
title('Z vs. X for UFS for last pathseek'), xlabel('X (mm)'), ylabel('Z
(mm)');
legend_handles = [plot_handles_1; plot_handles_2; plot_handles_3];
legend(legend_handles, 'flxn \rightarrow extn', 'extn \rightarrow flxn',
'flxn \rightarrow start');
%=========================================================================
====================================

% PLOT OF MOVEMENT OF COR IN GLOBAL C.S.
% Z vs. X in global c.s. (start, extn & flxn together, starting & ending
points only) for last pathseek
fh=figure('Position',[150 100 1000 900],'Color','w');
temp = 0;
hold on
for i = 1:size(z_ct0_sum,2)
    plot_handles_3 = plot([tGT0_posn(1,temp+1),
tGT0_posn(1,temp+1+z_ct0_total(1,i,end))],...
        [tGT0_posn(3,temp+1), tGT0_posn(3,temp+1+z_ct0_total(1,i,end))], '-
ob');
    temp = temp + 1 + z_ct0_total(1,i,end);
end
temp = 0;
for i = 1:size(z_ct400_sum,2)
    plot_handles_1 = plot([tGT400_posn(1,temp+1),
tGT400_posn(1,temp+1+z_ct400_total(1,i,end))],...
        [tGT400_posn(3,temp+1),
tGT400_posn(3,temp+1+z_ct400_total(1,i,end))], '-or');
    temp = temp + 1 + z_ct400_total(1,i,end);
end
temp = 0;
for i = 1:size(z_ct800_sum,2)
    plot_handles_2 = plot([tGT800_posn(1,temp+1),
tGT800_posn(1,temp+1+z_ct800_total(1,i,end))],...
        [tGT800_posn(3,temp+1),
tGT800_posn(3,temp+1+z_ct800_total(1,i,end))], '-ok');
    temp = temp + 1 + z_ct800_total(1,i,end);
end
hold off
title('Z vs. X for COR for last pathseek'), xlabel('X (mm)'), ylabel('Z
(mm)');
```

```
legend_handles = [plot_handles_1; plot_handles_2; plot_handles_3];
legend(legend_handles, 'flxn \rightarrow extn', 'extn \rightarrow flxn',
'flxn \rightarrow start');
%=========================================================================
====================================

% PLOT OF MOVEMENT OF pt. of interest, UFS & COR IN GLOBAL C.S.
% Z vs. X in global c.s. (start, extn & flxn together, starting & ending
points only) for last pathseek
fh=figure('Position',[150 100 1000 900],'Color','w');
temp = 0;
hold on
for i = 1:size(z_ct0_sum,2)
    plot_handles_3 = plot([tGT0_posn(1,temp+1),
tGT0_posn(1,temp+1+z_ct0_total(1,i,end))],...
        [tGT0_posn(3,temp+1), tGT0_posn(3,temp+1+z_ct0_total(1,i,end))], '.-
b',...
        [tGUFS0_posn(1,temp+1),
tGUFS0_posn(1,temp+1+z_ct0_total(1,i,end))],...
        [tGUFS0_posn(3,temp+1), tGUFS0_posn(3,temp+1+z_ct0_total(1,i,end))],
'-ob',...
        [tGPOI0_posn(1,temp+1),
tGPOI0_posn(1,temp+1+z_ct0_total(1,i,end))],...
        [tGPOI0_posn(3,temp+1), tGPOI0_posn(3,temp+1+z_ct0_total(1,i,end))],
'-sb', 'MarkerSize', 5);
    temp = temp + 1 + z_ct0_total(1,i,end);
end
temp = 0;
for i = 1:size(z_ct400_sum,2)
    plot_handles_1 = plot([tGT400_posn(1,temp+1),
tGT400_posn(1,temp+1+z_ct400_total(1,i,end))],...
        [tGT400_posn(3,temp+1),
tGT400_posn(3,temp+1+z_ct400_total(1,i,end))], '.-b',...
        [tGUFS400_posn(1,temp+1),
tGUFS400_posn(1,temp+1+z_ct400_total(1,i,end))],...
        [tGUFS400_posn(3,temp+1),
tGUFS400_posn(3,temp+1+z_ct400_total(1,i,end))], '-or',...
        [tGPOI400_posn(1,temp+1),
tGPOI400_posn(1,temp+1+z_ct400_total(1,i,end))],...
        [tGPOI400_posn(3,temp+1),
tGPOI400_posn(3,temp+1+z_ct400_total(1,i,end))], '-sr', 'MarkerSize', 5);
    temp = temp + 1 + z_ct400_total(1,i,end);
end
temp = 0;
for i = 1:size(z_ct800_sum,2)
    plot_handles_2 = plot([tGT800_posn(1,temp+1),
tGT800_posn(1,temp+1+z_ct800_total(1,i,end))],...
        [tGT800_posn(3,temp+1),
tGT800_posn(3,temp+1+z_ct800_total(1,i,end))], '.-b',...
        [tGUFS800_posn(1,temp+1),
tGUFS800_posn(1,temp+1+z_ct800_total(1,i,end))],...
        [tGUFS800_posn(3,temp+1),
tGUFS800_posn(3,temp+1+z_ct800_total(1,i,end))], '-ok',...
        [tGPOI800_posn(1,temp+1),
tGPOI800_posn(1,temp+1+z_ct800_total(1,i,end))],...
        [tGPOI800_posn(3,temp+1),
tGPOI800_posn(3,temp+1+z_ct800_total(1,i,end))], '-sk', 'MarkerSize', 5);
```

```
    temp = temp + 1 + z_ct800_total(1,i,end);
end
hold off
title('Z vs. X for point of interest, UFS & COR for last pathseek'),
xlabel('X (mm)'), ylabel('Z (mm)');
legend_handles = [plot_handles_1; plot_handles_2; plot_handles_3];
legend(legend_handles, 'COR: flxn \rightarrow extn ', 'UFS: flxn \rightarrow
extn', 'POI: flxn \rightarrow extn',...
    'COR: extn \rightarrow flxn', 'UFS: extn \rightarrow flxn', 'POI: extn
\rightarrow flxn',...
    'COR: flxn \rightarrow start', 'UFS: flxn \rightarrow start', 'POI: flxn
\rightarrow start', 0);
```

After finding the final passive path of the specimen, it is replayed to make sure no more pre-conditioning needs to be done (**spine3h_val_path2.m**).

```
% spine3h_val_path2.m
% replay flexion/extension
% converted from spine3h.v2
% Amy Loveless
% 7/31/2002

% Disable buttons on GUI until spine3h_val_path2.m is done running
buttons(guihandles, 'off');

% Input dialog box to get the number of times to run replay
prompt  = {'Enter the number of times you want to run the replay'};
title   = 'Number of Replays';
lines= 1;
def     = {''};
answer  = inputdlg(prompt,title,lines,def);
if isequal(answer,{}) == 1
    % Enable buttons on GUI
    buttons(guihandles, 'on');
else
    plays = str2num(answer{1});
end

% plays = str2num(answer{1});

% Clear variables created for inputdlg
clear prompt title lines def answer;

% Input dialog box to get the filename for data storage
prompt  = {'Enter Filename'};
title   = 'Filename';
lines= 1;
def     = {'c:\robot'};
answer  = inputdlg(prompt,title,lines,def);
if isequal(answer,{}) == 1
    % Enable buttons on GUI
    buttons(guihandles, 'on');
else
```

```
    filename = answer{1};
end

% Clear variables created for inputdlg
clear prompt title lines def answer;

% setup figure to graphically monitor loads
[fx, fy, fz, mx, my, mz, handles, fh] = val_path_display1;

% Arrays of constants
rot_angle0_replay =
flipdim(rot_angle0_end_pts(:,1:size_start_end_pts(end),end),1);
rot_angle400_replay = rot_angle400_end_pts(:,1:size_flxn_end_pts(end),end);
rot_angle800_replay = rot_angle800_end_pts(:,1:size_extn_end_pts(end),end);
rot_angle_replay = [rot_angle400_replay rot_angle800_replay];

% move specimen to flxn in incremental movements
for p = 1:size(start_replay1,2)
    ok = 0;
   flag = 3.1;
   fprintf(port1, [ok, flag]);
   fprintf(port1, start_replay1(1:6,p));

    done_moving = fscanf(port1);
    done_moving = sscanf(done_moving, '%f');

    %=========================================
   get_loads;  % measure: forces and moments
   %=========================================

