SANDIA REPORT

SAND98-8242
Unlimited Release
Printed May 1998

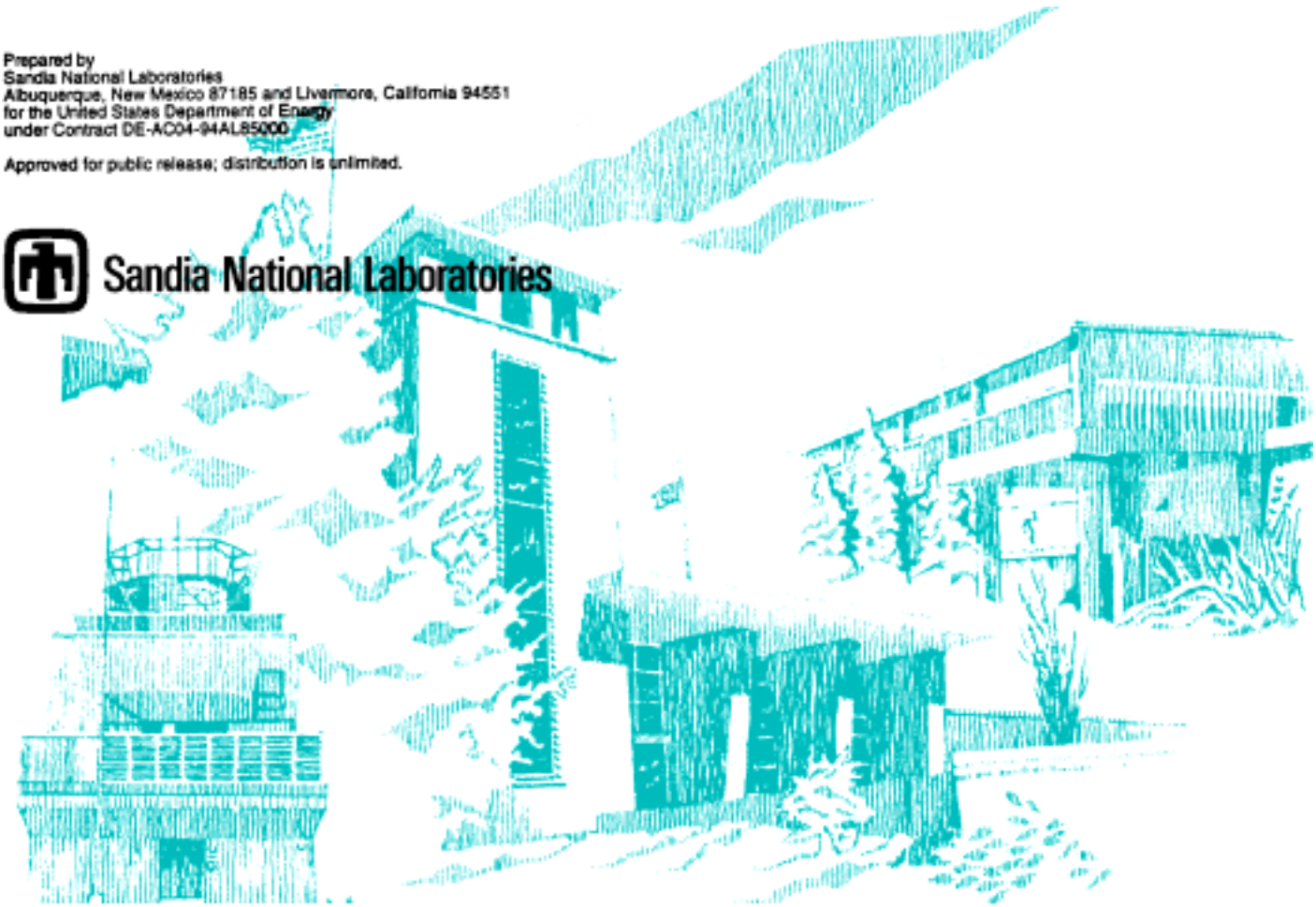
Some Approaches for Modeling and Analysis of a Parallel Mechanism with Stewart Platform Architecture

V. De Sapio

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94551
for the United States Department of Energy
under Contract DE-AC04-94AL85000.

Approved for public release; distribution is unlimited.

Sandia National Laboratories



SF2900Q(8-81)

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.
NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contrators.

SAND98-8242
Unlimited Release
Printed May 1998

Distribution
Category UC-706

Some Approaches for Modeling and Analysis of a Parallel Mechanism with Stewart Platform Architecture

Vincent De Sapio
Integrated Manufacturing Systems Center
Sandia National Laboratories
Livermore, CA 94550

Abstract

Parallel mechanisms represent a family of devices based on a closed kinematic architecture. This is in contrast to serial mechanisms, which are comprised of a chain-like series of joints and links in an open kinematic architecture. The closed architecture of parallel mechanisms offers certain benefits and disadvantages. A particular subset of parallel mechanisms is known as Stewart platforms (or hexapods). The classic Stewart platform is based on a set of six independently actuated struts. The struts translate through sphere joints which reside on a stationary platform. Each strut terminates at a ball joint residing on a mobile tool platform. All joints allow passive rotation. The effect of this kinematic architecture is that the tool platform can be controlled to move in six axes. This report addresses some analytic approaches for modeling and characterizing these devices. These approaches cover inverse and forward kinematics, Jacobian evaluation, path execution, workspace envelope modeling, tool velocity characterization, and structural stiffness. A characterization is also made of a hybrid parallel-serial architecture consisting of a Stewart platform and a two-axis serial tool head. Numerical algorithms are also provided along with source code for performing characterizations.

Contents

Nomenclature 5
Background 6
Introduction 7
Inverse Kinematics 7
Modeling an Articulated Tool Head 11
Inverse Jacobian 14
Path Execution. 16
Workspace Envelope 18
Velocity Envelopes 20
Stiffness 22
Forward Kinematics 24
Conclusion. 26
References 28

Figures

1 Illustration from “A Platform with Six Degrees of Freedom”, 1965, by D. Stewart. 6
2 Hexel hexapod design 7
3 6-3 node Stewart platform 8
4 Geometric layouts of upper and lower platforms[3] 8
5 6-3 node Stewart platform with tool head frame defined as the operational frame. 9
6 6-3 node Stewart platform modeled in Deneb Envision TR (inverse kinematics) 11
7 6-3 node Stewart platform with two-axis tool head frame. 11
8 Tool head kinematics 12
9 Tool head kinematics (Denavit- Hartenberg description) 13
10 Stewart platform with articulated tool head modeled in Deneb Envision TR 14
11 6-3 node Stewart platform 16
12 Tool path in Cartesian space 18
13 Lower workspace algorithm. 18
14 Lower workspace envelope plot 19
15 Upper workspace algorithm 19
16 Upper workspace envelope plot 20
17 Total workspace envelope 20
18 Velocity envelope algorithm 21
19 Velocity envelope plots 21
20 Velocity hypersurface plot showing variation of velocity envelopes within the workspace 22
21 Stiffness sample output 24
22 Forward displacement algorithm 25
23 Forward displacement sample output (blur effect used to convey animation) 25
24 Variant designs 26
25 Parallel mechanism toolbox 27
26 Simulation development/implementation 27

Tables

1 Denavit-Hartenberg parameters 12

Nomenclature

- "Cartesian space" will be used interchangeably with "operational space" to describe a mechanism's tool, in space, in terms of three rectilinear positions coordinates (and their derivatives) as well as three orientation parameters (and their derivatives).
- "Configuration space" will be used interchangeably with "joint space" and "strut space" to describe a mechanisms posture in terms of its joint variables (and their derivatives). In the case of a Stewart platform these joint variables are the strut lengths.
- Inverse kinematics refers to the problem of determining a mechanism's joint values given a particular Cartesian description of the tool (position and orientation). In the case of articulated serial mechanisms, more than one joint solution usually exists for a given tool position and orientation.
- Forward kinematics refers to the problem of determining the Cartesian description of a mechanism's tool (position and orientation) given a particular set of joint values. In the case of most parallel mechanisms, more than one Cartesian solution usually exists for a given set of joint values.
- A rotation matrix is a 3x3 matrix describing the orientation of one coordinate frame relative to another. The column vectors of a rotation matrix are the bases of the coordinate frame of interest, represented within the embedding frame.
- A homogenous transformation matrix is a 4x4 matrix describing both position and orientation of one coordinate frame relative to another. It is composed of a rotation matrix part and a position vector part.
- Euler angles are an ordered triplet of angles (α, β, γ), where each subsequent rotation is given about a chosen axis, relative to the current (intermittent) frame in the sequence.
- A Jacobian matrix defines the mapping of differential joint space displacements into differential Cartesian space displacements of the tool.

The following notation will be employed regarding coordinate frames, matrices, and vectors:

- A coordinate frame will be denoted with brackets, "{}". For example, $\{A\}$ denotes coordinate frame A.
- Rotation matrices will be represented as "R". Scripting will be employed on the left hand side to indicate the coordinate frame of interest and the embedding frame that it is being represented within. The frame of interest will be represented with a sub-script and the embedding frame will be represented with a super-script. For example, ${}^{A}_{B}R$ denotes the rotation matrix representing the orientation of frame B in frame A.
- Homogenous transformation matrices will use the same convention as rotation matrices. For example, ${}^{A}_{B}T$ denotes the transformation matrix representing the position and orientation of frame B in frame A.
- Jacobian matrices will be represented as "J". A super-script will be used to indicate the frame that the Jacobian is represented with respect to. For example, ${}^{B}J$ denotes the Jacobian represented in the Base frame. Since we will always be referring to the Jacobian in this frame, J without any superscript will, by default, refer to a Jacobian represented within the Base frame.
- A vector will be represented with a horizontal arrow. A super-script to the left will indicate the frame that it is being represented within. For example, ${}^{A}\vec{p}$ is a position vector represented in frame A.

Some Approaches for Modeling and Analysis of a Parallel Mechanism with Stewart Platform Architecture

Background

The classic Stewart platform was first proposed by D. Stewart in 1965[1]. The most common early application for these devices was use as flight simulators (see *Figure 1*). Stewart also indicated potential applications such as a vibration damping platform for shipboard situations. There was also some anticipation, by Stewart, of applications associated with automatic assembly and machine tools. However, no significant work related to these potential manufacturing applications was done in those early years.

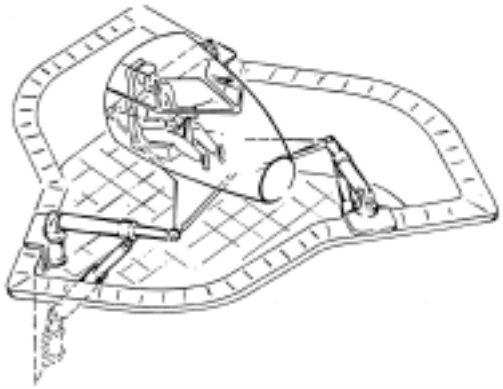


Figure 1. Illustration from "A Platform with Six Degrees of Freedom", 1965, by D. Stewart

In the past 30 years since Stewart's initial proposal the advent of computer controlled robots and machine tools have brought a greater interest in the application of Stewart platforms (and other parallel architecture variants) to factory automation and manufacturing problems. Currently there are a handful of hexapods in operation in the United States. Ingersoll, Giddings & Lewis, and Hexel (see *Figure 2*) are the major U.S. suppliers of these devices. There is an equal, if not greater, enthusiasm for hexapod technology outside of the U.S., among suppliers like Geodetics in Europe.

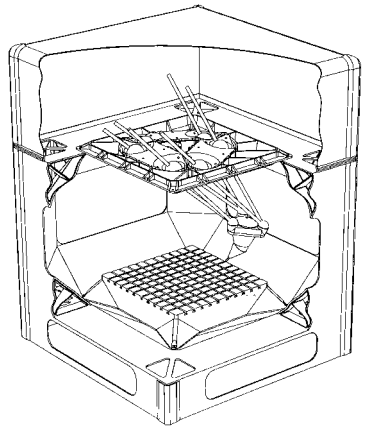


Figure 2. Hexel hexapod design

Real-time computer based control has been the most powerful impetus for the emergence of these machines (as it has been for traditional serial arm robots and CNC machine tools). A fundamental understanding and computer implementation of the kinematics, dynamics, controls aspects, etc. of these devices is essential. This report addresses these areas and provides an analytical framework. *Introduction to Robototics: Mechanics & Control*[2] provides some of the background fundamentals (kinematics, transformation and matrix algebra, etc.) relevant to this report.

Introduction

This report addresses the analysis of a class of parallel mechanisms known as Stewart platforms. The analysis focuses on inverse and forward kinematics, Jacobian evaluation, path execution, workspace envelope modeling, tool velocity characterization, and structural stiffness. Stewart platforms (and parallel architecture mechanisms in general) are being pursued as next generation motion mechanisms for machine tool applications, robotics, and other factory automation applications. For this reason it is of paramount necessity to characterize these devices both analytically and empirically.

Parallel mechanisms provide special challenges not associated with prismatic serial mechanisms. Workspace properties are not isotropic throughout the working envelope. Therefore it is important to create an analytical framework to characterize these devices so that mechanism designs can be optimized to meet particular objectives. For a pre-established design, analytical characterization is also very important. In this case, detailed performance maps are desired so that part fixturing and tool path plans are based on relevant criteria for the particular application (e.g., high-speed machining, part manipulation, high stiffness processing, etc.).

Inverse Kinematics

In many cases the inverse kinematics of parallel mechanisms are more conducive to a closed-form analytic solution, than the inverse kinematics of a serial link robot. In the case of both the 6-6 node Stewart platform and the 6-3 node Stewart platform, the inverse kinematics can be solved in a closed-form fashion (the forward kinematics will be addressed using a numerical algorithm in the *Forward Kinematics* section, page 24). The derivations that follow throughout this report specifically address a 6-3 Stewart platform, however all of the approaches used here are easily extensible to the 6-6 variety. Additionally, where numerical values are applied they are specific to the Hexel corporation hexapod.

First it will be necessary to establish a set of coordinate frames. As with conventional robots, a Base frame and a Tool (or operational frame) will be defined. While our choice of these frames is arbitrary we will use a

convenient frame layout shown in *Figure 3*. The Base frame lies on the plane defined by the six sphere joint centers (stationary platform). The operational frame lies on the plane defined by the three ball joint centers (mobile tool platform).

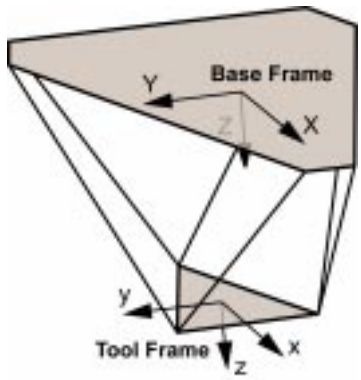


Figure 3. 6-3 node Stewart platform

We can now define position vectors representing the six base plate vertices in the Base frame, ${}^{B}\vec{p}_{bi}$ (i = 1 to 6), and the three tool plate vertices in the Tool frame, ${}^{T}\vec{p}_{ti}$ (i = 1 to 3).

