A Study of Isostatic Framework with Application to Manipulator Design

by

Babu Padmanabhan

Dissertation Submitted to the faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Mechanical Engineering

APPROVED:

C. F. Reinholtz, Chairman

H. H. Robertshaw

R. L. West

J. S. Bay

S. Jayaram

September 23, 1992

Blacksburg, Virginia

A Study of Isostatic Framework with Application to Manipulator Design

by

Babu Padmanabhan

Charles F. Reinholtz, Chairman

Mechanical Engineering

(ABSTRACT)

Isostatic frameworks are statically determinate trusses that are self contained (i.e. they exist independent of support or foundation). Isostatic frameworks have been widely used as supporting structures, and recently they have been used as the structure for parallel manipulators. These truss-based manipulators could potentially solve the problems facing conventional manipulators and could make the design of high-degree-of-freedom manipulators feasible. The rigorous scientific study of isostatic frameworks and manipulators based on their structure has been limited. Recent developments in the design of large space structures and truss-based manipulators, however, demand rigorous design and mathematical tools. This dissertation provides a general theory for the design of structures based on frameworks and methods to analyze the kinematics of truss-based manipulators.

The objective of the first part of this dissertation is to solve the problems of identification, generation and classification of isostatic frameworks in greater depth than in any past work in this area. Original methods are discussed for the enumeration and generation of isostatic frameworks. The first part also presents an

original method to determine the geometry of general frameworks and an improved method to find the forces in their members. The determination of geometry and forces are critical areas in structural design.

The second part of this dissertation presents a case study on one of the candidates for manipulator applications, the double-octahedral manipulator. The kinematic analyses of the double-octahedral manipulator includes methods to perform forward and inverse kinematic analysis, velocity and acceleration analysis, singularity analysis and workspace analysis. The closed-form solution to the inverse analysis presented herein is a major breakthrough in the development of the double-octahedral manipulator. Other analysis, such as velocity and acceleration, singularity, and workspace, depend on the inverse solution. It is believed that these solutions will help narrow the gap between theory and application of truss-based manipulators. The determination of singularities and workspaces are application of recent ideas of other researchers. However, original implementations of these ideas have yielded astonishing results. The Jacobian and Hessian matrix presented in this dissertation should help in developing the control scheme for this device. C-language program codes for several of the methods are also provided. The methods have been tested based on the results obtained from these programs. The position analysis algorithms have also been tested on real hardware. Some of the methods developed here have been successfully employed for simulated and experimental vibration control studies.

Acknowledgments

I like to specially thank Dr. Charles Reinholtz for being such a great positive influence on me. His creative ideas, constant encouragement and guidance has made this work possible. I owe him for my confidence in the field of kinematics and in taking on the challenges of the real world.

I am thankful to Dr. Harry Robertshaw for introducing this exciting topic to all of us and for proposing to use this research material for his new book. I am very appreciative of his willingness to allow my free use of his lab and other facilities. I am grateful to Dr. Bob West, Dr. John Bay, and Dr. Sankar Jayaram for their help and support. I am also grateful to Dr. Dhande for his guidance in the project on extruder screw manufacturing. I like to acknowledge the support of researchers at NASA Langley, especially, Dr. Garnet Horner.

My friends, Venki, Sriram, Hemanshu, Srinivas, Saikat and Vinita are the best and I am very thankful to them for their friendship. Thanks are due to Arun for motivating me to publish my results and helping me discover many of them. I like to thank Bob and Paul for their contribution to this research, in terms of productive discussions and helping with the gimbal paper. Many thanks to Kolady for making the workspace project some of the most interesting work I have ever done. My thanks are due to Ravi for helping me with the dissertation preparation. I am thankful to

John, Steve, Barry, Jon, Kevin, Allen and Will for helping me make most of my graduate studies. Many thanks to Jeri, Charlie, Amanda and Nicholas for being such good friends and helping me learn to water ski. I like to thank my tall brother for agreeing to be part of my industrial venture. Satish, you are a great asset.

I cannot thank my father and mother enough for encouraging me through all this and for having belief in me. Thank you, Mom. Thank you, Dad.

Table of Contents

List of Illustrations

1. Introduction

1.1 Research Area

Applications involving the mechanical positioning of one body with respect to another, with or without relative motion, require a rigid connection between the two bodies. In the absence of relative motion, the members that are part of this rigid connection are said to form a structure. In the presence of relative motion, the members form a manipulator capable of producing the required motion. Forces acting on the bodies and the connecting members produce stress and deformation in the members. The deformations are negligible in a rigid structure and the relative positioning between the two bodies is maintained. When the forces are small, a weak structure, such as a slender cantilevered beam, or a weak manipulator, such as the Remote Manipulator System[1] in the shuttle, will suffice. If high strength and rigidity are required in the presence of significant forces, then alternate designs for structures and manipulators may have to be considered.

Among the numerous structures that can be conceived, statically determinate trusses form an important class. Isostatic frameworks are part of this class. A property of the isostatic frameworks is that they can exist as structures independent

[1] Also called Canada Arm, the RMS is essentially a long serial chain of links that connect the shuttle to the payload. The RMS cannot support its own weight in a one-gravity (1-g) environment. The 0-g environment of space and the lack of atmosphere eliminates many of the structural strength requirements.

of support or foundation[2]. Due to their complexity, problems associated with isostatic frameworks have been studied to a very limited extent. The problem of identification of isostatic frameworks, generation of isostatic frameworks, representation of isostatic frameworks, classification of isostatic frameworks, determination of geometry (i.e. coordinates of the nodes), and the determination of forces have been explored in this dissertation. The methods developed here provide the designer of frameworks with a set of powerful tools for synthesis and analysis.

In the past, manipulator studies have been restricted to serial devices like the Canada Arm. Recent developments, like those based on the Stewart platform[3], have led to lighter and stiffer manipulators called parallel manipulators. Among the numerous manipulators that can be designed as parallel devices, an important class is based on isostatic frameworks. In this dissertation, these are simply referred to as truss-based manipulators. The design and control of these truss-based manipulators depend on efficient methods to solve the problems associated with the kinematics of the device. A comprehensive kinematic analysis of one of the candidates for a truss-based manipulator, the double-octahedral manipulator, is presented.

---

[2] Isostatic frameworks have been referred to as simply stiff trusses, minimally statically rigid trusses, and as self-contained statically determinate trusses by other authors.

[3] Stewart platform is a device that has a moving platform that is attached to the ground with six actuated legs. These six actuators provide all six degrees of freedom to the platform. A well known application of the Stewart platform is the flight simulator.

1.2 Motivation

The simplest connection between two bodies is achieved by attaching the two ends of a link to the two bodies. Such a structure is called a beam. In a given position, serial manipulators are formed as a series of cantilevered beams, and they suffer from the problem of low strength and rigidity. The structural flexibility and joint flexibility contribute toward this problem. The structural flexibility arises due to the inherent nature of the structure to transmit bending moments. The joint flexibility arises because the joints that transmit the forces must be actuated. The problem of adding structural strength and rigidity to a serial manipulator may be solved by the skillful combination of material resources, such as quantity of material, choice of material, and distribution of material in the structure. Likewise, the problem of flexibility at the joints can be minimized by proper drive train selection and careful mechanical design. Nevertheless, serial manipulators can only have a limited number of joints and a limited number of degrees of freedom. There are, however, applications that require the use of stiff high-DOF manipulators[4].

One way to satisfy the strength and rigidity requirements is to consider alternate designs for the structure. A better design would take advantage of the transmission of forces through the connecting members to achieve better material utilization. Isostatic frameworks possess such characteristics. The members of a framework carry only tensile or compressive stresses, and therefore are two-force

[4] Sometimes referred to as redundant or hyper-redundant manipulators, High-DOF manipulators are applied in an unstructured environment for tasks that require surmounting obstacles and access through narrow passageways or openings.

members. The variability in geometry of the framework, achieved by means of variable length link members, can be utilized to perform manipulation tasks. In such a design, the joints that transmit the forces are not actuated, therefore, the problem of joint flexibility is greatly minimized. The design and control of truss-based manipulator, however, is more challenging than that of serial manipulators. In the past, adequate scientific tools have not been available to design and analyze such devices.

1.3 Objective

The objective of this dissertation is to develop a general theoretical foundation for designing and analyzing isostatic framework (truss) based structures and manipulators. Various applications have been envisaged by researchers for such trusses in the literature. For several of these proposed applications to become reality, structural and kinematic analysis is required. This dissertation provides methods and techniques to solve problems in several areas of truss design.

1.4 Literature Review

1.4.1 Structural Analysis

Marshall and Nelson [1969] state that the function of a structure is to transmit forces from one point in space to another without any appreciable deformation of one part relative to another. Joints are critical in the transmission of forces through the members. In truss structures, the members of the structure are connected by spheric joints, providing three Degrees Of Freedom (DOF). Bending moments, torques and

shear forces cannot be transmitted from member to member. All the members of the truss experience either tension or compression. If the truss is designed properly, this feature can be exploited to achieve better material utilization. In practice, members of a structural "truss" are usually welded or riveted together. Because of these rigid joints, the links of the truss experience bending. Usually, the influence of this secondary bending is ignored [Timoshenko and Young, 1965].

Manipulation capability is achieved in an isostatic framework by introducing variability in the link dimensions. If all the members of the framework are jointed by spheric joints, then any or all members can be made variable. In practice, however, only those members that are necessary to provide the required degrees of freedom (DOF) and workspace are made variable. These variable links, or actuators, provide the manipulation capability by changing the geometry of the framework[5]. Therefore, in the case of truss-based manipulators, the members cannot be riveted or welded together, and the design of joints to connect the members of a framework becomes a difficult task. Depending on the choice of actuated links, several spheric joints may need to be centered about the same point. In such cases, structural characteristics may have to be compromised for ease of joint design. Rhodes and Mikulas [1985] built an octahedral truss at the NASA Langley research center to test space structure deployment concepts. Their approach introduced an extra link to provide additional connectivity points for joining links. Depending on the mobility

[5] This is a primary reason for employing statically determinate trusses as opposed to statically indeterminate trusses. In the latter case, several links may have to be moved in unison to achieve manipulation capability.

requirement, revolute joints can be substituted for spheric joints to improve long-term reliability and lower manufacturing costs. An octahedral truss built at VPI&SU has Hooke's coupling universal joints instead of spheric joints [Tidwell, 1989].

In the design of truss-based manipulators and structures the choice of topology is crucial. For centuries there have been several known isostatic frameworks. The first attempt to systematically study isostatic frameworks was made by Henneberg [1911]. Later, Cox [1936], like Henneberg, developed a number of rules for building larger isostatic frameworks from smaller ones, and for breaking down larger frameworks for the purpose of analysis. Laman [1975], Asimow & Roth [1978], and Recksi [1984] employ similar rules based techniques for proof of various properties of isostatic frameworks. Tay and Whitley [1985] present a unified approach of these principles, with a special focus on effective combinatorial characterization of the graphs of isostatic frameworks in space, to generate isostatic frameworks. Arun et al. [1990a] provide an excellent, but incomplete, approach based on mobility to classify and enumerate isostatic frameworks. In this work, an attempt is made to combine the advantages of these earlier approaches with new ones that are based on kinematic concepts. In this work, the mobility based approach is pursued more rigorously to identify and enumerate all possible isostatic frameworks.

Authors writing on the theory of structures classify statically determinate trusses as simple, compound and complex. This classification is based on the degree of difficulty of the static analysis. This classification can be extended to isostatic

frameworks as well. The only simple frameworks are those whose members are part of a tetrahedron. Every other topology falls into the category of complex framework. Hence this is not a useful classification scheme. However, an important observation can be made about trusses based on the above classification. Simple frameworks can be easily solved using the method of joints [Timoshenko and Young, 1965]. Complex frameworks need sophisticated methods such as the one developed by Henneberg [1911].

In the past, only those trusses and frameworks whose geometries are fixed have been analyzed. In a fixed structure, the geometry can be determined by measurement. In the trusses being investigated, the geometry can vary considerably during operation due to changes in dimensions of the actuated members of the truss. Hence, analytic determination of the geometry (i.e. generation of possible closures, selection of a particular closure and determination of nodal coordinates) from the truss topology and link dimensions is a crucial part of the structural analysis. Historically, the problem of determining the geometry of the framework has been very difficult. Only since the advent of analytic and coordinate geometry could this problem be handled effectively. Computer-based methods have also helped to solve the problems that require iterative solution. Coxeter [1973] provides excellent information on the history of polyhedra and analysis of their geometry. Many of the remarks in Coxeter's book are relevant, since the edges of deltahedra are known to form isostatic frameworks. The simplest framework is a four-noded six-linked tetrahedron. A solution to this geometry has been well known for several centuries. Griffis and Duffy

[1989] show that the solution to the geometry of the octahedron, which is the next simplest framework, can be reduced to a single sixteenth degree equation. A single equation to determine the geometry of the octahedron is also obtained by Nanua and Waldron [1989] and Innocenti and Parenti-Castelli [1990]. A generalized kinematic equivalent model for the octahedral framework that provides solutions through a simple iterative scheme is provided by Tidwell et al. [1990]. Arun et al. [1990b] use homotopy-based methods to obtain the geometry of more complex frameworks. In this dissertation, a simple scheme to determine the geometry that is applicable to all frameworks is presented. This method is shown to yield faster and more complete results compared to the methods that are currently available.

Several methods for conducting static analysis on statically determinate trusses can be found in the literature [Timoshenko and Young, 1965; Marshall and Nelson, 1969]. The well known method of joints is the simplest of all methods. The method of joints can only be used on frameworks built from tetrahedrons. Other methods include the method of sections, method of virtual displacement and Henneberg's method of link replacement. Since the advent of finite element methods, static analysis of complex trusses has usually been performed using finite elements. Lacy [1991] performs static analysis of the octahedral framework using this method. Static analysis based on Henneberg's method is provided in the latter sections. This method yields the stresses in the framework faster and with fewer computations than the finite element methods.

1.4.2 Kinematic Analysis

A manipulator is a device capable of transforming the position and orientation of one body relative to another body. This transformation is effected by means of some moving members called actuators. The position, velocity, and acceleration of the actuators controls the position, velocity, and acceleration of one body relative to the other. Kinematic analysis methods that provide actuator parameters that satisfy a given set of moving body parameters for the octahedral manipulator are presented in this dissertation. Once the actuators are locked in place, the manipulator should maintain the relative position between the two bodies, acting as a structure (i.e support a given load within a specified deflection). Structural load can be determined by performing a static analysis on the manipulator.

The study of truss-based manipulators can greatly benefit from the vast literature available in the field of parallel manipulators. The Stewart platform [Stewart, 1965] is one of the earliest parallel manipulators proposed. It is used as a flight simulator mechanism, among other applications. The Stewart platform is a device that has a free top platform and a fixed bottom platform. Six Spheric-Prismatic-Spheric (SPS) jointed links connect the two platforms together. By varying the length of the SPS leg, the position and orientation of the top platform can be controlled. Hunt [1978] initiates the idea of using the Stewart platform as the mechanism of a robot arm. Fichter and McDowell [1982] develop a new design for a manipulator arm based on Hunt's idea and called it the Stewart Platform based Manipulator Arm (SPMA).

The Stewart platform can be designed in several ways. Among those, the 3-3 Stewart platform[6] forms an octahedral framework. The determination of the octahedral geometry described earlier were the result of studying the 3-3 Stewart platform. Workspaces of Stewart platform based manipulators are studied by Behi [1986], Merlet [1987], Cleary and Arai [1991] and Gosselin [1990]. Gosselin's analytical work forms an excellent basis for the workspace determination of truss-based manipulators with closed-form inverse solutions. Gosselin and Angeles [1990] have studied the conditions for singularity in parallel manipulators. These results are also applicable while studying truss-based manipulators.

Miura [1985] coined the term 'Variable-Geometry-Truss Manipulator' (VGTM) and defined it as a statically determinate truss with some of its links replaced by variable length members. Most of the earlier work in this area has been carried out under the name of variable-geometry-truss manipulators. Several classes of Stewart platform are statically determinate trusses. They are, however, not called variable-geometry-truss manipulators. Since, the usage of the term variable-geometry-truss manipulators is ambiguous, a new term "truss-based manipulator" is used in this dissertation. Truss-based manipulators are defined as parallel manipulators with a structure of an isostatic framework.

[6] Defined by Griffis and Duffy [1989], a 3-3 Stewart Platform has triangular (3 nodes) moving and fixed plane. Two legs meet at each node of the triangles to form the octahedral configuration.

1.4.3 Application and Related Studies

The use of truss-based manipulator as manipulator arms has been studied by Miura et al. [1985], Sincarsin and Hughes [1987], and Reinholtz and Gokhale [1987]. Some of the potential applications of these devices in space are evidenced in the work by Rockwell International [1982], Cox and Nelson [1982], Miura and Furuya [1985], Rhodes and Mikulas [1987]. Salerno et al. [1990] and Naccarato and Hughes [1991] have developed methods to analyze the inverse kinematics of truss-type long-chain hyper-redundant manipulators for use in unstructured environment. An inverse analysis of a tetrahedral truss-based manipulator with one link actuated in each bay is discussed by Sohmshetty and Kramer [1989]. Recently, there has been further work on large adaptive truss-based space structures [Takamatsu and Onoda, 1991; Mikulas et al., 1991; Bush et al., 1991, Kawaguchi, 1990; Anderson et al., 1990, Davison, 1990; Dorsey and Mikulas, 1990; Matunaga et al., 1990; Burdisso and Haftka, 1989].

The ability of adaptive trusses to perform vibration control are valued by many researchers. Wada [1990] provides an overview of research on the dynamics of adaptive trusses. Robertshaw [1989] shows that the vibrations of a flexible slender rod appended to the bottom of a quadruple octahedral adaptive truss can be controlled effectively using a leadscrew-based mechanical actuator. Similar results for a thin, flexible beam in the gravity field are obtained by Wynn [1990]. Fanson [1989] and Natori[1987] perform vibration control of spatial structures with other actuation schemes such as piezoelectric and voice coil. Other vibration control results are found

in the works of Clark [1990], Robertshaw [1985], Juang [1986], and Alberts [1990]. Warrington [1991] uses the closed-form inverse kinematics results found in this dissertation in experiments on large-angle flexible beam control using the octahedral adaptive truss.

1.5 Summary of Contents

The contents of this dissertation are presented in two parts. The Second and Third Chapters cover structural analysis of isostatic frameworks. The Second Chapter includes sections on methods to identify and generate various isostatic framework topologies. The Third Chapter provides methods for classification, determination of geometry (i.e., generation and selection of geometry and determination of nodal coordinates), and determination of stresses in the members of the isostatic framework.

The second part of this dissertation, consisting of chapters 4 through 9, is a case study on the kinematic analysis of the double-octahedral manipulator. In the Fourth Chapter, the design of a double-octahedral manipulator is considered. In the Fifth Chapter, methods to perform the forward kinematic analysis of the double-octahedral manipulator are presented. The Sixth Chapter presents closed-form inverse kinematic analysis of three different problems proposed for the double-octahedral manipulator. The determination of the Jacobian and Hessian matrices are shown in the Seventh Chapter on velocity and acceleration analysis. Methods to determine the singular configurations of the manipulator are presented in the Eighth Chapter on Singularity analysis. The Ninth Chapter presents methods to determine

the workspace using the position relationship from the closed-form inverse analysis.

All the methods developed in chapters 2 through 7 were implemented on the computer using the C-language. The C-language program codes are presented as Appendices 1-5. Numerical results obtained using these programs are presented at the end of each method.

2. Identification and Generation of Isostatic Frameworks

2.1 Introduction

A link freely floating in space has six degrees of freedom. A linkage is formed by the process of joining links to one another and to the ground, constraining the motions of the individual links. The number of links, the number of joints, and the type of joints determine the mobility of the resulting system. The mobility of the system forms the basis for calling a linkage a mechanism, a statically determinate structure, or a statically indeterminate structure. A mechanism has positive mobility. On the application of a force or a torque, a mechanism will correspondingly change its geometry. The study of mechanisms forms an important part of kinematics. On the other hand, a structure resists forces and torques with negligible deflection by transmitting the forces through its members to the ground. A special nature of the statically determinate structure is that the links are just sufficiently constrained to remove all the degrees of freedom, resulting in a mobility of zero. For this reason, the forces in the links are uniquely defined. Due to their lack of mobility, structures have not been studied by kinematicians in the past. The current scope of applications, however, demands the study of trusses from a kinematic point of view. Therefore, mobility analysis forms the first step in the structural analysis of isostatic frameworks, which belong to the class of statically determinate structures.

2.2 Identification

2.2.1 Joints-Links Relationship

The total mobility of a system of n interconnected links is given by the Kutzbach equation [Mabie and Reinholtz, 1987].

$$
\begin{aligned}
&M = 6(n-1) - 5f_1 - 4f_2 - 3f_3 - 2f_4 - f_5 \\
&\textit{where} \\
&M = \textit{mobility, or number of degrees of freedom} \\
&n = \textit{total number of links, including the ground} \\
&f_1 = \textit{number of one-degree-of-freedom joints} \\
&f_2 = \textit{number of two-degree-of-freedom joints} \\
&f_3 = \textit{number of three-degree-of-freedom joints} \\
&f_4 = \textit{number of four-degree-of-freedom joints} \\
&f_5 = \textit{number of five-degree-of-freedom joints}
\end{aligned}
\qquad (2.1)
$$

In the case of a framework, all the joints are spheric (3 DOF). Therefore the Kutzbach equation can be written specifically for a framework, as shown in Eq. 2.2.

$$
\begin{aligned}
&M = 6(L - 1) - 3S \\
&\textit{where} \\
&L = \textit{Number of links in the framework} \\
&S = \textit{Number of spheric joints}
\end{aligned}
\qquad (2.2)
$$

When every member is connected to another by a spheric joint there is an idle degree of freedom, namely, the idle rotation of the members about their own axis. Since this does not contribute to the gross mobility, there are 'L' fewer degrees of freedom in the system. The gross mobility of the framework after adjusting for the idle rotation is given in Eq. 2.3.

$$M = 5L - 3S - 6 \tag{2.3}$$

The condition that the mobility of an isostatic framework is zero provides the relationship between the number of links and the number of spheric joints in the framework, as shown in Eq. 2.4.

$$5L - 3S - 6 = 0 \tag{2.4}$$

2.2.2 Joints-Nodes-Links Relationship

A framework is made of binary links, i.e each link has a joint at either end. The links combine to form 'N' vertices or nodes. Since two links combine to form only one joint, there are as many joints as twice the number of links less the number of nodes. This relationship is expressed in Eq. 2.5.

$$S = 2L - N \tag{2.5}$$

2.2.3 Necessary Condition

Combining Eqs. 2.4 and 2.5, a relationship between the number of nodes and the number of links is obtained.

$$3N - L - 6 = 0 \tag{2.6}$$

Equation 2.6 provides the <u>necessary</u> (but not sufficient) condition for identifying isostatic frameworks. This derivation, based on the mobility analysis, is also presented by Arun et al. [1990a]. Since the values for nodes and links can only be positive integers, the simplest isostatic framework is made from three nodes and three

links. This is a triangle and is a planar truss. The simplest framework in space has four nodes and six links. This is the tetrahedral framework. Equation 2.6 can be used to determine the number of links that are present in the framework for a given number of nodes.

2.2.4 Necessary and Sufficient Condition

The degree of a node is defined as the number of links that combine at that node. For example, three links join together to form a third degree node. In a spatial isostatic framework, there are no nodes that are second degree or less. Eq. (2.6), along with the above condition, forms the necessary and sufficient condition to obtain an isostatic framework. The proof for this condition can be realized in the following manner. The mobility equation provides the total mobility of the framework. It is possible, however, that part of the framework is statically indeterminate and the other part has some mobility. A node that is of second degree or less gives rise to such a condition. This can be shown by considering the removal of nodes from the structure beginning with a first degree node. The mobility equation before the removal of any node is given by Eq. 2.7.

$$M = 3N - L - 6 = 0 \qquad (2.7)$$

Consider the removal of a first degree node from the framework. This will result in the loss of one link. The mobility of the framework after the removal of the first degree node is given in Eq. 2.8.

$$\begin{aligned} M &= 3(N-1) - (L-1) - 6 \\ &= 3N - L - 6 - 2 = -2 \end{aligned} \qquad (2.8)$$

The removal of a second degree node, results in the loss of two links. The mobility of the resulting framework is shown in Eq. 2.9.

$$\begin{aligned} M &= 3(N-1)-(L-2)-6 \\ &= 3N-L-6-1=-1 \end{aligned} \tag{2.9}$$

The removal of a third degree node results in the loss of three links. The mobility equation now becomes

$$\begin{aligned} M &= 3(N-1)-(L-3)-6 \\ &= 3N-L-6=0 \end{aligned} \tag{2.10}$$

It is clear that the removal of a first or a second degree node leaves the structure statically indeterminate and the removal of a third degree node leaves the structure unaffected. On continuing the same scheme, it can be shown that the removal of nodes of higher degree will result in positive mobility in the structure. Therefore, only the presence of first and second degree nodes will induce local static indeterminacy. Hence the proof.

2.3 Generation

It is evident from Eq. 2.6 that several link and node combinations can form isostatic frameworks. The least number of nodes to form a spatial isostatic framework in space is four. Six links combine at four nodes to form a tetrahedron, the simplest isostatic framework. Tay and Whitley [1985] present a method for generating isostatic frameworks, called Henneberg's method. This method is based on the addition and removal of links and nodes and can potentially produce all possible isostatic frameworks. According to Tay and Whitley, "every graph of an isostatic framework

in space is generated from a Henneberg tree of graphs, each built from the one or two adjacent graphs by adding a new vertex, and deleting a few edges, using a simple local pattern of edges." However, the method has some limitations. The method does not guarantee the enumeration of all possible frameworks for a given number of nodes without an exhaustive search. This method also does not allow the selective generation of deltahedral frameworks[1]. Timoshenko and Young [1965] prove that the members of any self-contained space truss (isostatic framework) represent the edges of a closed polyhedron having triangular faces without internal diagonals. Due to the presence of internal diagonals the non-deltahedral frameworks have not been of much interest [Timoshenko and Young, 1965]. A heuristic scheme to generate deltahedral frameworks is provided by Arun et al. [1990a]. This generation scheme based on Eq. 2.6, however, provides only selected deltahedral frameworks.