   %=========================================
   fm_tare5;  % tare out bolt-up and fixture wt
   %=========================================

   % display f/m after taring out bolt-up and fixture wt
    val_path_display2([fm_tcs, fx, fy, fz], [mx, my, mz], handles,
[rot_angle0_replay(p), z_target(2)]);
%     for i = 1:3
%         if abs(fa(i)) > 0.5
%             line_color(i,1:3) = [1 0 0];
%         else
%             line_color(i,1:3) = [0 0.75 0];
%         end
%  end
%  subplot(2,1,1), set(fx, 'XData', [0 fa(1)], 'Color', line_color(1,:));
%  subplot(2,1,1), set(fy, 'XData', [0 fa(2)], 'Color', line_color(2,:));
%  subplot(2,1,1), set(fz, 'XData', [0 fa(3)], 'Color', line_color(3,:));
%  subplot(2,1,2), set(mx, 'XData', [0 fa(4)]);
%  subplot(2,1,2), set(my, 'XData', [0 fa(5)]);
%  subplot(2,1,2), set(mz, 'XData', [0 fa(6)]);
%     set(handles.w_now_edit, 'String', rot_angle0_replay(p));
%
%     drawnow

end

for j = 1:plays
```

```
    % Read position and load data for dir = 1200 (flxn -> extn, replay)
    for p = 1:size(flxn_replay,2)
        ok = 0;
      flag = 3.1;
      fprintf(port1, [ok, flag]);
      fprintf(port1, flxn_replay(1:6,p));

        done_moving = fscanf(port1);
        done_moving = sscanf(done_moving, '%f');

        ok = 0;
        flag = 1.1;
        fprintf(port1, [ok flag]);
        flxn1 = fscanf(port1);
        flxn1 = sscanf(flxn1, '%f');
        z_gt1200_val(1:6,p,j) = flxn1(1:6);
        flxn_val_jt_angles(1:6,p,j) = flxn1(7:12);

      %=========================================
      get_loads;  % measure: forces and moments
      %=========================================

      %=========================================
      fm_tare5;  % tare out bolt-up and fixture wt
      %=========================================

      % display f/m after taring out bolt-up and fixture wt
        val_path_display2([fm_tcs, fx, fy, fz], [mx, my, mz], handles,
[rot_angle400_replay(p), z_target(2)]);
%     for i = 1:3
%             if abs(fa(i)) > 0.5
%                 line_color(i,1:3) = [1 0 0];
%             else
%                 line_color(i,1:3) = [0 0.75 0];
%             end
%     end
%     subplot(2,1,1), set(fx, 'XData', [0 fa(1)], 'Color', line_color(1,:));
%     subplot(2,1,1), set(fy, 'XData', [0 fa(2)], 'Color', line_color(2,:));
%     subplot(2,1,1), set(fz, 'XData', [0 fa(3)], 'Color', line_color(3,:));
%     subplot(2,1,2), set(mx, 'XData', [0 fa(4)]);
%     subplot(2,1,2), set(my, 'XData', [0 fa(5)]);
%     subplot(2,1,2), set(mz, 'XData', [0 fa(6)]);
%         set(handles.w_now_edit, 'String', rot_angle400_replay(p));
%         set(handles.valpath_edit, 'String', j);
%
%     drawnow

        load1200_val(1:6,p,j) = fa';
   end

    % Read position and load data for dir = 1600 (extn -> flxn, replay)
   for p = 1:size(extn_replay,2)
        ok = 0;
      flag = 3.1;
      fprintf(port1, [ok, flag]);
      fprintf(port1, extn_replay(1:6,p));
```

```
        done_moving = fscanf(port1);
        done_moving = sscanf(done_moving, '%f');

        ok = 0;
        flag = 1.1;
        fprintf(port1, [ok flag]);
        extn1 = fscanf(port1);
        extn1 = sscanf(extn1, '%f');
        z_gt1600_val(1:6,p,j) = extn1(1:6);
        extn_val_jt_angles(1:6,p,j) = extn1(7:12);

      %=========================================
      get_loads;  % measure: forces and moments
      %=========================================

      %=========================================
      fm_tare5;  % tare out bolt-up and fixture wt
      %=========================================

      % display f/m after taring out bolt-up and fixture wt
        val_path_display2([fm_tcs, fx, fy, fz], [mx, my, mz], handles,
[rot_angle800_replay(p), z_target(2)]);
%         for i = 1:3
%             if abs(fa(i)) > 0.5
%                 line_color(i,1:3) = [1 0 0];
%             else
%                 line_color(i,1:3) = [0 0.75 0];
%             end
%     end
%     subplot(2,1,1), set(fx, 'XData', [0 fa(1)], 'Color', line_color(1,:));
%     subplot(2,1,1), set(fy, 'XData', [0 fa(2)], 'Color', line_color(2,:));
%     subplot(2,1,1), set(fz, 'XData', [0 fa(3)], 'Color', line_color(3,:));
%     subplot(2,1,2), set(mx, 'XData', [0 fa(4)]);
%     subplot(2,1,2), set(my, 'XData', [0 fa(5)]);
%     subplot(2,1,2), set(mz, 'XData', [0 fa(6)]);
%         set(handles.w_now_edit, 'String', rot_angle800_replay(p));
%         set(handles.valpath_edit, 'String', j);
%
%     drawnow

        load1600_val(1:6,p,j) = fa';
   end

end

% move specimen back to rotation angle = 0 in incremental movements
for p = 1:size(start_replay2,2)
    ok = 0;
   flag = 3.1;
   fprintf(port1, [ok, flag]);
   fprintf(port1, start_replay2(1:6,p));

    done_moving = fscanf(port1);
    done_moving = sscanf(done_moving, '%f');

   %=========================================
```

```
    get_loads;  % measure: forces and moments
    %=========================================

    %=========================================
    fm_tare5;  % tare out bolt-up and fixture wt
    %=========================================

    % display f/m after taring out bolt-up and fixture wt
     val_path_display2([fm_tcs, fx, fy, fz], [mx, my, mz], handles,
[rot_angle400_replay(p), z_target(2)]);
%     for i = 1:3
%         if abs(fa(i)) > 0.5
%             line_color(i,1:3) = [1 0 0];
%         else
%             line_color(i,1:3) = [0 0.75 0];
%         end
%  end
%  subplot(2,1,1), set(fx, 'XData', [0 fa(1)], 'Color', line_color(1,:));
%  subplot(2,1,1), set(fy, 'XData', [0 fa(2)], 'Color', line_color(2,:));
%  subplot(2,1,1), set(fz, 'XData', [0 fa(3)], 'Color', line_color(3,:));
%  subplot(2,1,2), set(mx, 'XData', [0 fa(4)]);
%  subplot(2,1,2), set(my, 'XData', [0 fa(5)]);
%  subplot(2,1,2), set(mz, 'XData', [0 fa(6)]);
%     set(handles.w_now_edit, 'String', rot_angle400_replay(p));
%
%  drawnow
end

% remove monitor loads figure from screen
delete(fh);

%=========================================
data_display_val_path2; % display data
%=========================================

% Enable buttons on GUI when spine3h_val_path2.m is done running
buttons(guihandles, 'on');
```

val_path_display1.m is a function called by spine3h_val_path2.m that sets up the plot to graphically monitor UFS loads.