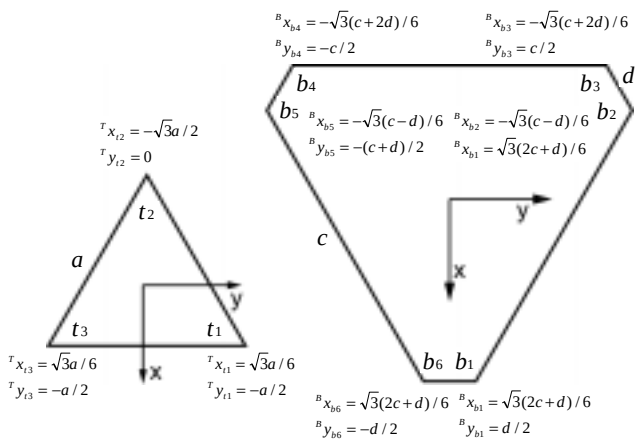


Figure 4. Geometric layouts of upper and lower platforms[3]

In the case of the Hexel machine,

$$a = 300mm$$
$$c = 572mm$$
$$d = 244mm$$

So, we have,

$$
\begin{aligned}
&{}^{T}x_{t1} = 86.60mm && {}^{B}x_{b1} = 400.68mm \\
&{}^{T}y_{t1} = 150mm && {}^{B}y_{b1} = 122mm \\
&{}^{T}x_{t2} = 173.20mm && {}^{B}x_{b2} = 94.69mm \\
&{}^{T}y_{t2} = 0 && {}^{B}y_{b2} = 408mm \\
&{}^{T}x_{t3} = 86.6mm && {}^{B}x_{b3} = 306mm \\
&{}^{T}y_{t3} = -150mm && {}^{B}y_{b3} = 286mm \\
&{}^{T}z_{ti} = 0 \quad (\mathrm{i} = 1 \text{ to } 3) && {}^{B}x_{b3} = -306mm \\
& && {}^{B}y_{b3} = -286mm \\
& && {}^{B}x_{b4} = -94.69mm \\
& && {}^{B}y_{b4} = 408mm \\
& && {}^{B}x_{b5} = 400.68mm \\
& && {}^{B}y_{b5} = -122mm \\
& && {}^{B}z_{bi} = 0 \quad (\mathrm{i} = 1 \text{ to } 6)
\end{aligned}
$$

If we take the operational frame to be at the end of a tool head (instead of being on the plane defined by the three ball joint centers) the representation of the three vertices is merely shifted in *z* (see *Figure 5*). This simply introduces a *z* component for the three points, equal to the shift distance in the negative *z* direction. In the case of the Hexel machine the shift distance is 305.1 mm nominally.

$$ {}^{T}z_{t1} = {}^{T}z_{t2} = {}^{T}z_{t3} = -305.1mm $$

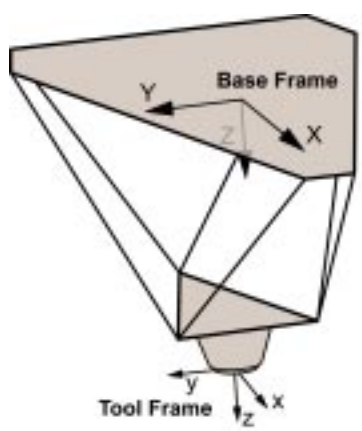


Figure 5. 6-3 node Stewart platform with tool head frame defined as the operational frame

We now need to express the three tool plate vertices with respect to the Base frame to obtain ${}^{B}\vec{p}_{ti}$ (i = 1 to 3). To do this we need to define a homogenous transformation matrix which represents the position and orientation of the Tool frame embedded within the Base frame, ${}^{B}_{T}T$. This transform will be a function of the *x, y, z* position values of the tool and three orientation parameters represented in the Base frame. A typical way of describing orientation as an ordered triplet utilizes angle set parameters α, β, γ. We may either use a fixed set or

an Euler set. We will use a *Z-Y-Z* Euler set[3]. The rotation matrix associated with this parameterization of orientation is,

$$ {}^{B}_{T}R(\alpha,\beta,\gamma) = \begin{bmatrix} \cos\beta\cos\gamma + \sin\alpha\sin\beta\sin\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\beta\sin\gamma & \cos\alpha\sin\beta \\ \cos\alpha\sin\gamma & \cos\alpha\cos\gamma & -\sin\alpha \\ \sin\alpha\cos\beta\sin\gamma - \sin\beta\cos\gamma & \sin\beta\sin\gamma + \sin\alpha\cos\beta\cos\gamma & \cos\alpha\cos\beta \end{bmatrix} $$

and,

$$ {}^{B}_{T}T(x,y,z,\alpha,\beta,\gamma) = \begin{bmatrix} \cos\beta\cos\gamma + \sin\alpha\sin\beta\sin\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\beta\sin\gamma & \cos\alpha\sin\beta & x \\ \cos\alpha\sin\gamma & \cos\alpha\cos\gamma & -\sin\alpha & y \\ \sin\alpha\cos\beta\sin\gamma - \sin\beta\cos\gamma & \sin\beta\sin\gamma + \sin\alpha\cos\beta\cos\gamma & \cos\alpha\cos\beta & z \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Consequently,

$$ {}^{B}\vec{p}_{ti} = {}^{B}_{T}T(x,y,z,\alpha,\beta,\gamma)\,{}^{T}\vec{p}_{ti} $$

Now a description of the strut lengths is obtained merely by taking the magnitudes of the vector differences between corresponding base plate and tool plate vertices.

$$ l_1 = \sqrt{({}^{B}x_{t1} - {}^{B}x_{b1})^2 + ({}^{B}y_{t1} - {}^{B}y_{b1})^2 + {}^{B}z_{t1}{}^{2}} $$
$$ l_2 = \sqrt{({}^{B}x_{t1} - {}^{B}x_{b2})^2 + ({}^{B}y_{t1} - {}^{B}y_{b2})^2 + {}^{B}z_{t1}{}^{2}} $$
$$ l_3 = \sqrt{({}^{B}x_{t2} - {}^{B}x_{b3})^2 + ({}^{B}y_{t2} - {}^{B}y_{b3})^2 + {}^{B}z_{t2}{}^{2}} $$
$$ l_4 = \sqrt{({}^{B}x_{t2} - {}^{B}x_{b4})^2 + ({}^{B}y_{t2} - {}^{B}y_{b4})^2 + {}^{B}z_{t2}{}^{2}} $$
$$ l_5 = \sqrt{({}^{B}x_{t3} - {}^{B}x_{b5})^2 + ({}^{B}y_{t3} - {}^{B}y_{b5})^2 + {}^{B}z_{t2}{}^{2}} $$
$$ l_6 = \sqrt{({}^{B}x_{t3} - {}^{B}x_{b6})^2 + ({}^{B}y_{t3} - {}^{B}y_{b6})^2 + {}^{B}z_{t2}{}^{2}} $$

An inverse kinematic model of the Hexel hexapod has been generated in the Deneb Envision software (see *Figure 6)*. A MATLAB program which couples the inverse and forward problems is detailed in the Forward Kinematics section.

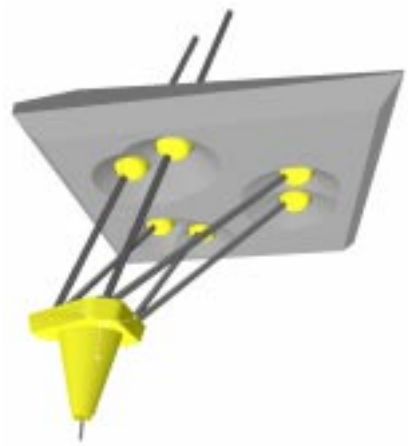


Figure 6. 6-3 node Stewart platform modeled in Deneb Envision TR (inverse kinematics)

Modeling an Articulated Tool Head

An articulated tool head offers a means of increasing the dexterity of a standard Stewart mechanism. In this case we will be characterizing a two-axis tool head. We will take the operational frame to be at the end of the two-axis tool head (see *Figure 7*).

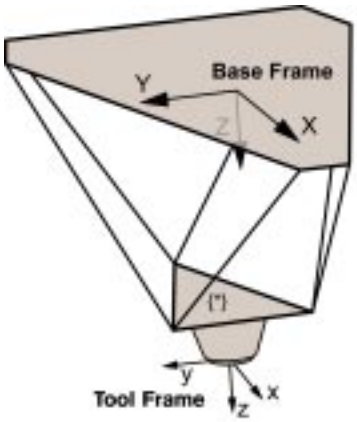


Figure 7. 6-3 node Stewart platform with two-axis tool head frame

Figure 8 describes the configuration of joints on the two-axis tool head.

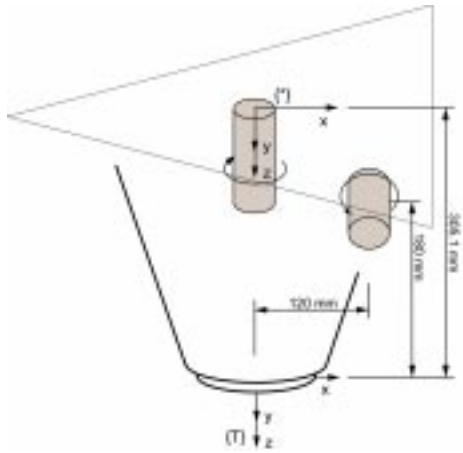


Figure 8. Tool head kinematics

We can find the transform representing the tool head frame, $\{T\}$, in the ball plate frame, $\{*\}$.

$$ {}^{*}_{T}T = R_z(\theta_1)\cdot D(120,0,125.1)\cdot R_x(-90)\cdot R_z(\theta_2)\cdot D(-120,-180,0)\cdot R_x(90) $$

$$ {}^{*}_{T}T = \begin{bmatrix} \cos\theta_1\cos\theta_2 & -\sin\theta_1 & \cos\theta_1\sin\theta_2 & (180\sin\theta_2 - 120\cos\theta_2 + 120)\cos\theta_1 \\ \sin\theta_1\cos\theta_2 & \cos\theta_1 & \sin\theta_1\sin\theta_2 & (180\sin\theta_2 - 120\cos\theta_2 + 120)\sin\theta_1 \\ -\sin\theta_2 & 0 & \cos\theta_2 & 180\cos\theta_2 + 120\sin\theta_2 + 125.1 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Alternately, we can use the Denavit-Hartenberg convention. The Denavit-Hartenberg parameters are†,

i	a_{i-1} *Link Length*	α_{i-1} *Link Twist*	d_i *Joint Offset*	θ_i *Joint Angle*
1	0	0°	125.1	θ_1
2	120	-90°	0	θ_2
3	-120	90°	180	0

Table 1. Denavit-Hartenberg parameters

†. Modified Denavit-Hartenberg approach[2].

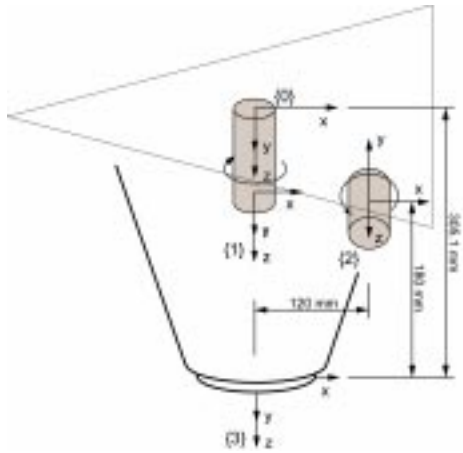


Figure 9. Tool head kinematics (Denavit- Hartenberg description)

We can input these parameters into the Denavit-Hartenberg transform matrices, defined as,

$$ {}^{i-1}_{\ \ i}T = R_x(\alpha_{i-1}) \cdot D_x(a_{i-1}) \cdot R_z(\theta_i) \cdot D_z(d_i) $$

Multiplying out ${}^{*}_{T}T = {}^{0}_{1}T\,{}^{1}_{2}T\,{}^{2}_{3}T$ yields the same result arrived at previously. The locations of the bifurcated ball joint centers in the tool head frame, ${}^{T}\vec{p}_{ti}$ (i = 1 to 3), are then,

$$ {}^{T}\vec{p}_{ti} = {}^{T}_{*}T\,{}^{*}\vec{p}_{ti} $$

where,

$$ {}^{T}_{*}T = {}^{*}_{T}T^{-1} $$

and,

$$ {}^{*}\vec{p}_{ti} = \begin{bmatrix} {}^{*}x_{ti} \\ {}^{*}y_{ti} \\ {}^{*}z_{ti} \end{bmatrix} $$

are the constant position vectors representing the bifurcated ball joint centers in the ball plate frame {*}, calculated earlier as,

$$ \begin{aligned} {}^{*}x_{t1} &= 86.60mm \\ {}^{*}y_{t1} &= 150mm \\ {}^{*}x_{t2} &= 173.20mm \\ {}^{*}y_{t2} &= 0 \\ {}^{*}x_{t3} &= 86.6mm \\ {}^{*}y_{t3} &= -150mm \\ {}^{*}z_{ti} &= 0 \quad \text{(i = 1 to 3)} \end{aligned} $$

A model of a two-axis articulated tool head has been generated in the Deneb Envision software (see *Figure 10*).

Figure 10. Stewart platform with articulated tool head modeled in Deneb Envision TR

Inverse Jacobian

For a serial mechanism a closed form solution of the forward kinematics can be obtained.

$$x = f_1(q_1,,,q_n)$$
$$y = f_2(q_1,,,q_n)$$
$$.$$
$$.$$

The application of the chain rule yields differentials of *x, y,* ... as functions of the differentials of q_i (i = 1 to n).

$$\delta x = \frac{\partial f_1}{\partial q_1}\delta q_1 +,,, \frac{\partial f_1}{\partial q_n}\delta q_n$$
$$\delta y = \frac{\partial f_2}{\partial q_1}\delta q_1 +,,, \frac{\partial f_2}{\partial q_n}\delta q_n$$
$$.$$
$$.$$

Dividing both sides by the differential time element δt and expressing in matrix form yields,

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ . \\ . \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial q_1} & . & . & \frac{\partial f_1}{\partial q_n} \\ \frac{\partial f_2}{\partial q_1} & . & . & \frac{\partial f_2}{\partial q_n} \\ . & . & . & . \\ . & . & . & . \end{bmatrix} \bullet \begin{bmatrix} \dot{q}_1 \\ . \\ . \\ \dot{q}_n \end{bmatrix}$$

Where the matrix is defined as the Jacobian, J. The Jacobian facilitates the mapping of the configuration space joint rate vector, $\dot{\vec{q}}$, into the Cartesian space velocity vector, $\vec{v}$.

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial q_1} & \cdot & \cdot & \frac{\partial f_1}{\partial q_n} \\ \frac{\partial f_2}{\partial q_1} & \cdot & \cdot & \frac{\partial f_2}{\partial q_n} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

$$\vec{v} = J\dot{\vec{q}}$$

For a parallel mechanism, like a Stewart platform, we have seen that a closed-form solution of the inverse kinematics can be obtained.

$$\begin{aligned} l_1 &= f_1(x, y, z, \alpha, \beta, \gamma) \\ l_2 &= f_2(x, y, z, \alpha, \beta, \gamma) \\ &\cdot \\ &\cdot \\ l_6 &= f_6(x, y, z, \alpha, \beta, \gamma) \end{aligned}$$

The application of the chain rule yields differentials of l_i (i = 1 to 6) as functions of the differentials of x, y, ...,

$$\begin{aligned} \delta l_1 &= \frac{\partial f_1}{\partial x}\delta x + \frac{\partial f_1}{\partial y}\delta y + \frac{\partial f_1}{\partial z}\delta z + \frac{\partial f_1}{\partial \alpha}\delta\alpha + \frac{\partial f_1}{\partial \beta}\delta\beta + \frac{\partial f_1}{\partial \gamma}\delta\gamma \\ \delta l_2 &= \frac{\partial f_2}{\partial x}\delta x + \frac{\partial f_2}{\partial y}\delta y + \frac{\partial f_2}{\partial z}\delta z + \frac{\partial f_2}{\partial \alpha}\delta\alpha + \frac{\partial f_2}{\partial \beta}\delta\beta + \frac{\partial f_2}{\partial \gamma}\delta\gamma \\ &\cdot \\ &\cdot \\ \delta l_6 &= \frac{\partial f_6}{\partial x}\delta x + \frac{\partial f_6}{\partial y}\delta y + \frac{\partial f_6}{\partial z}\delta z + \frac{\partial f_6}{\partial \alpha}\delta\alpha + \frac{\partial f_6}{\partial \beta}\delta\beta + \frac{\partial f_6}{\partial \gamma}\delta\gamma \end{aligned}$$

Dividing both sides by the differential time element δt and expressing in matrix form yields,

$$\begin{bmatrix} \dot{l}_1 \\ \dot{l}_2 \\ \cdot \\ \cdot \\ \cdot \\ \dot{l}_6 \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} & \frac{\partial f_1}{\partial \alpha} & \frac{\partial f_1}{\partial \beta} & \frac{\partial f_1}{\partial \gamma} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial z} & \frac{\partial f_2}{\partial \alpha} & \frac{\partial f_2}{\partial \beta} & \frac{\partial f_2}{\partial \gamma} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \frac{\partial f_6}{\partial x} & \frac{\partial f_6}{\partial y} & \frac{\partial f_6}{\partial z} & \frac{\partial f_6}{\partial \alpha} & \frac{\partial f_6}{\partial \beta} & \frac{\partial f_6}{\partial \gamma} \end{bmatrix} \bullet \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix}$$

Noting the standard Jacobian expression,

$$\vec{v} = J\dot{\vec{q}} \quad \text{or} \quad \vec{v} = J\dot{\vec{l}}$$

$$\dot{\vec{l}} = J^{-1}\vec{v}$$

$$J^{-1} = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} & \frac{\partial f_1}{\partial \alpha} & \frac{\partial f_1}{\partial \beta} & \frac{\partial f_1}{\partial \gamma} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \frac{\partial \dot{f}_6}{\partial x} & \frac{\partial \dot{f}_6}{\partial y} & \frac{\partial \dot{f}_6}{\partial z} & \frac{\partial \dot{f}_6}{\partial \alpha} & \frac{\partial \dot{f}_6}{\partial \beta} & \frac{\partial \dot{f}_6}{\partial \gamma} \end{bmatrix}$$

The matrix above is the inverse of the conventional Jacobian. This facilitates the mapping of the Cartesian space velocity vector, $\vec{v}$, into the configuration space strut displacement rate vector, $\dot{\vec{q}} = \dot{\vec{l}}$.

Path Execution

Having dealt with the inverse kinematics and the inverse Jacobian of a Stewart platform, it is now appropriate to consider the practical application of executing an arbitrary trajectory in 3-space.

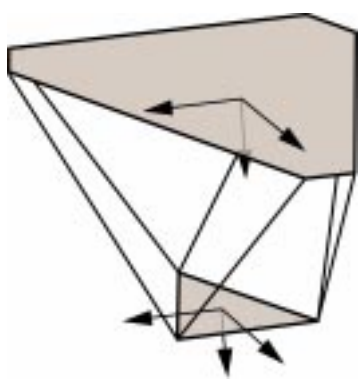


Figure 11. 6-3 node Stewart platform

Any arbitrary path in space can be represented by parametric relationships for the 6 Cartesian coordinates with respect to time, *t*.

$$\vec{p} = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \\ \alpha(t) \\ \beta(t) \\ \gamma(t) \end{bmatrix}$$

Where [*x(t), y(t), z(t)*] is the vector relating tool position along the path, and [$\alpha(t)$, $\beta(t)$, $\gamma(t)$] is the vector relating tool orientation along the path.

.