2.3.1 Method of degree sequences

An original method to partially enumerate isostatic frameworks for a given number of nodes can be obtained by considering the degrees of the nodes in the framework. If n_i is the number of nodes of degree i, then the sum of all nodes of various degrees should provide the total number of nodes. This is given in Eq. 2.11.

$$n_3 + n_4 + n_5 + n_6 + .. + n_i + .. = N \qquad (2.11)$$

In Eq. 2.11, i is greater than two due to the necessary condition developed in section 2.2. If there are N nodes, the greatest degree of a node can only be 'N-1', since every

[1] Polyhedra with just triangular faces are referred to as deltahedra.

node can only be connected to 'N-1' nodes. This leads to Eq. 2.12.

$$n_3 + n_4 + n_5 + .. + n_i + .. + n_{N-1} = N \qquad (2.12)$$

Since all the links of the isostatic framework are binary, the sum of the product of the number of nodes of each degree and the degree of the node is equal to twice the number of links. This relationship is presented in Eq. 2.13.

$$3n_3 + 4n_4 + 5n_5 + .. + (N-1)n_{N-1} = 2L = 6N - 12 \qquad (2.13)$$

All combinations of n_i's that satisfy the Eqs. 2.12 and 2.13 simultaneously form isostatic frameworks. Eqs. 2.12 and 2.13 are combined by eliminating n_3 to yield Eq. 2.14.

$$n_4 + 2n_5 + 3n_6 + ... + (N-4)n_{N-1} = 3N - 12 \qquad (2.14)$$

All positive integer results for Eq. 2.14 are valid frameworks.

Consider the case when 'N', the number of nodes, is equal to six. In this case, the number of links are twelve [using Eq. 2.6]. Equations 2.12 and 2.13 can be written for this specific node-link combination as

$$\begin{aligned} n_3 + n_4 + n_5 &= 6 \\ 3n_3 + 4n_4 + 5n_5 &= 24 \end{aligned} \qquad (2.15)$$

Combining the two equations to eliminate n_3, Eq. 2.16 is obtained.

$$n_4 + 2n_5 = 6 \qquad (2.16)$$

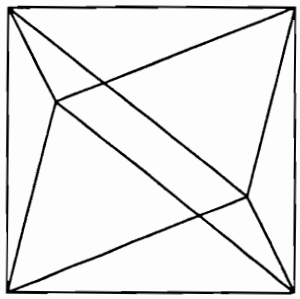
The positive integer values that satisfy Eq. 2.16 can be tabulated as follows.

Sequence No.	n_3	n_4	n_5
1	0	6	0
2	1	4	1
3	2	2	2
4	3	0	3

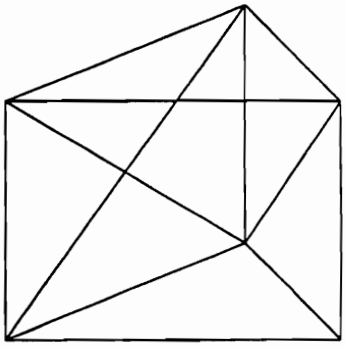
Figure 1 shows the four frameworks listed in the table. The second, third, and fourth cases yield six-noded isostatic frameworks that are formed from tetrahedrons. The presence of n_3 nodes are the reason for this result. It was earlier shown, using Eq. 2.10, that, the removal of a third degree node from a structure does not affect its mobility. In these three cases (i.e sequence numbers two, three, and four) nodes can be removed until the simpler tetrahedral framework results. The framework with the sequence number one, however, cannot be reduced further. This is the octahedral framework. In the case of the six-noded frameworks, a degree sequence, i.e the list of entries in a row for a specific sequence number, results in a unique framework. Therefore, it can be claimed that all possible six-noded frameworks have been enumerated. In the case of larger frameworks, however, a degree sequence need not yield a unique framework[2] (an example is discussed later).

It was stated earlier that a third degree node can always be removed from a framework without affecting the mobility of the structure. This process results in a

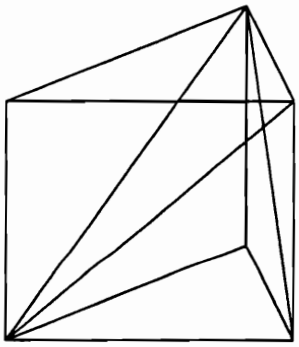
[2] For this reason, complete enumeration, using the method of degree sequences, is not possible.



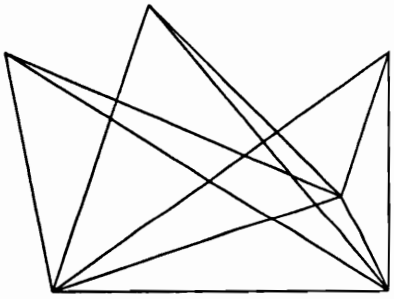
60 Octahedral Framework



141 Framework



222 Framework



303 Framework

Figure 1 Six-Noded Frameworks

simpler framework with each removal. Another way to reduce a larger framework to a simpler framework is by separation of parts that can exist as independent isostatic frameworks. Two isostatic frameworks can be fused together at a triangular face to yield a larger framework. In this dissertation, this process is called 'stacking' and the resulting framework is called a 'stacked' framework. The separation can also be carried out at the common triangle. The following proof shows that, if a smaller isostatic framework is separated from a larger framework by splitting a triangle between them, then the resulting framework is also an isostatic framework. A large framework with N nodes and L links, and a smaller framework which forms a part of the larger framework with N_1 nodes and L_1 links are considered. The mobility relationship, shown in Eq. 2.17, can be written for the two frameworks.

$$\begin{aligned} N - 3L - 6 &= 0 \\ N_1 - 3L_1 - 6 &= 0 \end{aligned} \qquad (2.17)$$

Suppose, the resulting framework after separation has N_2 nodes and L_2 links, the following relationship can be written between the three framework based on the method of separation (i.e. splitting a common triangle resulting in the duplication of three links and three nodes).

$$\begin{aligned} N_1 + N_2 - 3 &= N \\ L_1 + L_2 - 3 &= L \end{aligned} \qquad (2.18)$$

The mobility of the resultant framework can be written as

$$M = 3N_2 - L_2 - 6$$

Since from Eq. (17)

$$N_2 = N - N_1 + 3 \text{ and}$$

$$L_2 = L - L_1 + 3 \qquad (2.19)$$

$$\begin{aligned} M &= 3(N - N_1 + 3) - (L - L_1 + 3) - 6 \\ &= 3N - L - 6 - (3N_1 - L_1 - 6) \\ &= 0 \end{aligned}$$

Hence the proof.

2.3.2 Definition of a Unit Cell

A unit cell is an isostatic framework that cannot be broken down into simpler frameworks either by the removal of third degree nodes or by the separation of simpler frameworks at common triangles. In other words, a unit cell is an isostatic framework that does not contain a simpler unit cell as a part of itself. The tetrahedral framework and the octahedral framework are unit cells. The table of degree sequences can be used to enumerate unit cells. Consider the enumeration of eight-noded unit cells as an example. Avoiding n_3 nodes, since their presence leads to tetrahedrons, Eqs. 2.12 and 2.13 can be written for N = 8.

$$\begin{aligned} n_4 + n_5 + n_6 + n_7 &= 8 \\ 4n_4 + 5n_5 + 6n_6 + 7n_7 &= 36 \end{aligned} \qquad (2.20)$$

Equation 2.21 is obtained by combining the above equations.

$$n_5 + 2n_6 + 3n_7 = 4 \qquad (2.21)$$

The table of degree sequences can be formed from Eq. 2.21 as follows.

Sequence No.	n_4	n_5	n_6	n_7
1	4	4	0	0
2	5	2	1	0
3	6	0	2	0
4	6	1	0	1

Figure 2 shows the five eight-noded frameworks listed in the table. Sequence number two leads to two different frameworks, shown in Fig. 2.2a and 2.2b. This is an example of the non-uniqueness described earlier. It can be seen that the frameworks shown in Fig. 2.1, 2.2a, and 2.3, corresponding to sequence numbers one, two, and three, do not have simpler frameworks as part of their structure. Therefore, these are unit cells. However, there are no methods currently available to distinguish between unit cells and stacked frameworks without actually building the frameworks for the specific degree sequence. This is another limitation of the method of degree sequences.

2.3.3 Planar Graph Based Generation of Deltahedral Unit Cells

The table of degree sequences consists of frameworks that form the edges of a deltahedra and the ones that do not. Furthermore, they can be a unit cell or a stacked framework. The frameworks that form the edges of a deltahedra are called deltahedral frameworks. Since the commonly used frameworks are all deltahedral, the generation and representation of deltahedral unit cells holds the most practical importance. Due to the limitations in the method of degree sequences, a graphical approach is adopted for the selective generation of deltahedral unit cells. It is well-

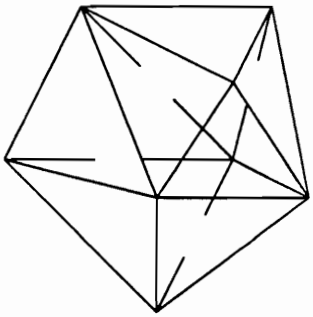


Fig. 2.1 4400 Framework

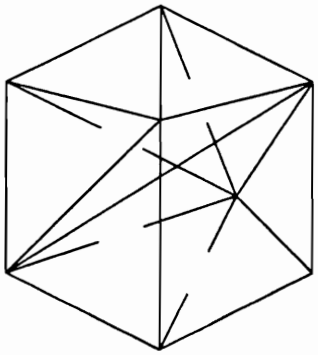


Fig. 2.2a 5210 Framework

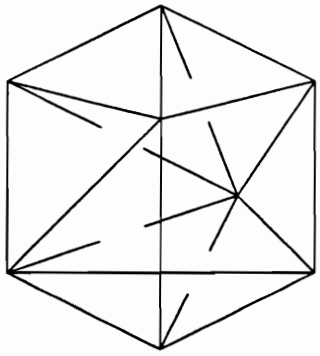


Fig. 2.2b 5210 Framework

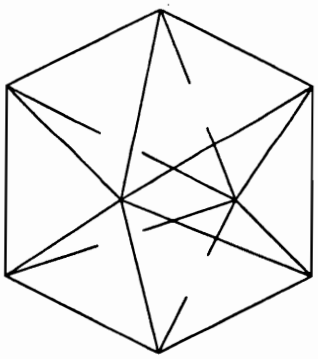


Fig. 2.3 6020 Framework

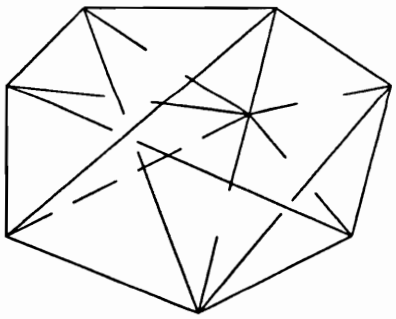


Fig. 2.4 6101 Framework

Figure 2 Eight-Noded Frameworks

known that convex polyhedra can be represented as planar graphs[3]. Since deltahedra are isomorphic to convex polyhedra, they can also be represented as fully triangulated planar graphs. The connectivity between the nodes is preserved in the graph giving a realistic representation.

Parameters of the graph

In the planar graph representation, the nodes of the framework correspond to vertices of the graph, the links to the edges, and the faces of the framework to the faces of the graph. These parameters are related as follows:

$$\text{Number of vertices in the graph } V = N$$
$$\text{Number of edges in the graph } E = L = 3N - 6$$
$$\text{Number of faces in the graph } F = 2(N - 2)$$

Maximum degree of a node in an N-noded Deltahedra

The maximum degree of a node in any framework can only be equal to 'N-1', however, such a framework cannot be represented in a plane without intersecting edges. Therefore, it cannot be a deltahedra either. A framework with a maximum degree of 'N-2' is possible. Such a framework can be visualized to be a chain of 'N-2' links to which two nodes are added, each connected to every node in the chain. Such frameworks are called bi-pyramids by some authors. Since, any N-noded deltahedral framework can be constructed with just fourth and fifth degree nodes (a partial proof can be found in the next section), the maximum degree of a node in a N-noded framework can vary from 5 to N-2.

---

[3] Planar graphs are graphs that do not have intersecting edges. In this dissertation, the planar graphs are strict graphs (i.e. without loops or multiple joins)

Minimum degree of a node in an N-noded Deltahedra

The minimum degree of a node in a framework is either four or five. Equations 2.12 and 2.13 are useful in explaining this result. Suppose, frameworks with a minimum nodal degree of six exist. Equations 2.12 and 2.13 can be rewritten for this case, as Eqs. (2.22).

$$\begin{aligned} n_6 + n_7 + n_8 + \ldots + n_{N-2} &= N \\ 6n_6 + 7n_7 + 8n_8 + \ldots + (N-2)n_{N-2} &= 6N - 12 \end{aligned} \qquad (2.22)$$

Combining the equations by eliminating n_6 yields Eq. 2.23.

$$n_7 + 2n_8 + \ldots + (N-8)n_{N-2} = -12 \qquad (2.23)$$

Since the right hand side of the equation is negative, it is clear that a framework without fourth or fifth degree nodes is not possible. It can be shown that, based on a similar technique, the number of nodes has to be greater than or equal to twelve for the lowest nodal degree to be equal to five. A deltahedra with the lowest nodal degree of four exists for all nodes[4].

Enumeration

All fully triangulated planar graphs with a minimum degree of four or five will produce deltahedral frameworks. The stacked frameworks are included in this set. It is, however, possible to separate the stacked frameworks and the unit cells by

[4] The tetrahedral framework with a maximum and minimum degree of three and the octahedral framework with a maximum and minimum degree of four are the only exceptions to the above rules.

considering the connectivity of the planar graphs. Stacked frameworks are all three-connected planar graphs[5]. Therefore, planar graphs that are four-connected are deltahedral unit cells. In this dissertation, it has not been attempted to enumerate the fully-triangulated four-connected N-noded planar graphs. However, a table of graphs upto N equal to nine is provided in Fig. 3. It is likely that an algorithm to enumerate these graphs is already available[6].

2.3.4 Cox's Method

Another interesting method to generate complex deltahedral unit cells is achieved by the combination of simpler frameworks. Cox [1936] presents this approach without generation as the ultimate objective. Theorem 2.3 of Cox is stated here. "If in a simply-stiff space (isostatic) framework the member between any two joints is replaced by a quartet with feet[7] at these two joints and at two other joints of the framework, the resulting framework is simply-stiff."

The theorem can be restated in a more general manner in the following form. When two adjacent faces of a unit cell are attached to two such similar faces of another unit cell, such that the triangles of one are fused with the other, then if the

[5] An n-connected graph is a graph in which a minimum of n nodes have to be removed to disconnect the graph. In stacked frameworks, removing the three nodes that form the common triangle disconnects one part from another.

[6] Tutte [1984] in his book on 'Graph Theory' explains most of the terms used here.

[7] A quartet with feet is a double-tetrahedral framework with one link that is part of the common triangle removed.

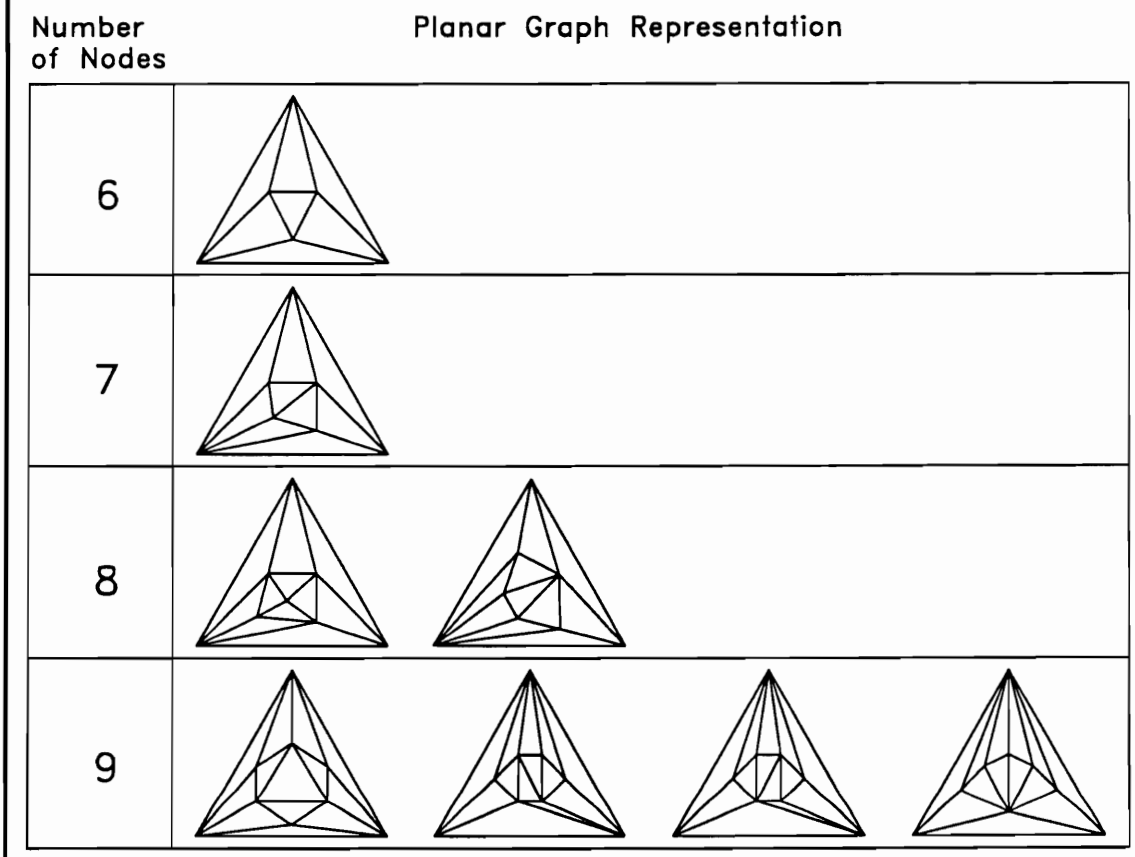


Figure 3 Deltahedral Unit Cells (Planar Graph Representation)

common link between the two triangles is removed, the resultant structure is an isostatic framework. The resultant framework is categorized as a unit cell because it cannot be reduced further into simpler frameworks. Only the double-tetrahedral framework is necessary as a building block to generate any complex framework. In this dissertation, this original method of generating deltahedral unit cells is called "Cox's method" and the process is called "fusion". Fig. 4a shows an octahedral framework formed by the fusion of two double-tetrahedral frameworks. The cube truss in Fig. 4b can be thought to be formed by the fusion of two eight-noded frameworks. The eight-noded framework itself is the result of the fusion of several double-tetrahedral frameworks. In principle, this method is similar to the Henneberg's method [Tay and Whitley, 1985] described earlier. The process of addition and removal of nodes in the Henneberg's methods can be related to the addition and removal of double-tetrahedral frameworks in Cox's method. Due to the presence of similar limitations to that of the Henneberg's method, this method is not recommended here.

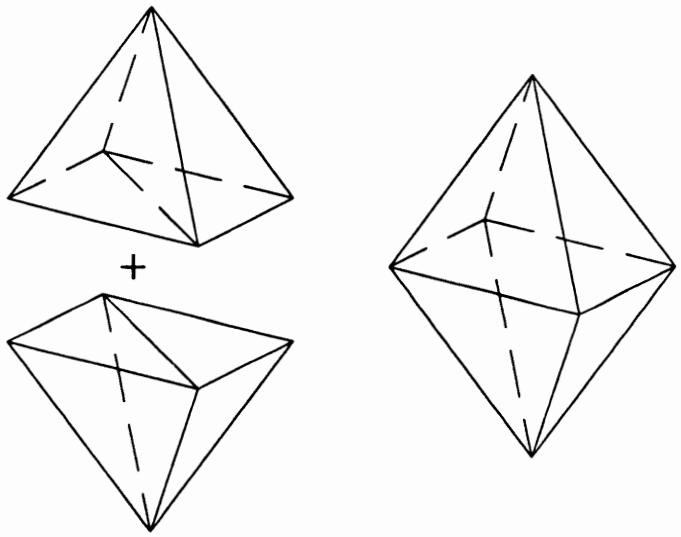


Fig. 4a Fusion of Two Double–Tetrahedral Frameworks

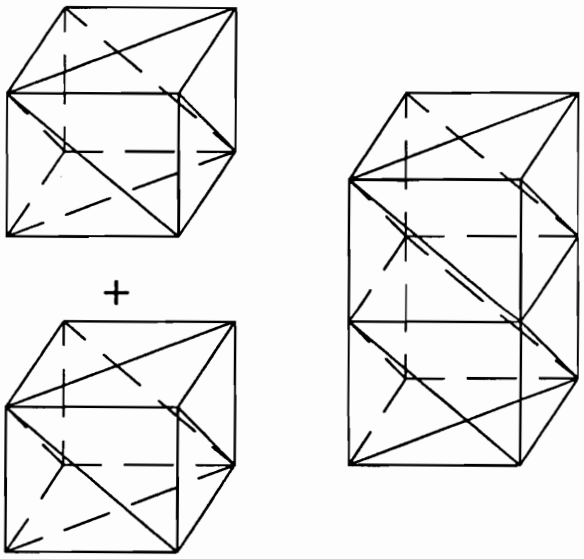


Fig. 4b Fusion of Two Eight–Noded Frameworks

Figure 4 Formation of Unit Cells (Cox's Method)

3. Classification, Geometry Determination and Static Analysis of Isostatic Frameworks

3.1 Classification

Typical isostatic frameworks are stacked deltahedral frameworks. A framework is often named based on the type of unit cell present in its structure. A double-tetrahedral and a double-octahedral framework are examples of such frameworks. For the purpose of structural analysis, it is sufficient to develop methods to solve the unit cells, since all larger framework are composed of some combinations of unit cells. Therefore, in the following sections, only the deltahedral unit cells will be dealt with.

3.1.1 Tetrahedrizing Unit Cells

The methods discussed later in the sections on the generation of geometry, determination of nodal coordinates and static analysis depend on the process of tetrahedrizing unit cells[1]. The process of tetrahedrizing unit cells is explained with the eight-noded framework as an example. Two deltahedral unit cells exist in the case of eight-noded frameworks. Projections of these two frameworks are shown in Fig. 5. In the case of the eight-noded framework, shown in Fig. 5.1, two links are added to minimally tetrahedrize the structure. By adding links 34 and 35, seven

[1] The process of tetrahedrizing unit cells results in every link being part of a tetrahedron. This is accomplished by the addition of links to the existing framework, making the framework statically indeterminate.

tetrahedrons 3412, 3425, 3456, 3467, 3471, 3528, and 3586 are formed. On the other hand, in the case of the other eight-noded framework, shown in Fig. 5.2, by adding the link 34, six tetrahedrons, 3412, 3425, 3456, 3467, 3478, and 3481 are formed[2].

It is interesting to note that the links removed in the generation of unit cells based on Cox's method, using the double-tetrahedral framework as the building block, are replaced for the purpose of tetrahedrizing the unit cells[3]. The octahedral framework shown in Fig. 4a is a good example. The replacement of the same links, however, does not guarantee that the unit cell is minimally tetrahedrized.

3.1.2 Classification Criterion

The number of links necessary to minimally tetrahedrize a unit cell is used as the criterion for classification in this dissertation. The number of tetrahedrons formed in the tetrahedrized framework is referred to as the tetrahedron number. The eight-noded framework shown in Fig. 5.1 has a tetrahedron number of seven. The framework shown in Fig. 5.2 has a tetrahedron number of six.

3.1.3 Naming Convention

The unit cell is named after the deltahedron (based on the number of faces) preceded by the degree sequence and followed by the tetrahedron number. If the

---

[2] The minimum number of links that have to be added to make every link part of a tetrahedron.

[3] This is probably an advantage for Cox's method.

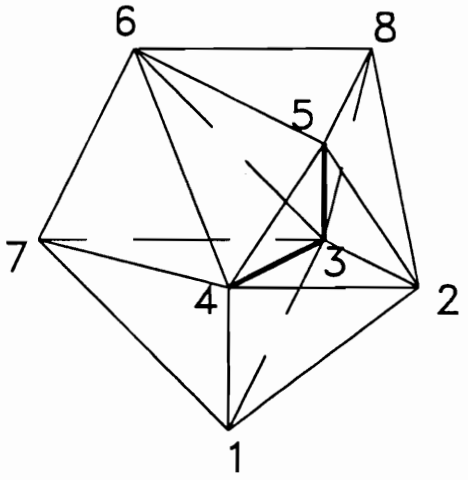


Fig. 5.1 4400 Dodecahedral-7 Framework

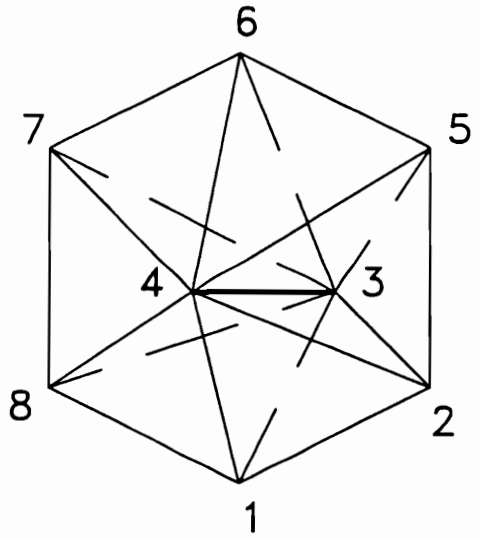


Fig. 5.2 6020 Dodecahedral-6 Framework

Figure 5 Tetrahedrizing Frameworks

degree sequence is too long, it may be omitted. The framework shown in Fig. 5.1 is called the 4400 dodecahedral-7 framework. The framework shown in Fig. 5.2 is called the 6020 dodecahedral-6 framework.

3.2 Generation of Geometry and Determination of Nodal Coordinates

3.2.1 Method of Tetrahedrons

The first step in the generation of geometry of a unit cell is to tetrahedrize the unit cell minimally. The dimensions of the links that are added to tetrahedrize the framework are assumed arbitrarily or based on some calculated guesses. The coordinates of the nodes are obtained in closed-form by solving tetrahedrons. In the process of solving the nodal coordinates, some link dimensions are not used. The number of links that are not used are the same as the number of added links. An iterative scheme can now be designed to correct the dimensions of the added links corresponding to the dimensions of the unused links.

The solution scheme, therefore, depends on solving tetrahedrons. There are two solutions or closures to every tetrahedron. The various combinations of these solutions lead to the various closures or solutions of the framework. A specific geometry can be solved by judiciously selecting one closure among the various possible closures. It is encouraging to note that the number of added links are few even in fairly complex deltahedral frameworks. When there are only one or two added links, an exhaustive search can be performed to determine the correct dimensions of the added links, however, a more efficient method may be designed using nonlinear

optimization techniques. A flow chart describing the method is presented in Fig. 6.

The problem of determination of nodal coordinates is also part of the direct or forward kinematic problem. The solution to this problem involves the determination of the coordinates of all the nodes when provided with the lengths of all links and the location of a base triangle in some coordinate system. In the case of the tetrahedral framework, the problem is solved easily by obtaining a single quadratic equation. This has been well known for centuries and forms the foundation for the method described here. Only recently, researchers have obtained a single sixteenth degree equation to solve the octahedral framework [Griffis and Duffy, 1989]. The solutions to more complex unit cells have required the use of sophisticated mathematical techniques [Arun et al. 1990b]. Therefore, the method of tetrahedrons is a powerful new method for the generation of geometry and the determination of nodal coordinates of the isostatic framework.