```
function [fx, fy, fz, mx, my, mz, handles, fh] = val_path_display1;

% setup figure to graphically monitor loads
fh = figure('Position',[400 300 600 600],'Color','w');
subplot(2,1,1)
set(gca,'XLim', [-50 50], 'YLim', [0 4], 'YTick', [1 2 3], 'YTickLabel', 'Fz
(N)|Fy (N)|Fx (N)')
title('Forces')
fx = line('XData', [0 0], 'YData', [3 3], 'LineWidth', 24, 'Color', [0 0.75
0]);
fy = line('XData', [0 0], 'YData', [2 2], 'LineWidth', 24, 'Color', [0 0.75
0]);
```

```
fz = line('XData', [0 0], 'YData', [1 1], 'LineWidth', 24, 'Color', [0 0.75
0]);
origin = line('XData', [0 0], 'YData', [0 4]);

subplot(2,1,2)
set(gca,'XLim', [-10 10], 'YLim', [0 4], 'YTick', [1 2 3], 'YTickLabel', 'Mz
(Nm)|My (Nm)|Mx (Nm)')
title('Moments')
mx = line('XData', [0 0], 'YData', [3 3], 'LineWidth', 24, 'Color', [0 0.75
0]);
my = line('XData', [0 0], 'YData', [2 2], 'LineWidth', 24, 'Color', [0 0.75
0]);
mz = line('XData', [0 0], 'YData', [1 1], 'LineWidth', 24, 'Color', [0 0.75
0]);
origin = line('XData', [0 0], 'YData', [0 4]);

uicontrol('Style', 'text', 'Tag', 'current_text',...
    'Position', [20 0 60 20], 'BackgroundColor', [1 1 1], 'FontSize', 12,
'String', 'Current:');
uicontrol('Style', 'edit', 'Tag', 'w_now_edit',...
    'Position', [135 20 60 20], 'BackgroundColor', [1 1 1], 'FontSize', 12);
uicontrol('Style', 'text', 'Tag', 'w_now_text',...
    'Position', [135 0 60 20], 'BackgroundColor', [1 1 1], 'FontSize', 12,
'String', 'Angle');
uicontrol('Style', 'edit', 'Tag', 'valpath_edit',...
    'Position', [335 20 60 20], 'BackgroundColor', [1 1 1], 'FontSize', 12);
uicontrol('Style', 'text', 'Tag', 'valpath_text',...
    'Position', [335 0 70 20], 'BackgroundColor', [1 1 1], 'FontSize', 12,
'String', 'Replay #');
handles = guihandles(fh);
guidata(fh, handles);

% any of these changes should make simple animations smooth
% zbuffer can be very slow and on my computer none of these are
% necessary to stop flashing
set(fh,'doublebuffer','on');
% set(fh,'renderer','zbuffer');
% set(hfig,'renderer','opengl');
```

val_path_display2.m is a function called by spine3h_val_path2.m that plots UFS loads.

```
function val_path_display2(forces, moments, handles, misc)

fa = forces(1:6);
fx = forces(7);
fy = forces(8);
fz = forces(9);

mx = moments(1);
my = moments(2);
mz = moments(3);

rot_angle_replay = misc(1);
limit = misc(2);
```

```
for i = 1:3
    if abs(fa(i)) > limit
        line_color(i,1:3) = [1 0 0];
    else
        line_color(i,1:3) = [0 0.75 0];
    end
end
subplot(2,1,1), set(fx, 'XData', [0 fa(1)], 'Color', line_color(1,:));
subplot(2,1,1), set(fy, 'XData', [0 fa(2)], 'Color', line_color(2,:));
subplot(2,1,1), set(fz, 'XData', [0 fa(3)], 'Color', line_color(3,:));
subplot(2,1,2), set(mx, 'XData', [0 fa(4)]);
subplot(2,1,2), set(my, 'XData', [0 fa(5)]);
subplot(2,1,2), set(mz, 'XData', [0 fa(6)]);
set(handles.w_now_edit, 'String', rot_angle_replay);

drawnow
```

data_display_val_path2.m is a script called by spine3h_val_path2.m that plots the data gathered during pathseek validation.

```
% data_display_val_path2.m
% display data
% Amy Loveless
% 7/31/02

% BUILD ARRAYS TO BE USED FOR PLOTTING
% Arrays of mx
for i = 1:plays
    mx1200_val(1,1:size(load1200_val,2),i) =
load1200_val(4,1:size(load1200_val,2),i);
    mx1600_val(1,1:size(load1600_val,2),i) =
load1600_val(4,1:size(load1600_val,2),i);
end

mx_end_pts = [flxn_mx_end_pts extn_mx_end_pts];

% Arrays of constants
rot_angle400_val = rot_angle400_end_pts(:,1:size_flxn_end_pts(end),end);
rot_angle800_val = rot_angle800_end_pts(:,1:size_extn_end_pts(end),end);
rot_angle_val = [rot_angle400_val rot_angle800_val];

for i = 1:length(rot_angle400_val)
    y_replay2(i) = 0;
end

clear legend_string
for i = 1:plays
    legend_string(i,1:8) = ['replay ', int2str(i)];
end

% Save workspace
save(filename)
disp('Data has been saved.')
```

```
%=========================================================================
==================================

% Plot of rotation angle vs. mx for passive pathseek (mx_end_pts) & first
replay (mx1200_val, mx1600_val)
fh=figure('Position',[150 100 1000 900],'Color','w');
set(gca, 'NextPlot', 'add');
plot(-rot_angle_val, mx_end_pts, '.-', -rot_angle_val, [mx1200_val(:,:,1)
mx1600_val(:,:,1)], '-o');
line('XData', get(gca, 'XLim'), 'YData', [0 0]);
line('XData', [0 0], 'YData', get(gca, 'YLim'));
title('Mx vs. rotation angle'), xlabel('rotation angle (deg)'), ylabel('Mx
(Nm)');
legend_handles = get(gca, 'Children');
legend(flipdim(legend_handles(3:4),1), 'last pathseek', 'intact replay', 2);
text(1, -2, 'flexion', 'Rotation', 30, 'FontSize', 14);
text(1, 2, 'extension', 'Rotation', 30, 'FontSize', 14);

% Plots of mx vs. rotation angle (for all replays, flxn -> extn)
fh=figure('Position',[150 100 1000 900],'Color','w');
set(gca, 'NextPlot', 'add');
line_color = get(gca, 'ColorOrder');
for i = 1:plays
    plot(-rot_angle_val, [mx1200_val(:,:,i) mx1600_val(:,:,i)], '.-',
'Color', line_color(i,:));
end
line('XData', get(gca, 'XLim'), 'YData', [0 0]);
line('XData', [0 0], 'YData', get(gca, 'YLim'));
title('Mx vs. rotation angle'), xlabel('rotation angle (deg)'), ylabel('Mx
(Nm)');
legend_handles = get(gca, 'Children');
legend(flipdim(legend_handles(3:plays+2),1), legend_string, 2);
text(1, -2, 'flexion', 'Rotation', 30, 'FontSize', 14);
text(1, 2, 'extension', 'Rotation', 30, 'FontSize', 14);
```

After the passive path has been validated, the cutting study begins using

spine3h_replay2.m.

```
% spine3h_replay2.m
% replay flexion/extension
% converted from spine3h.v2
% Amy Loveless
% 7/4/2002

% Disable buttons on GUI until spine3h_replay.m is done running
set(hok, 'Enable', 'off');
set(hbolt, 'Enable', 'off');
set(hbefore, 'Enable', 'off');
set(hafter, 'Enable', 'off');
set(hpath, 'Enable', 'off');
set(hval, 'Enable', 'off');
set(hreplay, 'Enable', 'off');
```

```
set(hend, 'Enable', 'off');

% Input dialog box to get the filename for data storage
prompt  = {'Enter Filename'};
title   = 'Filename';
lines= 1;
def     = {'c:\robot'};
answer  = inputdlg(prompt,title,lines,def);
filename = answer{1};

% Clear variables created for inputdlg
clear prompt title lines def answer;

% setup figure to graphically monitor loads
fh = figure('Position',[400 300 600 600],'Color','w');
subplot(2,1,1)
set(gca,'XLim', [-50 50], 'YLim', [0 4], 'YTick', [1 2 3], 'YTickLabel', 'Fz
(N)|Fy (N)|Fx (N)')
title('Forces')
fx = line('XData', [0 0], 'YData', [3 3], 'LineWidth', 24, 'Color', [0 0.75
0]);
fy = line('XData', [0 0], 'YData', [2 2], 'LineWidth', 24, 'Color', [0 0.75
0]);
fz = line('XData', [0 0], 'YData', [1 1], 'LineWidth', 24, 'Color', [0 0.75
0]);
origin = line('XData', [0 0], 'YData', [0 4]);

subplot(2,1,2)
set(gca,'XLim', [-10 10], 'YLim', [0 4], 'YTick', [1 2 3], 'YTickLabel', 'Mz
(Nm)|My (Nm)|Mx (Nm)')
title('Moments')
mx = line('XData', [0 0], 'YData', [3 3], 'LineWidth', 24, 'Color', [0 0.75
0]);
my = line('XData', [0 0], 'YData', [2 2], 'LineWidth', 24, 'Color', [0 0.75
0]);
mz = line('XData', [0 0], 'YData', [1 1], 'LineWidth', 24, 'Color', [0 0.75
0]);
origin = line('XData', [0 0], 'YData', [0 4]);

uicontrol('Style', 'text', 'Tag', 'current_text',...
    'Position', [20 0 60 20], 'BackgroundColor', [1 1 1], 'FontSize', 12,
'String', 'Current:');
uicontrol('Style', 'edit', 'Tag', 'w_now_edit',...
    'Position', [135 20 60 20], 'BackgroundColor', [1 1 1], 'FontSize', 12);
uicontrol('Style', 'text', 'Tag', 'w_now_text',...
    'Position', [135 0 60 20], 'BackgroundColor', [1 1 1], 'FontSize', 12,
'String', 'Angle');
handles = guihandles(fh);
guidata(fh, handles);