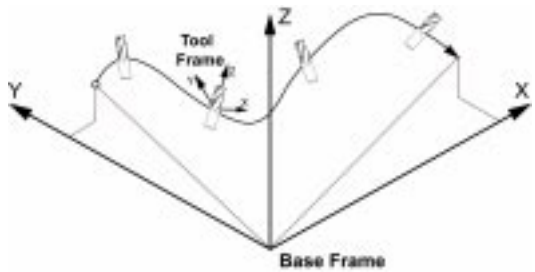


Figure 12. Tool path in Cartesian space

Given that,

$$
\begin{aligned}
l_1 &= f_1(x, y, z, \alpha, \beta, \gamma) \\
l_2 &= f_2(x, y, z, \alpha, \beta, \gamma) \\
&. \\
&. \\
l_6 &= f_6(x, y, z, \alpha, \beta, \gamma)
\end{aligned}
$$

we can substitute in the Cartesian time histories to obtain time functions for each of the strut lengths.

$$
\begin{aligned}
l_1(t) &= f_1(x(t), y(t), z(t), \alpha(t), \beta(t), \gamma(t)) \\
l_2(t) &= f_2(x(t), y(t), z(t), \alpha(t), \beta(t), \gamma(t)) \\
&. \\
&. \\
l_6(t) &= f_6(x(t), y(t), z(t), \alpha(t), \beta(t), \gamma(t))
\end{aligned}
$$

The displacement rates can be found by direct differentiation.

$$
\dot{\vec{l}} = \begin{bmatrix} \dot{l}_1(t) \\ . \\ . \\ . \\ \dot{l}_2(t) \\ \dot{l}_6(t) \end{bmatrix}
$$

Alternatively, using the inverse Jacobian, J^{-1},

$$
\dot{\vec{l}} = \begin{bmatrix} \dot{l}_1 \\ \dot{l}_2 \\ . \\ . \\ . \\ \dot{l}_6 \end{bmatrix} = \begin{bmatrix}
\frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} & \frac{\partial f_1}{\partial \alpha} & \frac{\partial f_1}{\partial \beta} & \frac{\partial f_1}{\partial \gamma} \\
\frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial z} & \frac{\partial f_2}{\partial \alpha} & \frac{\partial f_2}{\partial \beta} & \frac{\partial f_2}{\partial \gamma} \\
. & . & . & . & . & . \\
. & . & . & . & . & . \\
\frac{\partial f_6}{\partial x} & \frac{\partial f_6}{\partial y} & \frac{\partial f_6}{\partial z} & \frac{\partial f_6}{\partial \alpha} & \frac{\partial f_6}{\partial \beta} & \frac{\partial f_6}{\partial \gamma}
\end{bmatrix} \bullet \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \\ \dot{\alpha}(t) \\ \dot{\beta}(t) \\ \dot{\gamma}(t) \end{bmatrix} = J^{-1}\vec{p}
$$

Consequently, given a continuous tool path represented by a set of time functions for the six Cartesian coordinates, the set of strut lengths and displacement rates can be found directly. These lengths and displacement rates can be used as reference signals for the control of each individual strut.

In the case of piecewise linear tool path descriptions, where no single time function of each of the Cartesian coordinates exists, the strut lengths and displacement rates can still be found directly. In the case of a path described by a discrete set of "via" points, strut lengths would be calculated at the via points with simple linear interpolation of the strut displacements between via points (linear interpolation in "strut space" would not yield straight-line Cartesian motion between the via points). If specific Cartesian velocities are required at the via points these would be specified and the inverse Jacobian would be used to calculate strut displacement rates at the via points, corresponding to the specified Cartesian velocities.

Workspace Envelope

The workspace evaluation for the hexapod is divided into two parts. The first part seeks to find the workspace boundary associated with limits in strut travel. This is the lower workspace boundary.

A program written in MATLAB was used to search points in a rectangular grid to determine if they were inside or outside the workspace boundary, based on one or more struts exceeding their travel limits. Orientation of the tool head was maintained in the horizontal position ($\alpha=\beta=\gamma=0$). The same algorithm can be executed with other orientations of the tool head. See *Appendix A* for the program source code. A block diagram of the algorithm is shown in *Figure 13*, and sample output is shown in *Figure 14*.

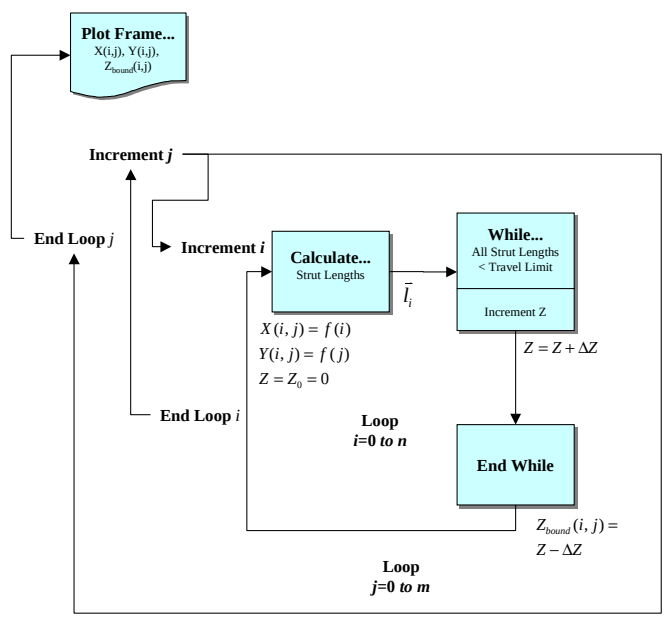


Figure 13. Lower workspace algorithm

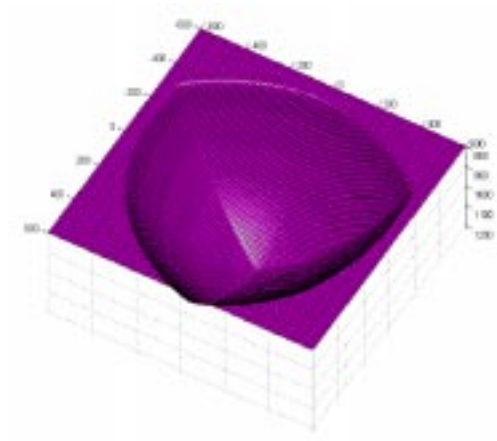


Figure 14. Lower workspace envelope plot

The second part seeks to find the workspace boundary determined by collisions between the struts and the top platform. This is the upper workspace boundary. A program written in MATLAB was used to search points in a rectangular grid to determine if they were inside or outside the workspace boundary, based on one or more struts exceeding an angle limit between the strut axis and the horizontal plane of the top platform. See *Appendix A* for the program source code. A block diagram of the algorithm is shown in *Figure 15*, and sample output is shown in *Figure 16*.

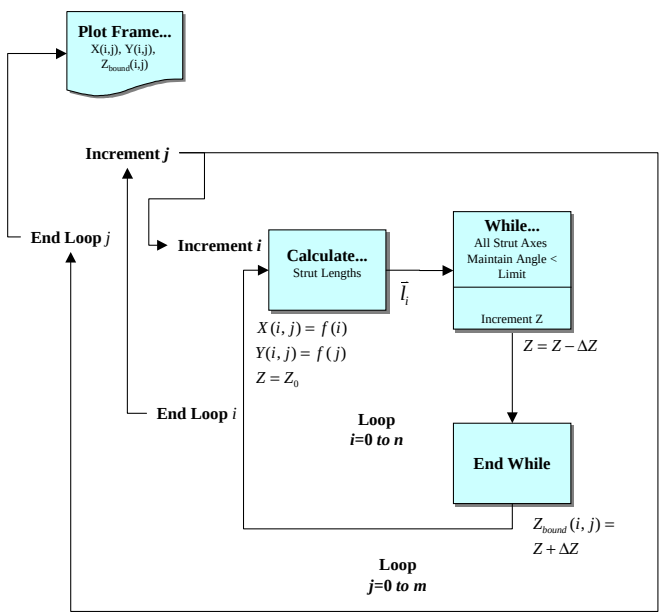


Figure 15. Upper workspace algorithm

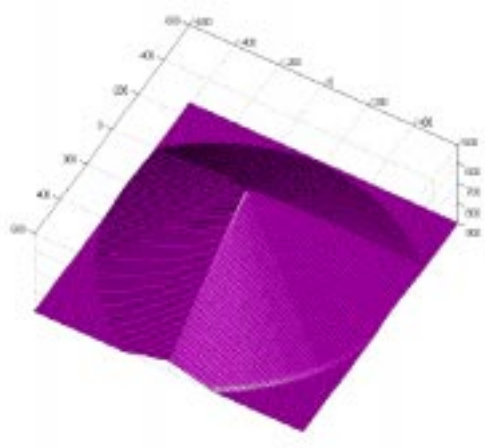


Figure 16. Upper workspace envelope plot

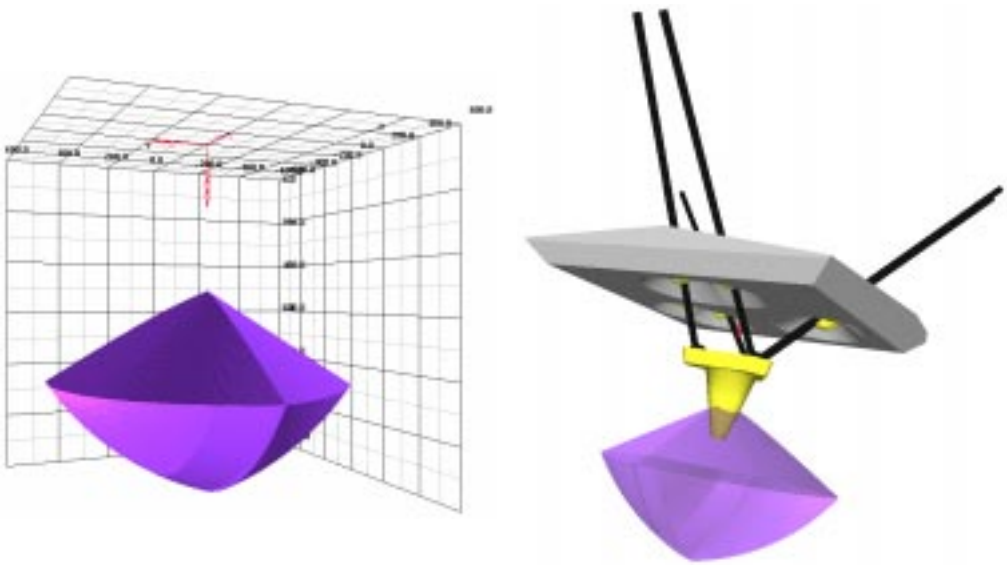


Figure 17. Total workspace envelope

Velocity Envelopes

A way of characterizing maximum achievable velocities of departure at different points in the hexapod workspace is sought. Envelopes are plotted indicating the maximum velocity in any direction, departing from a given point in the workspace.

A program written in MATLAB was used to sequence through a grid of unit vectors radiating from a specified point in the workspace. The inverse Jacobian was used to evaluate strut displacement rates associated with each of these unit velocity vectors. Based on one or more of the struts exceeding their displacement rates, a maximum Cartesian velocity in each unit direction was determined. See *Appendix B* for the program source code. A block diagram of the algorithm is shown in *Figure 18*. Sample output is shown in *Figure 19*.

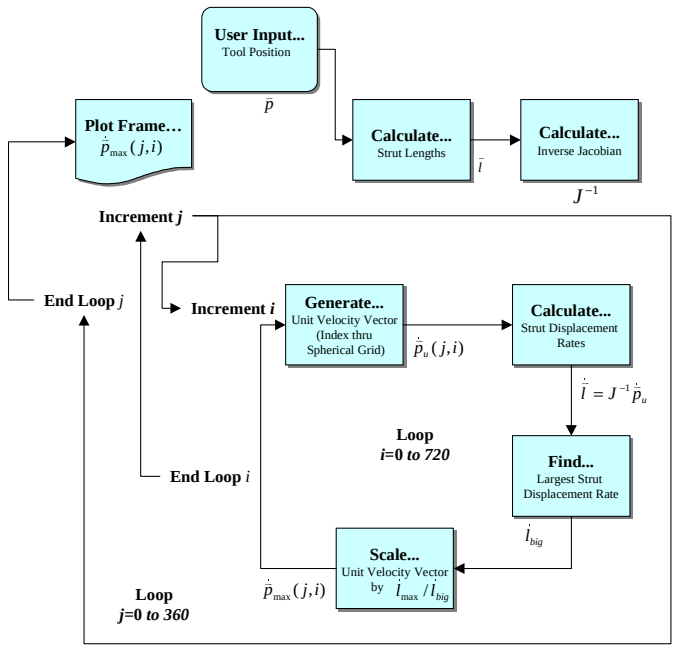


Figure 18. Velocity envelope algorithm

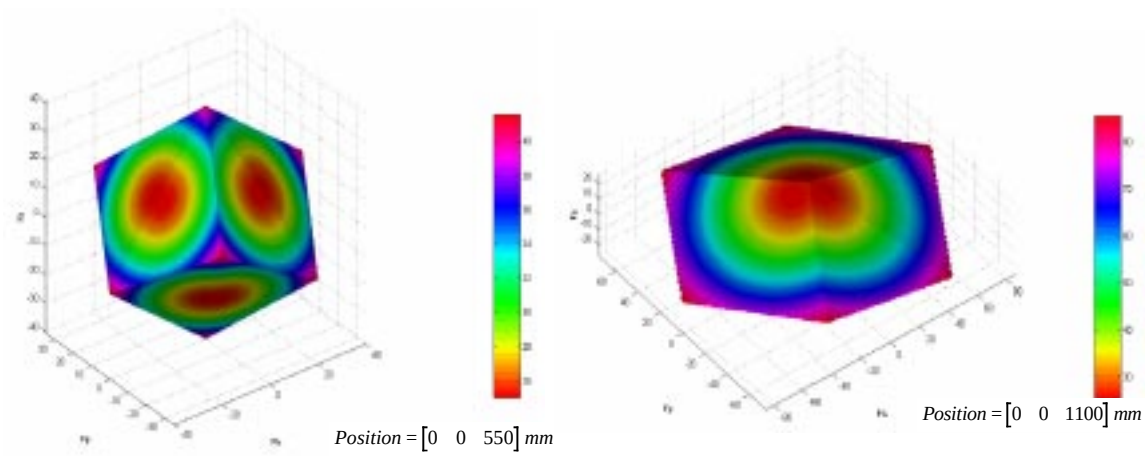


Figure 19. Velocity envelope plots

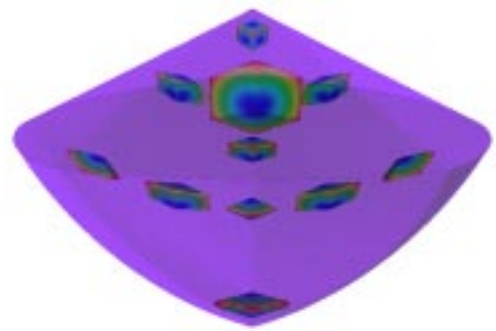


Figure 20. Velocity hypersurface plot showing variation of velocity envelopes within the workspace

Stiffness

Applying the principle of virtual work to an arbitrary mechanism allows us to equate work done in Cartesian space terms to work done in configuration space terms. Specifically, work in Cartesian terms is associated with a Cartesian force/torque vector, $\vec{F}$, applied at a mechanism's tool frame and acting through an infinitesimal Cartesian displacement, $\delta\vec{p}$. Work in configuration space terms is associated with a configuration space force/torque vector, $\vec{f}$, applied at a mechanism's joints and acting through infinitesimal joint displacements, $\delta\vec{q}$.

Work is calculated as the dot product of a force/torque vector with a displacement vector.

$$\vec{F} \bullet \delta\vec{p} = \vec{f} \bullet \delta\vec{q}$$

or,

$$\vec{F}^T \delta\vec{p} = \vec{f}^T \delta\vec{q}$$

and noting that, $\delta\vec{p} = J\delta\vec{q}$,

$$\vec{F}^T J\delta\vec{q} = \vec{f}^T \delta\vec{q}$$
$$\vec{F}^T J = \vec{f}^T$$

Transposing both sides we have,

$$(\vec{F}^T J)^T = \vec{f}$$
$$\vec{f} = J^T \vec{F}$$

From this we conclude that "actuating" a mechanism with a force/torque vector, $\vec{F}$, applied at the tool is equivalent to actuating that mechanism with a force/torque vector, $\vec{f}$, applied at the joints, when the same amount of virtual work is done in either case. For the hexapod we can relate an applied force, $\vec{F}$, at the tool to the resulting axial forces in the struts, $\vec{f}$.†

$$\vec{F} = J^{-T} \vec{f}$$

†. Positive values of *f* indicate struts in tension. Negative values indicate struts in compression.

Given pure axial loading,

$$\varepsilon = \delta l_i / l_i = \sigma / E = f_i / AE$$

$$f_i = (AE / l_i)\delta l_i$$

Where E is the elastic modulus of the strut material and A is the cross-sectional area of the strut.
In matrix form,

$$\vec{f} = \begin{bmatrix} AE/l_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & AE/l_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & AE/l_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & AE/l_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & AE/l_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & AE/l_6 \end{bmatrix} \delta\vec{l}$$

$$\vec{f} = K_s \delta\vec{l}$$

Where the matrix is identified as the strut space stiffness matrix, K_s. Noting that $\delta\vec{l} = J^{-1}\delta\vec{p}$,

$$\vec{f} = K_s J^{-1} \delta\vec{p}$$

and,

$$\vec{F} = J^{-T} K_s J^{-1} \delta\vec{p}$$
$$K_c = J^{-T} K_s J^{-1}$$
$$\vec{F} = K_c \delta\vec{p}$$

Where K_c is the Cartesian space stiffness matrix. Setting up an eigenvalue problem we can find principal stiffness axes, $\vec{\eta}_i$, and principal stiffnesses, λ_i.

$$\vec{F} = K_c \delta\vec{p} = \lambda_i \delta\vec{p}$$
$$(K_c - \lambda_i I_6)\delta\vec{p} = 0$$
$$\left| K_c - \lambda_i I_6 \right| = 0; \quad (i = 1 \text{ to } 6)$$

$\vec{\eta}_i$ is in the direction of $\delta\vec{p}_i$ where the above condition holds.