3.2.2 Results

The solution to the geometry of the two dodecahedral frameworks are presented as an example. Fig. 7 shows the 6020 dodecahedral-6 framework with the nodes identified. The base triangle coordinates (nodes 1, 2, and 3) and all the link lengths are known. The link 34 is added to form the tetrahedrons 3412, 3425, 3456, 3467, 3478, and 3481. A plot showing the variation of a function based on the unused dimension 18, for one of the 64 closures is shown in Fig. 8. The function used in this example is given in Eq. 3.1.

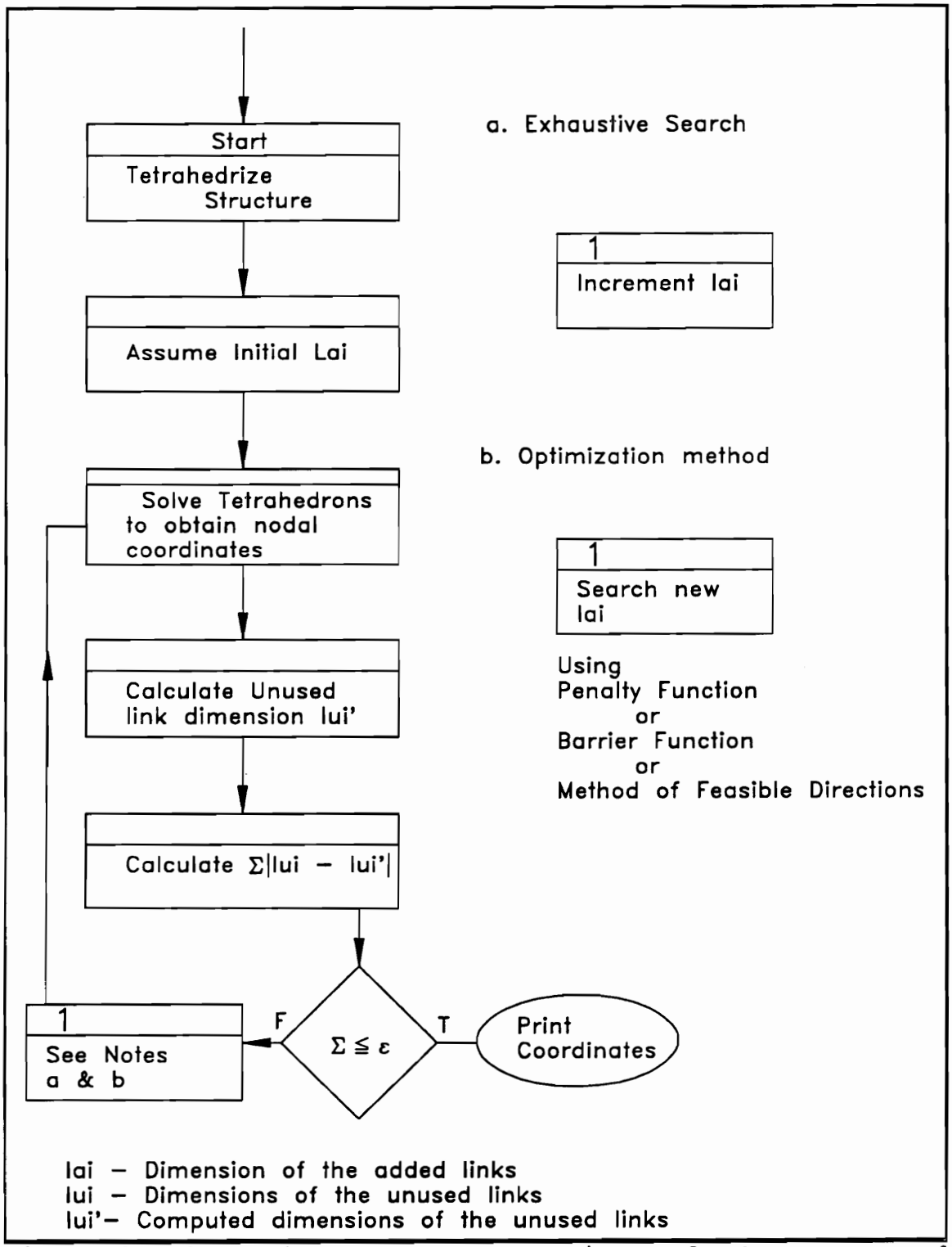
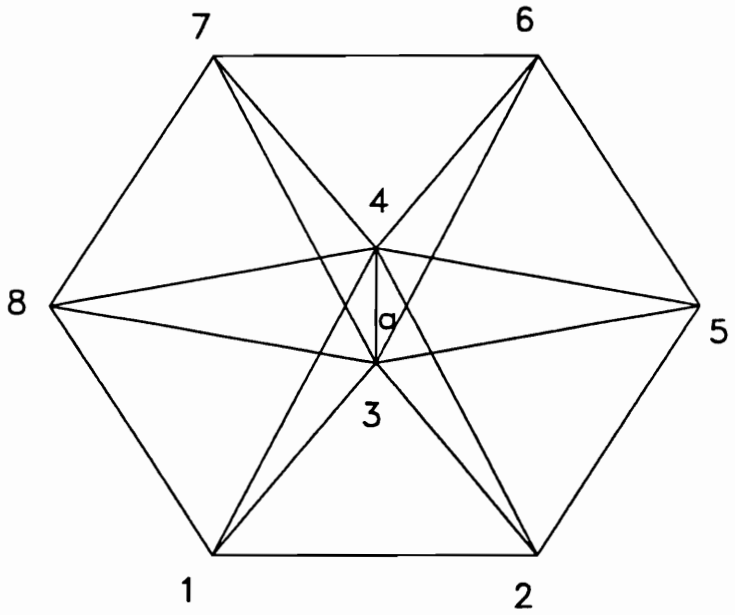


Figure 6 Flow Chart For Generation of Geometry and Determination of Nodal Coordinates



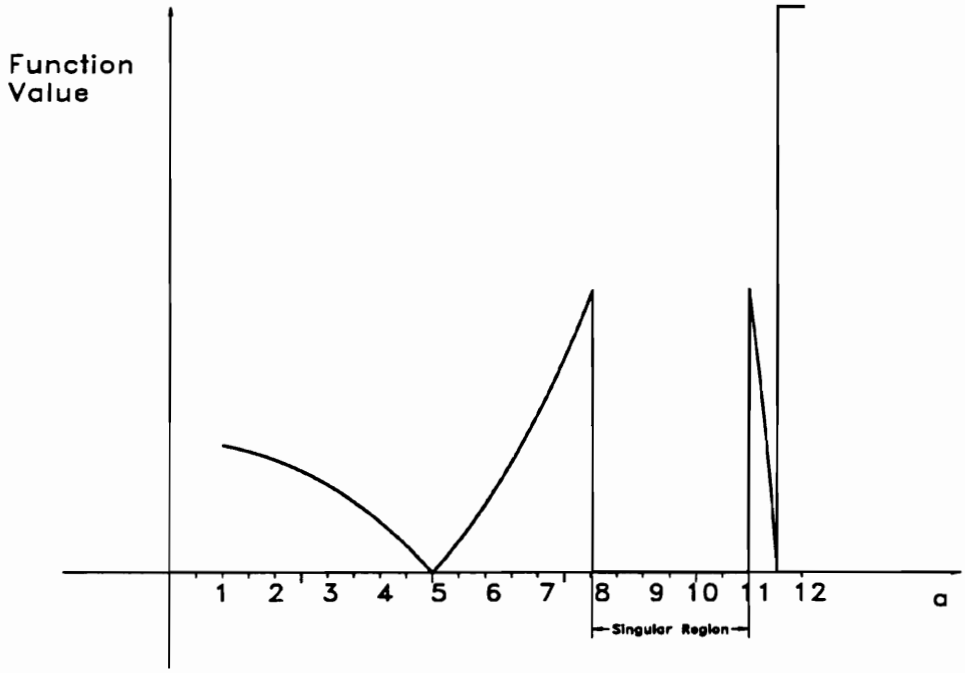
Given Nodal Coordinates

Node #	X	Y	Z
1	0.0000	0.0000	0.0000
2	6.0000	0.0000	0.0000
3	3.0000	5.7663	0.0000

Link Lengths

114	6.5
124	6.5
145	6.5
146	6.5
147	6.5
148	6.5
135	6.5
136	6.5
137	6.5
138	6.5
125	6.0
156	6.0
167	6.0
178	6.0
118	6.0

Figure 7 Geometry Determination of 6020 Dodecahedral-6 Framework I



Nodal Coordinates

Node #	X	Y	Z
1	0.0000	0.0000	0.0000
2	6.0000	0.0000	0.0000
3	3.0000	5.7663	0.0000
4	3.0000	3.5985	4.5056
5	9.0000	4.6824	2.2528
6	6.0000	9.3648	4.5056
7	0.0000	9.3648	4.5056
8	−3.0000	4.6824	2.2528

Figure 8 Geometry Determination of 6020 Dodecahedral-6 Framework II

$$f(a) = \sqrt{(l_{18} - \tilde{l}_{18})^2} \qquad (3.1)$$

Where

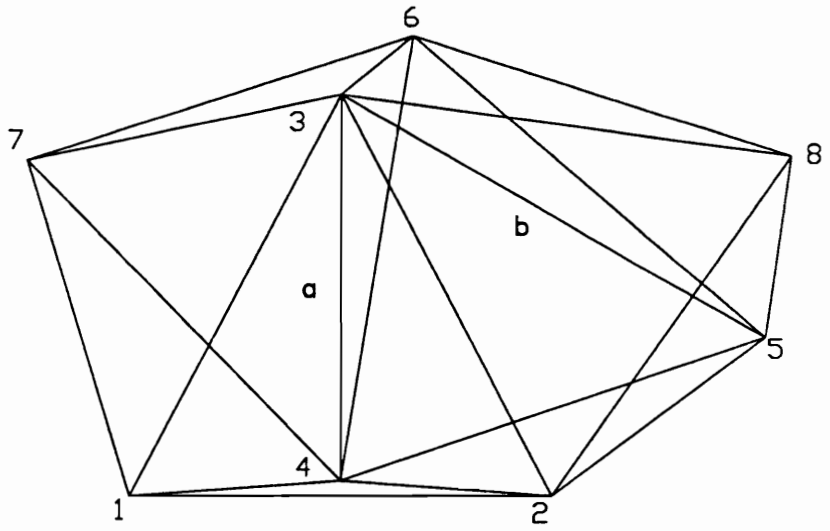
$$\tilde{l}_{18} = \textit{Computed Length of link } 18$$

The point where the curve intersects the X axis is the solution. At this point the computed value for the length of the link 18 is the same as the actual value.

The plot also shows a potential singular configuration for the dodecahedral framework. The merger of the solution curve with the X axis indicates the presence of a singularity. The deduction of such singularities in frameworks are pursued later in the singularity analysis section. A unique advantage of this method of geometry determination is the ability to detect such structural singularities. This is not possible with other analysis methods, including sophisticated methods such as the homotopy based continuation methods.

In the case of the 4400 dodecahedral-7 framework, shown in Fig. 9, two links are necessary to tetrahedrize the structure. Therefore, the solution space is two dimensional. The three-dimensional plot showing the variation of the function based on the unused dimensions 34 and 35 is shown in Fig. 10. The intersection of this solution surface with the XY plane gives the dimensions of links 34 and 35 and the solution.

The C-language program codes used to generate the plots for determining the geometry of the two dodecahedral framework are given in Appendix 1a and 1b. All



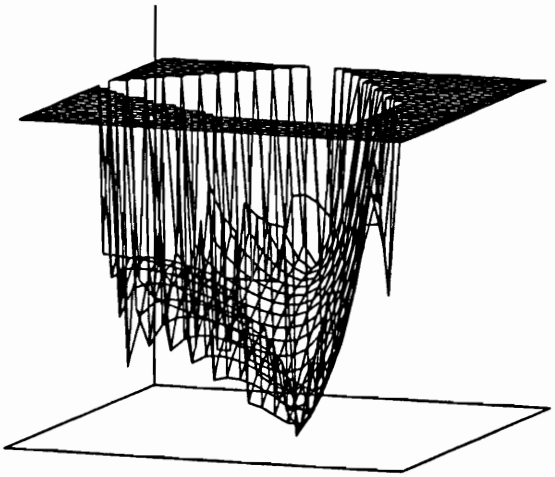
Given Nodal Coordinates

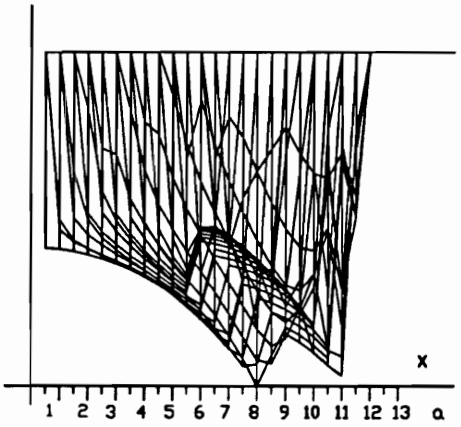
Node #	X	Y	Z
1	0.0000	0.0000	0.0000
2	6.0000	0.0000	0.0000
3	3.0000	5.7663	0.0000

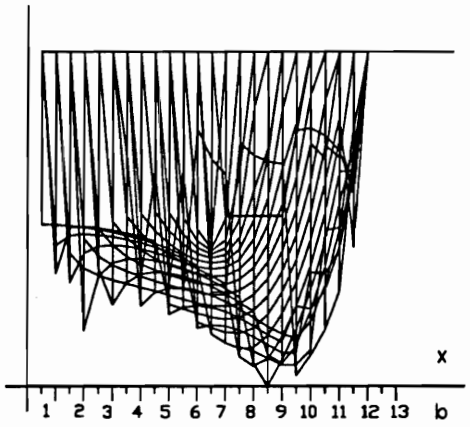
Link Lengths

l14	6.5
l24	6.5
l45	6.5
l46	6.5
l47	6.5
l25	6.5
l58	6.5
l56	6.5
l38	6.5
l36	6.5
l37	6.5
l28	6.0
l68	9.2315
l67	6.0
l17	6.8522

Figure 9 Geometry Determination of 4400 Dodecahedral-7 Framework I







Nodal Coordinates

Node #	X	Y	Z
1	0.0000	0.0000	0.0000
2	6.0000	0.0000	0.0000
3	3.0000	5.7663	0.0000
4	3.0000	0.2168	5.7622
5	9.0043	2.6252	5.1315
6	4.0178	6.6083	6.3644
7	−1.4471	4.8242	4.6461
8	9.3669	4.8733	−0.9566

Figure 10 Geometry Determination of 4400 Dodecahedral-7 Framework II

the nodal coordinates and the corresponding solution plots are shown in Figs. 8 and 10.

3.3 Static Analysis

3.3.1 Method

The unit cells except the tetrahedron belong to the category of complex trusses. Hence, the well-known methods of joints or sections cannot be used on any of the unit cells except the tetrahedron. Henneberg [1911] has devised a method to solve the forces in complex trusses [Timoshenko and Young, 1965]. In Henneberg's method, a complex truss is converted into a tetrahedral framework by the removal and addition of a suitable number of links in the truss. The forces in the members of a tetrahedral frameworks are easily solvable for the given loading condition and for a set of fictitious loads of unknown magnitude acting along the direction of the removed members. The forces in the added links due to the loads are superimposed and set to zero. The forces in the members can be obtained by solving for the unknown load.

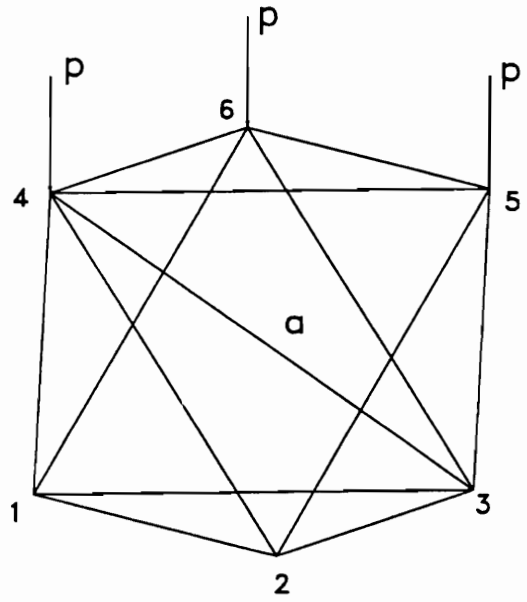
The process of generating a tetrahedral framework from a complex truss has been quite arbitrary. In this dissertation, using the tetrahedrizing technique, a logical method is provided. The tetrahedrized framework is used as a starting point for performing static analysis. In order to obtain the minimum number of unknowns in the analysis, the unit cell is minimally tetrahedrized. Appropriate links are then removed (equal to the number of links added to tetrahedrize the structure) to obtain a tetrahedral framework. The forces in the links of a tetrahedral framework can be

easily solved using the method of joints for a given loading condition. Once the forces in the links have been obtained, the structure is analyzed for a load applied along the direction of the removed links. This load is of an unknown magnitude {X}, where {X} is a vector representing one or more loads. The forces in the links can be computed for this new set of loads in terms of {X}. By superimposing the later loading condition with the given loading condition, the axial forces in the fictitious tetrahedral truss are calculated. Since, in the real framework, the added links do not exist, the unknown loads are solved by setting the loads in the added links to be zero. The forces in the removed links are given by the unknown loads. This gives a closed-form solution technique for finding the forces in any isostatic framework.

3.3.2 Results

Consider the analysis of the octahedral framework loaded as shown in Fig. 11. The base triangle coordinates and the link dimensions are also known. The tetrahedron-based geometry determination scheme is used to obtain the coordinates of all the nodes. Since the solution space is one dimensional, an exhaustive search approach was followed. The C-language code for the geometry determination is shown in Appendix 2a. The coordinates of all the nodes are determined (shown in Fig. 11).

Following the approach described earlier, link 34 is added to tetrahedrize the structure. In this example, link 56 is removed to make the structure a statically determinate tetrahedral framework. The method of joints is used to solve for the links forces in this fictitious truss using the given loading condition. These values are listed



Link Lengths

114	6.0
145	6.0
135	6.0
124	6.0
125	6.0
116	6.0
146	6.0
136	6.0
156	6.0

Nodal Coordinates

Node #	X	Y	Z
1	0.0000	0.0000	0.0000
2	6.0000	0.0000	0.0000
3	3.0000	5.1962	0.0000
4	3.0000	−1.7330	4.8986
5	5.9992	3.4636	4.8993
6	−0.0008	3.4627	4.8979

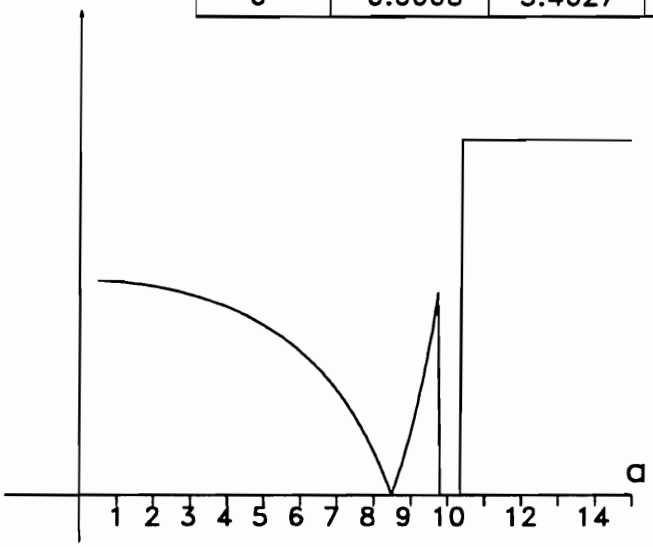
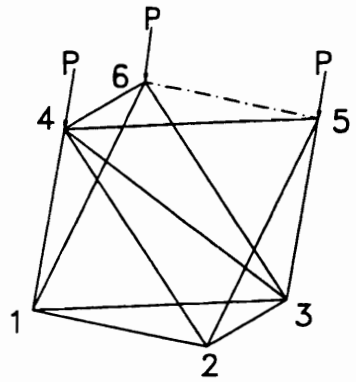
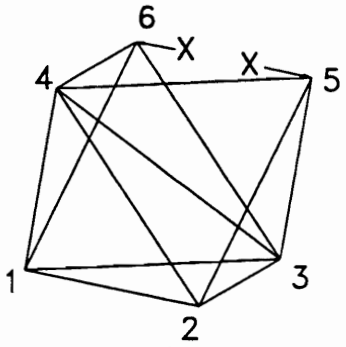


Figure 11 Geometry Determination of the 60 Octahedral-4 Framework

in the table under the heading 'Given Load' in Fig. 12. Now, a load of unknown quantity 'X' is applied along the removed link 56. The forces in the members are again calculated for this load. These are listed in the table under the heading 'Load X'. The sum of these two forces are given in the next column. By setting the force in link 34 to be zero, the unknown load X is solved. The last column in the table provides the value of forces in the links of the real truss. A simple C-language code for performing the method of joints is provided in Appendix 2b.



Given Load



Load X

P = 10 {0.0, 0.0, −1.0}

Link	Direction			Given Load	Load X	Actual Forces
	X	Y	Z			
116	0.0000	3.4627	4.8979	8.1653	X	6.1247
136	−3.0008	−1.7335	4.8979	4.0819	−X	6.1225
146	−3.0008	5.1951	−0.0007	−4.0816	−X	−2.0410
125	−0.0008	3.4636	4.8993	8.1645	+X	6.1239
135	2.9992	−1.7326	4.8993	4.0827	−X	6.1233
145	2.9992	5.1966	0.0007	−4.0805	−X	−2.0399
114	3.0000	−1.7330	4.8986	4.0875	−X	6.1281
124	−3.0000	−1.7330	4.8986	4.0839	−X	6.1245
134	0.0000	−6.9292	4.8986	5.7718	2.8285X	0.0
156	−5.9992	−0.0007	−0.0014	−	X=−2.0406	−2.0406

Figure 12 Static Analysis of the 60 Octahedral-4 Framework

4. Introduction to the Kinematic Analyses of the Double-Octahedral Manipulator

4.1 Introduction

Kinematic analyses is the first step in the process of designing and controlling a manipulator. A manipulator is a device capable of transforming the position and orientation of a moving coordinate frame relative to a reference coordinate frame. In the case of industrial robots, the moving frame describes the position and orientation of an end-effector. The task of a robotics engineer is to control the transformation of the end-effector relative to the reference frame. This transformation is effected by means of actuators connecting the moving frame to the reference frame. Various relationships between the moving coordinate frame and the actuators are established as the result of the analyses.

4.2 Degrees of Freedom

An object freely floating in three-dimensional space has six degrees of freedom, or six independent motions. These motions may be accounted for as translations in any three mutually perpendicular axes and rotations about any three mutually perpendicular axes. Since the object to be controlled is rigidly attached to the end-effector, controlling the motion of the object amounts to controlling the transformation of the moving frame.

4.2.1 Degrees of Freedom in Joint Space

The number of degrees of freedom of a manipulator in joint space is equal to the number of actuators present in the manipulator. The actuators usually provide either pure rotation or translation, but, in some cases, they provide more complex motion. The degrees of freedom in joint space forms the set of output parameters for the inverse kinematic analysis.

4.2.2 Degrees of Freedom in Object Space

The number of degrees of freedom of a manipulator in object space is the number of independent motions that can be specified for the moving frame. This forms part of the set of input parameters to the inverse kinematic analysis. The number of degrees of freedom in object space can only be a maximum of six. The set of input parameters, however, may include the position and orientation of intermediate links of the manipulator. This arises in the case of redundant manipulators. In these manipulators, the number of degrees of freedom in joint space are greater than the number of degrees of freedom in object space. These extra degrees of freedom may be used in applications involving obstacle avoidance in an unstructured environment.

4.3 The Octahedral Manipulator

4.3.1 Structural Design

When some links of an isostatic framework are replaced by variable length members (actuators), then if the framework has proper joints, it can function as a

manipulator. If the isostatic framework is an octahedral framework, then the manipulator is called the octahedral manipulator. A single octahedral manipulator is shown in Fig. 13a. Actuating the six legs of the octahedral framework, leaving the top and bottom triangles fixed, leads to the so-called 3-3 Stewart platform [Griffis and Duffy, 1989]. When two octahedral frameworks are stacked together, the resulting configuration is called a double-octahedral framework. This configuration, shown in Fig. 13b, consists of three triangular planes joined together by twelve links, such that there are six links between each plane. These three transverse triangular planes are referred to as the fixed plane, the mid-plane, and the moving plane. The members that form these triangular planes are called battens. The six members on either side of the mid-plane that connect the three planes together are called the longerons.

4.3.2 Actuation Scheme

In the double-octahedral framework, fifteen links (12 longerons and 3 mid-plane battens) have the potential to be made variable. When the mid-plane battens are made variable, called mid-plane actuation, the framework exhibits certain special properties. One such property is collapsibility. Another property is that the mid-plane acts as a plane of symmetry through out the entire range of actuation, provided the two unit cells are constructed as mirror images at the beginning. The usefulness of this property will be explained later. This double-octahedral manipulator has three degrees of freedom and is studied in the following chapters.

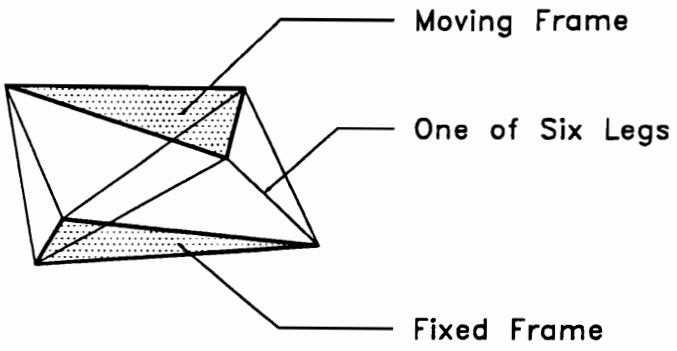


Fig. 13a Single–Octahedral Framework

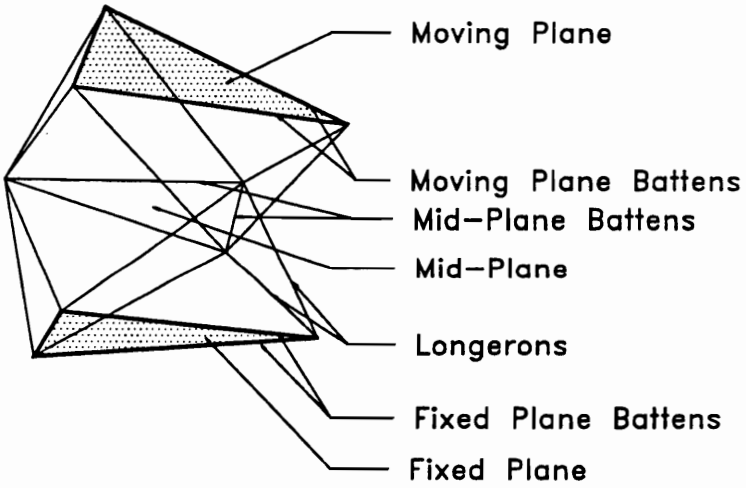


Fig. 13b Double–Octahedral Framework

Figure 13 Octahedral Frameworks

4.3.3 Joint Design

In a perfect framework, the joints must be designed with three rotational degrees of freedom between each connected link to provide the necessary relative motion. This eliminates the transmission of bending moments, shear forces and torques to the members of the framework. This also allows any member of the framework to be made variable. Only three links are, however, actuated in the case of the double-octahedral manipulator. This permits the possibility of reducing the number of degrees of freedom provided by some of the joints without affecting the required gross mobility. Note, however, that the framework will no longer be ideal and that small bending, torsion and shear loads may be induced in the links.