% any of these changes should make simple animations smooth
% zbuffer can be very slow and on my computer none of these are
% necessary to stop flashing
set(fh,'doublebuffer','on');
% set(fh,'renderer','zbuffer');
% set(hfig,'renderer','opengl');
```

```
% Arrays of constants
rot_angle0_replay =
flipdim(rot_angle0_end_pts(:,1:size_start_end_pts(end),end),1);
rot_angle400_replay = rot_angle400_end_pts(:,1:size_flxn_end_pts(end),end);
rot_angle800_replay = rot_angle800_end_pts(:,1:size_extn_end_pts(end),end);
rot_angle_replay = [rot_angle400_replay rot_angle800_replay];

% move specimen to flxn in incremental movements
for p = 1:size(start_replay1,2)
    ok = 0;
   flag = 3.1;
   fprintf(port1, [ok, flag]);
   fprintf(port1, start_replay1(1:6,p));

    done_moving = fscanf(port1);
    done_moving = sscanf(done_moving, '%f');

    %=========================================
   get_loads;  % measure: forces and moments
   %=========================================

   %=========================================
   fm_tare5;  % tare out bolt-up and fixture wt
   %=========================================

   % display f/m after taring out bolt-up and fixture wt
    for i = 1:3
        if abs(fa(i)) > 0.5
            line_color(i,1:3) = [1 0 0];
        else
            line_color(i,1:3) = [0 0.75 0];
        end
   end
   subplot(2,1,1), set(fx, 'XData', [0 fa(1)], 'Color', line_color(1,:));
   subplot(2,1,1), set(fy, 'XData', [0 fa(2)], 'Color', line_color(2,:));
   subplot(2,1,1), set(fz, 'XData', [0 fa(3)], 'Color', line_color(3,:));
   subplot(2,1,2), set(mx, 'XData', [0 fa(4)]);
   subplot(2,1,2), set(my, 'XData', [0 fa(5)]);
   subplot(2,1,2), set(mz, 'XData', [0 fa(6)]);
    set(handles.w_now_edit, 'String', rot_angle0_replay(p));

    drawnow

end

% Read position and load data for dir = 1200 (flxn -> extn, replay)
% cuts = page number of matrix
% cuts = cuts + 1 is for use with Matlab interface only
cuts = cuts + 1;

for p = 1:size(flxn_replay,2)
    ok = 0;
   flag = 3.1;
   fprintf(port1, [ok, flag]);
   fprintf(port1, flxn_replay(1:6,p));

    done_moving = fscanf(port1);
```

```
    done_moving = sscanf(done_moving, '%f');

    ok = 0;
    flag = 1.1;
    fprintf(port1, [ok flag]);
    flxn1 = fscanf(port1);
    flxn1 = sscanf(flxn1, '%f');
    z_gt1200(1:6,p,cuts) = flxn1(1:6);
    flxn_replay_jt_angles(1:6,p,cuts) = flxn1(7:12);

   %=========================================
   get_loads;  % measure: forces and moments
   %=========================================

   %=========================================
   fm_tare5;  % tare out bolt-up and fixture wt
   %=========================================

    load1200(1:6,p,cuts) = fa';
    fm_ufs1200(1:6,p,cuts) = fm_ufs';

   % display f/m after taring out bolt-up and fixture wt
    for i = 1:3
        if abs(fa(i)) > 0.5
            line_color(i,1:3) = [1 0 0];
        else
            line_color(i,1:3) = [0 0.75 0];
        end
   end
   subplot(2,1,1), set(fx, 'XData', [0 fa(1)], 'Color', line_color(1,:));
   subplot(2,1,1), set(fy, 'XData', [0 fa(2)], 'Color', line_color(2,:));
   subplot(2,1,1), set(fz, 'XData', [0 fa(3)], 'Color', line_color(3,:));
   subplot(2,1,2), set(mx, 'XData', [0 fa(4)]);
   subplot(2,1,2), set(my, 'XData', [0 fa(5)]);
   subplot(2,1,2), set(mz, 'XData', [0 fa(6)]);
    set(handles.w_now_edit, 'String', rot_angle400_replay(p));

    drawnow
end

% Read position and load data for dir = 1600 (extn -> flxn, replay)
% cuts = page number of matrix
for p = 1:size(extn_replay,2)
    ok = 0;
   flag = 3.1;
   fprintf(port1, [ok, flag]);
   fprintf(port1, extn_replay(1:6,p));

    done_moving = fscanf(port1);
    done_moving = sscanf(done_moving, '%f');

    ok = 0;
    flag = 1.1;
    fprintf(port1, [ok flag]);
    extn1 = fscanf(port1);
    extn1 = sscanf(extn1, '%f');
    z_gt1600(1:6,p,cuts) = extn1(1:6);
```

```
    extn_replay_jt_angles(1:6,p,cuts) = extn1(7:12);

   %=========================================
   get_loads;  % measure: forces and moments
   %=========================================

   %=========================================
   fm_tare5;  % tare out bolt-up and fixture wt
   %=========================================

    load1600(1:6,p,cuts) = fa';

   % display f/m after taring out bolt-up and fixture wt
    for i = 1:3
        if abs(fa(i)) > 0.5
            line_color(i,1:3) = [1 0 0];
        else
            line_color(i,1:3) = [0 0.75 0];
        end
   end
   subplot(2,1,1), set(fx, 'XData', [0 fa(1)], 'Color', line_color(1,:));
   subplot(2,1,1), set(fy, 'XData', [0 fa(2)], 'Color', line_color(2,:));
   subplot(2,1,1), set(fz, 'XData', [0 fa(3)], 'Color', line_color(3,:));
   subplot(2,1,2), set(mx, 'XData', [0 fa(4)]);
   subplot(2,1,2), set(my, 'XData', [0 fa(5)]);
   subplot(2,1,2), set(mz, 'XData', [0 fa(6)]);
    set(handles.w_now_edit, 'String', rot_angle800_replay(p));

    drawnow
end

% move specimen back to rotation angle = 0 in incremental movements
for p = 1:size(start_replay2,2)
    ok = 0;
   flag = 3.1;
   fprintf(port1, [ok, flag]);
   fprintf(port1, start_replay2(1:6,p));

    done_moving = fscanf(port1);
    done_moving = sscanf(done_moving, '%f');

    %=========================================
   get_loads;  % measure: forces and moments
   %=========================================

   %=========================================
   fm_tare5;  % tare out bolt-up and fixture wt
   %=========================================

   % display f/m after taring out bolt-up and fixture wt
    for i = 1:3
        if abs(fa(i)) > 0.5
            line_color(i,1:3) = [1 0 0];
        else
            line_color(i,1:3) = [0 0.75 0];
        end
   end
```

```
    subplot(2,1,1), set(fx, 'XData', [0 fa(1)], 'Color', line_color(1,:));
    subplot(2,1,1), set(fy, 'XData', [0 fa(2)], 'Color', line_color(2,:));
    subplot(2,1,1), set(fz, 'XData', [0 fa(3)], 'Color', line_color(3,:));
    subplot(2,1,2), set(mx, 'XData', [0 fa(4)]);
    subplot(2,1,2), set(my, 'XData', [0 fa(5)]);
    subplot(2,1,2), set(mz, 'XData', [0 fa(6)]);
     set(handles.w_now_edit, 'String', rot_angle400_replay(p));

     drawnow

end

delete(fh);

%=========================================
data_display_replay3; % display data
%=========================================

% Enable buttons on GUI when spine3h_replay.m is done running
set(hok, 'Enable', 'on');
set(hbolt, 'Enable', 'on');
set(hbefore, 'Enable', 'on');
set(hafter, 'Enable', 'on');
set(hpath, 'Enable', 'on');
set(hval, 'Enable', 'on');
set(hreplay, 'Enable', 'on');
set(hend, 'Enable', 'on');
```

data_display_replay3.m is a script called by spine3h_replay2.m that plots the data gathered during the cutting study.