A program has been written in MATLAB which performs the stiffness calculations described in this section. See *Appendix C* for the program source code. Sample output is shown in *Figure 21*.

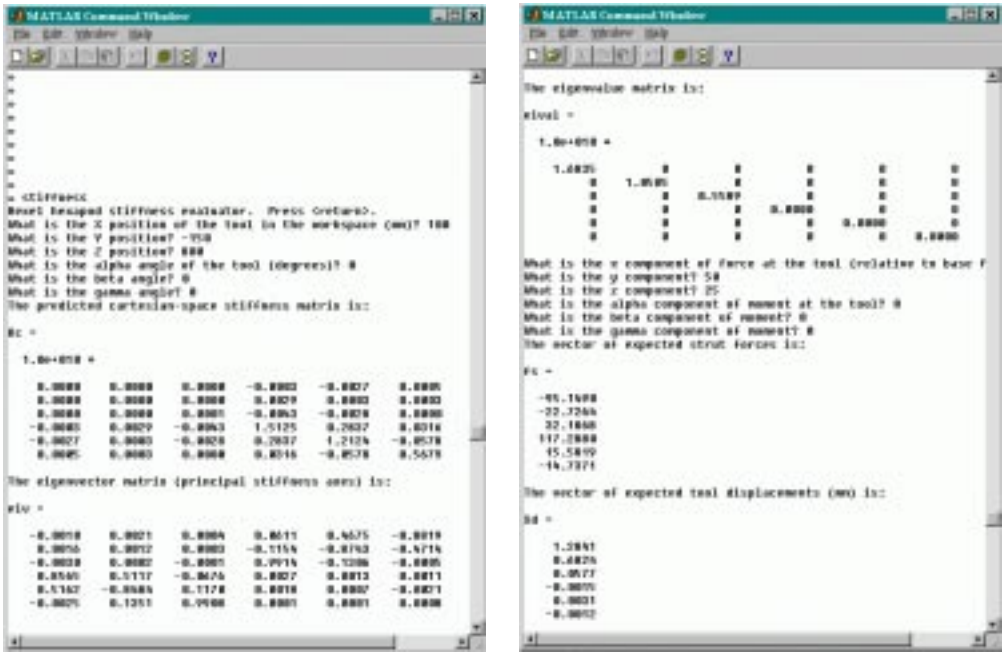


Figure 21. Stiffness sample output

Forward Kinematics

A numerical approach to solving the forward displacement problem will be presented here. Using the Jacobian we can determine an incremental Cartesian displacement of the tool frame, $\delta\vec{p}$, associated with an incremental set of strut displacements, $\delta\vec{l}$, since,

$$\delta\vec{p} = J\delta\vec{l}$$

From a numerical standpoint, error will accumulate based on using a finite step size, $\Delta\vec{l}$. Of course, the smaller step size we use the smaller the error. However, computation time will increase. A way of reducing the accumulation of numerical error associated with a given step size involves using the closed form inverse kinematics solution to evaluate the strut lengths, associated with the estimated tool position, at each iteration. The difference between the closed form calculated struts lengths and the set of goal strut lengths at a given iteration represents an error that can be accounted for in the $\Delta\vec{l}$ increment at the next iteration.

In this sense the algorithm is always aware of the strut lengths (calculated to double precision from the closed form equations) associated with the Jacobian-based estimate of the tool position at a given iteration. By compensating for any error, in the next $\Delta\vec{l}$ iteration we should maintain a numerical error that does not accumulate. A program has been written in MATLAB which performs the forward displacement computations. See *Appendix D* for the program source code. A block diagram of the algorithm is shown in *Figure 22*, and sample output is shown in *Figure 23*.

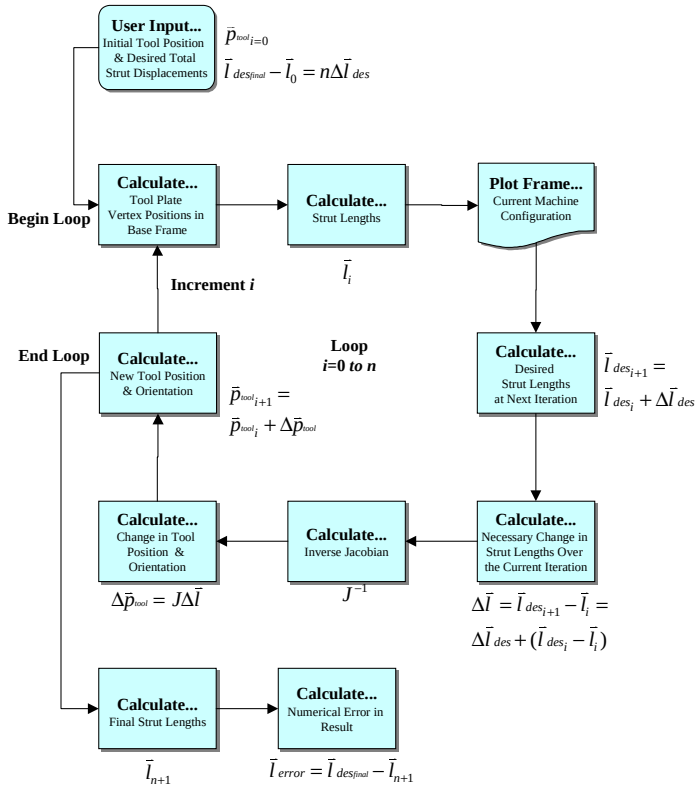


Figure 22. Forward displacement algorithm

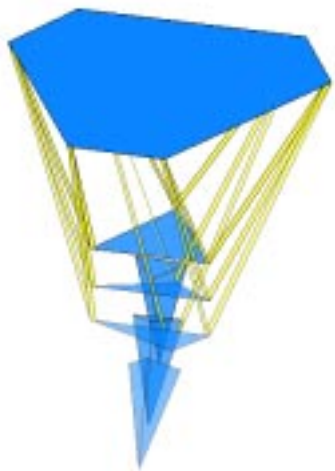


Figure 23. Forward displacement sample output (blur effect used to convey animation)

Conclusion

Stewart platform-based parallel mechanisms offer a great deal of potential, especially within the areas of manufacturing and factory automation. In order to realize this potential, a framework for designing and analyzing these mechanisms is essential. This report has presented elements of an analysis framework that can be applied to Stewart platforms. The integration of design (synthesis) techniques with analysis techniques offers the opportunity for improved hexapod designs in the future.

Additionally, the techniques presented in this report are amenable to other parallel architecture designs, which are variants of Stewart platforms. *Figure 24* depicts an articulated elbow variant known as a "rotopod". A prototype of this design is being created at Sandia (Bieg, et al). Also depicted is a variant known as a "hexaglide" which uses linear actuators to translate the top nodes of the platform. This design is being pursued at the Swiss Federal Institute of Technology. These designs represent two variant classes of parallel mechanisms with significant potential. An integrated synthesis/analysis approach is recommended in order to explore the design parameter space for a wide range of parallel mechanisms; in hopes of creating improved, if not optimal, parallel devices for any number of applications.

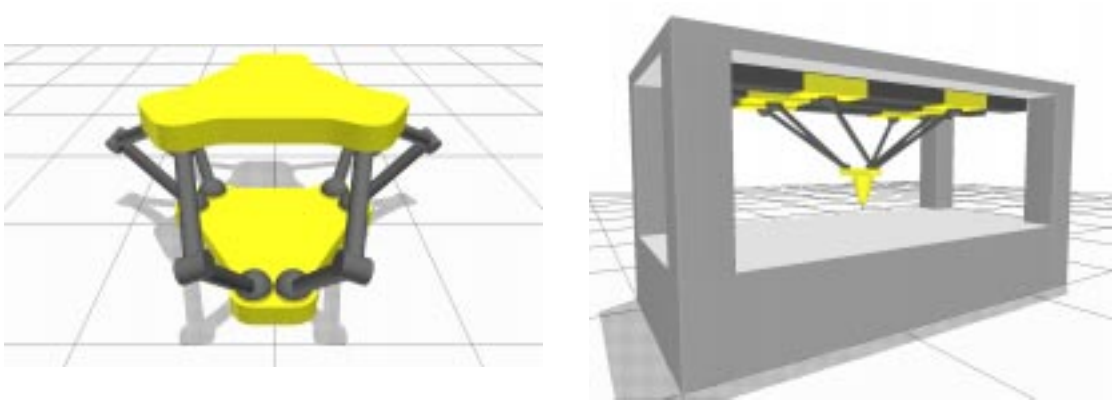


Figure 24. Variant designs

Modeling and computer simulation are becoming extremely powerful tools for conducting synthesis and analysis of mechanisms. In this report a number of numerical algorithms have been developed using MATLAB. Future work will be directed at expanding these algorithms and integrating them into an extensive parallel mechanism tool box (such toolboxes have been developed for serial arm robots[5]). *Figure 25* indicates some basic building blocks for such a toolbox.

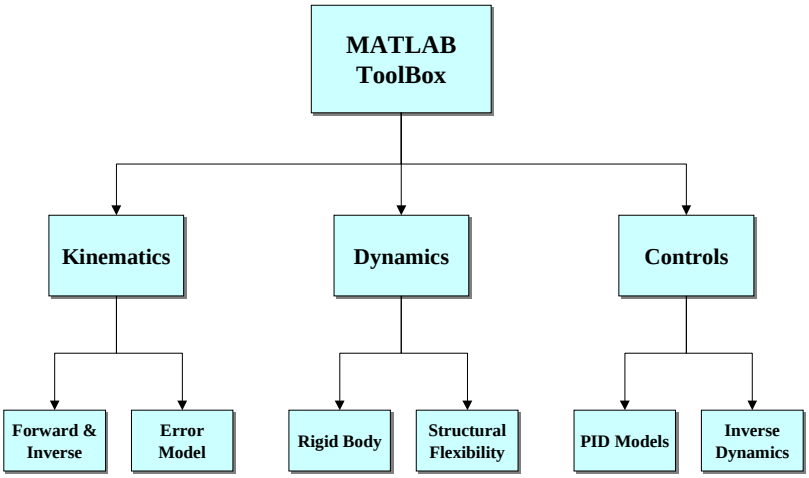


Figure 25. Parallel mechanism toolbox

Using this parallel mechanism toolbox as a developmental starting point, C code as well as Java would be generated and finally implemented via a web-based server/client or a high end simulation API/environmment. In the latter case, a product like Deneb's Envision TR or Division's dVISE would be employed. Both have high level API's for implementing custom source code. Such a development/implementation plan is depicted in *Figure 26*.

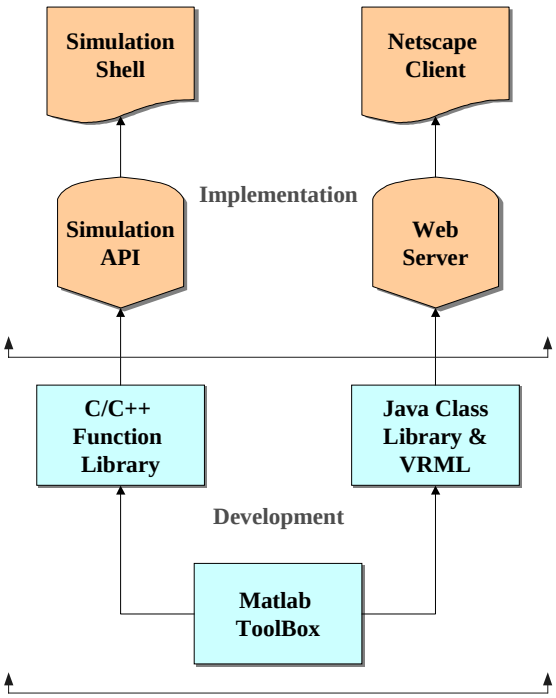


Figure 26. Simulation development/implementation

References

[1]D. Stewart, A Platform with Six Degrees of Freedom, Proceedings of the Institution of Mechanical Engineers, vol. 180, no. 15, May 1965.

[2]J.J. Craig, Introduction to Robotics: Mechanics and Control, Addison-Wesley, second edition, 1989.

[3]K. Liu, J.M. Fitzgerald, and F.L. Lewis, Kinematics Analysis of a Stewart Platform Manipulator, IEEE Transactions on Industrial Electronics, vol. 40, no. 2, April 1993.

[4]H. Yang and O. Masory, Vibration Analysis of Stewart-Platform, IEEE Transactions on Industrial Electronics, vol. 40, no. 2, April 1993.

[5]P. I. Corke, A Robotics Toolbox for MATLAB, IEEE Robotics & Automation Magazine, March 1996.

APPENDIX A: WORKSPACE ENVELOPE PLOTTER

```
% Partial workspace (lower envelope) plotter for a Hexel hexapod.
% Vince De SapioJanuary 16, 1997

% Initializing constants.

% Maximum strut displacement.
Lmax=970;

% Tool plate vertices (ball joint centers) in the tool coordinate sytem.
xt1=86.6;
yt1=150;
zt1=-305.1;
xt2=-173.2;
yt2=0;
zt2=-305.1;
xt3=86.6;
yt3=-150;
zt3=-305.1;

% Base plate vertices (ball joint centers) in the base coordinate sytem.
Xb1=400.68;
Yb1=122;
Xb2=-94.7;
Yb2=408;
Xb3=-306;
Yb3=286;
Xb4=-306;
Yb4=-286;
Xb5=-94.7;
Yb5=-408;
Xb6=400.7;
Yb6=-122;
Zb1=0;
Zb2=0;
Zb3=0;
Zb4=0;
Zb5=0;
Zb6=0;
a=0;
b=0;
c=0;

input('Hexel hexapod workspace evaluator.  Press <return>.');

% Index through a grid in X-Y plane.
for j=0:48

              for i=0:48

              % Initialize strut lengths.
              L(1)=0;
              L(2)=0;
              L(3)=0;
              L(4)=0;
              L(5)=0;
              L(6)=0;

              x(i+1,j+1)=(i-24)*25;
              y(i+1,j+1)=(j-24)*25;
```

```
            Xt1=x(i+1,j+1) + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt1 +
(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*yt1 + (cos(a)*sin(b))*zt1;
            Yt1=y(i+1,j+1) + (cos(a)*sin(c))*xt1 + (cos(a)*cos(c))*yt1 -
(sin(a))*zt1;
            Xt2=x(i+1,j+1) + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt2 +
(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*yt2 + (cos(a)*sin(b))*zt2;
            Yt2=y(i+1,j+1) + (cos(a)*sin(c))*xt2 + (cos(a)*cos(c))*yt2 -
(sin(a))*zt2;
            Xt3=x(i+1,j+1) + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt3 +
(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*yt3 + (cos(a)*sin(b))*zt3;
            Yt3=y(i+1,j+1) + (cos(a)*sin(c))*xt3 + (cos(a)*cos(c))*yt3 -
(sin(a))*zt3;

            za=0;

            % While conditions are satisfied such that current point is in the
workspace increment za.
            while (L(1)<Lmax & L(2)<Lmax & L(3)<Lmax & L(4)<Lmax & L(5)<Lmax &
L(6)<Lmax)
            za=za + 5;
            Zt1=za + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt1 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt1 + (cos(a)*cos(b))*zt1;
            Zt2=za + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt2 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt2 + (cos(a)*cos(b))*zt2;
            Zt3=za + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt3 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt3 + (cos(a)*cos(b))*zt3;

            % Calculate strut lengths.
            L(1)=sqrt((Xt1-Xb1)*(Xt1-Xb1) + (Yt1-Yb1)*(Yt1-Yb1) + (Zt1-Zb1)*(Zt1-
Zb1));
            L(2)=sqrt((Xt1-Xb2)*(Xt1-Xb2) + (Yt1-Yb2)*(Yt1-Yb2) + (Zt1-Zb2)*(Zt1-
Zb2));
            L(3)=sqrt((Xt2-Xb3)*(Xt2-Xb3) + (Yt2-Yb3)*(Yt2-Yb3) + (Zt2-Zb3)*(Zt2-
Zb3));
            L(4)=sqrt((Xt2-Xb4)*(Xt2-Xb4) + (Yt2-Yb4)*(Yt2-Yb4) + (Zt2-Zb4)*(Zt2-
Zb4));
            L(5)=sqrt((Xt3-Xb5)*(Xt3-Xb5) + (Yt3-Yb5)*(Yt3-Yb5) + (Zt3-Zb5)*(Zt3-
Zb5));
            L(6)=sqrt((Xt3-Xb6)*(Xt3-Xb6) + (Yt3-Yb6)*(Yt3-Yb6) + (Zt3-Zb6)*(Zt3-
Zb6));
            end

            % z value at workspace boundary.
            z(i+1,j+1)=za-5;
            end
end

index=0;
for j=0:48

            for i=0:48
            index=index+1;
            vect(index,1)=x(i+1,j+1);
            vect(index,2)=y(i+1,j+1);
            vect(index,3)=z(i+1,j+1);
            end
end
rect1=[210,350,423,350];
figure('Position',rect1)
```

```
whitebg('w')
axis('equal');
h=surf(x,y,z);
view([-60 45]);
upvector = [0 0 -1];
set(gca, 'CameraUpVector',upvector);
set(h,'FaceColor','magenta');
set(h,'FaceLighting','phong');
set(h,'SpecularExponent',1);
set(h,'AmbientStrength',8);
light('Position',[0 0 1800]);
material shiny;
axis('equal');
title([' Hexapod Lower Workspace Envelope
(mm)'],'FontName','times','FontAngle','italic','FontWeight','bold')
rect2=[640,350,350,350];
figure('Position',rect2);
h=surf(x,y,z);
view([-60 45]);
upvector = [0 0 -1];
set(gca, 'CameraUpVector',upvector);
shading interp;
set(h,'FaceColor','magenta');
set(h,'FaceLighting','phong');
set(h,'SpecularExponent',1);
set(h,'AmbientStrength',8);
light('Position',[0 0 1800]);
material shiny;
axis('equal');
axis('off');
ax(1)=-1;
ax(2)=1;
ax(3)=0;
M=moviein(24);
for q=1:24
             rotate(h,ax,15)
             M(:,q)=getframe;
end
axis('equal');
axis('off');
movie(M,15);
```