An important objective in the joint design for truss-based manipulators is to substitute revolute joints in the place of spheric joints whenever possible. Such a design will provide improved long-term reliability and low manufacturing costs. Rhodes and Mikulas [1985] developed the first double-octahedral manipulator at the NASA Langley research center to test space-structure deployment concepts. They found that the joints in the fixed plane and the moving plane can be simple revolute joints. The longerons need to only rotate about the non-actuated batten. The design of joints to connect the longerons and the actuated batten in the mid-plane, however, is not as simple. Six links must meet at each node in the mid-plane.

Rhodes and Mikulas' approach introduced an extra link in the joint as shown in Fig. 14a. This "joint link" contains two slotted spheric joints at either end and two

revolute joints in the center. It serves only to provide a connection point for joining links. Two longerons in the same octahedron come together at each spheric joint, leaving an offset between the two. Within this offset region, the two actuated battens are connected by revolute joints to the joint link. If the offset is small, its effect may be ignored in the static analysis, however, it must be considered in the kinematic analyses.

The double-octahedral manipulator built at VPI&SU [Tidwell, 1989] has only revolute joints. This joint concept is shown in Fig. 14b. This joint design also results in a joint offset. Kinematically, the design is similar to the NASA design except the spheric joints were replaced by Hooke's coupling universal joints. However, further mechanical design is necessary to add rigidity to this joint.

4.4 Manipulator Specifications

4.4.1 Input and Output Parameter Specifications

The three DOF double-octahedral manipulator with mid-plane actuation, shown in Fig. 15, forms an independent actuating unit. The lengths of the three actuated battens in the device are the output parameters of the inverse kinematic analysis. There are two important ways to specify the input parameters to the inverse problem for this device. When the position of a point rigidly attached to the moving plane (3 DOF) is specified, the inverse problem is called the positioning problem. The double-octahedral manipulator can also perform the task of a gimbal. This is achieved by specifying the orientation of a normal to the moving plane (2 DOF). A distance

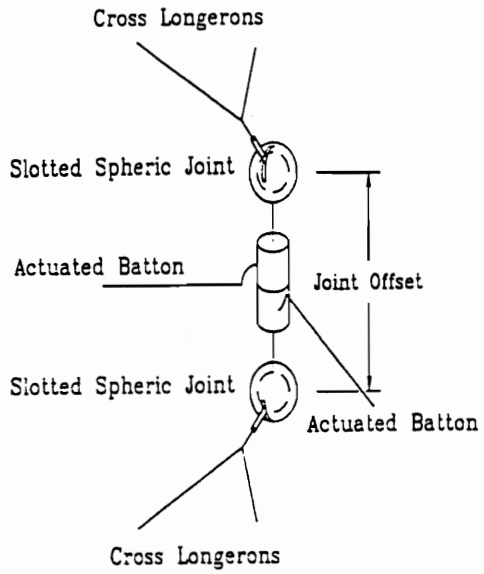


Fig. 14a Mikulas' Joint Design

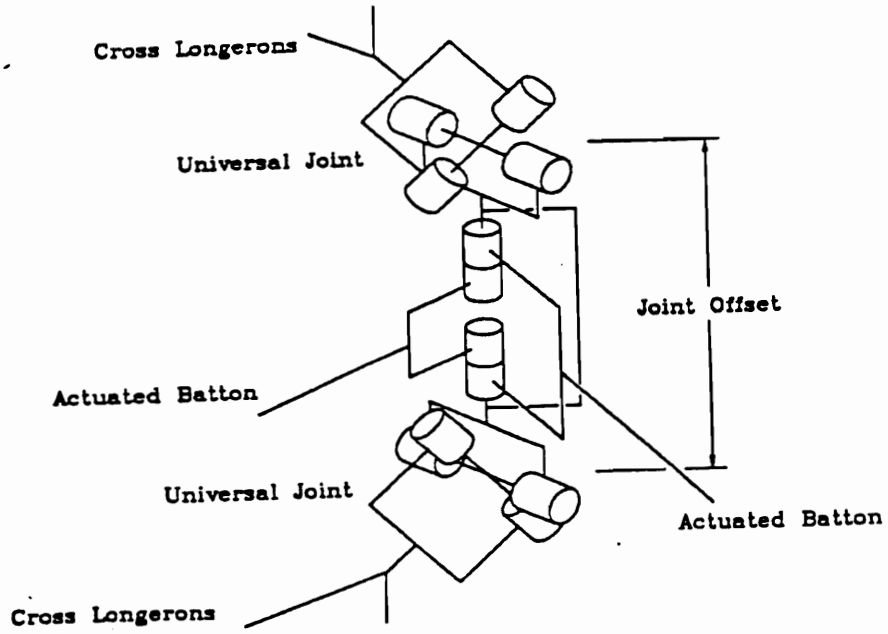


Fig. 14b Tidwell's Joint Design

Figure 14 Joint Design (courtesy : Tidwell [1989])

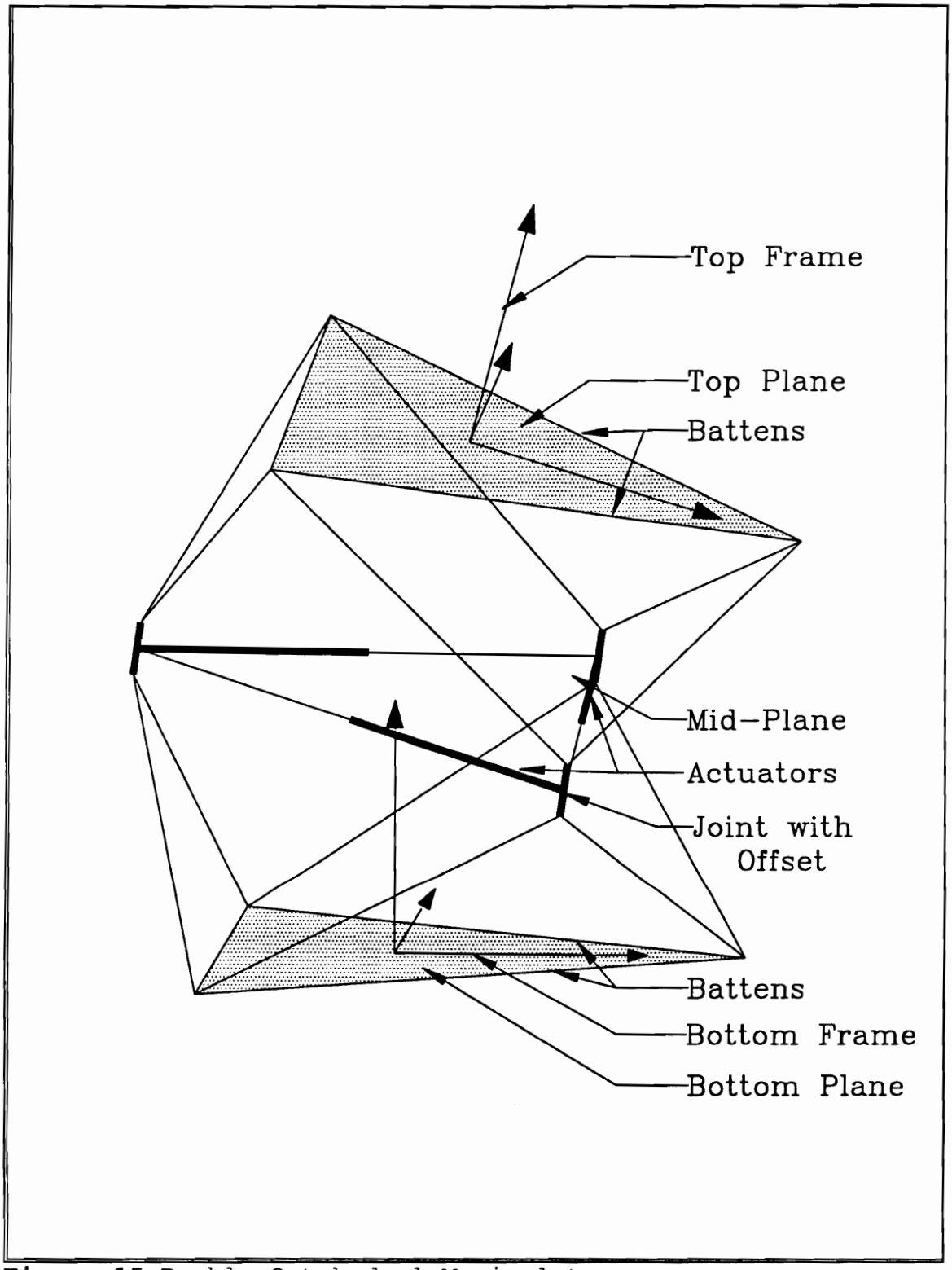


Figure 15 Double-Octahedral Manipulator

corresponding to the height of the framework (1DOF) is also specified using criteria that are explained later . This specification leads to the gimbal problem.

When two double-octahedral manipulators are stacked together, such that they share a common triangle, a six-degree-of-freedom quadruple-octahedral manipulator is formed. The lengths of the six actuated battens are the output parameters to the inverse kinematic analysis. This configuration is shown in Fig. 16. Although there are six degrees of freedom in the joint space (six actuators), there are only five degrees of freedom (position and orientation of the moving plane) in the object space due to the symmetric construction[1]. One of the motions of the moving frame (a rotation about the normal to the moving plane) cannot be specified independently. The extra degree of freedom leads to multiplicity of solutions. The potential use of this device in docking type applications where, two ships or spacecrafts can be brought together in a controlled fashion has led to the inverse problem being called the docking problem.

This dissertation describes the forward kinematic analysis, inverse kinematic analysis, velocity and acceleration analysis, singularity analysis and workspace analysis for the positioning and gimbal specifications to the double-octahedral manipulator. Solutions to the forward and inverse kinematics for the docking specification to the quadruple-octahedral manipulator are also presented.

---

[1] The quadruple-octahedral manipulator is a redundant manipulator because the number of degrees of freedom in joint space are greater than the number of degrees of freedom in object space.

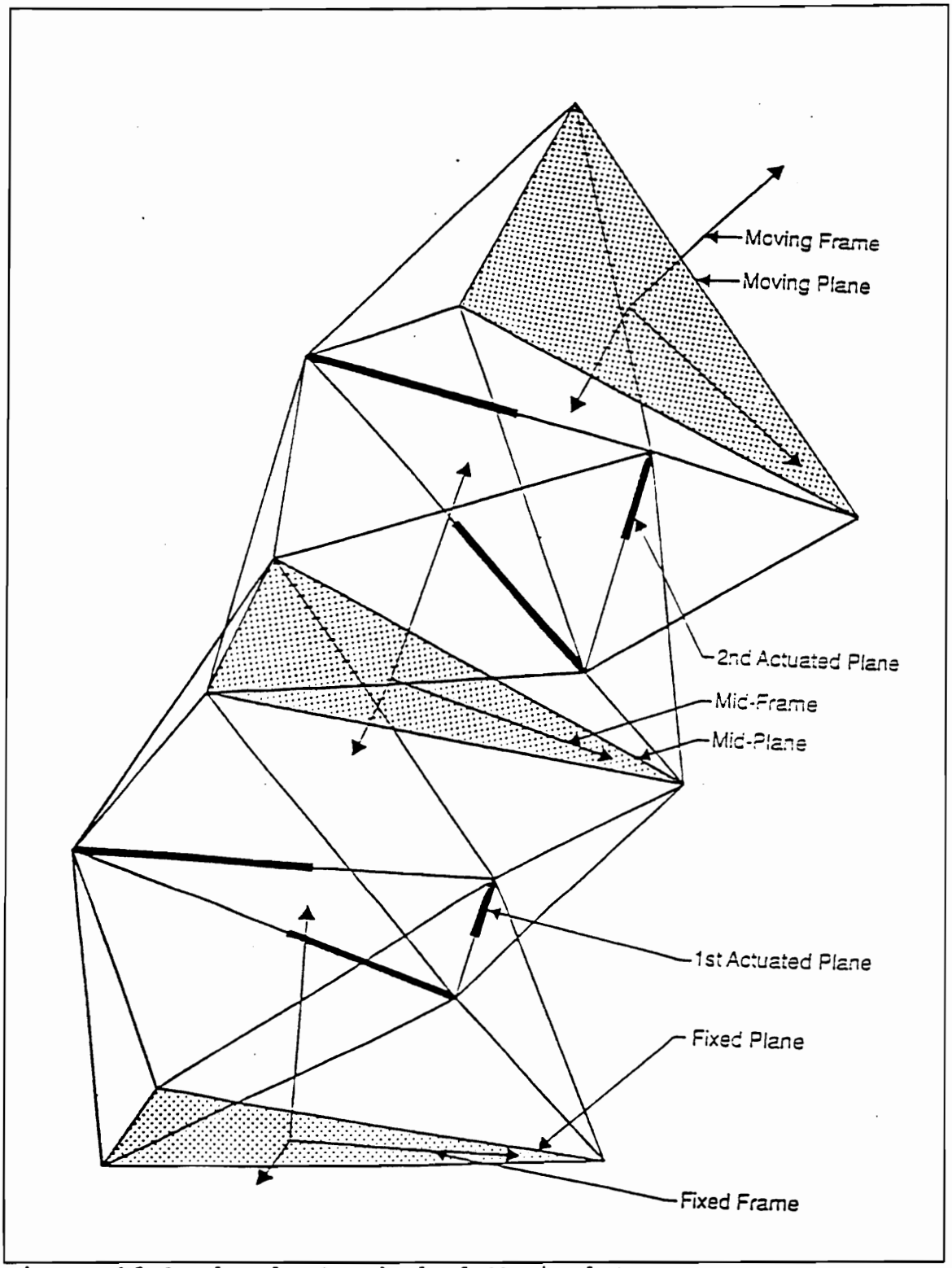


Figure 16 Quadruple-Octahedral Manipulator

4.4.2 Size Specification

The dimensions of the double-octahedral manipulator analyzed here correspond to that of Rhodes and Mikulas' adaptive truss [1987]. This manipulator is designed to have a batten length of 48 units and a longeron length of 34.5 units. Therefore, the octahedral unit cell can have a maximum height of 24.78 units. The three lead screw type actuators are designed to have approximately 8 units of travel between 39 and 47 units. The joints carry an offset of approximately 1.5 units, measured from the spheric joint in the bottom octahedron to the spheric joint in the top octahedron. The plane defined by the spheric joints is called the offset plane and is placed at a distance of 0.75 units from the mid-plane due to the 1.5 units joint offset.

5 Forward Kinematic Analysis

5.1 Objective

The objective of the forward kinematic analysis is usually to determine the position and orientation of one or more links in the manipulator corresponding to a given set of link dimensions. Therefore, in a broad sense, the output from the forward kinematic analysis is similar to the output from the geometry determination problem. In this dissertation, however, the required output from the forward kinematic analysis is limited to the determination of the moving frame parameters. The lengths of the actuators (actuated batten) form the input parameters to this analysis[1]. For convenience, the reference coordinate frame is attached to the fixed plane at the centroid of the bottom triangle such that one of its axis coincides with the normal to the fixed plane. In a similar manner, the moving coordinate frame is attached to the moving plane.

5.2 Method

5.2.1 Solution to the Octahedral Unit Cell

A special method to solve the forward kinematic problem of the double-octahedral manipulator using kinematic equivalents is presented by Reinholtz and

[1] In the case of the inverse kinematic analysis, the input parameters are the moving frame parameters and the output parameters are the actuator lengths. In this way the input and output parameters merely get interchanged between the forward and inverse kinematic analyses.

Gokhale [1987]. This model, shown in Fig. 17, was developed using the knowledge that the offset-plane nodes (spheric joints) follow a circular path. The fixed battens and longerons are replaced by kinematically equivalent Revolute-Spheric (RS) pairs. Therefore, each RS pair represents a longeron triangle. A coupled RSSR linkage is formed that has distinct solutions[2]. Based on the coupled RSSR linkage model, Eq. 5.1 is written defining the position of the offset plane nodes.

For i = 1, 2 and 3

$$\vec{B}_i = [R_{\theta_i, \hat{U}_i}]\{\vec{R}_i\} + \vec{O}_i \qquad (5.1)$$

Where

$\vec{B}_i$ - *Offset Plane Node Coordinates*

$[R_{\theta_i, \hat{U}_i}]$ - *Rotation Matrix*

θ_i - *Rotation angle of the RS Pair*

$\hat{U}_i$ - *Revolute Joint Axis*

$\vec{R}_i$ - *Initial Reference Vector*

$\vec{O}_i$ - *Origin of the Revolute Joint*

Based on the knowledge of the location and orientation of the bottom triangle relative to the reference coordinate frame, it is possible to compute the vectors U_i, R_i, and θ_i. The coordinates of the offset-plane nodes are solely functions of the angle θ_i, since all the other quantities are fixed by the bottom triangle. The condition that offset-plane

[2] A generalized method that is applicable to most frameworks was presented in the 'determination of geometry' section of the structural analysis. Using the tetrahedrizing method, all the possible closures can be obtained and any kind of actuation scheme can be easily handled. The kinematic equivalent model, however, proves to be a better scheme for this specific manipulator, because the model allows easy visualization of the various solutions.

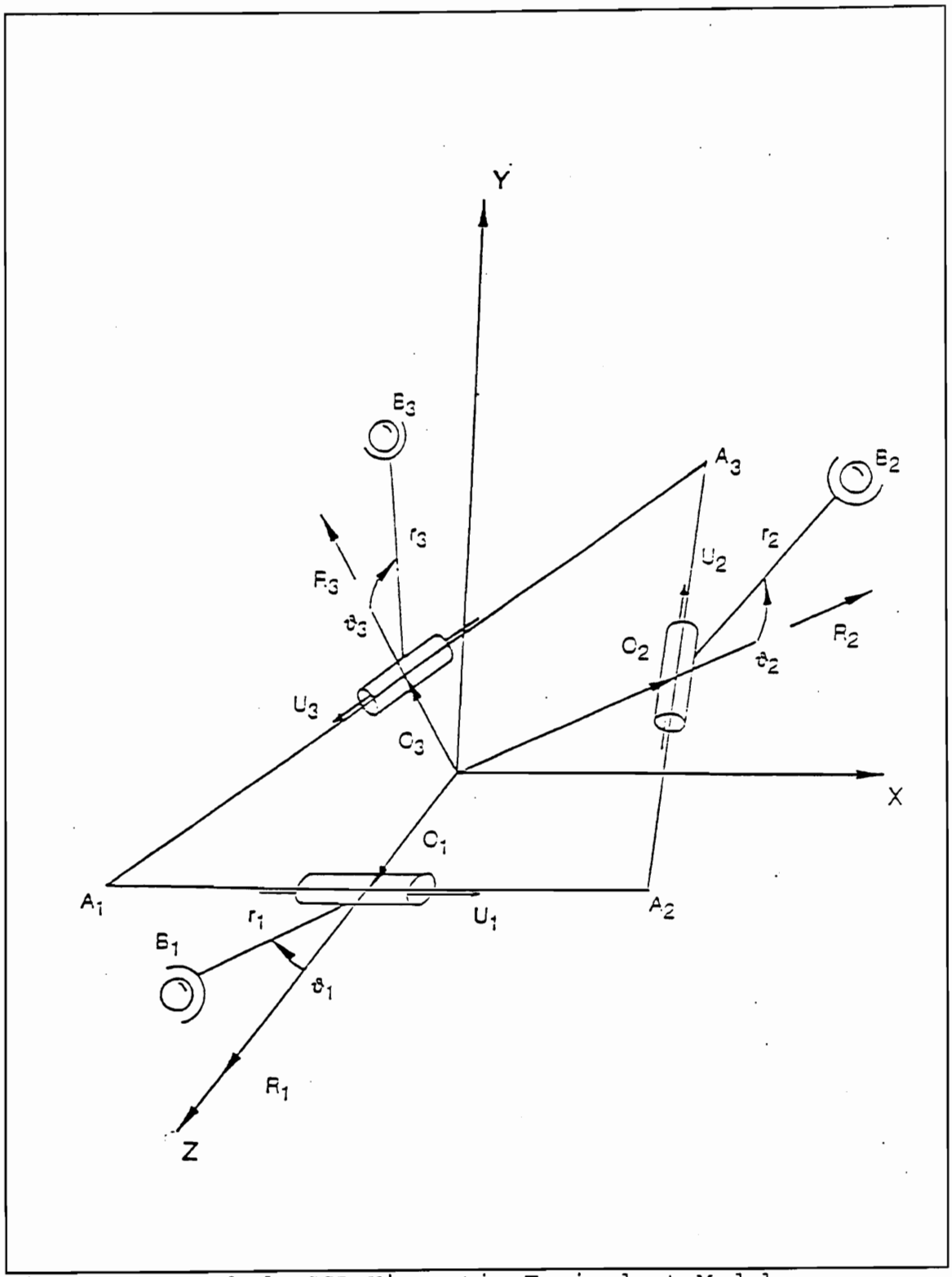


Figure 17 Coupled RSSR Kinematic Equivalent Model

nodes are separated by a known scalar distance, given by the actuator lengths, yields three scalar equation (Eqs. 5.2).

$$\begin{aligned} |\vec{B}_2 - \vec{B}_3| &= l_1 \\ |\vec{B}_3 - \vec{B}_1| &= l_2 \\ |\vec{B}_1 - \vec{B}_2| &= l_3 \end{aligned} \qquad (5.2)$$

An iterative scheme such as a Newton-Raphson method is used to obtain solutions to this non-linear system of three equations and three unknowns[3]. These three scalar equations yield a total of sixteen solutions, of which eight are mirror images. The choice of initial guesses is very important for obtaining solutions in the same closure, since Newton-Raphson's method can potentially lead to any one of the sixteen solutions (excluding special cases). The solution in the form of the coordinates of the offset-plane nodes defines the first octahedral unit cell completely.

5.2.2 Solution to the Double-Octahedral Manipulator

The preceding analysis can be carried out starting at the second offset plane (obtained by translating the first offset plane by the offset distance along its normal) to define the second octahedral unit cell. Another approach, valid for this particular geometry, exploits the symmetric nature of the truss to obtain an easier solution to the forward problem. Since the mid-plane, midway between the two offset planes, is the plane of symmetry, a transformation between the moving coordinate frame and the fixed coordinate frame can be written using this condition. The transformation

[3] Using a similar approach and after further manipulation Griffis and Duffy [1989], Nanua and Waldron [1989], Innocenti and Parenti-Castelli [1990] have obtained a single sixteenth degree equation to solve the geometry of the octahedron.

matrix can then be used to express the coordinates of a point attached to the moving frame with respect to the fixed frame of reference.

The transformation is obtained as follows. Fig. 18 shows a schematic of the double-octahedral manipulator. For clarity, only the transverse planes are shown. On solving the first octahedron, the normal to the offset plane is calculated from the knowledge of the location of the nodes. Equation 5.3 defines the normal by utilizing the vector cross product.

$$\hat{U}_1 = \frac{(\vec{B}_2 - \vec{B}_1) \; X \; (\vec{B}_3 - \vec{B}_1)}{|(\vec{B}_2 - \vec{B}_1) \; X \; (\vec{B}_3 - \vec{B}1)|} \qquad (5.3)$$

The distance along the normal to the offset plane from the centroid of the bottom triangle to the offset plane is given by Eq. 5.4.

$$\frac{n}{2} = \vec{B}_1 \cdot \hat{U}_1 \qquad (5.4)$$

The distance to the mid-plane is obtained by adding half the joint offset, since the mid-plane is mid-way between the two offset planes and parallel to both.

$$\frac{d}{2} = \frac{n}{2} + \frac{s}{2} \qquad (5.5)$$

where s - Joint Offset

The distance along the normal to the fixed plane from the centroid of the bottom triangle to the mid-plane is calculated in Eq. 5.6.