```
% data_display_replay3.m
% display data
% Amy Loveless
% from data_sto3c_flxn (7/10/02)

% BUILD TRANSFORMATIONS OF TOOL C.S. WRT GLOBAL C.S.
% Build transformations of tool c.s. wrt global c.s. for each location for
flxn -> extn
for i = 1:size(z_gt1200,2)
    tGT1200(1:4,i*4-3:4*i,cuts) = eul2tr([deg2rad(z_gt1200(4,i,cuts)),
deg2rad(z_gt1200(5,i,cuts)), deg2rad(z_gt1200(6,i,cuts))]);
    tGT1200(1:3,4*i,cuts) = [z_gt1200(1,i,cuts) z_gt1200(2,i,cuts)
z_gt1200(3,i,cuts)]';
end

% Build transformations of tool c.s. wrt global c.s. for each location for
extn -> flxn
for i = 1:size(z_gt1600,2)
    tGT1600(1:4,i*4-3:4*i,cuts) = eul2tr([deg2rad(z_gt1600(4,i,cuts)),
deg2rad(z_gt1600(5,i,cuts)), deg2rad(z_gt1600(6,i,cuts))]);
    tGT1600(1:3,4*i,cuts) = [z_gt1600(1,i,cuts) z_gt1600(2,i,cuts)
z_gt1600(3,i,cuts)]';
```

```
end

% Build array of position vectors of tool c.s. from tGT for flxn -> extn
for i = 1:size(tGT1200,2)/4
    tGT1200_posn(1:4,i,cuts) = tGT1200(:,i*4,cuts);
end

% Build array of position vectors of tool c.s. from tGT for extn -> flxn
for i = 1:size(tGT1600,2)/4
    tGT1600_posn(1:4,i,cuts) = tGT1600(:,i*4,cuts);
end

% BUILD TRANSFORMATIONS OF UFS WRT GLOBAL C.S.
% Build transformations of UFS wrt global c.s. for each location for flxn ->
extn
for i = 1:size(tGT1200,2)/4
    tGUFS1200(1:4,i*4-3:4*i,cuts) = tGT1200(1:4,i*4-3:i*4,cuts)*pinv(tUFST);
end

% Build transformations of UFS wrt global c.s. for each location for extn ->
flxn
for i = 1:size(tGT1600,2)/4
    tGUFS1600(1:4,i*4-3:4*i,cuts) = tGT1600(1:4,i*4-3:i*4,cuts)*pinv(tUFST);
end

% Build array of position vectors of UFS from tGUFS for flxn -> extn
for i = 1:size(tGUFS1200,2)/4
    tGUFS1200_posn(1:4,i,cuts) = tGUFS1200(1:4,i*4,cuts);
end

% Build array of position vectors of UFS from tGUFS for extn -> flxn
for i = 1:size(tGUFS1600,2)/4
    tGUFS1600_posn(1:4,i,cuts) = tGUFS1600(1:4,i*4,cuts);
end

% BUILD TRANSFORMATIONS OF PT. OF INTEREST WRT GLOBAL C.S.
% Build transformations of pt. of interest wrt global c.s. for each location
for flxn -> extn
for i = 1:size(tGT1200,2)/4
    tGPOI1200(1:4,i*4-3:4*i,cuts) = tGUFS1200(1:4,i*4-3:i*4,cuts)*tUFSPOI;
end

% Build transformations of pt. of interest wrt global c.s. for each location
for extn -> flxn
for i = 1:size(tGT1600,2)/4
    tGPOI1600(1:4,i*4-3:4*i,cuts) = tGUFS1600(1:4,i*4-3:i*4,cuts)*tUFSPOI;
end

% Build array of position vectors of pt. of interest from tGUFS for flxn ->
extn
for i = 1:size(tGPOI1200,2)/4
    tGPOI1200_posn(1:4,i,cuts) = tGPOI1200(1:4,i*4,cuts);
end

% Build array of position vectors of pt. of interest from tGUFS for extn ->
flxn
for i = 1:size(tGPOI1600,2)/4
```

```
    tGPOI1600_posn(1:4,i,cuts) = tGPOI1600(1:4,i*4,cuts);
end
%=========================================================================
===================================

% BUILD ARRAYS TO BE USED FOR PLOTTING
% Arrays of fy, fz & mx
fy1200(1,1:size(load1200,2),cuts) = load1200(2,1:size(load1200,2),cuts);
fz1200(1,1:size(load1200,2),cuts) = load1200(3,1:size(load1200,2),cuts);
mx1200(1,1:size(load1200,2),cuts) = load1200(4,1:size(load1200,2),cuts);

fy1600(1,1:size(load1600,2),cuts) = load1600(2,1:size(load1600,2),cuts);
fz1600(1,1:size(load1600,2),cuts) = load1600(3,1:size(load1600,2),cuts);
mx1600(1,1:size(load1600,2),cuts) = load1600(4,1:size(load1600,2),cuts);

mx_end_pts = [flxn_mx_end_pts extn_mx_end_pts];

% Arrays of cut fy, fz & mx resultant force (for flxn -> extn only)
if cuts ~= 1
    for i = 1:size(fy1200,2)
        fy_cut(1,i,cuts-1) = fy1200(1,i,cuts-1) - fy1200(1,i,cuts);
        fz_cut(1,i,cuts-1) = fz1200(1,i,cuts-1) - fz1200(1,i,cuts);
        mx_cut(1,i,cuts-1) = mx1200(1,i,cuts-1) - mx1200(1,i,cuts);
        fyz(1,i,cuts-1) = sqrt(fy_cut(1,i,cuts-1)^2 + fz_cut(1,i,cuts-1)^2);
    end
end

% Arrays of moment arms
if cuts ~= 1
   for i = 1:size(fy_cut,2)
        dyhero(1,i,cuts-1) = fz_cut(1,i,cuts-1)*mx_cut(1,i,cuts-
1)/(fyz(1,i,cuts-1)^2);
        dzhero(1,i,cuts-1) = -fy_cut(1,i,cuts-1)*mx_cut(1,i,cuts-
1)/(fyz(1,i,cuts-1)^2);
        dyz(1,i,cuts-1) = mx_cut(1,i,cuts-1)/fyz(1,i,cuts-1);
        dzz(1,i,cuts-1) = -mx_cut(1,i,cuts-1)/fy_cut(1,i,cuts-1);
        dyy(1,i,cuts-1) = mx_cut(1,i,cuts-1)/fz_cut(1,i,cuts-1);
    end
end

% Arrays of constants
for i = 1:length(last_rot_angle_end_pts)
    y_replay(i) = 0;
end
for i = 1:length(rot_angle400_replay)
    y_replay2(i) = 0;
end

% Build strings to be used in plot legends
clear legend_string
if cuts < 10
    for i = 1:cuts
        legend_string(i,1:9) = ['replay 0', int2str(i)];
    end
else
    for i = 10:cuts
        legend_string(i,1:9) = ['replay ', int2str(i)];
```

```
    end
end

if cuts < 10
    for i = 2:cuts
        cut_string(i-1,1:9) = ['replay 0', int2str(i)];
    end
else
    for i = 10:cuts
        cut_string(i-1,1:9) = ['replay ', int2str(i)];
    end
end

% Save workspace
save(filename)
disp('Data has been saved.')
%=========================================================================
====================================

% BUILD PLOTS LIKE THOSE IN TODD'S MATHEMATICA PROGRAM
% Plot of rotation angle vs. mx for passive pathseek (mx_end_pts) & intact
replay (mx1200, mx1600)
if cuts == 1
   fh=figure('Position',[150 100 1000 900],'Color','w');
   set(gca, 'NextPlot', 'add');
%  plot(-rot_angle_replay, mx_end_pts, '.-', -rot_angle_replay,
[mx1200(:,:,1) mx1600(:,:,1)], '-o')
    plot(-rot_angle400_replay, mx_end_pts(1:length(mx1200)), '.-');
    plot(-rot_angle800_replay, mx_end_pts(length(mx1200)+1:end), '*-');
    plot(-rot_angle400_replay, mx1200(:,:,1), '.-', 'Color', [0 0.5 0]);
    plot(-rot_angle800_replay, mx1600(:,:,1), '*-', 'Color', [0 0.5 0]);
   line('XData', get(gca, 'XLim'), 'YData', [0 0]);
   line('XData', [0 0], 'YData', get(gca, 'YLim'));
   title('Mx vs. rotation angle'), xlabel('rotation angle (deg)'), ylabel('Mx
(Nm)');
   legend_handles = get(gca, 'Children');
   legend(flipdim(legend_handles(3:6),1), 'last pathseek (flxn \rightarrow
extn)', 'last pathseek (extn \rightarrow flxn)',...
        'intact replay (flxn \rightarrow extn)', 'intact replay (extn
\rightarrow flxn)', 2);
%  text(1, -2, 'flexion', 'Rotation', 30, 'FontSize', 14);
%  text(1, 2, 'extension', 'Rotation', 30, 'FontSize', 14);
end