APPENDIX A: WORKSPACE ENVELOPE PLOTTER

```
% Partial workspace (upper envelope) plotter for a Hexel hexapod.
% Vince De SapioFebruary 24, 1997

% Initializing constants.

% Maximum strut displacement.
Lmax=970;

% SIN of angle at which strut collides with upper stationary platform.
sin_ang=.55;

% Z value at which tool platform collides with upper stationary platform.
zmin=454;

% Tool plate vertices (ball joint centers) in the tool coordinate sytem.
xt1=86.6;
yt1=150;
zt1=-305.1;
xt2=-173.2;
yt2=0;
zt2=-305.1;
xt3=86.6;
yt3=-150;
zt3=-305.1;

% Base plate vertices (ball joint centers) in the base coordinate sytem.
Xb1=400.68;
Yb1=122;
Xb2=-94.7;
Yb2=408;
Xb3=-306;
Yb3=286;
Xb4=-306;
Yb4=-286;
Xb5=-94.7;
Yb5=-408;
Xb6=400.7;
Yb6=-122;
Zb1=0;
Zb2=0;
Zb3=0;
Zb4=0;
Zb5=0;
Zb6=0;
a=0;
b=0;
c=0;

input('Hexel hexapod workspace evaluator.  Press <return>.');

% Index through a grid in X-Y plane.
for j=0:48

              for i=0:48

              % Initial Z value.
              za=1800;

              % Initialize strut lengths.
```

```
            L(1)=0;
            L(2)=0;
            L(3)=0;
            L(4)=0;
            L(5)=0;
            L(6)=0;

            x(i+1,j+1)=(i-24)*25;
            y(i+1,j+1)=(j-24)*25;
            Xt1=x(i+1,j+1) + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt1 +
(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*yt1 + (cos(a)*sin(b))*zt1;
            Yt1=y(i+1,j+1) + (cos(a)*sin(c))*xt1 + (cos(a)*cos(c))*yt1 -
(sin(a))*zt1;
            Xt2=x(i+1,j+1) + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt2 +
(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*yt2 + (cos(a)*sin(b))*zt2;
            Yt2=y(i+1,j+1) + (cos(a)*sin(c))*xt2 + (cos(a)*cos(c))*yt2 -
(sin(a))*zt2;
            Xt3=x(i+1,j+1) + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt3 +
(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*yt3 + (cos(a)*sin(b))*zt3;
            Yt3=y(i+1,j+1) + (cos(a)*sin(c))*xt3 + (cos(a)*cos(c))*yt3 -
(sin(a))*zt3;
            Zt1=za + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt1 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt1 + (cos(a)*cos(b))*zt1;
            Zt2=za + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt2 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt2 + (cos(a)*cos(b))*zt2;
            Zt3=za + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt3 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt3 + (cos(a)*cos(b))*zt3;

            % Calculate strut lengths.
            L(1)=sqrt((Xt1-Xb1)*(Xt1-Xb1) + (Yt1-Yb1)*(Yt1-Yb1) + (Zt1-Zb1)*(Zt1-
Zb1));
            L(2)=sqrt((Xt1-Xb2)*(Xt1-Xb2) + (Yt1-Yb2)*(Yt1-Yb2) + (Zt1-Zb2)*(Zt1-
Zb2));
            L(3)=sqrt((Xt2-Xb3)*(Xt2-Xb3) + (Yt2-Yb3)*(Yt2-Yb3) + (Zt2-Zb3)*(Zt2-
Zb3));
            L(4)=sqrt((Xt2-Xb4)*(Xt2-Xb4) + (Yt2-Yb4)*(Yt2-Yb4) + (Zt2-Zb4)*(Zt2-
Zb4));
            L(5)=sqrt((Xt3-Xb5)*(Xt3-Xb5) + (Yt3-Yb5)*(Yt3-Yb5) + (Zt3-Zb5)*(Zt3-
Zb5));
            L(6)=sqrt((Xt3-Xb6)*(Xt3-Xb6) + (Yt3-Yb6)*(Yt3-Yb6) + (Zt3-Zb6)*(Zt3-
Zb6));

            % While conditions are satisfied such that current point is in the
workspace increment za.
            while ((Zt1-Zb1)/L(1)>sin_ang & ((Zt1-Zb2)/L(2))>sin_ang & ((Zt2-Zb3)/
L(3))>sin_ang & ((Zt2-Zb4)/L(4))>sin_ang & ((Zt3-Zb5)/L(5))>sin_ang & ((Zt3-Zb6)/
L(6))>sin_ang & za>zmin)
            za=za - 10;
            Zt1=za + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt1 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt1 + (cos(a)*cos(b))*zt1;
            Zt2=za + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt2 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt2 + (cos(a)*cos(b))*zt2;
            Zt3=za + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt3 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt3 + (cos(a)*cos(b))*zt3;
            L(1)=sqrt((Xt1-Xb1)*(Xt1-Xb1) + (Yt1-Yb1)*(Yt1-Yb1) + (Zt1-Zb1)*(Zt1-
Zb1));
            L(2)=sqrt((Xt1-Xb2)*(Xt1-Xb2) + (Yt1-Yb2)*(Yt1-Yb2) + (Zt1-Zb2)*(Zt1-
Zb2));
```

```
            L(3)=sqrt((Xt2-Xb3)*(Xt2-Xb3) + (Yt2-Yb3)*(Yt2-Yb3) + (Zt2-Zb3)*(Zt2-
Zb3));
            L(4)=sqrt((Xt2-Xb4)*(Xt2-Xb4) + (Yt2-Yb4)*(Yt2-Yb4) + (Zt2-Zb4)*(Zt2-
Zb4));
            L(5)=sqrt((Xt3-Xb5)*(Xt3-Xb5) + (Yt3-Yb5)*(Yt3-Yb5) + (Zt3-Zb5)*(Zt3-
Zb5));
            L(6)=sqrt((Xt3-Xb6)*(Xt3-Xb6) + (Yt3-Yb6)*(Yt3-Yb6) + (Zt3-Zb6)*(Zt3-
Zb6));
            end

            % z value at workspace boundary.
            z(i+1,j+1)=za+10;
            end
end

index=0;
for j=0:48

            for i=0:48
            index=index+1;
            vect(index,1)=x(i+1,j+1);
            vect(index,2)=y(i+1,j+1);
            vect(index,3)=z(i+1,j+1);
            end
end
rect1=[210,350,423,350];
figure('Position',rect1)
whitebg('w')
axis('equal');
h=surf(x,y,z);
view([-60 45]);
upvector = [0 0 -1];
set(gca, 'CameraUpVector',upvector);
set(h,'FaceColor','magenta');
set(h,'FaceLighting','phong');
set(h,'SpecularExponent',1);
set(h,'AmbientStrength',8);
light('Position',[0 0 1800]);
material shiny;
axis('equal');
title([' Hexapod Upper Workspace Envelope
(mm)'],'FontName','times','FontAngle','italic','FontWeight','bold')
rect2=[640,350,350,350];
figure('Position',rect2);
h=surf(x,y,z);
view([-60 45]);
upvector = [0 0 -1];
set(gca, 'CameraUpVector',upvector);
shading interp;
set(h,'FaceColor','magenta');
set(h,'FaceLighting','phong');
set(h,'SpecularExponent',1);
set(h,'AmbientStrength',8);
light('Position',[0 0 1800]);
material shiny;
axis('equal');
axis('off');
ax(1)=-1;
ax(2)=1;
```

```
ax(3)=0;
M=moviein(24);
for q=1:24
             rotate(h,ax,15)
             M(:,q)=getframe;
end
axis(’equal’);
axis(’off’);
movie(M,15);
```

APPENDIX B: VELOCITY ENVELOPE PLOTTER

```
% Velocity envelope plotter for a Hexel hexapod.
% Vince De SapioJune 22, 1996
% Updated    October 4, 1996

% Initializing constants.

% Maximum strut displacement rate.
Ldotmax=25;
% Maximum strut displacement.
Lmax=970;

Ldot=0;
Pdot=0;
xdot=0;
ydot=0;
zdot=0;
x2dot=0;
y2dot=0;
z2dot=0;

% Tool plate vertices (ball joint centers) in the tool coordinate sytem.
xt1=86.6;
yt1=150;
zt1=-305.1;
xt2=-173.2;
yt2=0;
zt2=-305.1;
xt3=86.6;
yt3=-150;
zt3=-305.1;

% Base plate vertices (ball joint centers) in the base coordinate sytem.
Xb1=400.68;
Yb1=122;
Xb2=-94.7;
Yb2=408;
Xb3=-306;
Yb3=286;
Xb4=-306;
Yb4=-286;
Xb5=-94.7;
Yb5=-408;
Xb6=400.7;
Yb6=-122;
Zb1=0;
Zb2=0;
Zb3=0;
Zb4=0;
Zb5=0;
Zb6=0;

input('Hexel hexapod velocity envelope plotter.  Press <return>.');
x=input('What is the X position of the tool in the workspace (mm)? ');
y=input('What is the Y position? ');
z=input('What is the Z position? ');
adeg=input('What is the alpha angle of the tool (degrees)? ');
bdeg=input('What is the beta angle? ');
cdeg=input('What is the gamma angle? ');
a=pi*adeg/180;
```

```
b=pi*bdeg/180;
c=pi*cdeg/180;

% Hexapod inverse Jacobian
% Values for inverse Jacobian calculation
Xt1=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt1 + (sin(a)*sin(b)*cos(c) -
cos(b)*sin(c))*yt1 + (cos(a)*sin(b))*zt1;
Yt1=y + (cos(a)*sin(c))*xt1 + (cos(a)*cos(c))*yt1 - (sin(a))*zt1;
Zt1=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt1 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt1 + (cos(a)*cos(b))*zt1;
Xt2=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt2 + (sin(a)*sin(b)*cos(c) -
cos(b)*sin(c))*yt2 + (cos(a)*sin(b))*zt2;
Yt2=y + (cos(a)*sin(c))*xt2 + (cos(a)*cos(c))*yt2 - (sin(a))*zt2;
Zt2=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt2 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt2 + (cos(a)*cos(b))*zt2;
Xt3=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt3 + (sin(a)*sin(b)*cos(c) -
cos(b)*sin(c))*yt3 + (cos(a)*sin(b))*zt3;
Yt3=y + (cos(a)*sin(c))*xt3 + (cos(a)*cos(c))*yt3 - (sin(a))*zt3;
Zt3=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt3 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt3 + (cos(a)*cos(b))*zt3;
dXt1a=(cos(a)*sin(b)*sin(c))*xt1 + (cos(a)*sin(b)*cos(c))*yt1 - (sin(a)*sin(b))*zt1;
dYt1a=-(sin(a)*sin(c))*xt1 - (sin(a)*cos(c))*yt1 - (cos(a))*zt1;
dZt1a=(cos(a)*cos(b)*sin(c))*xt1 + (cos(a)*cos(b)*cos(c))*yt1 - (sin(a)*cos(b))*zt1;
dXt2a=(cos(a)*sin(b)*sin(c))*xt2 + (cos(a)*sin(b)*cos(c))*yt2 - (sin(a)*sin(b))*zt2;
dYt2a=-(sin(a)*sin(c))*xt2 - (sin(a)*cos(c))*yt2 - (cos(a))*zt2;
dZt2a=(cos(a)*cos(b)*sin(c))*xt2 + (cos(a)*cos(b)*cos(c))*yt2 - (sin(a)*cos(b))*zt2;
dXt3a=(cos(a)*sin(b)*sin(c))*xt3 + (cos(a)*sin(b)*cos(c))*yt3 - (sin(a)*sin(b))*zt3;
dYt3a=-(sin(a)*sin(c))*xt3 - (sin(a)*cos(c))*yt3 - (cos(a))*zt3;
dZt3a=(cos(a)*cos(b)*sin(c))*xt3 + (cos(a)*cos(b)*cos(c))*yt3 - (sin(a)*cos(b))*zt3;
dXt1b=(sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt1 + (sin(a)*cos(b)*cos(c) +
sin(b)*sin(c))*yt1 + (cos(a)*cos(b))*zt1;
dYt1b=0;
dZt1b=-(sin(a)*sin(b)*sin(c) + cos(b)*cos(c))*xt1 + (cos(b)*sin(c) -
sin(a)*sin(b)*cos(c))*yt1 - (cos(a)*sin(b))*zt1;
dXt2b=(sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt2 + (sin(a)*cos(b)*cos(c) +
sin(b)*sin(c))*yt2 + (cos(a)*cos(b))*zt2;
dYt2b=0;
dZt2b=-(sin(a)*sin(b)*sin(c) + cos(b)*cos(c))*xt2 + (cos(b)*sin(c) -
sin(a)*sin(b)*cos(c))*yt2 - (cos(a)*sin(b))*zt2;
dXt3b=(sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt3 + (sin(a)*cos(b)*cos(c) +
sin(b)*sin(c))*yt3 + (cos(a)*cos(b))*zt3;
dYt3b=0;
dZt3b=-(sin(a)*sin(b)*sin(c) + cos(b)*cos(c))*xt3 + (cos(b)*sin(c) -
sin(a)*sin(b)*cos(c))*yt3 - (cos(a)*sin(b))*zt3;
dXt1c=(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*xt1 - (cos(b)*cos(c) +
sin(a)*sin(b)*sin(c))*yt1;
dYt1c=(cos(a)*cos(c))*xt1 - (cos(a)*sin(c))*yt1;
dZt1c=(sin(a)*cos(b)*cos(c) + sin(b)*sin(c))*xt1 + (sin(b)*cos(c) -
sin(a)*cos(b)*sin(c))*yt1;
dXt2c=(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*xt2 - (cos(b)*cos(c) +
sin(a)*sin(b)*sin(c))*yt2;
dYt2c=(cos(a)*cos(c))*xt2 - (cos(a)*sin(c))*yt2;
dZt2c=(sin(a)*cos(b)*cos(c) + sin(b)*sin(c))*xt2 + (sin(b)*cos(c) -
sin(a)*cos(b)*sin(c))*yt2;
dXt3c=(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*xt3 - (cos(b)*cos(c) +
sin(a)*sin(b)*sin(c))*yt3;
dYt3c=(cos(a)*cos(c))*xt3 - (cos(a)*sin(c))*yt3;
dZt3c=(sin(a)*cos(b)*cos(c) + sin(b)*sin(c))*xt3 + (sin(b)*cos(c) -
sin(a)*cos(b)*sin(c))*yt3;
```

```
% Strut lengths
L(1)=sqrt((Xt1-Xb1)*(Xt1-Xb1) + (Yt1-Yb1)*(Yt1-Yb1) + (Zt1-Zb1)*(Zt1-Zb1));
L(2)=sqrt((Xt1-Xb2)*(Xt1-Xb2) + (Yt1-Yb2)*(Yt1-Yb2) + (Zt1-Zb2)*(Zt1-Zb2));
L(3)=sqrt((Xt2-Xb3)*(Xt2-Xb3) + (Yt2-Yb3)*(Yt2-Yb3) + (Zt2-Zb3)*(Zt2-Zb3));
L(4)=sqrt((Xt2-Xb4)*(Xt2-Xb4) + (Yt2-Yb4)*(Yt2-Yb4) + (Zt2-Zb4)*(Zt2-Zb4));
L(5)=sqrt((Xt3-Xb5)*(Xt3-Xb5) + (Yt3-Yb5)*(Yt3-Yb5) + (Zt3-Zb5)*(Zt3-Zb5));
L(6)=sqrt((Xt3-Xb6)*(Xt3-Xb6) + (Yt3-Yb6)*(Yt3-Yb6) + (Zt3-Zb6)*(Zt3-Zb6));

% Check if workspace envelope is exceeded.
if L(1)>Lmax
             input('You specified a position outside of the workspace envelope (strut
1 exceeding travel limit)');
end
if L(2)>Lmax
             input('You specified a position outside of the workspace envelope (strut
2 exceeding travel limit)');
end
if L(3)>Lmax
             input('You specified a position outside of the workspace envelope (strut
3 exceeding travel limit)');
end
if L(4)>Lmax
             input('You specified a position outside of the workspace envelope (strut
4 exceeding travel limit)');
end
if L(5)>Lmax
             input('You specified a position outside of the workspace envelope (strut
5 exceeding travel limit)');
end
if L(6)>Lmax
             input('You specified a position outside of the workspace envelope (strut
6 exceeding travel limit)');
end

% Hexapod inverse Jacobian with respect to the base frame
J(1,1)=(Xt1-Xb1)/L(1);
J(2,1)=(Xt1-Xb2)/L(2);
J(3,1)=(Xt2-Xb3)/L(3);
J(4,1)=(Xt2-Xb4)/L(4);
J(5,1)=(Xt3-Xb5)/L(5);
J(6,1)=(Xt3-Xb6)/L(6);
J(1,2)=(Yt1-Yb1)/L(1);
J(2,2)=(Yt1-Yb2)/L(2);
J(3,2)=(Yt2-Yb3)/L(3);
J(4,2)=(Yt2-Yb4)/L(4);
J(5,2)=(Yt3-Yb5)/L(5);
J(6,2)=(Yt3-Yb6)/L(6);
J(1,3)=(Zt1-Zb1)/L(1);
J(2,3)=(Zt1-Zb2)/L(2);
J(3,3)=(Zt2-Zb3)/L(3);
J(4,3)=(Zt2-Zb4)/L(4);
J(5,3)=(Zt3-Zb5)/L(5);
J(6,3)=(Zt3-Zb6)/L(6);
J(1,4)=((Xt1-Xb1)*dXt1a + (Yt1-Yb1)*dYt1a + (Zt1-Zb1)*dZt1a)/L(1);
J(2,4)=((Xt1-Xb2)*dXt1a + (Yt1-Yb2)*dYt1a + (Zt1-Zb2)*dZt1a)/L(2);
J(3,4)=((Xt2-Xb3)*dXt2a + (Yt2-Yb3)*dYt2a + (Zt2-Zb3)*dZt2a)/L(3);
J(4,4)=((Xt2-Xb4)*dXt2a + (Yt2-Yb4)*dYt2a + (Zt2-Zb4)*dZt2a)/L(4);
J(5,4)=((Xt3-Xb5)*dXt3a + (Yt3-Yb5)*dYt3a + (Zt3-Zb5)*dZt3a)/L(5);
```

```
J(6,4)=((Xt3-Xb6)*dXt3a + (Yt3-Yb6)*dYt3a + (Zt3-Zb6)*dZt3a)/L(6);
J(1,5)=((Xt1-Xb1)*dXt1b + (Yt1-Yb1)*dYt1b + (Zt1-Zb1)*dZt1b)/L(1);
J(2,5)=((Xt1-Xb2)*dXt1b + (Yt1-Yb2)*dYt1b + (Zt1-Zb2)*dZt1b)/L(2);
J(3,5)=((Xt2-Xb3)*dXt2b + (Yt2-Yb3)*dYt2b + (Zt2-Zb3)*dZt2b)/L(3);
J(4,5)=((Xt2-Xb4)*dXt2b + (Yt2-Yb4)*dYt2b + (Zt2-Zb4)*dZt2b)/L(4);
J(5,5)=((Xt3-Xb5)*dXt3b + (Yt3-Yb5)*dYt3b + (Zt3-Zb5)*dZt3b)/L(5);
J(6,5)=((Xt3-Xb6)*dXt3b + (Yt3-Yb6)*dYt3b + (Zt3-Zb6)*dZt3b)/L(6);
J(1,6)=((Xt1-Xb1)*dXt1c + (Yt1-Yb1)*dYt1c + (Zt1-Zb1)*dZt1c)/L(1);
J(2,6)=((Xt1-Xb2)*dXt1c + (Yt1-Yb2)*dYt1c + (Zt1-Zb2)*dZt1c)/L(2);
J(3,6)=((Xt2-Xb3)*dXt2c + (Yt2-Yb3)*dYt2c + (Zt2-Zb3)*dZt2c)/L(3);
J(4,6)=((Xt2-Xb4)*dXt2c + (Yt2-Yb4)*dYt2c + (Zt2-Zb4)*dZt2c)/L(4);
J(5,6)=((Xt3-Xb5)*dXt3c + (Yt3-Yb5)*dYt3c + (Zt3-Zb5)*dZt3c)/L(5);
J(6,6)=((Xt3-Xb6)*dXt3c + (Yt3-Yb6)*dYt3c + (Zt3-Zb6)*dZt3c)/L(6);

% Genetrating a spherical grid of unit velocity vectors.
for j=0:60

              for i=0:120
                           xdot(j+1,i+1)=cos(pi*i/60)*cos(pi*j/60);
                           ydot(j+1,i+1)=cos(pi*i/60)*sin(pi*j/60);
                           zdot(j+1,i+1)=sin(pi*i/60);
                           Pdot(1,1)=xdot(j+1,i+1);
                           Pdot(2,1)=ydot(j+1,i+1);
                           Pdot(3,1)=zdot(j+1,i+1);
                           Pdot(4,1)=0;
                           Pdot(5,1)=0;
                           Pdot(6,1)=0;

                           % Strut displacement rates associated with unit velocity
vectors.
                           Ldot=J*Pdot;

                           % Finding the largest absolute strut displacement rate.
                           big=abs(Ldot(1));

                           if abs(Ldot(2))>big
                                        big=abs(Ldot(2));
                           end

                           if abs(Ldot(3))>big
                                        big=abs(Ldot(3));
                           end

                           if abs(Ldot(4))>big
                                        big=abs(Ldot(4));
                           end

                           if abs(Ldot(5))>big
                                        big=abs(Ldot(5));
                           end

                           if abs(Ldot(6))>big
                                        big=abs(Ldot(6));
                           end

                           % Scaling unit velocity vector to yield maximum strut
displacement rate.
                           k=Ldotmax/big;
                           x2dot(j+1,i+1)=k*xdot(j+1,i+1);
```

```
                          y2dot(j+1,i+1)=k*ydot(j+1,i+1);
      z2dot(j+1,i+1)=k*zdot(j+1,i+1);

m2dot(j+1,i+1)=sqrt(x2dot(j+1,i+1)*x2dot(j+1,i+1)+y2dot(j+1,i+1)*y2dot(j+1,i+1)+z2dot(
j+1,i+1)*z2dot(j+1,i+1));
             end

end

ndex=0;
for i=0:120

             for j=0:60
             index=index+1;
             vect(index,1)=x2dot(j+1,i+1);
             vect(index,2)=y2dot(j+1,i+1);
   vect(index,3)=z2dot(j+1,i+1);
   vect(index,4)=m2dot(j+1,i+1);
             end
end

rect1=[210,350,423,350];
figure('Position',rect1)
whitebg('w')
axis('equal');
h=surf(x2dot,y2dot,z2dot,m2dot);
colormap hsv;
shading interp;
colorbar2;
set(h,'FaceLighting','phong');
set(h,'SpecularExponent',1);
set(h,'AmbientStrength',10);
light('Position',[-300 -300 300]);
material shiny;
axis('equal');
title([' Hexapod Velocity Envelope (mm/s); at Position
[',num2str(x),',',num2str(y),',',num2str(z),']'],'FontName','times','FontAngle','itali
c','FontWeight','bold')
xlabel('Vx','FontName','times','FontWeight','bold','FontAngle','italic');
ylabel('Vy','FontName','times','FontAngle','italic');
zlabel('Vz','FontName','times','FontAngle','italic');
rect2=[640,350,350,350];
figure('Position',rect2);
h=surf(x2dot,y2dot,z2dot,m2dot);
colormap hsv;
shading interp;
set(h,'FaceLighting','phong');
set(h,'SpecularExponent',1);
set(h,'AmbientStrength',10);
light('Position',[-300 -300 300]);
material shiny;
axis('equal');
axis('off');
ax(1)=0;
ax(2)=0;
ax(3)=1;
M=moviein(24);
for q=1:24
             rotate(h,ax,15)
```

```
            M(:,q)=getframe;
end

axis('equal');
axis('off');
movie(M,15);

Vx=input('What is the x component of the desired tool velocity (relative to base
frame, in mm/s)? ');
Vy=input('What is the y component? ');
Vz=input('What is the z component? ');
Radeg=input('What is the alpha component of desired tool rotation rate (relative to
base frame, in deg/s)? ');
Rbdeg=input('What is the beta component? ');
Rcdeg=input('What is the gamma component? ');
Ra=pi*Radeg/180;
Rb=pi*Rbdeg/180;
Rc=pi*Rcdeg/180;

Vc(1)=Vx;
Vc(2)=Vy;
Vc(3)=Vz;
Vc(4)=Ra;
Vc(5)=Rb;
Vc(6)=Rc;

% Required strut displacement rates.
Lrate=J*Vc';
input('The vector of required strut displacement rates is: ');Lrate
```