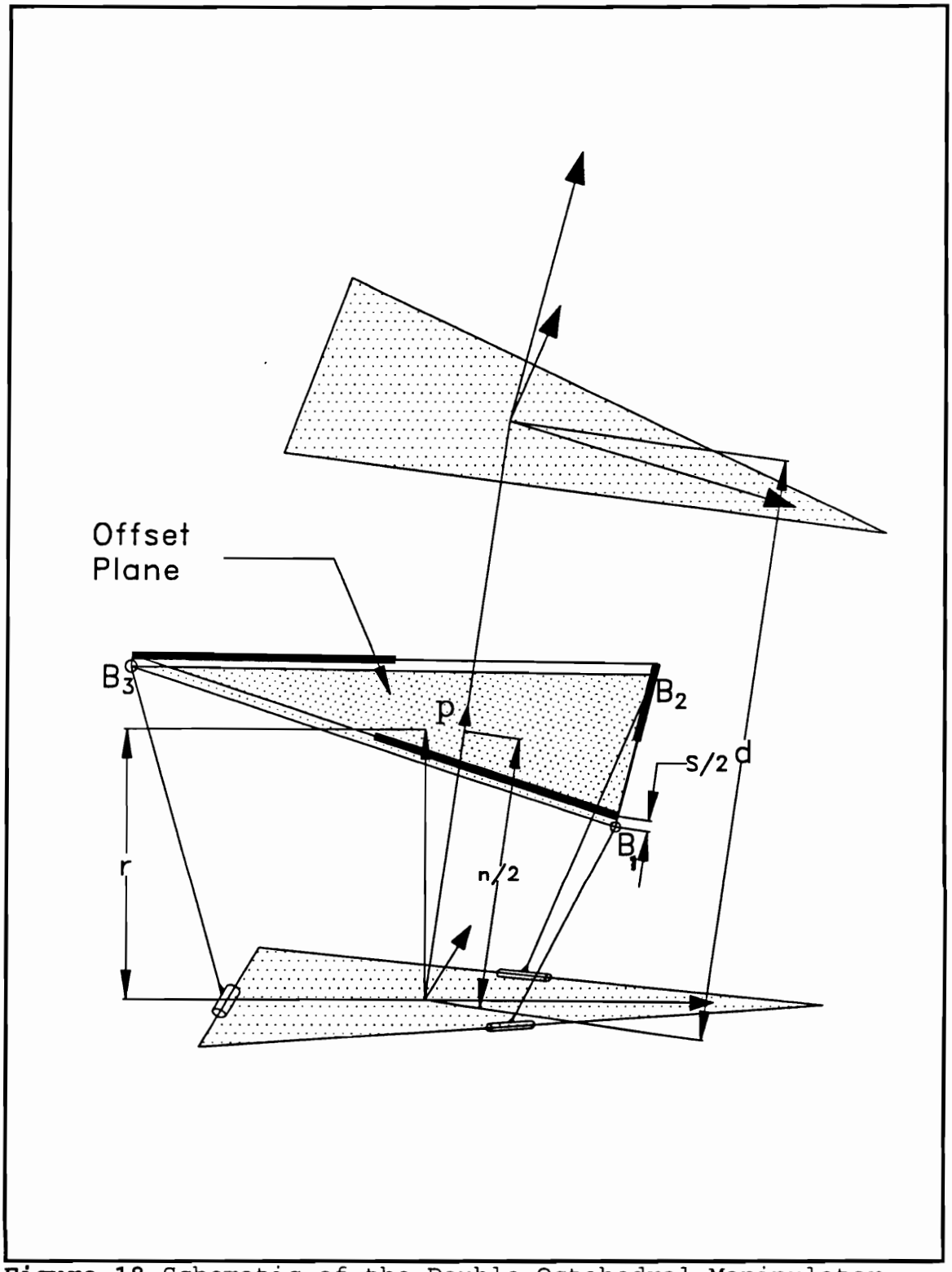


Figure 18 Schematic of the Double-Octahedral Manipulator

$$r = \frac{d}{2(\hat{U}_0 \cdot \hat{U}_1)} \tag{5.6}$$

Using the notation presented in Fig. 18, the following loop-closure equation can be written that provides the origin of the moving coordinate frame.

$$\vec{P} = r(\hat{U}_0 + \hat{U}_2) = d\hat{U}_1 \tag{5.7}$$

Eq. 5.7 is rearranged to solve for the normal to the moving plane in Eq. 5.8.

$$\hat{U}_2 = \frac{d}{r}\hat{U}_1 - \hat{U}_0 \tag{5.8}$$

Where

$\hat{U}_0$ - *Unit Normal to the Fixed Plane*
$\hat{U}_1$ - *Unit Normal to the Mid-plane*
$\hat{U}_2$ - *Unit Nœmal to the Bottom Plane*

Using the knowledge that the moving plane can be obtained as the reflection of the fixed plane with the mid-plane acting as the mirror, the transformation from the moving plane to the fixed plane can be written as follows.

$$[{}^{F}_{M}T] = \begin{bmatrix} 1-\frac{u_{2X}^2}{1+u_{2Z}} & \frac{-u_{2X}u_{2Y}}{1+u_{2Z}} & u_{2X} & X \\ \frac{-u_{2X}u_{2Y}}{1+u_{2Z}} & 1-\frac{u_{2Y}^2}{1+u_{2Z}} & u_{2Y} & Y \\ -u_{2X} & -u_{2Y} & u_{2Z} & Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.9}$$

Where

$\{U_{2X}\ U_{2Y}\ U_{2Z}\}^T$ - *Unit Normal to the Moving Plane*
$\{X\ Y\ Z\}^T$ - $\vec{P}$, *Origin of the Moving Frame*

This transformation can be thought of as a sequence of three rotations or as an equivalent axis rotation [Suh and Radcliffe, 1983]. The same transformation matrix is provided in a slightly different form by Miura et al. [1987].

5.3 Results

5.3.1 Positioning Specification

The output parameters to the forward kinematic analysis of the positioning problem are the coordinates of a point rigidly attached to the moving frame expressed in the fixed coordinate frame. The vector **V**, shown in Fig. 19, is the end-effector location relative to the moving coordinate frame. Using the transformation matrix in Eq. 5.9, the same point is expressed in the fixed coordinate frame in Eq. 5.10.

$$\vec{P}_e = [{}^F_M T] \ \vec{V} \qquad (5.10)$$

A program written using the C-language compiler, Turbo C, that solves the forward problem with the positioning specification using a Quasi-Newton method is shown in Appendices 3a and 3d. The program uses closed-form partial derivatives for iteratively finding the roots. The end-effector was assumed to be rigidly attached to the moving frame such that it is perpendicular to the moving plane and 10 units long. Using the sizes mentioned in section 4.4.2, a sample run of the program with actuator lengths l_1 = 42.0 units, l_2 = 45.0 units, and l_3 = 39.0 units, yielded the end-effector location X = -5.0513 units, Y = -0.0180 units, and Z = 56.0611 units. These results were verified by substituting them back as inputs to the inverse problem.

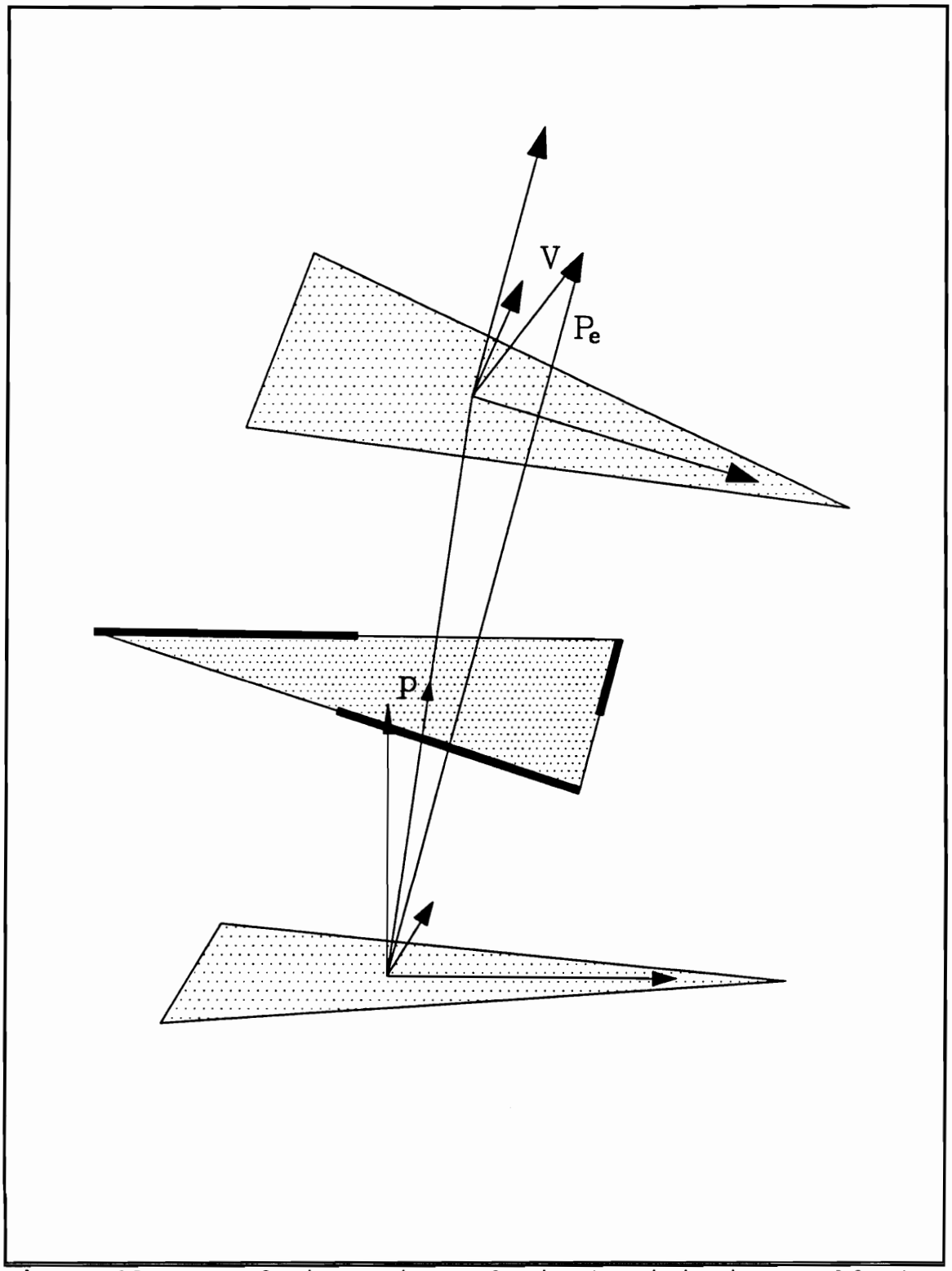


Figure 19 Forward Kinematic Analysis (Positioning Problem)

5.3.2 Gimbal Specification

The orientation of the normal to the top plane, forming part of the set of input parameters, is expressed by a fixed axis *Y-Z* rotation with respect to the bottom frame, assuming that the *Z* axes are defined to be normal to the fixed and moving planes. The *Y-Z* rotations are defined as follows: first, a rotation about the fixed *Y* axis by an amount β, followed by a rotation about the fixed *Z* axis by an amount θ. Eq. 5.11 and 5.12 give the angles β and θ in terms of the unit vectors along the moving *X, Y,* and *Z* axes expressed in the fixed reference frame. Fig. 20 is a schematic of the fixed, moving, and actuated triangles of the truss-based gimbal.

$$\beta = \tan^{-1}\left(\frac{\sqrt{U_{2X}^2 + U_{2Y}^2}}{U_{2Z}}\right) \qquad (5.11)$$

$$\theta = \tan^{-1}\left(\frac{U_{2Y}}{U_{2X}}\right) \qquad (5.12)$$

The gimbal can be designed such that fictitious spheric joint is located on the bottom plane or on the mid-plane. In other words, rotations of the moving plane can be made to occur at either of these locations. Based on this location of the fixed frame, either *d* or *r* will be an input parameter to the inverse problem along with β and θ. Although these distances can be made variable while the gimbal is in operation, they can also be held constant for a particular task, as is the case in a true two-DOF gimbal.

A program written using the C-language compiler, Turbo C, used to solve the

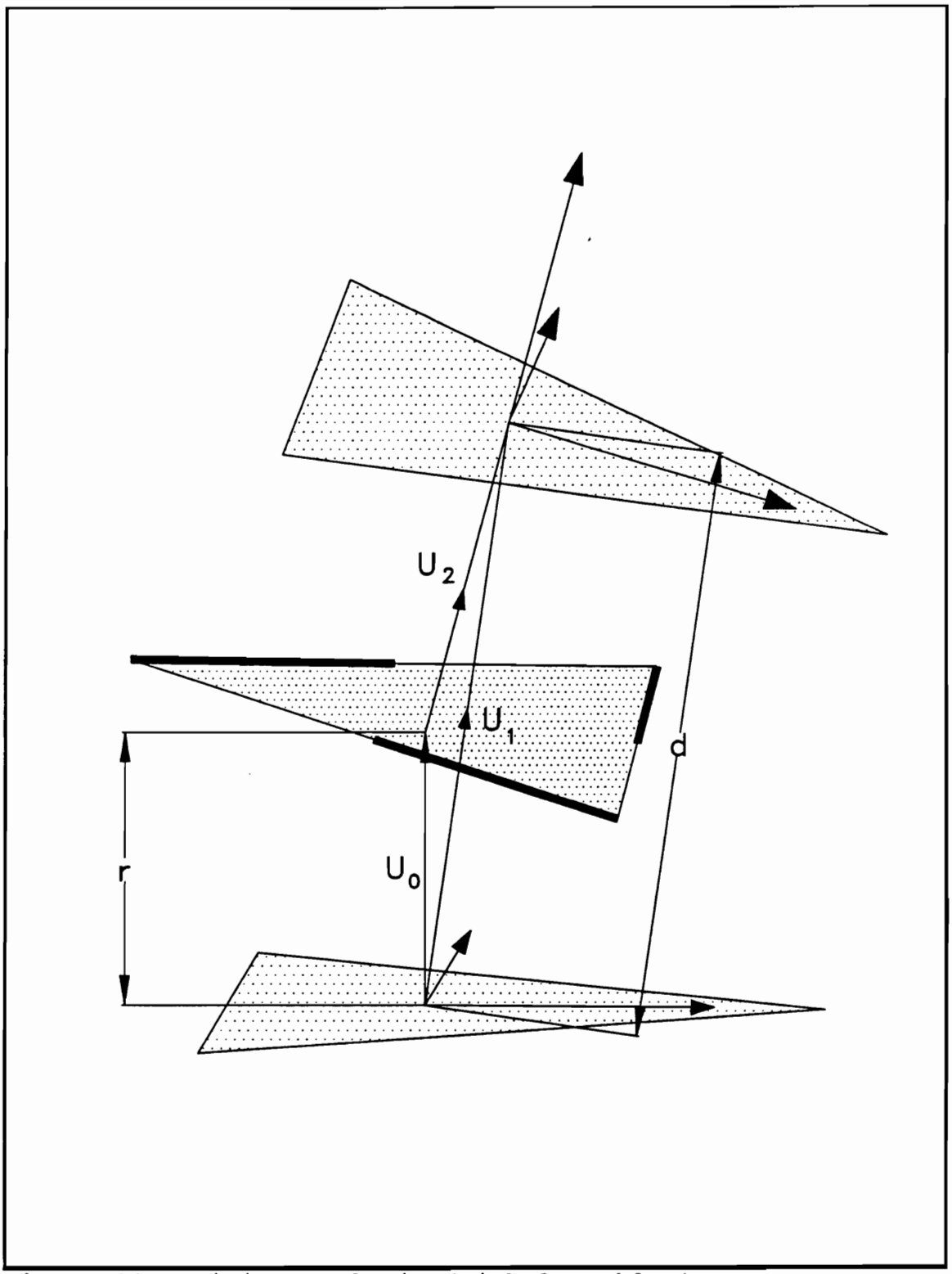


Figure 20 Position Analysis (Gimbal Problem)

forward kinematic analysis for the gimbal specification is given in Appendices 3b and 3d. For a sample run with actuator lengths l_1 = 42.0 units, l_2 = 45.0 units, and l_3 = 39.0 units, the gimbal parameters β = 8.7452 degrees, θ = -179.7960 degrees, and d = 46.3121 units were obtained. These results were again verified in a similar manner described earlier.

5.3.3 Docking Specification

Solving for the transformation matrix twice in succession for the two double-octahedral manipulators that are stacked together and taking the product of the two matrices yields the relationship between the moving and the fixed coordinate frame of the quadruple-octahedral manipulator. The C-language program used to solve the forward kinematic analysis for this case is given in Appendices 3c and 3d. The transformation matrix obtained with the actuator lengths l_1 = 46.784 units, l_2 = 41.3302 units, l_3 = 43.9983 units, l_4 = 40.3098 units, l_5 = 45.8085 units, and l_6 = 45.9199 units is given below.

$$[{}^{F}_{M}T\ ] = \begin{bmatrix} 0.9971 & 0.0083 & 0.0755 & 5.0003 \\ -0.0050 & 0.9990 & -0.0436 & 5.0002 \\ -0.0758 & 0.0431 & 0.9962 & 90.0000 \\ 0.0000 & 0.0000 & 0.0000 & 1.0000 \end{bmatrix}$$

These results were also verified by substituting back as inputs to the inverse problem.

6 Inverse Kinematic Analysis

6.1 Objective

The objective of the inverse analysis is to determine the set of output parameters (actuator lengths) that satisfy a given set of input parameters. All the other fixed quantities in the manipulator, such as the fixed batten and longeron length and the reference coordinate frame location, are assumed to be known.

6.2 Method

The method for solving the inverse problem also takes advantage of the symmetric nature of the double-octahedral manipulator under investigation. Closed-form solutions are obtained for the positioning, gimbal and docking inverse problems by exploiting this property. The solution procedure begins by first converting the input specifications to all three problems into a common form. It is assumed that from all three input specifications, the origin of the moving coordinate frame of the double-octahedral manipulator can be computed[1]. Once the origin of the moving frame is known, the equations of all the transverse planes can be easily written using symmetry. Since the mid-plane is the plane of symmetry, a line joining the origin of the reference frame to the origin of the moving frame is always perpendicular to the mid-plane. Also the mid-point of such a line is on the mid-plane. The last two

[1] The "results" sections show methods of computing the origin of the moving coordinate frame for the different problems.

statements completely define the parameters of the mid-plane. Using the above results, the equation of the mid-plane can be written directly for i = 1, 2, and 3

$$\left(\vec{b}_i - \frac{\vec{P}}{2} \right) \cdot \hat{U} = 0 \qquad (6.1)$$

Where **P** is the origin of the moving coordinate frame given by {*X, Y, Z*}, and $\mathbf{b}_i$ is the vector location of the mid-plane nodes. The unit normal to the mid-plane is shown in Eq. 6.2.

$$\hat{U} = \frac{\vec{P}}{|\vec{P}|} \qquad (6.2)$$

The spheric joints are, however, located on the offset plane. Therefore, the equation to the offset plane is needed. Since the offset plane is parallel to the mid-plane and translated by half the joint offset, the equation of the offset plane can be written as shown in Eq. 6.3.

$$\left(\vec{B}_i - \frac{\vec{P}}{2} + \frac{s\hat{U}}{2} \right) \cdot \hat{U} = 0 \qquad (6.3)$$

Equation 6.3 is expanded to yield Eq. 6.4.

$$B_{xi}X + B_{yi}Y + B_{zi}Z - \frac{|\vec{P}|^2}{2} + \frac{s|\vec{P}|}{2} = 0 \qquad (6.4)$$

The location of the spheric joints can be found by imposing the condition that the spheric joints lie on the offset plane. The solution is obtained by substituting the spheric joint location given in Eq. 5.1 in the equation of the offset plane given in Eq.

6.4. Equation 5.1 is expanded and separated into components of θ_i to yield Eq. 6.5. For i = 1, 2 and 3

$$\begin{aligned} B_{xi} &= k_{1i}COS(\theta_i) + k_{4i}SIN(\theta_i) + k_{7i} \\ B_{yi} &= k_{2i}COS(\theta_i) + k_{5i}SIN(\theta_i) + k_{8i} \\ B_{zi} &= k_{3i}COS(\theta_i) + k_{6i}SIN(\theta_i) + k_{9i} \end{aligned} \tag{6.5}$$

Where

$$\begin{aligned} k_{1i} &= R_{xi} - U_{xi}^2 R_{xi} - U_{xi}U_{yi}R_{yi} - U_{xi}U_{zi}R_{zi} \\ k_{2i} &= R_{yi} - U_{yi}^2 R_{yi} - U_{yi}U_{xi}R_{xi} - U_{yi}U_{zi}R_{zi} \\ k_{3i} &= R_{zi} - U_{zi}^2 R_{zi} - U_{zi}U_{xi}R_{xi} - U_{zi}U_{yi}R_{yi} \\ k_{4i} &= U_{yi}R_{zi} - U_{zi}R_{yi} \\ k_{5i} &= U_{zi}R_{xi} - U_{xi}R_{zi} \\ k_{6i} &= U_{xi}R_{yi} - U_{yi}R_{xi} \\ k_{7i} &= U_{xi}^2 R_{xi} + U_{xi}U_{yi}R_{yi} + U_{xi}U_{zi}R_{zi} + O_{xi} \\ k_{8i} &= U_{yi}^2 R_{yi} + U_{yi}U_{xi}R_{xi} + U_{yi}U_{zi}R_{zi} + O_{yi} \\ k_{9i} &= U_{zi}^2 R_{zi} + U_{zi}U_{xi}R_{xi} + U_{zi}U_{yi}R_{yi} + O_{zi} \end{aligned}$$

Substituting the coordinates of the nodes defined in Eq. 6.5 in the equation of the plane, defined in Eq. 6.4, an equation for θ_i can be written and solved.

$$\begin{aligned} &a_{1i}\cos\theta_i + a_{2i}\sin\theta_i + a_{3i} = 0 \\ &\textit{where} \\ &a_{1i} = k_{1i}X + k_{2i}Y + k_{3i}Z \\ &a_{2i} = k_{4i}X + k_{5i}Y + k_{6i}Z \\ &a_{3i} = k_{7i}X + k_{8i}Y + k_{9i}Z - \frac{d^2}{2} + \frac{sd}{2} \end{aligned} \tag{6.6}$$

The rotation angle of the RS link, θ_i, can now be obtained by using Cosine and Sine substitutions. This result is provided in Eq. 6.7.

$$Cos\theta_i = \frac{-a_{1i}a_{3i} \pm a_{2i}\sqrt{a_{1i}^2 + a_{2i}^2 - a_{3i}^2}}{a_{1i}^2 + a_{2i}^2}$$
$$Sin\theta_i = \frac{-a_{2i}a_{3i} \mp a_{1i}\sqrt{a_{1i}^2 + a_{2i}^2 - a_{3i}^2}}{a_{1i}^2 + a_{2i}^2} \qquad (6.7)$$

A solution will exist only when all three discriminants are greater than zero. This can be expressed by the following inequality.

$$a_{1i}^2 + a_{2i}^2 - a_{3i}^2 \geq 0. \qquad (6.8)$$

If the inequality is satisfied, then there can be a total of eight solutions[2]. For a particular closure, a unique combination of the sines and cosines exist.

The actuator lengths can be calculated from the knowledge of the location of the mid-plane nodes (spheric joint locations) in the following manner.

$$\begin{aligned} l_1 &= |B_2 - B_3| \\ l_2 &= |B_3 - B_1| \\ l_3 &= |B_1 - B_2| \end{aligned} \qquad (6.9)$$

The actuator lengths form the set of output parameters and hence complete the solution to the inverse problem.

[2] When one or more of the discriminants are exactly zero, there are fewer solutions than eight.

6.3 Results

6.3.1 Positioning Problem

In the case of the positioning problem, the input parameter consists of the location of a point (the end effector location) rigidly attached to the moving plane. The origin of the moving coordinate frame is obtained by considering the reflection of this point about the mid-plane. Since the mid-plane is the plane of symmetry, the vector V_r, shown in Fig. 21, is stationary relative to the fixed frame of reference. Therefore, the vector V_r can be calculated one time and used subsequently. This calculation is the easiest when the planes are parallel. Except for the Z coordinate, which takes a negative sign, the other coordinates of the vector remain identical. The origin of the moving coordinate frame is given in Eq. 6.10.

$$\vec{P} = \frac{(\vec{P}_e + \vec{V}_r)\bullet(\vec{P}_e - \vec{V}_r)}{(\vec{P}_e - \vec{V}_r)\bullet(\vec{P}_e - \vec{V}_r)}(\vec{P}_e - \vec{V}_r) \qquad (6.10)$$

Where $\vec{P}$ - *Vector Location of the Origin of the Moving Frame*

$\vec{P}_e$ - *Vector Location of the End-effector*

$\vec{V}_r$ - *Reflection of* $\vec{V}$ *about the Mid-plane*

The Turbo C code that solves the positioning problem is shown in Appendices 4a and 4d. The end-effector was assumed to be rigidly attached to the moving frame such that it is perpendicular to the moving plane and 10 units long. Using the sizes mentioned in section 4.4.2, a sample run of the program with end-effector location X = -5.0513 units, Y = -0.0180 units, and Z = 56.0611 units yielded the actuator lengths l_1 = 42.0 units, l_2 = 45.0 units, and l_3 = 39.0 units. These results can be seen to match with the results provided in section 5.3.1 of the forward kinematic analysis.

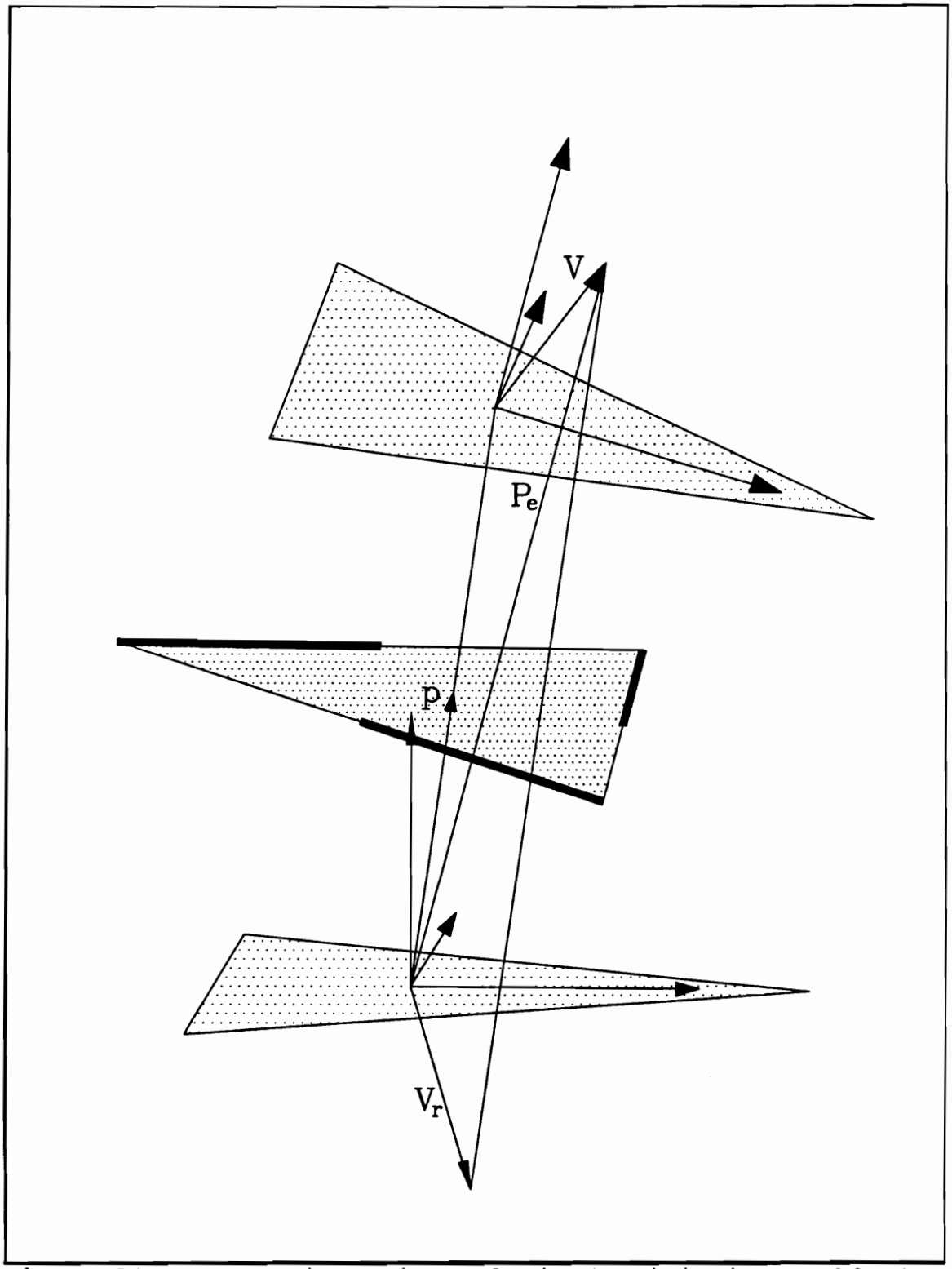


Figure 21 Inverse Kinematic Analysis (Positioning Problem)

6.3.2 Gimbal Problem

In the case of the gimbal problem, the input parameters includes the orientation of the moving coordinate frame and the distance d or r (Fig. 20). Since the orientation of the gimbal is the same as the normal to the moving plane, using the rotation convention described in section 5.3.2, the normal to the moving plane can be calculated, as in Eq. 6.11.

$$\hat{U}_2 = \begin{Bmatrix} Cos\theta\ Sin\beta \\ Sin\theta\ Sin\beta \\ Cos\beta \end{Bmatrix} \qquad (6.11)$$

If the dimension d is specified, then the point of rotation of the gimbal is taken to be located on the fixed frame. For this case, the origin of the moving coordinate frame is given in Eq. 6.12.

$$\vec{P} = \frac{d}{\sqrt{2\,(1 + \hat{U}_0 \bullet \hat{U}_2)}}\,(\hat{U}_0 + \hat{U}_2) \qquad (6.12)$$

Specifying the dimension r results in the point of rotation of the gimbal being on the mid-plane.

$$\vec{P} = r\,(\hat{U}_0 + \hat{U}_2) \qquad (6.13)$$

At this stage, the method presented in section 6.2 is followed to obtain the actuator lengths.