% Plots of fy vs. rotation angle (flxn -> extn)
% put negative sign on fy1200 on 08-21-02 (why does this have to be done? is
it related to the difference in testing axes and specimen axes?)
fh=figure('Position',[150 100 1000 900],'Color','w');
set(gca, 'NextPlot', 'add');
% for i = 1:cuts
%     plot(-rot_angle400_replay, fy1200(1,1:size(fy1200,2),i), '.-')
% end
plot(-rot_angle400_replay, -fy1200(1,1:size(fy1200,2),cuts), '.-');
line('XData', get(gca, 'XLim'), 'YData', [0 0]);
line('XData', [0 0], 'YData', get(gca, 'YLim'));
title('Fy vs. rotation angle'), xlabel('rotation angle (deg)'), ylabel('Fy
(N)');
```

```
legend_handles = get(gca, 'Children');
% legend(flipdim(legend_handles(3:cuts+2),1), legend_string, 2);
% legend(flipdim(legend_handles(3),1), legend_string(cuts,1:9), 2);
legend(legend_handles(3), legend_string(cuts,1:9), 2);

% Plots of fz vs. rotation angle (flxn -> extn)
fh=figure('Position',[150 100 1000 900],'Color','w');
set(gca, 'NextPlot', 'add');
% for i = 1:cuts
%     plot(-rot_angle400_replay, fz1200(1,1:size(fz1200,2),i), '.-')
% end
plot(-rot_angle400_replay, fz1200(1,1:size(fz1200,2),cuts), '.-');
line('XData', get(gca, 'XLim'), 'YData', [0 0]);
line('XData', [0 0], 'YData', get(gca, 'YLim'));
title('Fz vs. rotation angle'), xlabel('rotation angle (deg)'), ylabel('Fz
(N)');
legend_handles = get(gca, 'Children');
% legend(flipdim(legend_handles(3:cuts+2),1), legend_string, 2);
% legend(flipdim(legend_handles(3),1), legend_string(cuts,1:9), 2);
legend(legend_handles(3), legend_string(cuts,1:9), 2);

% Plots of mx vs. rotation angle (flxn -> extn)
fh=figure('Position',[150 100 1000 900],'Color','w');
set(gca, 'NextPlot', 'add');
% for i = 1:cuts
%     plot(-rot_angle400_replay, mx1200(1,1:size(mx1200,2),i), '.-')
% end
plot(-rot_angle400_replay, mx1200(1,1:size(mx1200,2),cuts), '.-');
line('XData', get(gca, 'XLim'), 'YData', [0 0]);
line('XData', [0 0], 'YData', get(gca, 'YLim'));
title('Mx vs. rotation angle'), xlabel('rotation angle (deg)'), ylabel('Mx
(Nm)');
legend_handles = get(gca, 'Children');
% legend(flipdim(legend_handles(3:cuts+2),1), legend_string, 2);
% legend(flipdim(legend_handles(3),1), legend_string(cuts,1:9), 2);
legend(legend_handles(3), legend_string(cuts,1:9), 2);

if cuts ~= 1
    % Plots of fy vs. rotation angle for cuts
    % put negative sign on fy_cut on 08-21-02 (why does this have to be done?
is it related to the difference in testing axes and specimen axes?)
    fh=figure('Position',[150 100 1000 900],'Color','w');
    set(gca, 'NextPlot', 'add');
%     for i = 2:cuts
%         plot(-rot_angle400_replay, fy_cut(1,1:size(fy_cut,2),i-1), '.-')
%     end
    plot(-rot_angle400_replay, -fy_cut(1,1:size(fy_cut,2),cuts-1), '.-');
    line('XData', get(gca, 'XLim'), 'YData', [0 0]);
    line('XData', [0 0], 'YData', get(gca, 'YLim'));
    title('Fy of cut structure vs. rotation angle'), xlabel('rotation angle
(deg)'), ylabel('Fy (N)');
    legend_handles = get(gca, 'Children');
%     legend(flipdim(legend_handles(3:cuts+1),1), cut_string, 2);
%     legend(flipdim(legend_handles(3),1), legend_string(cuts,1:9), 2);
    legend(legend_handles(3), legend_string(cuts,1:9), 2);

    % Plots of fz vs. rotation angle for cuts
```

```
   fh=figure('Position',[150 100 1000 900],'Color','w');
    set(gca, 'NextPlot', 'add');
%     for i = 2:cuts
%         plot(-rot_angle400_replay, fz_cut(1,1:size(fz_cut,2),i-1), '.-')
%     end
    plot(-rot_angle400_replay, fz_cut(1,1:size(fz_cut,2),cuts-1), '.-');
    line('XData', get(gca, 'XLim'), 'YData', [0 0]);
    line('XData', [0 0], 'YData', get(gca, 'YLim'));
    title('Fz of cut structure vs. rotation angle'), xlabel('rotation angle
(deg)'), ylabel('Fz (N)');
    legend_handles = get(gca, 'Children');
%     legend(flipdim(legend_handles(3:cuts+1),1), cut_string, 2);
%     legend(flipdim(legend_handles(3),1), legend_string(cuts,1:9), 2);
    legend(legend_handles(3), legend_string(cuts,1:9), 2);

    % Plots of mx vs. rotation angle for cuts
   fh=figure('Position',[150 100 1000 900],'Color','w');
    set(gca, 'NextPlot', 'add');
%     for i = 2:cuts
%         plot(-rot_angle400_replay, mx_cut(1,1:size(mx_cut,2),i-1), '.-')
%     end
    plot(-rot_angle400_replay, mx_cut(1,1:size(mx_cut,2),cuts-1), '.-');
    line('XData', get(gca, 'XLim'), 'YData', [0 0]);
    line('XData', [0 0], 'YData', get(gca, 'YLim'));
    title('Mx of cut structure vs. rotation angle'), xlabel('rotation angle
(deg)'), ylabel('Mx (Nm)');
    legend_handles = get(gca, 'Children');
%     legend(flipdim(legend_handles(3:cuts+1),1), cut_string, 2);
%     legend(flipdim(legend_handles(3),1), legend_string(cuts,1:9), 2);
    legend(legend_handles(3), legend_string(cuts,1:9), 2);

   % Plot of resultant force vs. rotation angle
   fh=figure('Position',[150 100 1000 900],'Color','w');
    set(gca, 'NextPlot', 'add');
%     for i = 2:cuts
%         plot(-rot_angle400_replay, fyz(1,1:size(fyz,2),i-1), '.-')
%     end
    plot(-rot_angle400_replay, fyz(1,1:size(fyz,2),cuts-1), '.-');
    line('XData', get(gca, 'XLim'), 'YData', [0 0]);
    line('XData', [0 0], 'YData', get(gca, 'YLim'));
    title('force resultant of cut structure vs. rotation angle'),
xlabel('rotation angle (deg)'), ylabel('Fyz (N)');
    legend_handles = get(gca, 'Children');
%     legend(flipdim(legend_handles(3:cuts+1),1), cut_string, 2);
%     legend(flipdim(legend_handles(3),1), legend_string(cuts,1:9), 2);
    legend(legend_handles(3), legend_string(cuts,1:9), 2);

%  % Plots of moment arm (dyz) vs. rotation angle
%  fh=figure('Position',[150 100 1000 900],'Color','w');
%     hold on
%     for i = 1:cuts
%         plot(rot_angle400_replay, dyz(1,1:size(dyz,2),cuts-1), '.-',...
%             rot_angle400_replay, y_replay2, '-k', y_replay2,
dyz(1,1:size(dyz,2),cuts-1), '-k'),...
%             title('moment arm vs. rotation angle'), xlabel('rotation angle
(deg)'), ylabel('moment arm (mm)');
%     end
```