APPENDIX C: STIFFNESS EVALUATOR

```
% Stiffness evaluator for a Hexel hexapod.
% Vince De SapioOctober 3, 1996; Modified October 17, 1997

% Initializing constants.

% Maximum strut displacement.
Lmax=1500;

% Tool plate vertices (ball joint centers) in the tool coordinate sytem.
xt1=86.6;
yt1=150;
zt1=-305.1;
xt2=-173.2;
yt2=0;
zt2=-305.1;
xt3=86.6;
yt3=-150;
zt3=-305.1;

% Base plate vertices (ball joint centers) in the base coordinate sytem.
Xb1=400.68;
Yb1=122;
Xb2=-94.7;
Yb2=408;
Xb3=-306;
Yb3=286;
Xb4=-306;
Yb4=-286;
Xb5=-94.7;
Yb5=-408;
Xb6=400.7;
Yb6=-122;
Zb1=0;
Zb2=0;
Zb3=0;
Zb4=0;
Zb5=0;
Zb6=0;

% Strut Cross-sectional area and elastic modulus
Area=.00042333;
EM=203250000000;

input('Hexel hexapod stiffness evaluator.  Press <return>.');
x=input('What is the X position of the tool in the workspace (mm)? ');
y=input('What is the Y position? ');
z=input('What is the Z position? ');
adeg=input('What is the alpha angle of the tool (degrees)? ');
bdeg=input('What is the beta angle? ');
cdeg=input('What is the gamma angle? ');
a=pi*adeg/180;
b=pi*bdeg/180;
c=pi*cdeg/180;

% Values for inverse Jacobian calculation
Xt1=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt1 + (sin(a)*sin(b)*cos(c) -
cos(b)*sin(c))*yt1 + (cos(a)*sin(b))*zt1;
Yt1=y + (cos(a)*sin(c))*xt1 + (cos(a)*cos(c))*yt1 - (sin(a))*zt1;
```

```
Zt1=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt1 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt1 + (cos(a)*cos(b))*zt1;
Xt2=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt2 + (sin(a)*sin(b)*cos(c) -
cos(b)*sin(c))*yt2 + (cos(a)*sin(b))*zt2;
Yt2=y + (cos(a)*sin(c))*xt2 + (cos(a)*cos(c))*yt2 - (sin(a))*zt2;
Zt2=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt2 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt2 + (cos(a)*cos(b))*zt2;
Xt3=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt3 + (sin(a)*sin(b)*cos(c) -
cos(b)*sin(c))*yt3 + (cos(a)*sin(b))*zt3;
Yt3=y + (cos(a)*sin(c))*xt3 + (cos(a)*cos(c))*yt3 - (sin(a))*zt3;
Zt3=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt3 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt3 + (cos(a)*cos(b))*zt3;
dXt1a=(cos(a)*sin(b)*sin(c))*xt1 + (cos(a)*sin(b)*cos(c))*yt1 - (sin(a)*sin(b))*zt1;
dYt1a=-(sin(a)*sin(c))*xt1 - (sin(a)*cos(c))*yt1 - (cos(a))*zt1;
dZt1a=(cos(a)*cos(b)*sin(c))*xt1 + (cos(a)*cos(b)*cos(c))*yt1 - (sin(a)*cos(b))*zt1;
dXt2a=(cos(a)*sin(b)*sin(c))*xt2 + (cos(a)*sin(b)*cos(c))*yt2 - (sin(a)*sin(b))*zt2;
dYt2a=-(sin(a)*sin(c))*xt2 - (sin(a)*cos(c))*yt2 - (cos(a))*zt2;
dZt2a=(cos(a)*cos(b)*sin(c))*xt2 + (cos(a)*cos(b)*cos(c))*yt2 - (sin(a)*cos(b))*zt2;
dXt3a=(cos(a)*sin(b)*sin(c))*xt3 + (cos(a)*sin(b)*cos(c))*yt3 - (sin(a)*sin(b))*zt3;
dYt3a=-(sin(a)*sin(c))*xt3 - (sin(a)*cos(c))*yt3 - (cos(a))*zt3;
dZt3a=(cos(a)*cos(b)*sin(c))*xt3 + (cos(a)*cos(b)*cos(c))*yt3 - (sin(a)*cos(b))*zt3;
dXt1b=(sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt1 + (sin(a)*cos(b)*cos(c) +
sin(b)*sin(c))*yt1 + (cos(a)*cos(b))*zt1;
dYt1b=0;
dZt1b=-(sin(a)*sin(b)*sin(c) + cos(b)*cos(c))*xt1 + (cos(b)*sin(c) -
sin(a)*sin(b)*cos(c))*yt1 - (cos(a)*sin(b))*zt1;
dXt2b=(sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt2 + (sin(a)*cos(b)*cos(c) +
sin(b)*sin(c))*yt2 + (cos(a)*cos(b))*zt2;
dYt2b=0;
dZt2b=-(sin(a)*sin(b)*sin(c) + cos(b)*cos(c))*xt2 + (cos(b)*sin(c) -
sin(a)*sin(b)*cos(c))*yt2 - (cos(a)*sin(b))*zt2;
dXt3b=(sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt3 + (sin(a)*cos(b)*cos(c) +
sin(b)*sin(c))*yt3 + (cos(a)*cos(b))*zt3;
dYt3b=0;
dZt3b=-(sin(a)*sin(b)*sin(c) + cos(b)*cos(c))*xt3 + (cos(b)*sin(c) -
sin(a)*sin(b)*cos(c))*yt3 - (cos(a)*sin(b))*zt3;
dXt1c=(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*xt1 - (cos(b)*cos(c) +
sin(a)*sin(b)*sin(c))*yt1;
dYt1c=(cos(a)*cos(c))*xt1 - (cos(a)*sin(c))*yt1;
dZt1c=(sin(a)*cos(b)*cos(c) + sin(b)*sin(c))*xt1 + (sin(b)*cos(c) -
sin(a)*cos(b)*sin(c))*yt1;
dXt2c=(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*xt2 - (cos(b)*cos(c) +
sin(a)*sin(b)*sin(c))*yt2;
dYt2c=(cos(a)*cos(c))*xt2 - (cos(a)*sin(c))*yt2;
dZt2c=(sin(a)*cos(b)*cos(c) + sin(b)*sin(c))*xt2 + (sin(b)*cos(c) -
sin(a)*cos(b)*sin(c))*yt2;
dXt3c=(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*xt3 - (cos(b)*cos(c) +
sin(a)*sin(b)*sin(c))*yt3;
dYt3c=(cos(a)*cos(c))*xt3 - (cos(a)*sin(c))*yt3;
dZt3c=(sin(a)*cos(b)*cos(c) + sin(b)*sin(c))*xt3 + (sin(b)*cos(c) -
sin(a)*cos(b)*sin(c))*yt3;

% Strut lengths
L(1)=sqrt((Xt1-Xb1)*(Xt1-Xb1) + (Yt1-Yb1)*(Yt1-Yb1) + (Zt1-Zb1)*(Zt1-Zb1));
L(2)=sqrt((Xt1-Xb2)*(Xt1-Xb2) + (Yt1-Yb2)*(Yt1-Yb2) + (Zt1-Zb2)*(Zt1-Zb2));
L(3)=sqrt((Xt2-Xb3)*(Xt2-Xb3) + (Yt2-Yb3)*(Yt2-Yb3) + (Zt2-Zb3)*(Zt2-Zb3));
L(4)=sqrt((Xt2-Xb4)*(Xt2-Xb4) + (Yt2-Yb4)*(Yt2-Yb4) + (Zt2-Zb4)*(Zt2-Zb4));
L(5)=sqrt((Xt3-Xb5)*(Xt3-Xb5) + (Yt3-Yb5)*(Yt3-Yb5) + (Zt3-Zb5)*(Zt3-Zb5));
L(6)=sqrt((Xt3-Xb6)*(Xt3-Xb6) + (Yt3-Yb6)*(Yt3-Yb6) + (Zt3-Zb6)*(Zt3-Zb6));
```

```
% Check if workspace envelope is exceeded.
if L(1)>Lmax
             input('You specified a position outside of the workspace envelope (strut
1 exceeding travel limit)');
end
if L(2)>Lmax
             input('You specified a position outside of the workspace envelope (strut
2 exceeding travel limit)');
end
if L(3)>Lmax
             input('You specified a position outside of the workspace envelope (strut
3 exceeding travel limit)');
end
if L(4)>Lmax
             input('You specified a position outside of the workspace envelope (strut
4 exceeding travel limit)');
end
if L(5)>Lmax
             input('You specified a position outside of the workspace envelope (strut
5 exceeding travel limit)');
end
if L(6)>Lmax
             input('You specified a position outside of the workspace envelope (strut
6 exceeding travel limit)');
end

% Strut lengths
L(1)=sqrt((Xt1-Xb1)*(Xt1-Xb1) + (Yt1-Yb1)*(Yt1-Yb1) + (Zt1-Zb1)*(Zt1-Zb1));
L(2)=sqrt((Xt1-Xb2)*(Xt1-Xb2) + (Yt1-Yb2)*(Yt1-Yb2) + (Zt1-Zb2)*(Zt1-Zb2));
L(3)=sqrt((Xt2-Xb3)*(Xt2-Xb3) + (Yt2-Yb3)*(Yt2-Yb3) + (Zt2-Zb3)*(Zt2-Zb3));
L(4)=sqrt((Xt2-Xb4)*(Xt2-Xb4) + (Yt2-Yb4)*(Yt2-Yb4) + (Zt2-Zb4)*(Zt2-Zb4));
L(5)=sqrt((Xt3-Xb5)*(Xt3-Xb5) + (Yt3-Yb5)*(Yt3-Yb5) + (Zt3-Zb5)*(Zt3-Zb5));
L(6)=sqrt((Xt3-Xb6)*(Xt3-Xb6) + (Yt3-Yb6)*(Yt3-Yb6) + (Zt3-Zb6)*(Zt3-Zb6));

% Hexapod inverse Jacobian with respect to the base frame
J(1,1)=(Xt1-Xb1)/L(1);
J(2,1)=(Xt1-Xb2)/L(2);
J(3,1)=(Xt2-Xb3)/L(3);
J(4,1)=(Xt2-Xb4)/L(4);
J(5,1)=(Xt3-Xb5)/L(5);
J(6,1)=(Xt3-Xb6)/L(6);
J(1,2)=(Yt1-Yb1)/L(1);
J(2,2)=(Yt1-Yb2)/L(2);
J(3,2)=(Yt2-Yb3)/L(3);
J(4,2)=(Yt2-Yb4)/L(4);
J(5,2)=(Yt3-Yb5)/L(5);
J(6,2)=(Yt3-Yb6)/L(6);
J(1,3)=(Zt1-Zb1)/L(1);
J(2,3)=(Zt1-Zb2)/L(2);
J(3,3)=(Zt2-Zb3)/L(3);
J(4,3)=(Zt2-Zb4)/L(4);
J(5,3)=(Zt3-Zb5)/L(5);
J(6,3)=(Zt3-Zb6)/L(6);
J(1,4)=((Xt1-Xb1)*dXt1a + (Yt1-Yb1)*dYt1a + (Zt1-Zb1)*dZt1a)/L(1);
J(2,4)=((Xt1-Xb2)*dXt1a + (Yt1-Yb2)*dYt1a + (Zt1-Zb2)*dZt1a)/L(2);
J(3,4)=((Xt2-Xb3)*dXt2a + (Yt2-Yb3)*dYt2a + (Zt2-Zb3)*dZt2a)/L(3);
J(4,4)=((Xt2-Xb4)*dXt2a + (Yt2-Yb4)*dYt2a + (Zt2-Zb4)*dZt2a)/L(4);
J(5,4)=((Xt3-Xb5)*dXt3a + (Yt3-Yb5)*dYt3a + (Zt3-Zb5)*dZt3a)/L(5);
```

```
J(6,4)=((Xt3-Xb6)*dXt3a + (Yt3-Yb6)*dYt3a + (Zt3-Zb6)*dZt3a)/L(6);
J(1,5)=((Xt1-Xb1)*dXt1b + (Yt1-Yb1)*dYt1b + (Zt1-Zb1)*dZt1b)/L(1);
J(2,5)=((Xt1-Xb2)*dXt1b + (Yt1-Yb2)*dYt1b + (Zt1-Zb2)*dZt1b)/L(2);
J(3,5)=((Xt2-Xb3)*dXt2b + (Yt2-Yb3)*dYt2b + (Zt2-Zb3)*dZt2b)/L(3);
J(4,5)=((Xt2-Xb4)*dXt2b + (Yt2-Yb4)*dYt2b + (Zt2-Zb4)*dZt2b)/L(4);
J(5,5)=((Xt3-Xb5)*dXt3b + (Yt3-Yb5)*dYt3b + (Zt3-Zb5)*dZt3b)/L(5);
J(6,5)=((Xt3-Xb6)*dXt3b + (Yt3-Yb6)*dYt3b + (Zt3-Zb6)*dZt3b)/L(6);
J(1,6)=((Xt1-Xb1)*dXt1c + (Yt1-Yb1)*dYt1c + (Zt1-Zb1)*dZt1c)/L(1);
J(2,6)=((Xt1-Xb2)*dXt1c + (Yt1-Yb2)*dYt1c + (Zt1-Zb2)*dZt1c)/L(2);
J(3,6)=((Xt2-Xb3)*dXt2c + (Yt2-Yb3)*dYt2c + (Zt2-Zb3)*dZt2c)/L(3);
J(4,6)=((Xt2-Xb4)*dXt2c + (Yt2-Yb4)*dYt2c + (Zt2-Zb4)*dZt2c)/L(4);
J(5,6)=((Xt3-Xb5)*dXt3c + (Yt3-Yb5)*dYt3c + (Zt3-Zb5)*dZt3c)/L(5);
J(6,6)=((Xt3-Xb6)*dXt3c + (Yt3-Yb6)*dYt3c + (Zt3-Zb6)*dZt3c)/L(6);

% Strut-space stiffness matrix (f=Ks*(delta_L))
Ks(6,6)=0;
Ks=0;
Ks(1,1)=Area*EM/L(1);
Ks(2,2)=Area*EM/L(2);
Ks(3,3)=Area*EM/L(3);
Ks(4,4)=Area*EM/L(4);
Ks(5,5)=Area*EM/L(5);
Ks(6,6)=Area*EM/L(6);

% Cartesian-space stiffness matrix (F=Kc(delta_X))
Kc=J'*Ks*J;

% Eigenvalues and eigenvectors of Kc
[eiv,eival]=eig(Kc);

input('The predicted cartesian-space stiffness matrix is: ');Kc
input('The eigenvector matrix (principal stiffness axes) is: ');eiv
input('The eigenvalue matrix is: ');eival

Fx=input('What is the x component of force at the tool (relative to base frame)? ');
Fy=input('What is the y component? ');
Fz=input('What is the z component? ');
Ma=input('What is the alpha component of moment at the tool? ');
Mb=input('What is the beta component of moment? ');
Mc=input('What is the gamma component of moment? ');

Fc(1)=Fx;
Fc(2)=Fy;
Fc(3)=Fz;
Fc(4)=Ma;
Fc(5)=Mb;
Fc(6)=Mc;

% Strut forces
fs=inv(J')*Fc';
% Tool displacements
Xd=1000*inv(Kc)*Fc';

input('The vector of expected strut forces is: ');fs
input('The vector of expected tool displacements (mm) is: ');Xd
```