The Turbo C code used to solve the gimbal problem is given in Appendices 4b and 4d. For a sample run with the gimbal parameters β = 8.7452 degrees, θ = -179.7960 degrees, and d = 46.3121 units, the actuator lengths l_1 = 42.0 units, l_2 = 45.0

units, and l_3 = 39.0 units were obtained. These results match with the results in section 5.3.2 of the forward kinematic analysis.

6.3.3 Docking Problem

Figure 22 shows a schematic diagram of the quadruple-octahedral manipulator. The solution to the docking problem is obtained by breaking it down into solutions of two double-octahedral manipulators. The set of input parameters specified in this problem are the position vector **P** and orientation vector $\mathbf{U}_4$. The origin of the moving frame for the two double-octahedral parts are obtained as follows. The following loop closure equation can be written using the notation in Fig. 22.

$$r_1\hat{U}_0 + r_1\hat{U}_2 + r_2\hat{U}_2 + r_2\hat{U}_4 = \vec{P}_e \qquad (6.14)$$

Eq. 6.15 defines $\mathbf{U}_2$ as a unit vector.

$$\hat{U}_2 \bullet \hat{U}_2 = 1 \qquad (6.15)$$

Using the notation that the ratio r_1/r_2 is equal to c, Eq. 6.16 is a quadratic equation in the unknown r_2.

$$k_1 r_2^2 + k_2 r_2 + k_3 r_2 = 0 \qquad (6.16)$$

where the k_i's are given as

$$k_1 = 2c(\hat{U}_0 \bullet \hat{U}_4 - 1)$$
$$k_2 = -2(c\hat{U}_0 + \hat{U}_4) \bullet \vec{P}$$
$$k_3 = \vec{P} \bullet \vec{P}$$

The unit normal $\mathbf{U}_2$ can be calculated as shown in Eq. 6.17

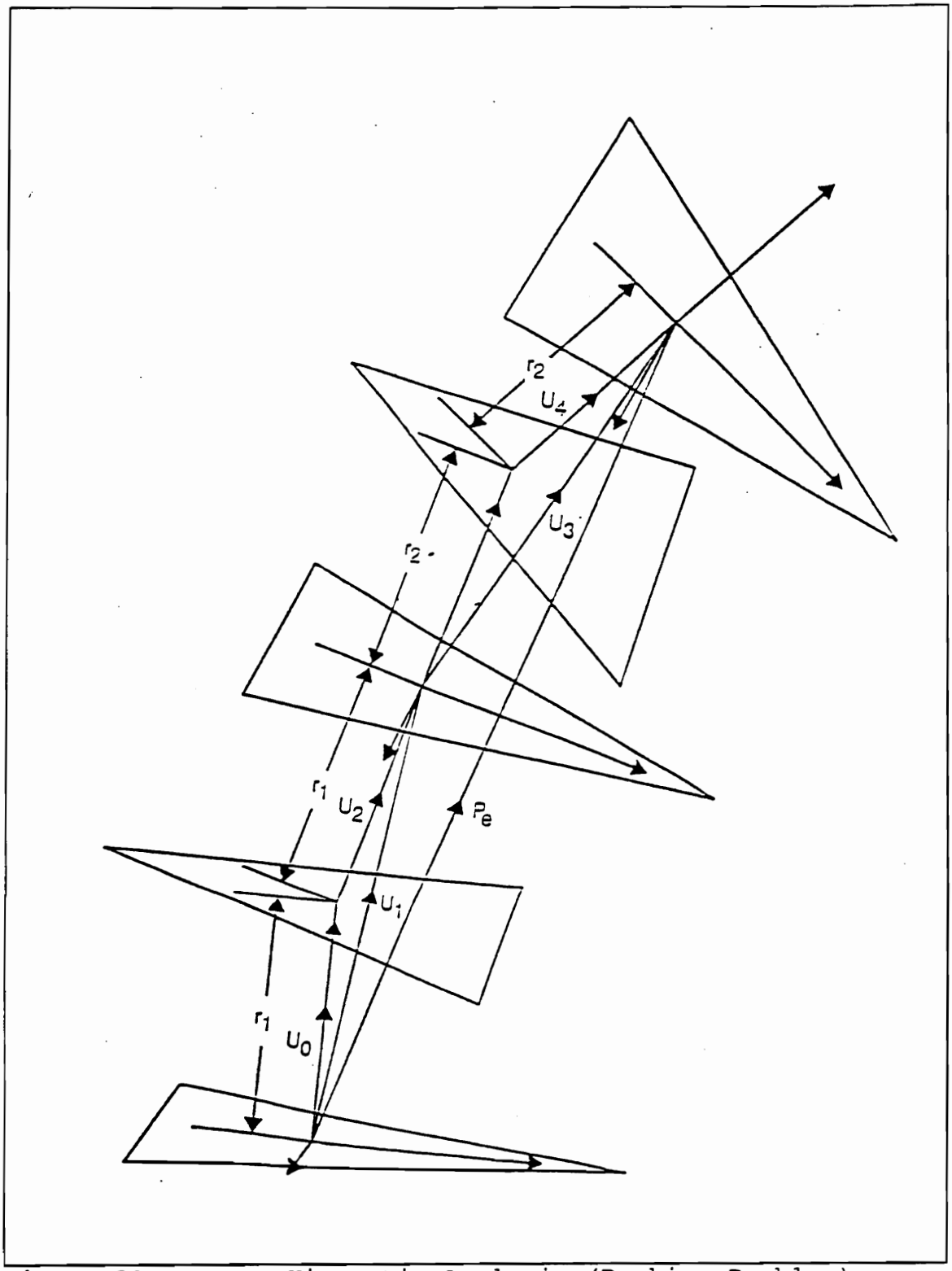


Figure 22 Inverse Kinematic Analysis (Docking Problem)

$$U_2 = \frac{\vec{P}}{r_2(c+1)} - \frac{c\hat{U}_0 + \hat{U}_4}{c+1} \qquad (6.17)$$

This enables the problem to be broken into the gimbal problem of the double-octahedral manipulator. The first double-octahedral manipulator is solved by using the gimbal specification r_1 and $\mathbf{U_2}$. To solve the second part, the rotational part of the transformation matrix, shown in Eq. 5.9, is constructed using the orientation vector $\mathbf{U_2}$. The unit vector $\mathbf{U_4}$ is transformed by multiplying with the transpose of this transformation matrix. By using the transformed unit vector $\mathbf{U_4}$ and r_2, the second double-octahedral manipulator is solved.

The Turbo C code used to solve the docking problem is shown in Appendices 4c and 4d. A sample run with the input specification X = 5.0 units, Y = 5.0 units, Z = 90.0 units, β = 5 degrees, θ = -30 degrees and c = 1.0 is satisfied by the actuator lengths l_1 = 46.784 units, l_2 = 41.3302 units, l_3 = 43.9983 units, l_4 = 40.3098 units, l_5 = 45.8085 units, and l_6 = 45.9199 units. These results can be seen to agree with the results shown in section 5.3.3 on the forward kinematic analysis of the quadruple-octahedral manipulator with docking specification.

7 Velocity and Acceleration Analysis

7.1 Objective

The objective of the velocity and acceleration analysis is to relate the velocities and accelerations of the moving coordinate frame to the velocities and accelerations of the actuators. This relationship, expressed in terms of matrices, takes the form of the Jacobian and Hessian matrix. It is also sometimes necessary to find the velocities of all link members in the manipulator. The Jacobian matrix also provides the relationship between the input and output forces.

7.2 Method

7.2.1 Velocity Analysis

The closed-form inverse solutions to the positioning problem and the gimbal problem provide a relationship between the position and orientation of the moving coordinate frame and the actuator lengths. This is called the position relationship. This relationship is highly non-linear in terms of the position parameters. The relationship between the velocities of the moving frame and the actuators are, however, always linear. The relationship between the velocities can be obtained by taking time derivatives of the position relationship. Since, the velocity relationship is always linear, it can be expressed in a matrix form. This matrix is called the Jacobian matrix. The Jacobian matrix for the double-octahedral manipulator can be obtained by starting at Eq. 6.9 and moving through the inverse solution taking

derivatives with respect to time at each step. Eq. 6.9 is expressed in a slightly different form in Eq. 7.1.

$$\begin{aligned} l_1^2 &= \vec{B}_{23} \bullet \vec{B}_{23} \\ l_2^2 &= \vec{B}_{13} \bullet \vec{B}_{13} \\ l_3^2 &= \vec{B}_{12} \bullet \vec{B}_{12} \end{aligned} \tag{7.1}$$

using the convention that

$$\vec{B}_{ij} = \vec{B}_i - \vec{B}_j$$

Equation 7.2 can be written by differentiating Eq. 7.1 with respect to time.

$$\begin{aligned} 2l_1\dot{l}_1 &= 2\vec{B}_{23} \bullet \dot{B}_{23} \\ 2l_2\dot{l}_2 &= 2\vec{B}_{13} \bullet \dot{B}_{13} \\ 2l_3\dot{l}_3 &= 2\vec{B}_{12} \bullet \dot{B}_{12} \end{aligned} \tag{7.2}$$

In general, taking the partial derivative of $\mathbf{B}_{ij}$ with respect to any three input parameters A, B, and Γ and using the chain rule to find time derivatives gives

$$\dot{B}_{ij} = \frac{\partial \vec{B}_{ij}}{\partial \mathbf{A}} \frac{d\mathbf{A}}{dt} + \frac{\partial \vec{B}_{ij}}{\partial \mathbf{B}} \frac{d\mathbf{B}}{dt} + \frac{\partial \vec{B}_{ij}}{\partial \Gamma} \frac{d\Gamma}{dt} \tag{7.3}$$

The above result can be expressed in the matrix form shown in Eq. 7.4.

$$\begin{bmatrix} l_1 & 0 & 0 \\ 0 & l_2 & 0 \\ 0 & 0 & l_3 \end{bmatrix} \begin{Bmatrix} \dot{l}_1 \\ \dot{l}_2 \\ \dot{l}_3 \end{Bmatrix} - \begin{bmatrix} \vec{B}_{23} \cdot \frac{\partial \vec{B}_{23}}{\partial \mathbf{A}} & \vec{B}_{23} \cdot \frac{\partial \vec{B}_{23}}{\partial \mathbf{B}} & \vec{B}_{23} \cdot \frac{\partial \vec{B}_{23}}{\partial \Gamma} \\ \vec{B}_{31} \cdot \frac{\partial \vec{B}_{31}}{\partial \mathbf{A}} & \vec{B}_{31} \cdot \frac{\partial \vec{B}_{31}}{\partial \mathbf{B}} & \vec{B}_{31} \cdot \frac{\partial \vec{B}_{31}}{\partial \Gamma} \\ \vec{B}_{12} \cdot \frac{\partial \vec{B}_{12}}{\partial \mathbf{A}} & \vec{B}_{12} \cdot \frac{\partial \vec{B}_{12}}{\partial \mathbf{B}} & \vec{B}_{12} \cdot \frac{\partial \vec{B}_{12}}{\partial \Gamma} \end{bmatrix} \begin{Bmatrix} \dot{\mathbf{A}} \\ \dot{\mathbf{B}} \\ \dot{\Gamma} \end{Bmatrix} = 0 \tag{7.4}$$

Referring to Eq. 6.5, the partial derivative of $\mathbf{B}_{ij}$ with respect to A is found to be

$$\frac{\partial \vec{B}_{ij}}{\partial A} = \begin{Bmatrix} k_{1i}\frac{\partial Cos\theta_i}{\partial A} + k_{4i}\frac{\partial Sin\theta_i}{\partial A} - k_{1j}\frac{\partial Cos\theta_j}{\partial A} - k_{4j}\frac{\partial Sin\theta_j}{\partial A} \\ k_{2i}\frac{\partial Cos\theta_i}{\partial A} + k_{5i}\frac{\partial Sin\theta_i}{\partial A} - k_{2j}\frac{\partial Cos\theta_j}{\partial A} - k_{5j}\frac{\partial Sin\theta_j}{\partial A} \\ k_{3i}\frac{\partial Cos\theta_i}{\partial A} + k_{6i}\frac{\partial Sin\theta_i}{\partial A} - k_{3j}\frac{\partial Cos\theta_j}{\partial A} - k_{6j}\frac{\partial Sin\theta_j}{\partial A} \end{Bmatrix} \quad (7.5)$$

Similarly equations can be developed for the partial derivatives of $\mathbf{B}_{ij}$ with respect to B and Γ. The partial derivatives of Cos θ_i and Sin θ_i in terms of the truss parameters from Eq. 6.7, are given below.

$$\frac{\partial Cos\theta_i}{\partial A} = \frac{1}{a_{1i}^2 + a_{2i}^2}\left[- a_{1i}\frac{\partial a_{3i}}{\partial A} - a_{3i}\frac{\partial a_{1i}}{\partial A} + \sqrt{a_{1i}^2 + a_{2i}^2 - a_{3i}^2}\frac{\partial a_{2i}}{\partial A} + \frac{a_{2i}\left(a_{1i}\frac{\partial a_{1i}}{\partial A} + a_{2i}\frac{\partial a_{2i}}{\partial A} - a_{3i}\frac{\partial a_{3i}}{\partial A}\right)}{\sqrt{a_{1i}^2 + a_{2i}^2 - a_{3i}^2}}\right] - \frac{2\left(-a_{1i}\,a_{3i} + a_{2i}\sqrt{a_{1i}^2 + a_{2i}^2 - a_{3i}^2}\right)\left(a_{1i}\frac{\partial a_{1i}}{\partial A} + a_{2i}\frac{\partial a_{2i}}{\partial A}\right)}{(a_{1i}^2 + a_{2i}^2)^2} \quad (7.6)$$

$$\frac{\partial Sin\theta_i}{\partial A} = \frac{1}{a_{1i}^2 + a_{2i}^2}\left[- a_{2i}\frac{\partial a_{3i}}{\partial A} - a_{3i}\frac{\partial a_{2i}}{\partial A} + \sqrt{a_{1i}^2 + a_{2i}^2 - a_{3i}^2}\frac{\partial a_{1i}}{\partial A} + \frac{a_{1i}\left(a_{1i}\frac{\partial a_{1i}}{\partial A} + a_{2i}\frac{\partial a_{2i}}{\partial A} - a_{3i}\frac{\partial a_{3i}}{\partial A}\right)}{\sqrt{a_{1i}^2 + a_{2i}^2 - a_{3i}^2}}\right] - \frac{2\left(-a_{2i}\,a_{3i} + a_{1i}\sqrt{a_{1i}^2 + a_{2i}^2 - a_{3i}^2}\right)\left(a_{1i}\frac{\partial a_{1i}}{\partial A} + a_{2i}\frac{\partial a_{2i}}{\partial A}\right)}{(a_{1i}^2 + a_{2i}^2)^2} \quad (7.7)$$

Where

$$\begin{aligned} \frac{\partial a_{1i}}{\partial A} &= k_{1i}\frac{\partial X}{\partial A} + k_{2i}\frac{\partial Y}{\partial A} + k_{3i}\frac{\partial Z}{\partial A} \\ \frac{\partial a_{2i}}{\partial A} &= k_{4i}\frac{\partial X}{\partial A} + k_{5i}\frac{\partial Y}{\partial A} + k_{6i}\frac{\partial Z}{\partial A} \\ \frac{\partial a_{3i}}{\partial A} &= k_{7i}\frac{\partial X}{\partial A} + k_{8i}\frac{\partial Y}{\partial A} + k_{9i}\frac{\partial Z}{\partial A} - \left(X\frac{\partial X}{\partial A} + Y\frac{\partial Y}{\partial A} + Z\frac{\partial Z}{\partial A}\right) \end{aligned} \quad (7.8)$$

This completes the preliminary step in the determination of the Jacobian matrix. The determination of the Jacobian matrices for positioning problem and the gimbal problem are shown below.

Positioning Problem

Eq. 6.10 of the inverse positioning problem is rewritten here, for convenience.

$$\vec{P} = \frac{(\vec{P}_e + \vec{V}_r)\bullet(\vec{P}_e - \vec{V}_r)}{(\vec{P}_e - \vec{V}_r)\bullet(\vec{P}_e - \vec{V}_r)}(\vec{P}_e - \vec{V}_r) \qquad (7.9)$$

where **P** is the origin of the moving coordinate frame given by {*X, Y, Z*} and $\mathbf{P}_e$ is the end-effector location given by $\{X_e, Y_e, Z_e\}$. Taking $A = X_e$, $B = Y_e$ and $\Gamma = Z_e$, and taking the derivative of **P**, the following equation can be written.

$$\frac{\partial \vec{P}}{\partial X_e} = \frac{\left[(\vec{P}_e + \vec{V}_r)\bullet(\vec{P}_e - \vec{V}_r)\right]\frac{\partial \vec{P}_e}{\partial X_e} + 2\left[\vec{P}e\bullet\frac{\partial \vec{P}e}{\partial X_e}\right](\vec{P}_e - \vec{V}_r)}{(\vec{P}_e - \vec{V}_r)\bullet(\vec{P}_e - \vec{V}_r)} - \frac{2\left[(\vec{P}_e + \vec{V}_r)\bullet(\vec{P}_e - \vec{V}_r)\right]\left[(\vec{P}_e - \vec{V}_r)\bullet\frac{\partial \vec{P}_e}{\partial X_e}\right](\vec{P}_e - \vec{V}_r)}{\left[(\vec{P}_e - \vec{V}_r)\bullet(\vec{P}_e - \vec{V}_r)\right]^2} \qquad (7.10)$$

where

$$\frac{\partial \vec{P}}{\partial X_e} = \begin{Bmatrix} \frac{\partial X}{\partial X_e} \\ \frac{\partial Y}{\partial Y_e} \\ \frac{\partial Z}{\partial Z_e} \end{Bmatrix}$$

and

$$\frac{\partial \vec{P}_e}{\partial X_e} = \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} \qquad (7.11)$$

Similar results can be developed for Y_e and Z_e. Substituting the results in Eq. 7.10 back in Eq. 7.8 completes the derivation of the Jacobian matrix for the positioning problem. The Turbo C code written for the determination of the Jacobian matrix is shown in Appendix 5a. Using this program, for the input specification to the positioning problem shown in section 6.3.1, the following Jacobian matrix was obtained.

$$\begin{bmatrix} 42.0 & 0.0 & 0.0 \\ 0.0 & 45.0 & 0.0 \\ 0.0 & 0.0 & 39.0 \end{bmatrix} \begin{Bmatrix} \dot{l}_1 \\ \dot{l}_2 \\ \dot{l}_3 \end{Bmatrix} - \begin{bmatrix} 13.6968 & 28.9303 & -78.0324 \\ -13.7591 & -15.1333 & -73.3195 \\ 37.4476 & -13.4153 & -78.9947 \end{bmatrix} \begin{Bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{Bmatrix} = 0$$

The values of this jacobian matrix were verified with a finite difference approximation scheme for finding the Jacobian matrices.

Gimbal Problem

The velocity analysis of the double-octahedral manipulator for the gimbal specification of dimension r is presented here. Eq. 6.13 in the inverse solution is expanded and written in the following form.

$$\begin{aligned} X &= r(U_{X0} + Cos\theta\ Sin\beta) \\ Y &= r(U_{Y0} + Sin\theta\ Sin\beta) \\ Z &= r(U_{Z0} + Cos\beta) \end{aligned} \qquad (7.12)$$

Using A = θ, B = β, and Γ = r, and taking partial derivatives of {X, Y, Z}, the following equations can be written.

$$\frac{\partial X}{\partial \theta} = -r\ Sin\theta\ Sin\beta$$
$$\frac{\partial Y}{\partial \theta} = r\ Cos\theta\ Sin\beta \qquad (7.13)$$
$$\frac{\partial Z}{\partial \theta} = 0$$

$$\frac{\partial X}{\partial \beta} = r\ Cos\beta\ Cos\theta$$
$$\frac{\partial Y}{\partial \beta} = r\ Cos\beta\ Sin\theta \qquad (7.14)$$
$$\frac{\partial Z}{\partial \beta} = -r\ Sin\beta$$

and

$$\frac{\partial X}{\partial r} = U_{X0} + Cos\theta\ Sin\beta$$
$$\frac{\partial Y}{\partial r} = U_{Y0} + Sin\theta\ Sin\beta \qquad (7.15)$$
$$\frac{\partial Z}{\partial r} = U_{Z0} + Cos\beta$$

Substituting these results in Eq. 7.8 completes the determination of the Jacobian matrix for the gimbal problem. The Turbo C code written for this process is shown in Appendix 5b. For the solution to the gimbal problem shown in section 6.3.2, the following Jacobian matrix was obtained by running this program.

This solution was also verified by checking the values with a finite difference approximation to the Jacobian matrix.

$$\begin{bmatrix} 42.0 & 0.0 & 0.0 \\ 0.0 & 45.0 & 0.0 \\ 0.0 & 0.0 & 39.0 \end{bmatrix} \begin{Bmatrix} \dot{l}_1 \\ \dot{l}_2 \\ \dot{l}_3 \end{Bmatrix} - \begin{bmatrix} -145.8885 & -58.9726 & -157.2533 \\ 76.1952 & 823.9361 & -143.6847 \\ 68.4388 & -829.0701 & -162.7548 \end{bmatrix} \begin{Bmatrix} \dot{\theta} \\ \dot{\beta} \\ \dot{r} \end{Bmatrix} = 0$$

7.2.2 Acceleration Analysis

Acceleration analysis easily follows from the velocity analysis. The partial derivative of the offset plane node constraint equations, given in Eq. 7.2 is rewritten in a general form as follows:

$$l_k \dot{l}_k = \vec{B}_{ij} \bullet \dot{B}_{ij} \tag{7.16}$$

Differentiating with respect to time, Eq. 7.17 can be written.

$$\dot{l}^{2_k} + l_k \ddot{l}_k = B_{ij} \bullet \ddot{B}_{ij} + \dot{B}_{ij} \bullet \dot{B}_{ij} \tag{7.17}$$

Again, by using the chain rule

$$\begin{aligned}
\ddot{B}_{ij} &= \frac{d}{dt}(\dot{B}_{ij}) \\
&= \frac{d}{dt}\left(\frac{\partial \vec{B}_{ij}}{\partial \mathbf{A}}\frac{d\mathbf{A}}{dt} + \frac{\partial \vec{B}_{ij}}{\partial \mathbf{B}}\frac{d\mathbf{B}}{dt} + \frac{\partial \vec{B}_{ij}}{\partial \Gamma}\frac{d\Gamma}{dt}\right) \\
&= \frac{d}{dt}\left(\frac{\partial \vec{B}_{ij}}{\partial \mathbf{A}}\right)\frac{d\mathbf{A}}{dt} + \frac{d}{dt}\left(\frac{\partial \vec{B}_{ij}}{\partial \mathbf{B}}\right)\frac{d\mathbf{B}}{dt} + \frac{d}{dt}\left(\frac{\partial \vec{B}_{ij}}{\partial \Gamma}\right)\frac{d\Gamma}{dt} \\
&+ \frac{\partial \vec{B}_{ij}}{\partial \mathbf{A}}\frac{d^2\mathbf{A}}{dt^2} + \frac{\partial \vec{B}_{ij}}{\partial \mathbf{B}}\frac{d^2\mathbf{B}}{dt^2} + \frac{\partial \vec{B}_{ij}}{\partial \Gamma}\frac{d^2\Gamma}{dt^2}
\end{aligned} \tag{7.18}$$

Expressed in matrix form, Eq. 7.18 becomes

$$\ddot{\vec{B}}_{ij} = \{\dot{A}\ \dot{B}\ \dot{\Gamma}\} \begin{bmatrix} \frac{\partial^2 \vec{B}_{ij}}{\partial A^2} & \frac{\partial^2 \vec{B}_{ij}}{\partial B \partial A} & \frac{\partial^2 \vec{B}_{ij}}{\partial \Gamma \partial A} \\ \frac{\partial^2 \vec{B}_{ij}}{\partial A \partial B} & \frac{\partial^2 \vec{B}_{ij}}{\partial B^2} & \frac{\partial^2 \vec{B}_{ij}}{\partial \Gamma \partial B} \\ \frac{\partial^2 \vec{B}_{ij}}{\partial A \partial \Gamma} & \frac{\partial^2 \vec{B}_{ij}}{\partial B \partial \Gamma} & \frac{\partial^2 \vec{B}_{ij}}{\partial \Gamma^2} \end{bmatrix} \begin{Bmatrix} \dot{A} \\ \dot{B} \\ \dot{\Gamma} \end{Bmatrix} + \begin{Bmatrix} \frac{\partial \vec{B}_{ij}}{\partial A} & \frac{\partial \vec{B}_{ij}}{\partial B} & \frac{\partial \vec{B}_{ij}}{\partial \Gamma} \end{Bmatrix} \begin{Bmatrix} \ddot{A} \\ \ddot{B} \\ \ddot{\Gamma} \end{Bmatrix} \qquad (7.19)$$

When Eq. 7.19 is written in its expanded form (i.e for all three i and j combinations), the matrix of partial derivatives becomes a three dimensional Hessian matrix. The Jacobian matrix is also part of this acceleration equation (the two dimensional form of the vector of partial derivatives in Eq. 7.19). The derivation of the Hessian matrix can be completed in a manner similar to the Jacobian matrix, but, it is not extended beyond Eq. 7.19 here. Upon substituting the appropriate Jacobian and Hessian matrix, the accelerations can be obtained[1].

[1] A clever method to obtain the relationship between the nodal velocities and accelerations and the link (including the actuators) velocities and accelerations, without relying on the inverse solution, is presented by Tidwell et al. [1990].

8 Singularity Analysis

8.1 Objective

Positions of a manipulator where the Jacobian matrix becomes singular are called singular positions. Since singularities lead to a loss or a gain of one or more degrees of freedom in a manipulator, the determination of these positions is important. Therefore, the objective of the singularity analysis is to identify the singular positions of a manipulator.