```
%     hold off
%
%     % Plots of moment arm (dyy) vs. rotation angle
%  fh=figure('Position',[150 100 1000 900],'Color','w');
%     hold on
%     for i = 1:cuts
%         plot(rot_angle400_replay, dyy(1,1:size(dyy,2),cuts-1), '.-',...
%             rot_angle400_replay, y_replay2, '-k', y_replay2,
dyz(1,1:size(dyz,2),cuts-1), '-k'),...
%             title('moment arm vs. rotation angle'), xlabel('rotation angle
(deg)'), ylabel('moment arm (mm)');
%     end
%     hold off
%
%     % Plots of moment arm (dzz) vs. rotation angle
%     fh=figure('Position',[150 100 1000 900],'Color','w');
%     hold on
%     for i = 1:cuts
%         plot(rot_angle400_replay, dzz(1,1:size(dzz,2),cuts-1), '.-',...
%             rot_angle400_replay, y_replay2, '-k', y_replay2,
dzz(1,1:size(dzz,2),cuts-1), '-k'),...
%             title('moment arm vs. rotation angle'), xlabel('rotation angle
(deg)'), ylabel('moment arm (mm)');
%     end
%     hold off
%
%     % Plots of moment arm (dzhero) vs. rotation angle
%  fh=figure('Position',[150 100 1000 900],'Color','w');
%     hold on
%     for i = 1:cuts
%         plot(rot_angle400_replay, dzhero(1,1:size(dzhero,2),cuts-1), '.-
',...
%             rot_angle400_replay, y_replay2, '-k', y_replay2,
dzhero(1,1:size(dzhero,2),cuts-1), '-k'),...
%             title('moment arm vs. rotation angle'), xlabel('rotation angle
(deg)'), ylabel('moment arm (mm)');
%     end
%     hold off
%
%     % Plots of moment arm (dyhero) vs. rotation angle
%  fh=figure('Position',[150 100 1000 900],'Color','w');
%     hold on
%     for i = 1:cuts
%         plot(rot_angle400_replay, dyhero(1,1:size(dyhero,2),cuts-1), '.-
',...
%             rot_angle400_replay, y_replay2, '-k', y_replay2,
dyhero(1,1:size(dyhero,2),cuts-1), '-k'),...
%             title('moment arm vs. rotation angle'), xlabel('rotation angle
(deg)'), ylabel('moment arm (mm)');
%     end
%     hold off
end
%=========================================================================
====================================

% % PLOT OF MOVEMENT OF POINT OF INTEREST IN GLOBAL C.S.
```

```
% % Z vs. X in global c.s. (start, extn & flxn together, starting & ending
points only)
% fh=figure('Position',[150 100 1000 900],'Color','w');
% set(gca, 'NextPlot', 'add')
% for i = 1:size(tGPOI1200_posn,2)
%     plot_handles_1 = plot(tGPOI1200_posn(1,i,cuts),
tGPOI1200_posn(3,i,cuts), '-or');
% end
% for i = 1:size(tGPOI1600_posn,2)
%     plot_handles_2 = plot(tGPOI1600_posn(1,i,cuts),
tGPOI1600_posn(3,i,cuts), '-ok');
% end
% title('Z vs. X for point of interest'), xlabel('X (mm)'), ylabel('Z (mm)');
% legend_handles = [plot_handles_1; plot_handles_2];
% legend(legend_handles, 'flxn \rightarrow extn', 'extn \rightarrow flxn');
%
% % PLOT OF MOVEMENT OF UFS IN GLOBAL C.S.
% % Z vs. X in global c.s. (start, extn & flxn together, starting & ending
points only)
% fh=figure('Position',[150 100 1000 900],'Color','w');
% set(gca, 'NextPlot', 'add')
% for i = 1:size(tGUFS1200_posn,2)
%     plot_handles_1 = plot(tGUFS1200_posn(1,i,cuts),
tGUFS1200_posn(3,i,cuts), '-or');
% end
% for i = 1:size(tGUFS1600_posn,2)
%     plot_handles_2 = plot(tGUFS1600_posn(1,i,cuts),
tGUFS1600_posn(3,i,cuts), '-ok');
% end
% title('Z vs. X for UFS'), xlabel('X (mm)'), ylabel('Z (mm)');
% legend_handles = [plot_handles_1; plot_handles_2];
% legend(legend_handles, 'flxn \rightarrow extn', 'extn \rightarrow flxn');
%
% % PLOT OF MOVEMENT OF COR IN GLOBAL C.S.
% % Z vs. X in global c.s. (start, extn & flxn together, starting & ending
points only)
% fh=figure('Position',[150 100 1000 900],'Color','w');
% set(gca, 'NextPlot', 'add')
% for i = 1:size(tGT1200_posn,2)
%     plot_handles_1 = plot(tGT1200_posn(1,i,cuts), tGT1200_posn(3,i,cuts),
'-or');
% end
% for i = 1:size(tGT1600_posn,2)
%     plot_handles_2 = plot(tGT1600_posn(1,i,cuts), tGT1600_posn(3,i,cuts),
'-ok');
% end
% title('Z vs. X for COR'), xlabel('X (mm)'), ylabel('Z (mm)');
% legend_handles = [plot_handles_1; plot_handles_2];
% legend(legend_handles, 'flxn \rightarrow extn', 'extn \rightarrow flxn');
%
% % PLOT OF MOVEMENT OF pt. of interest, UFS & COR IN GLOBAL C.S.
% % Z vs. X in global c.s. (start, extn & flxn together, starting & ending
points only)
% fh=figure('Position',[150 100 1000 900],'Color','w');
% set(gca, 'NextPlot', 'add')
% for i = 1:size(tGT1200_posn,2)
```