APPENDIX D: FORWARD KINEMATICS ANIMATOR

```
% Forward Kinematics Animator(Error Correction Inside Numerical Iteration Loop).
% Vince De SapioMay 2, 1997; Modified September 28, 1997

% Initializing constants.

clear all

% Maximum strut displacement.
Lmax=1500;

% Tool plate vertices (ball joint centers) in the tool coordinate system.
xt1=86.6;
yt1=150;
zt1=-305.1;
xt2=-173.2;
yt2=0;
zt2=-305.1;
xt3=86.6;
yt3=-150;
zt3=-305.1;
half_side=70;
xc1=half_side/.8660;
yc1=0;
zc1=-240;
xc2=-.5*half_side;
yc2=-.8660*half_side;
zc2=-240;
xc3=-.5*half_side;
yc3=.8660*half_side;
zc3=-240;

% Base plate vertices (ball joint centers) in the base coordinate system.
Xb1=400.68;
Yb1=122;
Xb2=-94.7;
Yb2=408;
Xb3=-306;
Yb3=286;
Xb4=-306;
Yb4=-286;
Xb5=-94.7;
Yb5=-408;
Xb6=400.7;
Yb6=-122;
Zb1=0;
Zb2=0;
Zb3=0;
Zb4=0;
Zb5=0;
Zb6=0;

% Initial Position in Cartesian Space.

input('Hexel hexapod forward kinematics animator.  Press <return>.');
x=input('Current position in mm; X? ');
y=input('Y? ');
z=input('Z? ');
adeg=input('Current orientation in degrees; alpha? ');
bdeg=input('beta? ');
```

```
cdeg=input('gamma? ');
a=pi*adeg/180;
b=pi*bdeg/180;
c=pi*cdeg/180;

% Desired Strut Displacements.
ldisp1=input('Desired strut displacements in mm; Strut 1? ');
ldisp2=input('Strut 2? ');
ldisp3=input('Strut 3? ');
ldisp4=input('Strut 4? ');
ldisp5=input('Strut 5? ');
ldisp6=input('Strut 6? ');
dl1=ldisp1/50;
dl2=ldisp2/50;
dl3=ldisp3/50;
dl4=ldisp4/50;
dl5=ldisp5/50;
dl6=ldisp6/50;

% Tool plate vertices (ball joint centers) in the base coordinate system.
Xt1=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt1 + (sin(a)*sin(b)*cos(c) -
cos(b)*sin(c))*yt1 + (cos(a)*sin(b))*zt1;
Yt1=y + (cos(a)*sin(c))*xt1 + (cos(a)*cos(c))*yt1 - (sin(a))*zt1;
Zt1=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt1 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt1 + (cos(a)*cos(b))*zt1;
Xt2=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt2 + (sin(a)*sin(b)*cos(c) -
cos(b)*sin(c))*yt2 + (cos(a)*sin(b))*zt2;
Yt2=y + (cos(a)*sin(c))*xt2 + (cos(a)*cos(c))*yt2 - (sin(a))*zt2;
Zt2=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt2 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt2 + (cos(a)*cos(b))*zt2;
Xt3=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt3 + (sin(a)*sin(b)*cos(c) -
cos(b)*sin(c))*yt3 + (cos(a)*sin(b))*zt3;
Yt3=y + (cos(a)*sin(c))*xt3 + (cos(a)*cos(c))*yt3 - (sin(a))*zt3;
Zt3=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt3 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt3 + (cos(a)*cos(b))*zt3;


% Initial strut lengths.
Linitial(1)=sqrt((Xt1-Xb1)*(Xt1-Xb1) + (Yt1-Yb1)*(Yt1-Yb1) + (Zt1-Zb1)*(Zt1-Zb1));
Linitial(2)=sqrt((Xt1-Xb2)*(Xt1-Xb2) + (Yt1-Yb2)*(Yt1-Yb2) + (Zt1-Zb2)*(Zt1-Zb2));
Linitial(3)=sqrt((Xt2-Xb3)*(Xt2-Xb3) + (Yt2-Yb3)*(Yt2-Yb3) + (Zt2-Zb3)*(Zt2-Zb3));
Linitial(4)=sqrt((Xt2-Xb4)*(Xt2-Xb4) + (Yt2-Yb4)*(Yt2-Yb4) + (Zt2-Zb4)*(Zt2-Zb4));
Linitial(5)=sqrt((Xt3-Xb5)*(Xt3-Xb5) + (Yt3-Yb5)*(Yt3-Yb5) + (Zt3-Zb5)*(Zt3-Zb5));
Linitial(6)=sqrt((Xt3-Xb6)*(Xt3-Xb6) + (Yt3-Yb6)*(Yt3-Yb6) + (Zt3-Zb6)*(Zt3-Zb6));

% Desired final strut lengths.
Ldesiref(1)=Linitial(1) + ldisp1;
Ldesiref(2)=Linitial(2) + ldisp2;
Ldesiref(3)=Linitial(3) + ldisp3;
Ldesiref(4)=Linitial(4) + ldisp4;
Ldesiref(5)=Linitial(5) + ldisp5;
Ldesiref(6)=Linitial(6) + ldisp6;

rect1=[350,250,490,450];
figure('Position',rect1)
whitebg('w')

for indexer=0:49
```

```
            % Values for inverse Jacobian calculation

            % Tool plate vertices (ball joint centers) in the base coordinate system.
            Xt1=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt1 +
(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*yt1 + (cos(a)*sin(b))*zt1;
            Yt1=y + (cos(a)*sin(c))*xt1 + (cos(a)*cos(c))*yt1 - (sin(a))*zt1;
            Zt1=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt1 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt1 + (cos(a)*cos(b))*zt1;
            Xt2=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt2 +
(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*yt2 + (cos(a)*sin(b))*zt2;
            Yt2=y + (cos(a)*sin(c))*xt2 + (cos(a)*cos(c))*yt2 - (sin(a))*zt2;
            Zt2=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt2 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt2 + (cos(a)*cos(b))*zt2;
            Xt3=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt3 +
(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*yt3 + (cos(a)*sin(b))*zt3;
            Yt3=y + (cos(a)*sin(c))*xt3 + (cos(a)*cos(c))*yt3 - (sin(a))*zt3;
            Zt3=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt3 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt3 + (cos(a)*cos(b))*zt3;

            Xc1=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xc1 +
(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*yc1 + (cos(a)*sin(b))*zc1;
            Yc1=y + (cos(a)*sin(c))*xc1 + (cos(a)*cos(c))*yc1 - (sin(a))*zc1;
            Zc1=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xc1 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yc1 + (cos(a)*cos(b))*zc1;
            Xc2=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xc2 +
(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*yc2 + (cos(a)*sin(b))*zc2;
            Yc2=y + (cos(a)*sin(c))*xc2 + (cos(a)*cos(c))*yc2 - (sin(a))*zc2;
            Zc2=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xc2 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yc2 + (cos(a)*cos(b))*zc2;
            Xc3=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xc3 +
(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*yc3 + (cos(a)*sin(b))*zc3;
            Yc3=y + (cos(a)*sin(c))*xc3 + (cos(a)*cos(c))*yc3 - (sin(a))*zc3;
            Zc3=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xc3 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yc3 + (cos(a)*cos(b))*zc3;

            dXt1a=(cos(a)*sin(b)*sin(c))*xt1 + (cos(a)*sin(b)*cos(c))*yt1 -
(sin(a)*sin(b))*zt1;
            dYt1a=-(sin(a)*sin(c))*xt1 - (sin(a)*cos(c))*yt1 - (cos(a))*zt1;
            dZt1a=(cos(a)*cos(b)*sin(c))*xt1 + (cos(a)*cos(b)*cos(c))*yt1 -
(sin(a)*cos(b))*zt1;
            dXt2a=(cos(a)*sin(b)*sin(c))*xt2 + (cos(a)*sin(b)*cos(c))*yt2 -
(sin(a)*sin(b))*zt2;
            dYt2a=-(sin(a)*sin(c))*xt2 - (sin(a)*cos(c))*yt2 - (cos(a))*zt2;
            dZt2a=(cos(a)*cos(b)*sin(c))*xt2 + (cos(a)*cos(b)*cos(c))*yt2 -
(sin(a)*cos(b))*zt2;
            dXt3a=(cos(a)*sin(b)*sin(c))*xt3 + (cos(a)*sin(b)*cos(c))*yt3 -
(sin(a)*sin(b))*zt3;
            dYt3a=-(sin(a)*sin(c))*xt3 - (sin(a)*cos(c))*yt3 - (cos(a))*zt3;
            dZt3a=(cos(a)*cos(b)*sin(c))*xt3 + (cos(a)*cos(b)*cos(c))*yt3 -
(sin(a)*cos(b))*zt3;
            dXt1b=(sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt1 + (sin(a)*cos(b)*cos(c)
+ sin(b)*sin(c))*yt1 + (cos(a)*cos(b))*zt1;
            dYt1b=0;
            dZt1b=-(sin(a)*sin(b)*sin(c) + cos(b)*cos(c))*xt1 + (cos(b)*sin(c) -
sin(a)*sin(b)*cos(c))*yt1 - (cos(a)*sin(b))*zt1;
            dXt2b=(sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt2 + (sin(a)*cos(b)*cos(c)
+ sin(b)*sin(c))*yt2 + (cos(a)*cos(b))*zt2;
            dYt2b=0;
```

```
            dZt2b=-(sin(a)*sin(b)*sin(c) + cos(b)*cos(c))*xt2 + (cos(b)*sin(c) -
sin(a)*sin(b)*cos(c))*yt2 - (cos(a)*sin(b))*zt2;
            dXt3b=(sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt3 + (sin(a)*cos(b)*cos(c)
+ sin(b)*sin(c))*yt3 + (cos(a)*cos(b))*zt3;
            dYt3b=0;
            dZt3b=-(sin(a)*sin(b)*sin(c) + cos(b)*cos(c))*xt3 + (cos(b)*sin(c) -
sin(a)*sin(b)*cos(c))*yt3 - (cos(a)*sin(b))*zt3;
            dXt1c=(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*xt1 - (cos(b)*cos(c) +
sin(a)*sin(b)*sin(c))*yt1;
            dYt1c=(cos(a)*cos(c))*xt1 - (cos(a)*sin(c))*yt1;
            dZt1c=(sin(a)*cos(b)*cos(c) + sin(b)*sin(c))*xt1 + (sin(b)*cos(c) -
sin(a)*cos(b)*sin(c))*yt1;
            dXt2c=(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*xt2 - (cos(b)*cos(c) +
sin(a)*sin(b)*sin(c))*yt2;
            dYt2c=(cos(a)*cos(c))*xt2 - (cos(a)*sin(c))*yt2;
            dZt2c=(sin(a)*cos(b)*cos(c) + sin(b)*sin(c))*xt2 + (sin(b)*cos(c) -
sin(a)*cos(b)*sin(c))*yt2;
            dXt3c=(sin(a)*sin(b)*cos(c) - cos(b)*sin(c))*xt3 - (cos(b)*cos(c) +
sin(a)*sin(b)*sin(c))*yt3;
            dYt3c=(cos(a)*cos(c))*xt3 - (cos(a)*sin(c))*yt3;
            dZt3c=(sin(a)*cos(b)*cos(c) + sin(b)*sin(c))*xt3 + (sin(b)*cos(c) -
sin(a)*cos(b)*sin(c))*yt3;

            % Strut length calculations.
            L(1)=sqrt((Xt1-Xb1)*(Xt1-Xb1) + (Yt1-Yb1)*(Yt1-Yb1) + (Zt1-Zb1)*(Zt1-
Zb1));
            L(2)=sqrt((Xt1-Xb2)*(Xt1-Xb2) + (Yt1-Yb2)*(Yt1-Yb2) + (Zt1-Zb2)*(Zt1-
Zb2));
            L(3)=sqrt((Xt2-Xb3)*(Xt2-Xb3) + (Yt2-Yb3)*(Yt2-Yb3) + (Zt2-Zb3)*(Zt2-
Zb3));
   L(4)=sqrt((Xt2-Xb4)*(Xt2-Xb4) + (Yt2-Yb4)*(Yt2-Yb4) + (Zt2-Zb4)*(Zt2-Zb4));
   L(5)=sqrt((Xt3-Xb5)*(Xt3-Xb5) + (Yt3-Yb5)*(Yt3-Yb5) + (Zt3-Zb5)*(Zt3-Zb5));
   L(6)=sqrt((Xt3-Xb6)*(Xt3-Xb6) + (Yt3-Yb6)*(Yt3-Yb6) + (Zt3-Zb6)*(Zt3-Zb6));

   % Desired strut lengths at next iteration.
   Ldesire(1)=Linitial(1) + (indexer+1)*dl1;
   Ldesire(2)=Linitial(2) + (indexer+1)*dl2;
   Ldesire(3)=Linitial(3) + (indexer+1)*dl3;
   Ldesire(4)=Linitial(4) + (indexer+1)*dl4;
   Ldesire(5)=Linitial(5) + (indexer+1)*dl5;
   Ldesire(6)=Linitial(6) + (indexer+1)*dl6;

   % Change in strut lengths.
   dell(1)=Ldesire(1)-L(1);
   dell(2)=Ldesire(2)-L(2);
   dell(3)=Ldesire(3)-L(3);
   dell(4)=Ldesire(4)-L(4);
   dell(5)=Ldesire(5)-L(5);
   dell(6)=Ldesire(6)-L(6);