8.2 Literature Review

Singularity analysis has been performed earlier on several parallel manipulators. Hunt (1983) used screw theory to perform such an analysis and pointed out one singular configuration for the Stewart Platform. Fichter (1986) studied a prototype Stewart Platform for singular configurations. Behi (1988) used screw theory to identify two singular positions for his Stewart Platform based parallel mechanism. Merlet (1989) used Grassmann line geometry to investigate singularities in the INRIA prototype manipulator, which is a variant of the Stewart Platform. Cleary and Arai (1991) performed singularity analysis on a prototype parallel link manipulator having a slightly different configuration than the Stewart Platform.

8.3 Method

8.3.1 Jacobian Matrix Based Singularity Determination

Singularity analysis of closed-loop kinematic chains based on the properties of the Jacobian matrices of the chain is described by Gosselin and Angeles (1990). They differentiate the relationship between the input and output coordinates with respect to time to obtain the relationship between the input and output velocities. In general, this can be written as follows:

$$[B]\dot{l} + [A]\dot{x} = 0 \qquad (8.1)$$

They have identified three kinds of singularities when either matrix [A] or matrix [B] or both become singular. The first kind of singularity occurs at workspace boundaries when matrix [B] becomes singular resulting in the loss of one or more DOFs. The second kind of singularity occurs when matrix [A] becomes singular. When matrix [A] and matrix [B] simultaneously become singular the third kind of singularity occurs. When the last two kinds of singularities occur, it is likely that the output links gain one or more degree-of-freedom. Under this condition, the manipulator cannot resist forces or moments. These types of singular configurations are highly undesirable and hence need to be determined so they can be avoided.

In the case of the double-octahedral manipulator, the Jacobian matrices [A] and [B] are presented in Eq. 7.4. On inspecting Eq. 7.4, it is clear that a singularity arises in matrix [B] when any of the actuator lengths go to zero. When this condition occurs, two of the mid-plane nodes coincide, making one of the rows of the A matrix all zero. Therefore, A matrix becomes singular as well. Using Gosselin and Angeles' result,

this singularity is characterized as the third kind of singularity. This leads to the gain of mobility in the manipulator.

Another singularity occurs when the mid-plane nodes become collinear. Eq. 8.2 gives the condition for collinearity of the mid-plane nodes.

$$\vec{B}_3 = \lambda \vec{B}_1 + (1-\lambda)\vec{B}_2 \tag{8.2}$$

Using the above condition, Eq. 7.4 can be rewritten as

$$\begin{bmatrix} l_1 & 0 & 0 \\ 0 & l_2 & 0 \\ 0 & 0 & l_3 \end{bmatrix} \begin{Bmatrix} l_1 \\ l_2 \\ l_3 \end{Bmatrix} - \begin{bmatrix} \lambda^2 \vec{B}_{12} \cdot \frac{\partial \vec{B}_{12}}{\partial X} & \lambda^2 \vec{B}_{12} \cdot \frac{\partial \vec{B}_{12}}{\partial Y} & \lambda^2 \vec{B}_{12} \cdot \frac{\partial \vec{B}_{12}}{\partial Z} \\ \vec{B}_{31} \cdot \frac{\partial \vec{B}_{31}}{\partial X} & \vec{B}_{31} \cdot \frac{\partial \vec{B}_{31}}{\partial Y} & \vec{B}_{31} \cdot \frac{\partial \vec{B}_{31}}{\partial Z} \\ \vec{B}_{12} \cdot \frac{\partial \vec{B}_{12}}{\partial X} & \vec{B}_{12} \cdot \frac{\partial \vec{B}_{12}}{\partial Y} & \vec{B}_{12} \cdot \frac{\partial \vec{B}_{12}}{\partial Z} \end{bmatrix} \begin{Bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{Bmatrix} = 0 \tag{8.3}$$

It is clear that the [A] matrix in Eq. 8.3 is singular, since the first row is a multiple of the third row. This results in the second kind of singularity. This also leads to the gain of a degree of freedom at the output link.

The first kind of singularity occurs at the boundary of the workspace. At the workspace boundary, at least one of the discriminants from Eq. 6.8 is exactly zero. This condition is shown in Eq. 8.4.

$$a_{1i}^2 + a_{2i}^2 - a_{3i}^2 = 0 \qquad i = 1,2,3 \tag{8.4}$$

On inspecting the derivation of the Jacobian matrix more closely, it is clear that the square root of the discriminant forms one of the denominators in Eqs. 7.6 and 7.7. If this term is used as an multiplication factor on both matrices [A] and [B], then when

the discriminant becomes zero, there are still some terms in the matrix [B] that are non-zero. However, the diagonal of matrix [A] becomes zero and hence [A] becomes a singular matrix. Since [B] is not singular, this singularity is the first kind of singularity and corresponds to the case where finite motion of the inputs produces no motion at the outputs.

8.3.2 Method-of-Tetrahedrons-Based Singularity Determination

The mobility equation is a useful tool for predicting the mobility of a system. A comprehensive mobility analysis was performed on the isostatic frameworks in the structural analysis section of this dissertation. Various conditions were developed for the framework to be isostatic. There are, however, special cases that cannot be determined using the mobility analysis that lead to the gain of one or more degrees of freedom in the framework. These singularities can be determined using the tetrahedron method for geometry determination. The solution curve to the geometry determination problem Is used to determine these singularities. For one of the geometries of the 60 octahedral-4 framework, the solution curve is generated using the Turbo C code shown in Appendix 2a. This curve is shown in Fig. 23. The merger of the curve with the X axis suggests that, there is a range of solutions for a range of the added link length. This leads to a singularity. The configuration of the octahedral framework in the two extreme positions are shown in Fig. 23.

In the case of the double-octahedral manipulator studied in this dissertation this singularity does not occur. The geometry chosen for the manipulator is such that

distinct solutions exist to the geometry through out the range of actuation. Only the singularities discussed using the method of Jacobian matrices apply to this geometry. However, the singularity analysis using the method of tetrahedrons has to be performed if any modifications to the geometry of the double-octahedral manipulator is considered.

This kind of singularity can also be identified through the method of Jacobian matrices. The relationship between the unactuated member lengths are, however, not part of the Jacobian matrix determined in this dissertation. Therefore, a different jacobian matrix is necessary to identify this singularity. It was mentioned earlier that Tidwell et al. [1990] developed a method for determining the Jacobian matrix. Their approach yields a generalized Jacobian matrix that contains the elements required to determine this kind of singularity. However, preliminary singularity analysis using this matrix found any singularity determination virtually intractable.

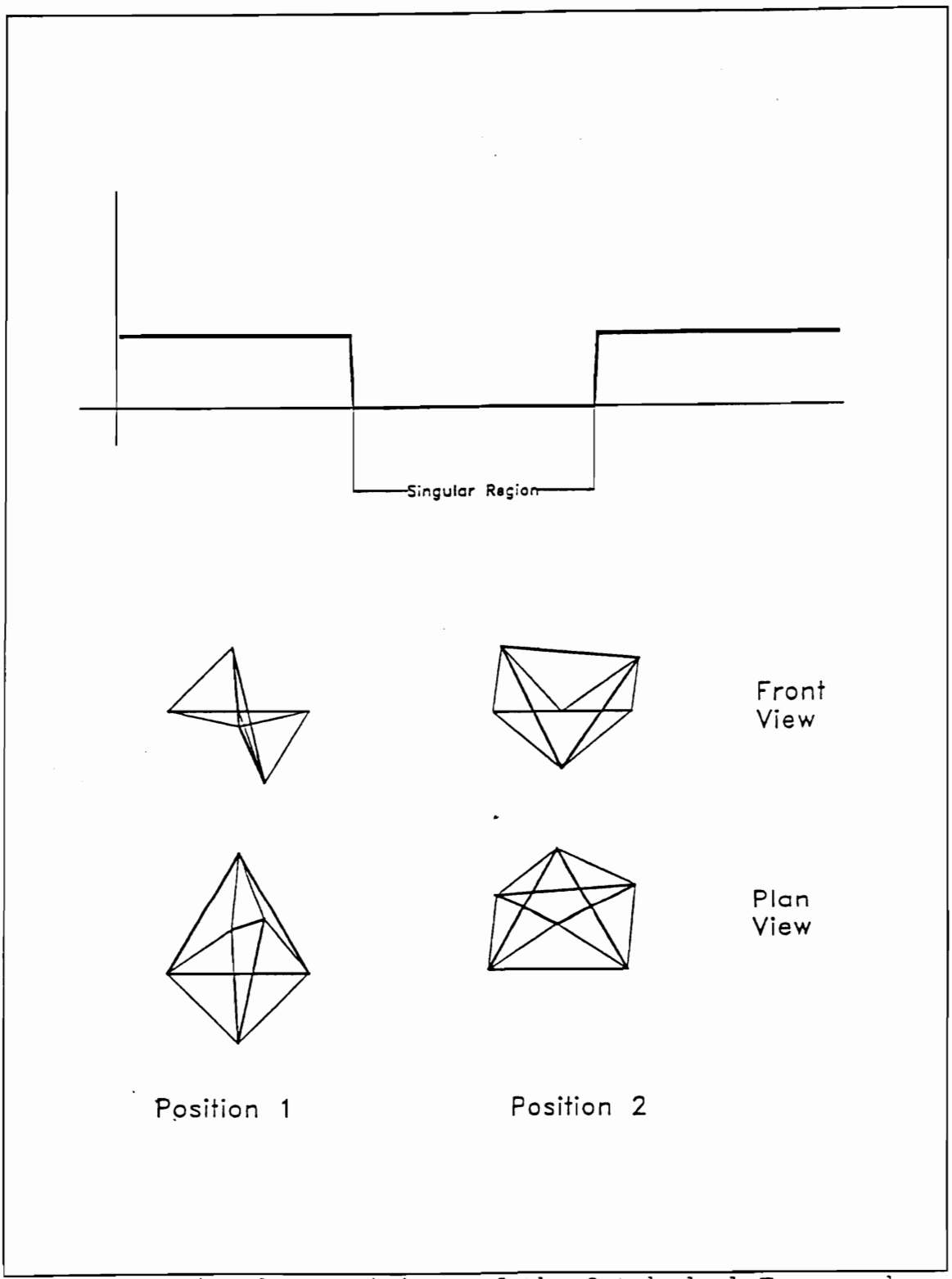


Figure 23. Singular positions of the Octahedral Framework

9 Workspace Analysis

9.1 Objective

The objective of the workspace analysis is to find the range of input parameters (end-effector location in the case of positioning problem; orientation of the moving plane in the case of the gimbal problem) that satisfy a given range of output parameters (actuator lengths).

9.2 Literature Review

Among parallel manipulators, the workspace of the Stewart platform has been studied most. A limited investigation of its workspace, using a special case, is performed by Yang and Lee [1984]. Fichter [1986] developed a simulation to investigate some of the kinematics of the Stewart platform. One of the outputs of the simulation is a plot of a cross-section of the work-envelope of the manipulator. Cwiakala [1986] used a partial scanning technique called "The Optimum Path Search Technique" to find the workspace cross-section for a number of cases for the Stewart Platform.

Merlet [1987] obtained workspace plots for the INRIA prototype manipulator, which is a variant of the Stewart Platform. Behi [1988] used a simulation to examine the workspace of a Stewart Platform based parallel mechanism with 3 PRPS sub-chains connecting the output platform to the base. Cleary and Arai [1991] have used

a simulation to plot workspaces for a prototype manipulator with a slightly different configuration from the Stewart Platform. The simulation results were verified using the prototype. Funabashi et al., [1991] have investigated the relationship between workspace and selected parameters of a 6-DOF parallel manipulator, each leg of which has three revolute pairs, and a spheric pair. Lee and Shah [1988] have used a kinematic simulation to plot the work envelope of a three-DOF parallel manipulator.

All the above efforts were based on a discretization of Cartesian space. Gosselin [1990] is the first to geometrically obtain sections of the workspace of a six-DOF parallel manipulator. Any three-dimensional section of the positioning workspace can be determined using Gosselin's method. The method is briefly described here. First, the inverse kinematic problem for the manipulator is solved to establish regions in space whose intersection will result in the workspace. For a given orientation of the platform, the workspace of the parallel manipulator in three-dimensional Cartesian space is described as the intersection of six regions, each region being the volume between two concentric spheres. A section of the workspace is obtained by taking the intersection of the spheres with a plane, which amounts to finding the intersection of six annular regions. An algorithm that computes the intersection and also the volume of the workspace is presented in the paper. The INRIA prototype (based on the Stewart platform) was used to demonstrate the method.

9.3 Method

9.3.1 Positioning Workspace

The range of motion of the end effector location for a given range of actuator lengths can be described either as a solid volume that consists of every reachable points, or as the outer surface forming a shell of reachable points.

Volumetric Definition of the Workspace

The common volume defined by the implicit equation, given previously as Eq. 6.8, applied simultaneously to all three RS links, provides the work volume of the double-octahedral manipulator when there are no restrictions on the range of the actuators and when link interferences are neglected. When constraints are imposed on the range of movement of the actuators Eq. 6.9 must be considered in the workspace definition. Eq. 6.9 provides the relationship between the actuator lengths and the moving frame location. If each of the link lengths, l_i, vary from l_{imin} to l_{imax}, they will describe three regions in space defined by the following equations.

$$\begin{aligned} l_{1min} \le |B_2 - B_3| \le l_{1max} \\ l_{2min} \le |B_3 - B_1| \le l_{2max} \\ l_{3min} \le |B_1 - B_2| \le l_{3max} \end{aligned} \qquad (9.1)$$

The volume that is common to the regions described by Eq. 9.1 (which is satisfied only when Eq. 6.9 is true) is the reachable workspace of the double-octahedral manipulator, with actuator length constraints.

It is also possible to add external constraints to the volume definition to get

specific properties. One such constraint to eliminate the link interferences can be achieved by setting bounds on the rotation angle of the RS link defined in Fig. 17. This result is obtained by observing the double-octahedral manipulator and keeps all the links in the same closure. The constraint is given in Eq. 9.2.

$$0 \leq \theta_i \leq \pi \qquad i = 1,2,3 \tag{9.2}$$

where θ_i, the rotational angle of the RS link, is defined in Fig. 15.

Results

Algorithm for Workspace Evaluation

Most volume rendering techniques build on cuboids or tetrahedra for the representation. Such techniques usually result in jagged representations of the defined volume. To obtain a smooth surface representation, an initial seed, in the shape of an infinitesimally small sphere (or an extremely large sphere) was taken and iteratively expanded (or contracted) till it converged onto the surface of the work volume. This rudimentary method works well for convex workspaces, but finds only some concavities and protrusions. Forsey [1991] has worked on hierarchial surfaces for animation characters. An algorithm was developed by K. V. Kolady, Research Associate at the CAD/CAM Laboratory at VPI&SU using a similar hierarchical surface definition in order to define the workspace concavities with more precision. The algorithm was implemented on a Silicon Graphics Iris 4D/80 GT machine, and the code was written in GL.

Figure 24a shows the work volume of the manipulator without considering

restrictions in the range of actuators and neglecting any link interferences for the dimensions stated earlier. Figure 24b shows the work volume with an actuator range of 34 inches to 64 inches, neglecting any link interferences. Figure 25a shows the work volume without restrictions in the range of actuators, but with additional constraints to avoid link interferences, given in Eq. 9.2. Figure 25b shows the work volume with restrictions in actuator movement, and also with constraints to avoid link interferences.

Surface Definition of the Workspace Boundary

The points on the boundary of the workspace is described by the implicit equation given in Eq. 9.3.

$$a_{1i}^2+a_{2i}^2-a_{3i}^2 = 0 \qquad (9.3)$$

This marks the boundary between a region of solutions and no solutions. When the actuated members are restricted to move between limits, Eqs. 9.4 are needed in addition to the previous equation to define the surface.

$$\begin{aligned} l_{1min} &= |B_2 - B_3| \\ l_{2min} &= |B_3 - B_1| \\ l_{3min} &= |B_1 - B_2| \\ l_{1max} &= |B_2 - B_3| \\ l_{2max} &= |B_3 - B_1| \\ l_{3max} &= |B_1 - B_2| \end{aligned} \qquad (9.4)$$

The approach taken here is to determine the workspace by employing sections and finding a common area.

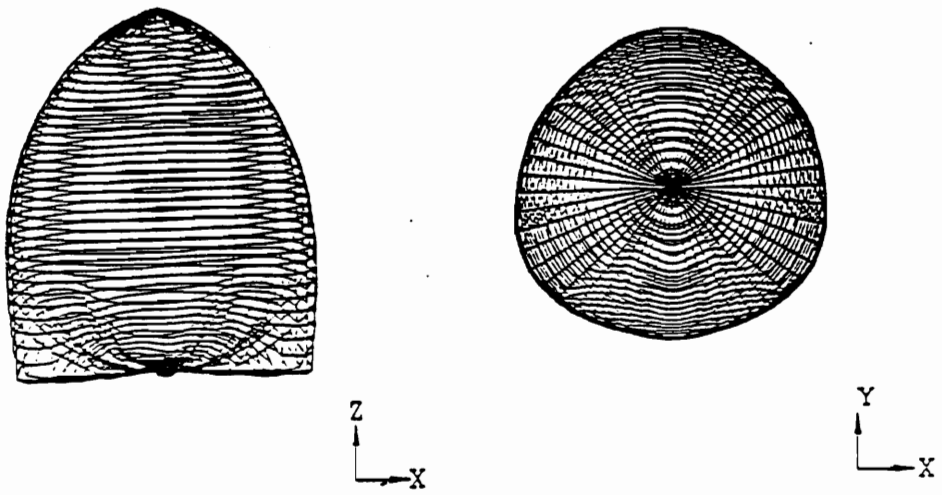


Fig. 24a Without Constraints on Actuator Range

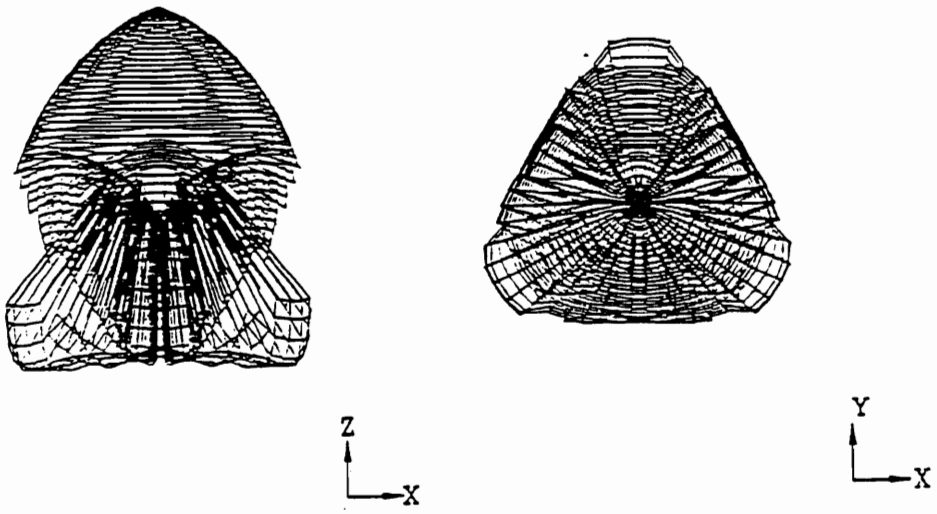


Fig. 24b With Constraints on Actuator Range

Figure 24 Volume Representation (Neglecting Link Interferences)

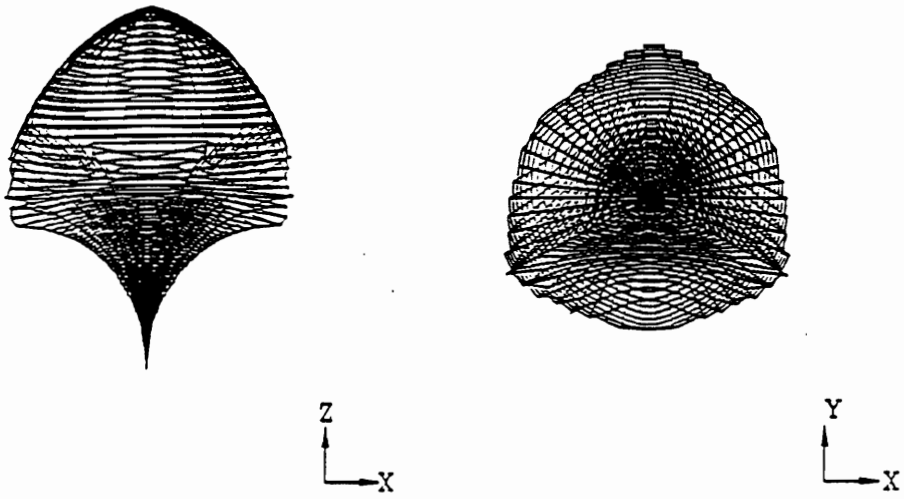


Fig. 25a Without Constraints on Actuator Range

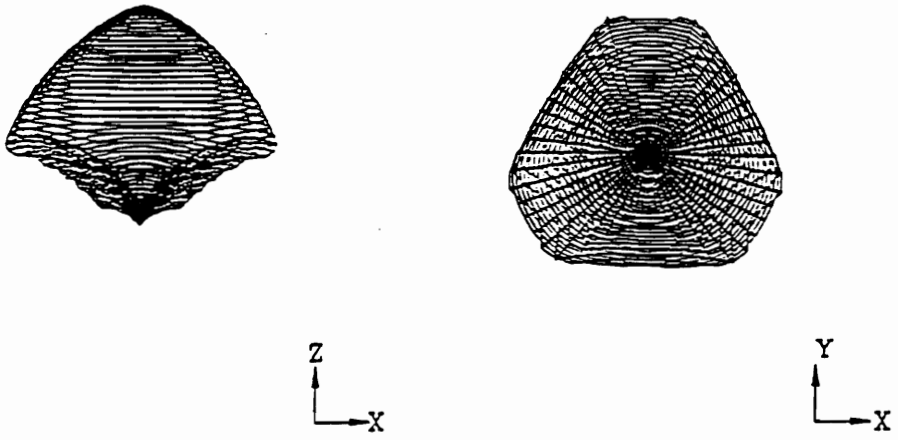


Fig. 25b With Constraints on Actuator Range

Figure 25 Volume Representation (Avoiding Link Interferences)

Results

Figure 26a shows the sections of the work space of the manipulator, the result of cutting the boundary with a plane parallel to the fixed plane, without considering restrictions in the range of actuators and neglecting any link interferences for the dimensions stated earlier. Figure 26b shows sections of the workspace with an actuator range of 34 inches to 64 inches, neglecting any link interferences. Figure 27a shows sections of the workspace without restrictions in the range of actuators, and additional constraints to avoid link interferences, given in Eq. 9.2. Figure 27b shows sections of the workspace with restrictions in actuator movement, and also with additional constraints to avoid link interferences.

9.3.2 Gimbal Workspace

The range of possible orientations for a given range of actuator motion is called the workspace of the gimbal. The workspace of the double-octahedral gimbal is based on the distance parameter 'd' or 'r'. The method used to graphically represent the workspace is to plot the boundary of all possible orientations the gimbal is capable of obtaining. This is repeated for various values of 'd'. This value is parametrized by using the ratio d/h. where h is the maximum height for each unit cell. Figure 28 provides the workspace plots for the batten to longeron (b/l) ratio of 0.5. Figure 29 does the same for the ratio 1.0.