```
%     plot_handles_1 = plot(tGT1200_posn(1,i,cuts), tGT1200_posn(3,i,cuts),
'.b-',...
%         tGUFS1200_posn(1,i,cuts), tGUFS1200_posn(3,i,cuts), '-or',...
%         tGPOI1200_posn(1,i,cuts), tGPOI1200_posn(3,i,cuts), '-sr');
% end
% for i = 1:size(tGT1600_posn,2)
%     plot_handles_2 = plot(tGT1600_posn(1,i,cuts), tGT1600_posn(3,i,cuts),
'.b-',...
%         tGUFS1600_posn(1,i,cuts), tGUFS1600_posn(3,i,cuts), '-ok',...
%         tGPOI1600_posn(1,i,cuts), tGPOI1600_posn(3,i,cuts), '-sk');
% end
% title('Z vs. X for point of interest, COR & UFS'), xlabel('X (mm)'),
ylabel('Z (mm)');
% legend_handles = [plot_handles_1; plot_handles_2];
% legend(legend_handles, 'COR: flxn \rightarrow extn ', 'UFS: flxn
\rightarrow extn', 'POI: flxn \rightarrow extn',...
%     'COR: extn \rightarrow flxn', 'UFS: extn \rightarrow flxn', 'POI: extn
\rightarrow flxn', 0);
```

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Wiesel, S. W., and International Society for Study of the Lumbar Spine. 1996. *The Lumbar spine*. 2nd ed. 2 vols. Philadelphia: Saunders.

2. Frymoyer, J. W., and T. B. Ducker. 1991. *The adult spine : principles and practice*. New York: Raven Press.

3. Nishida, K., J. D. Kang, et al. 1999. Modulation of the biologic activity of the rabbit intervertebral disc by gene therapy: An in vivo study of adenovirus-mediated transfer of the human transforming growth factor beta 1 encoding gene. *Spine* 24 (23):2419-2425.

4. Panjabi, M. M. 1992. The stabilizing system of the spine. Part I. Function, dysfunction, adaptation, and enhancement. *Journal of Spinal Disorders* 5 (4):383-9; discussion 397.

5. White, A. A., and M. M. Panjabi. 1990. *Clinical biomechanics of the spine*. 2nd ed. Philadelphia: Lippincott.

6. Panjabi, M. M. 1988. Biomechanical evaluation of spinal fixation devices: I. A conceptual framework. *Spine* 13:1129-1134.

7. Panjabi, M. M., R. A. Brand, et al. 1976a. Mechanical properties of the human thoracic spine as shown by three-dimensional load-displacement curves. *Journal of Bone & Joint Surgery - American Volume* 58:642-652.

8. Panjabi, M. M., R. A. Brand, et al. 1976b. Three-dimensional flexibility and stiffness properties of the human thoracic spine. *Journal of Biomechanics* 9:185-192.

9. Edwards, W. T., W. C. Hayes, et al. 1987. Variation of lumbar spine stiffness with load. *Journal of Biomechanical Engineering* 109:35-42.

10. Adams, M., and W. Hutton. 1981. The relevance of torsion to the mechanical derangement of the lumbar spine. *Spine* 6:241-248.

11. Goodwin, R. R., K. S. James, et al. 1994. Distraction and Compression Loads Enhance Spine Torsional Stiffness. *Journal of Biomechanics* 27 (8):1049-1057.

12. Goel, V. K., D. G. Wilder, et al. 1995. Biomechanical testing of the spine. Load-controlled versus displacement-controlled analysis. *Spine* 20 (21):2354-7.

13. Doehring, T. C., Ph.D. 2000. Delineation of In-Vitro Lumbar Spine Structural Properties Using a Robotic/UFS Testing System with Hybrid Control: Experiments and Analytical Simulation. Doctoral, University of Pittsburgh, Pittsburgh.

14. Brown, T., R. J. Hansen, et al. 1957. Some mechanical tests on the lumbosacral spine with particular reference to the intervertebral discs. A preliminary report. *Journal of Bone & Joint Surgery - American Volume* 39:1135-1164.

15. Markolf, K. L. 1972. Deformation of the thoracolumbar intervertebral joints in response to external loads: a biomechanical study using autopsy material. *Journal of Bone & Joint Surgery - American Volume* 54 (3):511-33.

16. Tencer, A. F., and A. M. Ahmed. 1981. The role of secondary variables in the measurement of the mechanical properties of the lumbar intervertebral joint. *Journal of Biomechanical Engineering* 103:129-137.

17. Lee, C. K., and N. A. Langrana. 1984. Lumbosacral spinal fusion. A biomechanical study. *Spine* 9 (6):574-81.

18. Goel, V. K., S. Goyal, et al. 1985. Kinematics of the whole lumbar spine. Effect of discectomy. *Spine* 10 (6):543-54.

19. Yamamoto, I., M. M. Panjabi, et al. 1989. Three-dimensional movements of the whole lumbar spine and lumbosacral joint. *Spine* 14 (11):1256-60.

20. Shea, M., W. T. Edwards, et al. 1992. Variations of Stiffness and Strength Along the Human Cervical-Spine - Response. *Journal of Biomechanics* 25 (6):690-690.

21. Kunz, D. N., R. P. Mccabe, et al. 1994. A Multi-Degree-of-Freedom System for Biomechanical Testing. *Journal of Biomechanical Engineering-Transactions of the Asme* 116 (3):371-373.

22. Wilke, H. J., L. Claes, et al. 1994. A universal spine tester for in vitro experiments with muscle force simulation. *European Spine Journal* 3 (2):91-7.

23. Crawford, N. R., A. G. U. Brantley, et al. 1995. An Apparatus for Applying Pure Nonconstraining Moments to Spine Segments in-Vitro. *Spine* 20 (19):2097-2100.

24. Patwardhan, A. G., R. M. Havey, et al. 1999. A follower load increases the load-carrying capacity of the lumbar spine in compression. *Spine* 24 (10):1003-1009.

25. Cobbold, R. S. C. 1974. *Transducers for biomedical measurements: Principles and applications*. New York: John Wiley & Sons.

26. Norton, H. N. 1989. *Handbook of transducers*. Englewood Cliffs: Prentice-Hall, Inc.

27. Fujie, H., G. A. Livesay, et al. 1995. The use of a universal force-moment sensor to determine in-situ forces in ligaments: a new methodology. *Journal of Biomechanical Engineering* 117 (1):1-7.

28. Koogle, T. A., R. L. Piziali, et al. 1977. A motion transducer for use in the intact in-vitro human lumbar spine. *Journal of Biomechanical Engineering* 99:160-165.

29. Panjabi, M. M., M. H. Krag, et al. 1981. A technique for measurement and description of three-dimensional six degree-of-freedom motion of a body joint with an application to the human spine. *Journal of Biomechanics* 14:447-460.

30. Goel, V. K., T. A. Nye, et al. 1987. A technique to evaluate an internal spinal device by use of the Selspot system: An application to the Luque closed loop. *Spine* 12:150-159.

31. Moeini, S. M. R., J. E. Lemons, et al. 1996. Investigation of video techniques for dynamic measurements of relative vertebral-body motion in vitro. *Biomedical Instrumentation and Technology* 30:62-70.

32. Raibert, M. H., and J. J. Craig. 1981. Hybrid position/force control of manipulators. *Journal of Dynamic Systems, Measurement and Control* 102:375-382.

33. Fujie, H., K. Mabuchi, et al. 1993. The Use of Robotics Technology to Study Human Joint Kinematics - a New Methodology. *Journal of Biomechanical Engineering-Transactions of the Asme* 115 (3):211-217.

34. Rudy, T. W., G. A. Livesay, et al. 1996. A combined robotic/universal force sensor approach to determine in situ forces of knee ligaments. *Journal of Biomechanics* 29 (10):1357-1360.

35. Carlin, G. J., G. A. Livesay, et al. 1996. In-situ forces in the human posterior cruciate ligament in response to posterior tibial loading. *Annals of Biomedical Engineering* 24 (2):193-197.

36. Livesay, G. A., H. Fujie, et al. 1995. Determination of the in-Situ Forces and Force Distribution within the Human Anterior Cruciate Ligament. *Annals of Biomedical Engineering* 23 (4):467-474.

37. Wu, G., S. Siegler, et al. 2002. ISB recommendation on definitions of joint coordinate system of various joints for the reporting of human joint motion - part 1: ankle, hip, and spine. *Journal of Biomechanics* 35 (4):543-548.

38. Wu, G., and P. R. Cavanagh. 1995. Isb Recommendations for Standardization in the Reporting of Kinematic Data. *Journal of Biomechanics* 28 (10):1257-1260.

39. Grood, E. S., and W. J. Suntay. 1983. A joint coordinate system for the clinical description of three-dimensional motions: application to the knee. *Journal of Biomechanical Engineering* 105 (2):136-44.

40. Grassmann, S., T. R. Oxland, et al. 1998. Constrained testing conditions affect the axial rotation response of lumbar functional spinal units. *Spine* 23 (10):1155-1162.

41. Goel, V. K., and J. N. Weinstein, eds. 1990. *Biomechanics of the spine : clinical and surgical perspective*. Boca Raton, FL: CRC Press.

42. Spiegelman, J. J., and S. L.-Y. Woo. 1987. A rigid-body method for finding centers of rotation and angular displacements of planar joint motion. *Journal of Biomechanics* 20 (7):715-721.

43. Crisco, J. J., X. B. Chen, et al. 1994. Optimal Marker Placement for Calculating the Instantaneous Center of Rotation. *Journal of Biomechanics* 27 (9):1183-1187.

44. Challis, J. H. 2001. Estimation of the finite center of rotation in planar movements. *Medical Engineering & Physics* 23 (3):227-233.

45. Melkerson, M. N., S. L. Griffith, et al. 2003. *Spinal implants : are we evaluating them appropriately?* West Conshohocken, PA: ASTM International.

46. Wilke, H. J., G. Russo, et al. 1997. A mechanical model of human spinal motion segments. *Biomedizinische Technik* 42 (11):327-331.

47. Ren, W. X., X. G. Tan, et al. 1999. Nonlinear analysis of plane frames using rigid body-spring discrete element method. *Computers & Structures* 71 (1):105-119.

48. Staubli. 1997. *RX90 Family Robot CS7 Instruction Manual*. D.280.190.54.G - 11/99 ed: Staubli Faverges.

49. Adept. 1995. *AdeptForce VME User's Guide*. Part Number 00211-00000, Rev. B ed: Adept Technology, Inc.

50. Pires, N. J., MATJR3PCI for MATLAB 5, 2001. Mechanical Engineering Department, Laboratory, Internal Report No GCG.RCL.001.02, Coimbra, Portugal.

51. Gilbertson, L. G., T. C. Doehring, et al. 1999. Improvement of accuracy in a high-capacity, six degree-of-freedom load cell: Application to robotic testing of musculoskeletal joints. *Annals of Biomedical Engineering* 27 (6):839-843.

52. Hoffman, J. D. 2001. *Numerical methods for engineers and scientists*. 2nd , rev. and expand ed. New York: Marcel Dekker.

53. Winkelstein, B. A., and B. S. Myers. 2002. Importance of nonlinear and multivariable flexibility coefficients in the prediction of human cervical spine motion. *Journal of Biomechanical Engineering* 124 (5):504-11.