            xs=[Xb1 Xt1 Xb2 Xb3 Xt2 Xb4 Xb5 Xt3 Xb6 Xb1 Xb2 Xb3 Xb4 Xb5 Xb6 Xt3 Xt1
Xt2 Xt3]';
            ys=[Yb1 Yt1 Yb2 Yb3 Yt2 Yb4 Yb5 Yt3 Yb6 Yb1 Yb2 Yb3 Yb4 Yb5 Yb6 Yt3 Yt1
Yt2 Yt3]';
            zs=[Zb1 Zt1 Zb2 Zb3 Zt2 Zb4 Zb5 Zt3 Zb6 Zb1 Zb2 Zb3 Zb4 Zb5 Zb6 Zt3 Zt1
Zt2 Zt3]';
            xf1=[Xb1 Xb2 Xb3 Xb4 Xb5 Xb6];
            yf1=[Yb1 Yb2 Yb3 Yb4 Yb5 Yb6];
            zf1=[Zb1 Zb2 Zb3 Zb4 Zb5 Zb6];
```

```
            xf2=[Xt1 Xt2 Xt3];
            yf2=[Yt1 Yt2 Yt3];
            zf2=[Zt1 Zt2 Zt3];
            xf3=[Xt1 Xb1 Xb1+15 Xt1+15];
            yf3=[Yt1 Yb1 Yb1 Yt1];
            zf3=[Zt1 Zb1 Zb1 Zt1];
            xf4=[Xt1 Xb2 Xb2+15 Xt1+15];
            yf4=[Yt1 Yb2 Yb2 Yt1];
            zf4=[Zt1 Zb2 Zb2 Zt1];
            xf5=[Xt2 Xb3 Xb3+15 Xt2+15];
            yf5=[Yt2 Yb3 Yb3 Yt2];
            zf5=[Zt2 Zb3 Zb3 Zt2];
            xf6=[Xt2 Xb4 Xb4+15 Xt2+15];
            yf6=[Yt2 Yb4 Yb4 Yt2];
            zf6=[Zt2 Zb4 Zb4 Zt2];
            xf7=[Xt3 Xb5 Xb5-15 Xt3-15];
            yf7=[Yt3 Yb5 Yb5 Yt3];
            zf7=[Zt3 Zb5 Zb5 Zt3];
            xf8=[Xt3 Xb6 Xb6-15 Xt3-15];
            yf8=[Yt3 Yb6 Yb6 Yt3];
            zf8=[Zt3 Zb6 Zb6 Zt3];
            xf9=[Xc1 Xc2 x];
            yf9=[Yc1 Yc2 y];
            zf9=[Zc1 Zc2 z];
            xf10=[Xc2 Xc3 x];
            yf10=[Yc2 Yc3 y];
            zf10=[Zc2 Zc3 z];
            xf11=[Xc1 Xc3 x];
            yf11=[Yc1 Yc3 y];
            zf11=[Zc1 Zc3 z];

            c1=[0 .5 1];
            c2=[1 1 0];

            fill3(xf1, yf1, zf1, c1, xf2, yf2, zf2, c1, xf3, yf3, zf3, c2, xf4, yf4,
zf4, c2, xf5, yf5, zf5, c2, xf6, yf6, zf6, c2, xf7, yf7, zf7, c2, xf8, yf8, zf8, c2,
xf9, yf9, zf9, c1, xf10, yf10, zf10, c1, xf11, yf11, zf11, c1);
   view([-45 -25])
   upvector = [0 0 -1];
   set(gca, 'CameraUpVector',upvector);
   axis('equal')
   axis([-500 500 -500 500 0 1000]);
   axis('off');
   title(['Hexapod - Forward & Inverse Kinematics
Simulation'],'FontName','times','FontAngle','italic','FontWeight','bold')
   M(:,indexer + 1)=getframe;

            % Hexapod inverse Jacobian with respect to the base frame.
            J(1,1)=(Xt1-Xb1)/L(1);
            J(2,1)=(Xt1-Xb2)/L(2);
            J(3,1)=(Xt2-Xb3)/L(3);
            J(4,1)=(Xt2-Xb4)/L(4);
            J(5,1)=(Xt3-Xb5)/L(5);
            J(6,1)=(Xt3-Xb6)/L(6);
            J(1,2)=(Yt1-Yb1)/L(1);
            J(2,2)=(Yt1-Yb2)/L(2);
            J(3,2)=(Yt2-Yb3)/L(3);
            J(4,2)=(Yt2-Yb4)/L(4);
            J(5,2)=(Yt3-Yb5)/L(5);
```

```
            J(6,2)=(Yt3-Yb6)/L(6);
            J(1,3)=(Zt1-Zb1)/L(1);
            J(2,3)=(Zt1-Zb2)/L(2);
            J(3,3)=(Zt2-Zb3)/L(3);
            J(4,3)=(Zt2-Zb4)/L(4);
            J(5,3)=(Zt3-Zb5)/L(5);
            J(6,3)=(Zt3-Zb6)/L(6);
            J(1,4)=((Xt1-Xb1)*dXt1a + (Yt1-Yb1)*dYt1a + (Zt1-Zb1)*dZt1a)/L(1);
            J(2,4)=((Xt1-Xb2)*dXt1a + (Yt1-Yb2)*dYt1a + (Zt1-Zb2)*dZt1a)/L(2);
            J(3,4)=((Xt2-Xb3)*dXt2a + (Yt2-Yb3)*dYt2a + (Zt2-Zb3)*dZt2a)/L(3);
            J(4,4)=((Xt2-Xb4)*dXt2a + (Yt2-Yb4)*dYt2a + (Zt2-Zb4)*dZt2a)/L(4);
            J(5,4)=((Xt3-Xb5)*dXt3a + (Yt3-Yb5)*dYt3a + (Zt3-Zb5)*dZt3a)/L(5);
            J(6,4)=((Xt3-Xb6)*dXt3a + (Yt3-Yb6)*dYt3a + (Zt3-Zb6)*dZt3a)/L(6);
            J(1,5)=((Xt1-Xb1)*dXt1b + (Yt1-Yb1)*dYt1b + (Zt1-Zb1)*dZt1b)/L(1);
            J(2,5)=((Xt1-Xb2)*dXt1b + (Yt1-Yb2)*dYt1b + (Zt1-Zb2)*dZt1b)/L(2);
            J(3,5)=((Xt2-Xb3)*dXt2b + (Yt2-Yb3)*dYt2b + (Zt2-Zb3)*dZt2b)/L(3);
            J(4,5)=((Xt2-Xb4)*dXt2b + (Yt2-Yb4)*dYt2b + (Zt2-Zb4)*dZt2b)/L(4);
            J(5,5)=((Xt3-Xb5)*dXt3b + (Yt3-Yb5)*dYt3b + (Zt3-Zb5)*dZt3b)/L(5);
            J(6,5)=((Xt3-Xb6)*dXt3b + (Yt3-Yb6)*dYt3b + (Zt3-Zb6)*dZt3b)/L(6);
            J(1,6)=((Xt1-Xb1)*dXt1c + (Yt1-Yb1)*dYt1c + (Zt1-Zb1)*dZt1c)/L(1);
            J(2,6)=((Xt1-Xb2)*dXt1c + (Yt1-Yb2)*dYt1c + (Zt1-Zb2)*dZt1c)/L(2);
            J(3,6)=((Xt2-Xb3)*dXt2c + (Yt2-Yb3)*dYt2c + (Zt2-Zb3)*dZt2c)/L(3);
            J(4,6)=((Xt2-Xb4)*dXt2c + (Yt2-Yb4)*dYt2c + (Zt2-Zb4)*dZt2c)/L(4);
            J(5,6)=((Xt3-Xb5)*dXt3c + (Yt3-Yb5)*dYt3c + (Zt3-Zb5)*dZt3c)/L(5);
            J(6,6)=((Xt3-Xb6)*dXt3c + (Yt3-Yb6)*dYt3c + (Zt3-Zb6)*dZt3c)/L(6);

            delp=inv(J)*dell';
            x=x+delp(1);
            y=y+delp(2);
            z=z+delp(3);
            a=a+delp(4);
            b=b+delp(5);
            c=c+delp(6);

end

% Tool plate vertices (ball joint centers) in the base coordinate system.
Xt1=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt1 + (sin(a)*sin(b)*cos(c) -
cos(b)*sin(c))*yt1 + (cos(a)*sin(b))*zt1;
Yt1=y + (cos(a)*sin(c))*xt1 + (cos(a)*cos(c))*yt1 - (sin(a))*zt1;
Zt1=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt1 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt1 + (cos(a)*cos(b))*zt1;
Xt2=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt2 + (sin(a)*sin(b)*cos(c) -
cos(b)*sin(c))*yt2 + (cos(a)*sin(b))*zt2;
Yt2=y + (cos(a)*sin(c))*xt2 + (cos(a)*cos(c))*yt2 - (sin(a))*zt2;
Zt2=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt2 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt2 + (cos(a)*cos(b))*zt2;
Xt3=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xt3 + (sin(a)*sin(b)*cos(c) -
cos(b)*sin(c))*yt3 + (cos(a)*sin(b))*zt3;
Yt3=y + (cos(a)*sin(c))*xt3 + (cos(a)*cos(c))*yt3 - (sin(a))*zt3;
Zt3=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xt3 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yt3 + (cos(a)*cos(b))*zt3;

Xc1=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xc1 + (sin(a)*sin(b)*cos(c) -
cos(b)*sin(c))*yc1 + (cos(a)*sin(b))*zc1;
Yc1=y + (cos(a)*sin(c))*xc1 + (cos(a)*cos(c))*yc1 - (sin(a))*zc1;
Zc1=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xc1 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yc1 + (cos(a)*cos(b))*zc1;
```

```
Xc2=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xc2 + (sin(a)*sin(b)*cos(c) -
cos(b)*sin(c))*yc2 + (cos(a)*sin(b))*zc2;
Yc2=y + (cos(a)*sin(c))*xc2 + (cos(a)*cos(c))*yc2 - (sin(a))*zc2;
Zc2=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xc2 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yc2 + (cos(a)*cos(b))*zc2;
Xc3=x + (cos(b)*cos(c) + sin(a)*sin(b)*sin(c))*xc3 + (sin(a)*sin(b)*cos(c) -
cos(b)*sin(c))*yc3 + (cos(a)*sin(b))*zc3;
Yc3=y + (cos(a)*sin(c))*xc3 + (cos(a)*cos(c))*yc3 - (sin(a))*zc3;
Zc3=z + (sin(a)*cos(b)*sin(c) - sin(b)*cos(c))*xc3 + (sin(b)*sin(c) +
sin(a)*cos(b)*cos(c))*yc3 + (cos(a)*cos(b))*zc3;

% Final strut lengths.
L(1)=sqrt((Xt1-Xb1)*(Xt1-Xb1) + (Yt1-Yb1)*(Yt1-Yb1) + (Zt1-Zb1)*(Zt1-Zb1));
L(2)=sqrt((Xt1-Xb2)*(Xt1-Xb2) + (Yt1-Yb2)*(Yt1-Yb2) + (Zt1-Zb2)*(Zt1-Zb2));
L(3)=sqrt((Xt2-Xb3)*(Xt2-Xb3) + (Yt2-Yb3)*(Yt2-Yb3) + (Zt2-Zb3)*(Zt2-Zb3));
L(4)=sqrt((Xt2-Xb4)*(Xt2-Xb4) + (Yt2-Yb4)*(Yt2-Yb4) + (Zt2-Zb4)*(Zt2-Zb4));
L(5)=sqrt((Xt3-Xb5)*(Xt3-Xb5) + (Yt3-Yb5)*(Yt3-Yb5) + (Zt3-Zb5)*(Zt3-Zb5));
L(6)=sqrt((Xt3-Xb6)*(Xt3-Xb6) + (Yt3-Yb6)*(Yt3-Yb6) + (Zt3-Zb6)*(Zt3-Zb6));

% Error between desired final strut lengths and actual final calculated lengths.
Lerror(1)=Ldesiref(1) - L(1);
Lerror(2)=Ldesiref(2) - L(2);
Lerror(3)=Ldesiref(3) - L(3);
Lerror(4)=Ldesiref(4) - L(4);
Lerror(5)=Ldesiref(5) - L(5);
Lerror(6)=Ldesiref(6) - L(6);

xs=[Xb1 Xt1 Xb2 Xb3 Xt2 Xb4 Xb5 Xt3 Xb6 Xb1 Xb2 Xb3 Xb4 Xb5 Xb6 Xt3 Xt1 Xt2 Xt3]';
ys=[Yb1 Yt1 Yb2 Yb3 Yt2 Yb4 Yb5 Yt3 Yb6 Yb1 Yb2 Yb3 Yb4 Yb5 Yb6 Yt3 Yt1 Yt2 Yt3]';
zs=[Zb1 Zt1 Zb2 Zb3 Zt2 Zb4 Zb5 Zt3 Zb6 Zb1 Zb2 Zb3 Zb4 Zb5 Zb6 Zt3 Zt1 Zt2 Zt3]';
xf1=[Xb1 Xb2 Xb3 Xb4 Xb5 Xb6];
yf1=[Yb1 Yb2 Yb3 Yb4 Yb5 Yb6];
zf1=[Zb1 Zb2 Zb3 Zb4 Zb5 Zb6];
xf2=[Xt1 Xt2 Xt3];
yf2=[Yt1 Yt2 Yt3];
zf2=[Zt1 Zt2 Zt3];
xf3=[Xt1 Xb1 Xb1+15 Xt1+15];
yf3=[Yt1 Yb1 Yb1 Yt1];
zf3=[Zt1 Zb1 Zb1 Zt1];
xf4=[Xt1 Xb2 Xb2+15 Xt1+15];
yf4=[Yt1 Yb2 Yb2 Yt1];
zf4=[Zt1 Zb2 Zb2 Zt1];
xf5=[Xt2 Xb3 Xb3+15 Xt2+15];
yf5=[Yt2 Yb3 Yb3 Yt2];
zf5=[Zt2 Zb3 Zb3 Zt2];
xf6=[Xt2 Xb4 Xb4+15 Xt2+15];
yf6=[Yt2 Yb4 Yb4 Yt2];
zf6=[Zt2 Zb4 Zb4 Zt2];
xf7=[Xt3 Xb5 Xb5-15 Xt3-15];
yf7=[Yt3 Yb5 Yb5 Yt3];
zf7=[Zt3 Zb5 Zb5 Zt3];
xf8=[Xt3 Xb6 Xb6-15 Xt3-15];
yf8=[Yt3 Yb6 Yb6 Yt3];
zf8=[Zt3 Zb6 Zb6 Zt3];
xf9=[Xc1 Xc2 x];
yf9=[Yc1 Yc2 y];
zf9=[Zc1 Zc2 z];
xf10=[Xc2 Xc3 x];
yf10=[Yc2 Yc3 y];
```

```
zf10=[Zc2 Zc3 z];
xf11=[Xc1 Xc3 x];
yf11=[Yc1 Yc3 y];
zf11=[Zc1 Zc3 z];

c1=[0 .5 1];
c2=[1 1 0];

fill3(xf1, yf1, zf1, c1, xf2, yf2, zf2, c1, xf3, yf3, zf3, c2, xf4, yf4, zf4, c2, xf5,
yf5, zf5, c2, xf6, yf6, zf6, c2, xf7, yf7, zf7, c2, xf8, yf8, zf8, c2, xf9, yf9, zf9,
c1, xf10, yf10, zf10, c1, xf11, yf11, zf11, c1);
view([-45 -25])
upvector = [0 0 -1];
set(gca, 'CameraUpVector',upvector);
axis('equal')
axis([-500 500 -500 500 0 1000]);
axis('off');
title(['Hexapod - Forward & Inverse Kinematics
Simulation'],'FontName','times','FontAngle','italic','FontWeight','bold')
M(:,indexer + 2)=getframe;

input('The final position is:');
x
y
z
a
b
c
input('The final strut lengths are:');
L

input('Error between desired strut length and final calculated strut length
(microns)');
Lerror*1000
movie(M,-10)
```

DISTRIBUTION

3	9018	Central Technical Files, 8940-2
4	0899	Technical Library, 4916
1	9021	Technical Communications Department, 8815/ Technical Library, MS 0899, 4916
2	9021	Technical Communications Department, 8815 For DOE/OSTI
1	0958	Lothar Bieg, 1484
1	0958	Donald Plymale, 1484
1	0958	Jon Munford, 1484
1	1004	Ray Harrigan, 9623
1	1007	David Kozlowski, 9672
1	0958	Gilbert Benavides, 1484
5	9420	Al West, 8200
1	9430	Anton West, 8240
1	9420	Mike Rogers, 8220
1	9430	Louie Tallerico, 8204
1	9430	Calvin King, 8240
1	9430	Robert Hillaire, 8240
1	9214	Gary Templet, 8920
1	9430	Charles Steinhaus, 8240