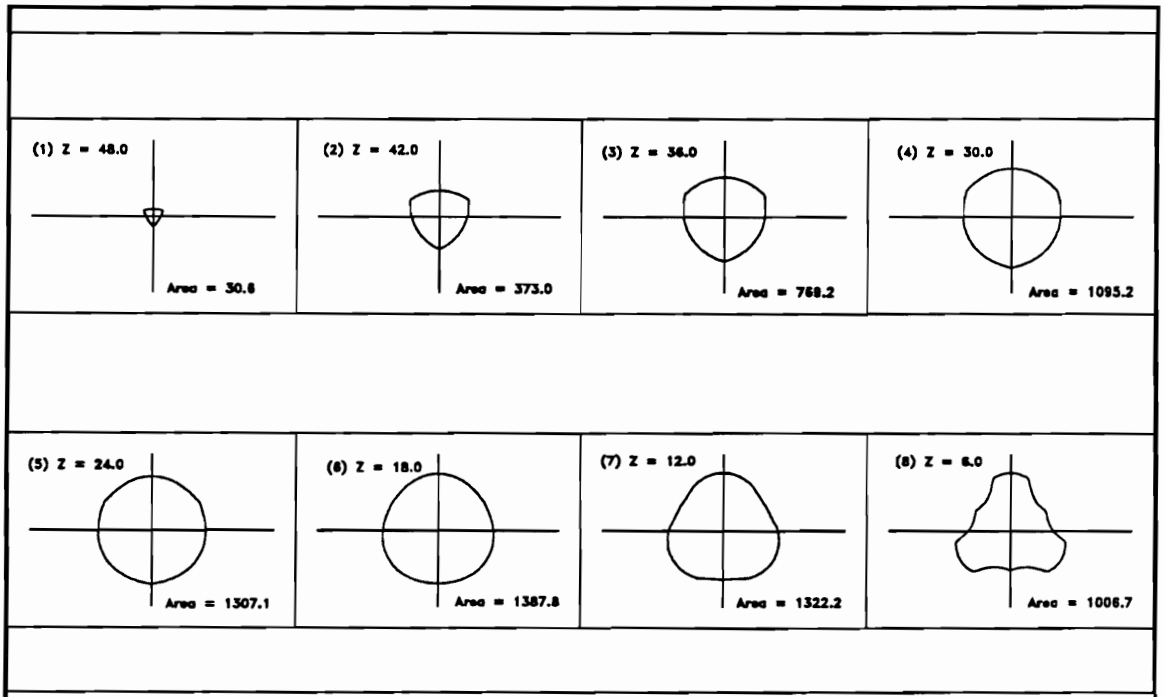


Fig. 26a Without Constraints on Actuator Range

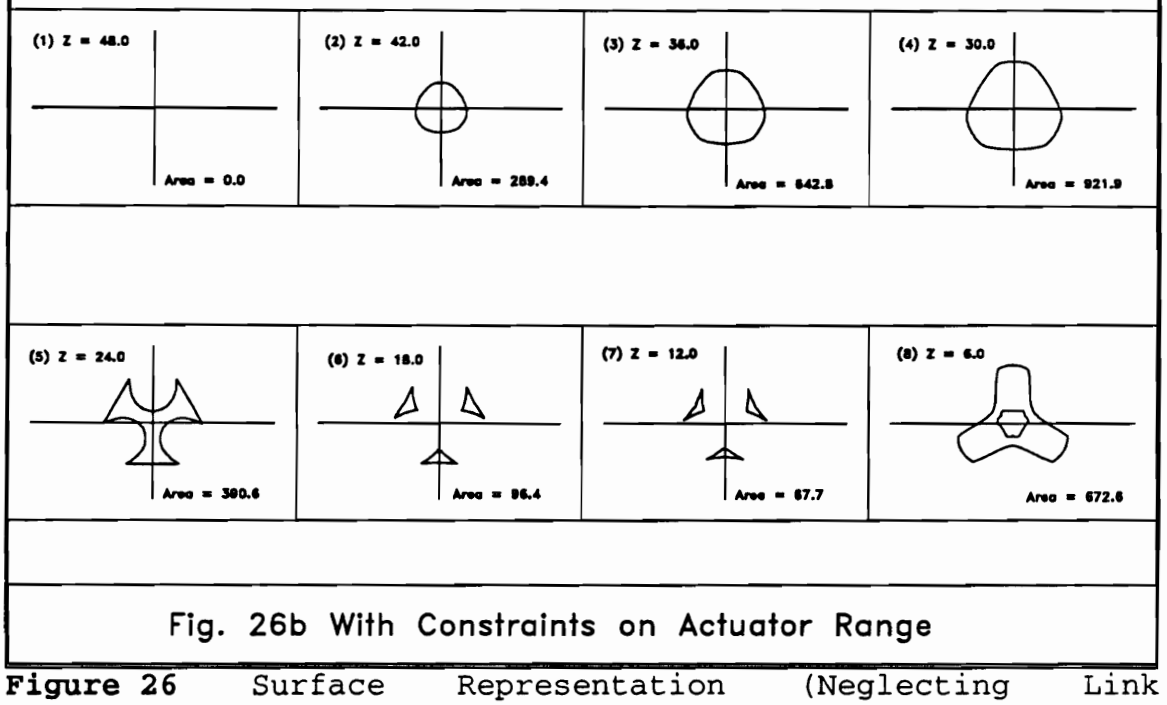


Fig. 26b With Constraints on Actuator Range

Figure 26 Surface Representation (Neglecting Link Interferences)

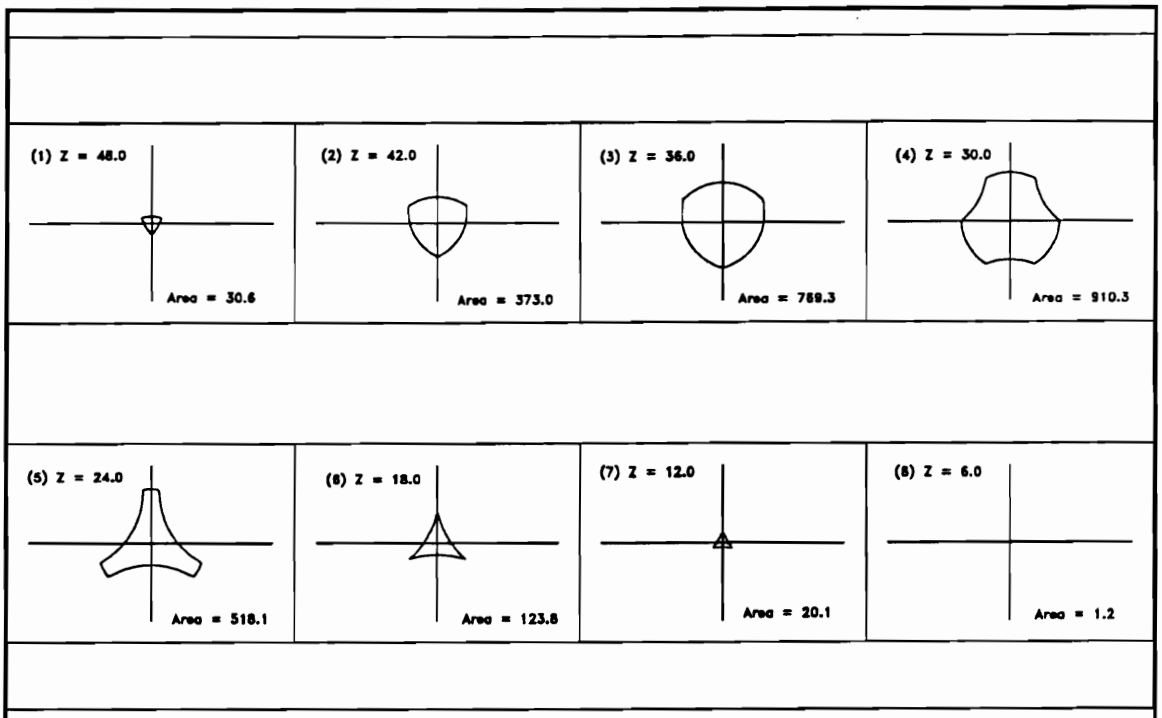


Fig. 27a Without Constraints on Actuator Range

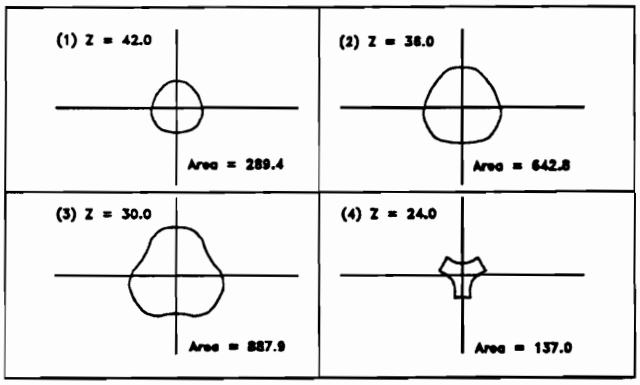


Fig. 27b With Constraints on Actuator Range

Figure 27 Surface Representation (Avoiding Link Interferences)

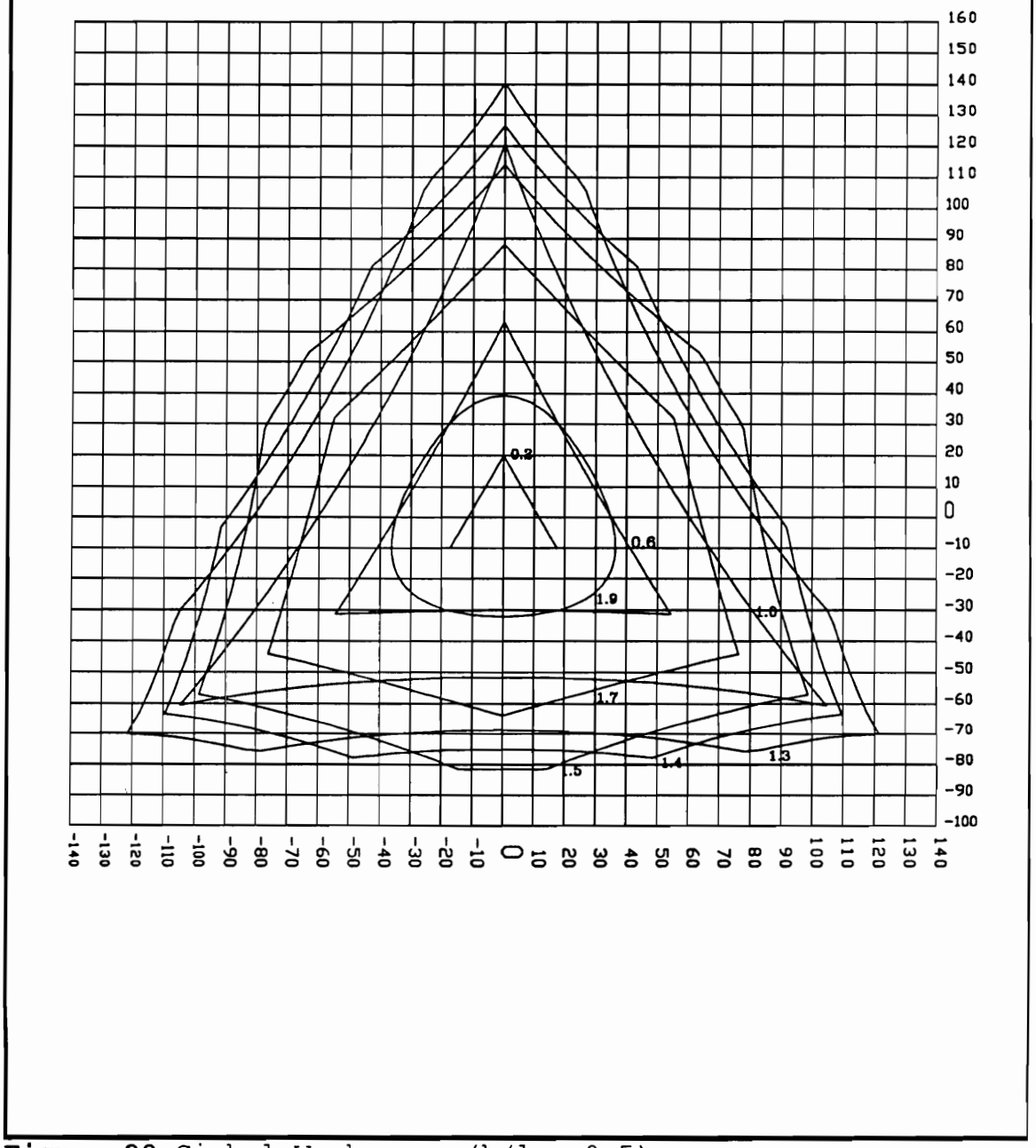


Figure 28 Gimbal Workspace (b/l = 0.5)

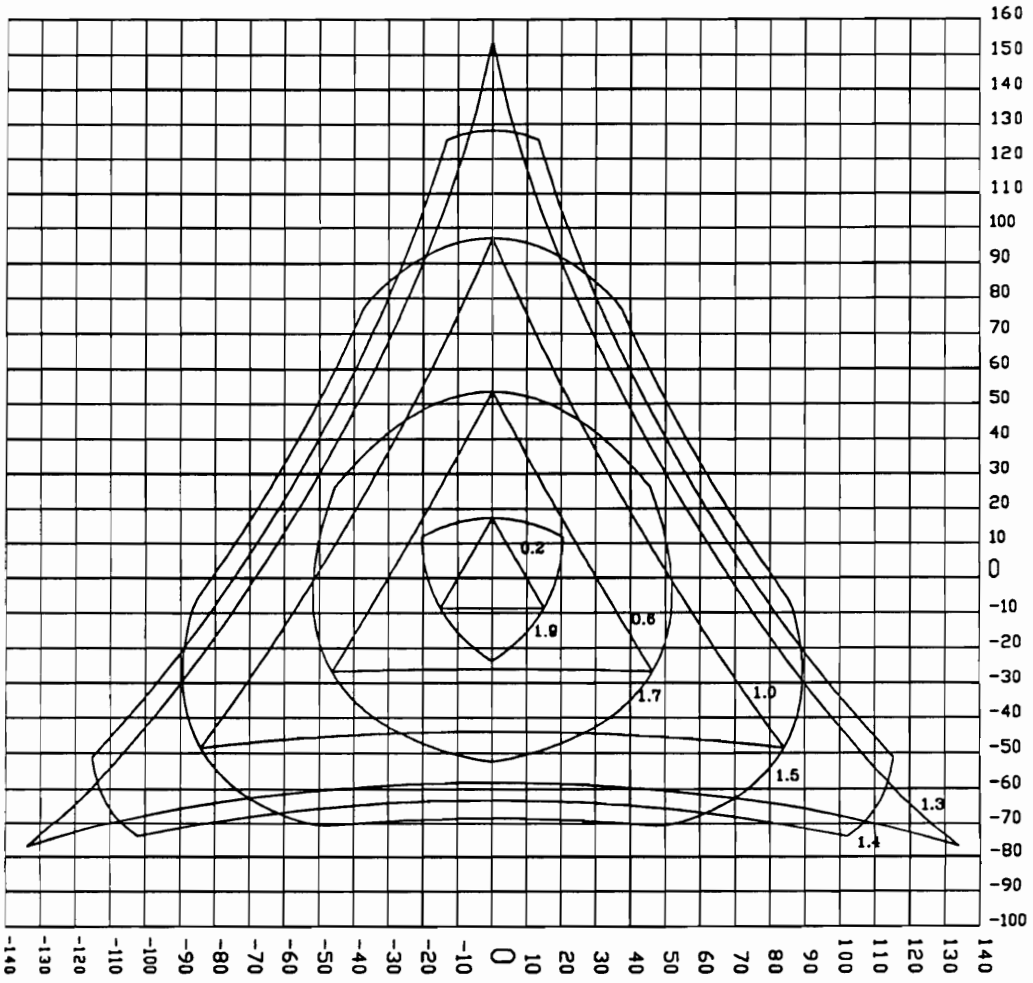


Figure 29 Gimbal Workspace (b/l =1.0)

10. Conclusions and Recommendations

This dissertation has presented many original methods for the design and analysis of isostatic frameworks and truss-based manipulators. The original methods presented on the design and analysis of isostatic frameworks and truss-based manipulators are as follows.

- Generation of general isostatic frameworks using the method of degree sequences.
- Generation of deltahedral unit cells using the planar graph based approach and Cox's method.
- Generation of geometries from a given topology and the determination of nodal coordinates using the method of tetrahedrons
- Closed-form solution to the inverse kinematic problem with extensions to the determination of the Jacobian and Hessian matrices.

In addition to the original contributions listed above, the following methods were extended and improved in this dissertation.

- Determination of forces in the members using the Henneberg's method.
- Forward kinematic analysis of the double-octahedral manipulator with positioning, gimbal and docking specification.

Also, currently available concepts for the design of joints in the double-octahedral manipulator were provided. Applications of current methods in the

determination of workspaces and singularities were presented.

The design of devices in such diverse areas as new structures, novel manipulators, crawling machines, constant-velocity joints, and as yet undiscovered devices depend on the ability to generate and classify the innumerable framework topologies.

The design of isostatic framework involves the generation of the isostatic framework topologies, generation of possible geometries, determination of nodal coordinates, and determination of forces. In the past, the solution to these problems were unavailable and a systematic design of structures was unexplored. The methods for generation, classification, geometry determination, and force determination presented in this dissertation provides the necessary tools to systematically and rigorously design new isostatic frameworks.

Manipulators are needed in increasing frequency to perform tasks in limited-access obstacle-ridden unstructured environments. These tasks require the use of High-DOF manipulators. Serial manipulators are not capable of providing the required strength and rigidity. Truss-based manipulators provide a potential answer to this problem. This dissertation provided an exhaustive kinematic analysis of an important truss-based manipulator, the double-octahedral manipulator. A long-chain of double-octahedral manipulators acting as actuated gimbals connecting static frameworks would be capable of performing tasks that are impossible for current

serial manipulators. One such task proposed for the device is the inspection of nuclear waste sites, such as the one at Hanford, Washington. Salerno et al. [1991] have studied the kinematics of such long-chain manipulators.

The double-octahedral manipulator as an independent actuating unit can be used as a positioning device or as an actuated gimbal-like device. Complete analysis is provided in this dissertation for both of these applications. The quadruple-octahedral manipulator is capable of providing five degrees-of-freedom at the end-effector and is recommended for docking-type applications. Such manipulators have the capacity to support and actuate large structures such as antennas and reflectors in space and on the ground.

The double-octahedral manipulator has been shown to have good vibration control characteristics [Robertshaw et al., 1989, Wada, 1990]. Vibration damping and isolation applications in space are likely for these devices[1]. Warrington [1991] and Lacy [1991] have used methods presented in this dissertation to perform their studies in vibration control and dynamic analysis.

Recommendations for Future Work

In the second chapter of this dissertation, principles for generating deltahedral unit cells were discussed. One method was to represent the deltahedral unit cells as

[1] The manipulator is referred to as an adaptive truss by authors working on the dynamics and control of these devices.

four-connected fully-triangulated planar graphs. An excellent extension of the present work would be to develop an algorithm to generate the four-connected fully-triangulated planar graphs for a given number of nodes. Another method discussed for the generation of deltahedral unit cells was Cox's method. Developing a graphical representation of this method is also recommended as an extension of the present work. It is likely that a good representation would lead to further development of this method and could lend some insight into the tetrahedrizing problem. The problem of minimally tetrahedrizing deltahedral unit cells is also recommended for further study. Such an algorithm will help to automate both geometry generation, determination of nodal coordinates and force analysis. All of the above-mentioned problems are suggested to a researcher in graph theory because these problems extensively rely upon current concepts and publications in this field. The motivation for solving these problems is that, once the algorithms are known, software can be developed that will allow a designer of isostatic frameworks to evaluate many possibilities quickly and easily.

The creation of software to assist the user in designing spatial isostatic frameworks is also recommended. Problems to be solved include generation, representation, generation of geometry, determination of nodal coordinates and determination of forces in the members of the framework. Such software would be an ideal design tool to bring together the conceptual design and analysis of isostatic framework. The creation of a comprehensive software to study specific tasks of the double-octahedral manipulator is also recommended. Appendices 1-5 of this

dissertation provide C-language codes to various general methods of analysis of isostatic frameworks and truss-based manipulators. While this code is carefully written and well documented, it is not the kind of user-friendly software necessary for a practicing designer to put the ideas into everyday use.

Design of a 3-DOF joint that is capable of providing the required rigidity and that can bring the structure of the framework closer to ideal is recommended to a design engineer. Current joints are mechanically complex and contain joint offsets that induce small amounts of bending, shear and torsion in the members. The use of elastomeric and composite materials could result in a new design that eliminates the mechanical complexity and joint offsets (like the joints in the human body). A design of an actuator that is capable of working under extreme conditions, such as in space or a nuclear storage facility is also recommended to the design engineer. Many actuators are necessary in the high-DOF devices. Therefore, any advance in actuator design will critically affect the development of long-chain manipulators.

References

1. Alberts, T. E., Love, L. J., Bayo, E., Moulin, H., 1990, "Experiments with End-Point Control of a Flexible Link Using the Inverse Dynamics Approach and Passive Damping," *Proceedings of the 1990 American Control Conference*, Vol. 1, pp. 350-355.

2. Anderson, E. H., Moore, D. M., Fanson, J. L., Ealey, M. A., 1990, "Development of an Active Truss Element for Control of Precision Structures," *Optical Engineering*, Vol. 29, No. 11, pp. 1333-1341.

3. Arun, V., Padmanabhan, B., Kolady, K., and Reinholtz, C. F., 1992, "Determination of the Workspace of the 3-DOF Double-Octahedral Variable-Geometry-Truss Manipulator," To be presented at the *22nd ASME Mechanisms Conference*, Phoenix, AZ.

4. Arun, V., Reinholtz, C. F., and Watson, L. T., 1990a "Enumeration and Analysis of Variable-Geometry Truss Manipulators," *Proceedings of the 1990 ASME Mechanisms Conference*, DE-VOL 26, Cams, Gears, and Mechanisms Design, pp. 199-205.

5. Arun, V., Reinholtz, C. F., and Watson, L. T., 1990b "Application of New Homotopy Techniques to Variable-Geometry Trusses," *Proceedings of the 1990 ASME Mechanisms Conference*, DE-VOL 26, Cams, Gears, and Mechanisms Design, pp. 87-92.

6. Asimow, L., and Roth, B., 1978, "Rigidity of Graphs," *Transactions of American Mathematical Society*, Vol. 245, pp. 279-289.

7. Behi, F., 1988, "Kinematic Analysis for a Six-Degree-of-Freedom 3-PRPS Parallel Mechanism," *IEEE Journal of Robotics and Automation*, Vol. 4, No. 5, pp. 561-565.

8. Burdisso, R. A., and Haftka, R. T., 1989, "Optimal Location of Actuators for Correcting Distortions in Large Truss Structures," *AIAA Journal*, Vol. 27, No. 10, pp. 1406- 1411.

9. Bush, H. G., Herstrom, C. L., Heard, W. L. Jr., Collins, T. J., Fichter, W. B., Wallsom, R. E., Phelps, J. E., 1991, "Design and Fabrication of an erectable truss for Precision Segmented Reflector Application," *Journal of Spacecraft and Rockets*, Vol. 28, No. 2, pp. 251-257.

10. Clark, W. W., Kimiavi, B., Robertshaw, H. H., 1989, "Control of Flexible Beams Using a Free-Free Active Truss," *Proceedings of the 1989 ASME Winter Annual Meeting - Adaptive Structures Session*, pp. 61-68

11. Cleary, K., and Arai, T., 1991, "A Prototype Parallel Manipulator : Kinematics, Construction, Software, Workspace Results, and Singularity Analysis," *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, pp. 566-571

12. Cox, H. R., 1936, "On the Synthesis and Analysis of Simply-Stiff Frameworks," *Proceedings of London Mathematical Society*, Vol. 40, pp. 203-216.

13. Cox, R. L., and Nelson R. A., 1982, "Development of Deployable Structures for Large Space Platform Systems," Rep. No. 2-32300/2R-53215 (Contract NAS8-34678), Vought Corporation.

14. Coxeter, H. S. M., 1973, "Regular Polytopes," Third Edition, *Dover Publication*, pp. 9-10.

15. Cwiakala, M., 1986, "Workspace of a Closed-Loop Manipulator", ASME publication 86-DET-95.

16. Davison, W., 1990, "Design Strategies for Very Large Telescopes," *Proceedings of SPIE - The International Society for Optical Engineering*, Vol. 1236, Pt. 2, Published by International Society for Optical Engineering, Bellingham, WA, USA, pp. 878-883.

17. Dorsey, J. T., and Mikulas, M. M. Jr., 1990, "Preliminary Design of a Large Tetrahedral Truss / Hexagonal Panel Aerobrake Structural System," *Collection of Technical Papers - 31st SDM conference*, Pt. 1, Published by AIAA, NY, USA, pp. 533-546.

18. Edelsbrunner, H., Preparata, F. P., and West, D. B., 1990, "Tetrahedrizing Point Sets in Three Dimensions," *Journal of Symbolic Computation*, Vol. 10, 335-347.

19. Fanson, J. L., 1987, "An Experimental Investigation of Vibration Suppression in Large Space Structures Using Position Feedback," *Ph. D. Thesis*, California Institute of Technology, Pasadena, CA.

20. Fichter, E. F., 1985, "A Stewart Platform-Based Manipulator: General Theory and Practical Construction", *The International Journal of Robotic Research*, Vol. 5, No. 2.

21. Fichter, E. F., and McDowell, E. D., 1982, "A Novel Design for a Robot Arm," *Advances in Computer Technology*, ASME Publication, pp. 250-256.

22. Forsey, D. R., 1991, "Hierarchical Free-Form Surface Modeling", Course Notes, Topics in the Construction, Manipulation and Assessment of Spline Surfaces, pp. 5-0 - 5-46, SIGGRAPH 1991, Las Vegas.

23. Funabashi, H., Horie, M., Kubota, T., and Takeda, Y., 1991, "Development of Spatial Parallel Manipulators with Six Degrees of Freedom", *JSME International Journal, Series III*, Vol. 34, No. 3, pp 382-387.

24. Gosselin, C., 1990, "Determination of the Workspace of 6-DOF Parallel Manipulators," *Journal of Mechanical Design*, Vol. 112, pp. 331-336.

25. Gosselin, C., and Angeles, J., 1990, "Singularity Analysis of Closed-Loop Kinematic Chains," *IEEE Transactions on Robotics and Automation*, Vol. 6, No. 3, pp.281-290.

26. Griffis, M., and Duffy, J., 1989, "A Forward Displacement Analysis of a Class of Stewart Platforms," *Journal of Robotic Systems*, Vol. 6, No. 6, pp. 703-720.

27. Henneberg, L., 1911, "*Die Graphische Statik der Starren Systeme*," Liepzig 1911, Johnson Reprint 1968.

28. Hunt, K. H., 1978, "*Kinematic Geometry of Mechanisms*", Oxford University Press, London.

29. Hunt, K. H., 1983, "Structural Kinematics of In-Parallel-Actuated Robot Arms," *Journal of Mechanisms, Transmissions, and Automation in Design*, Vol. 105, No. 4, pp. 705-712.

30. Innocenti, C., and Parenti-Castelli, V., 1990, "Direct Position Analysis of the Stewart Platform Mechanism," *Mechanisms and Machine Theory*, Vol. 25, No. 6, pp. 611-621.

31. Juang, J. N., Horta, L. G., Robertshaw, H. H., 1986, "A Slewing Control of Flexible Structures," *Journal of Guidance, Control, and Dynamics*, Vol. 9, No. 5.

32. Kawaguchi, M., 1990, "Space Structures with Changing Geometries," *Bulletin of the International Association for Shell and Spatial Structures*," Vol. 31, No. 102-103, pp.33-45.

33. Lacy, D. T., 1991, "An Automated Methodology of Force Analysis in Adaptive Trusses," *M. S. Thesis*, Virginia Polytechnic Institute and State University, Blacksburg, VA.

34. Laman, G., 1970, "On Graphs and Rigidity of Plane Skeletal Structures," *Journal of Engineering Mathematics*, Vol. 4, pp. 331-340

35. Lee, K-M., and Shah, D., 1988, "Kinematic Analysis of a Three-Degrees-of-Freedom In-Parallel Actuated Manipulator", *IEEE Journal of Robotics and Automation*, Vol. 4, No. 3, pp 354-360.

36. Mabie, H. H., and Reinholtz, C. F., 1987, "*Mechanisms and Dynamics of Machinery*," Fourth Edition, John Wiley & Sons, pp. 582-593.

37. Marshall, W. T., and Nelson, H. M., 1969, "*Structures*," Sir Isaac Pitman & Sons Ltd, London.

38. Matunaga, S., Miura, K., Natori, M., "Construction Concept of Large Space Structure Using Intelligent/Adaptive Structures," *Collection of Technical Papers - 31st SDM Conference*, Pt. 4, Published by AIAA, NY, USA, pp. 2298-2305.

39. Merlet, J-P, 1987, "Parallel Manipulator : Kinematics, Singular Configurations and Compliance," *Proceedings of the 3rd International Conference on Advanced Robotics*, pp. 125-135.

40. Mikulas, Martin M., Collins, T. J., Hedgepeth, J. M., 1991, "Preliminary Design Considerations for 10-40 meter-diameter Precision Truss Reflectors," *Journal of Spacecraft and Rockets*, Vol. 28, No. 4, pp. 439-447.

41. Miura, K., and Furuya, H., 1985, "An Adaptive Structure Concept for Future Space Applications," IAF-85-211, *Proceedings of the 36th Congress of the International Astronautical Federation*, Pergamon Press.

42. Miura, K., Furuya, H., Suzuki, K., 1985, "Variable-Geometry Truss and its applications to Deployable Truss and Space Crane Arm," *Acta Astronautica*, Vol. 12, No. 7-8, pp.599-607.

43. Naccarato, F., and Hughes, P., 1991, "Inverse Kinematics of Variable-Geometry Truss Manipulators," *Journal of Robotic Systems*, Vol. 8, No. 2, pp. 249-266.

44. Nanua, P., and Waldron, K. J., 1989, " ," *IEEE Proceedings of the conference on Robotic Automation*, p. 431.

45. Natori, M., Iwasaki, K., Kuwaco, F., 1987, "Adaptive Planar Truss Structures and Their Vibration Characteristics," *Proceedings of the 28th SDM Conference*, AIAA-87-0743, pp. 143-151.

46. Padmanabhan, B., Tidwell, P. H., Salerno, R. J., Reinholtz, C. F., 1992, "VGT-Based Gimbals : Practical Construction and General Theory," To be Presented at the *22nd ASME Mechanisms Conference*, Phoenix, AZ.

47. Padmanabhan, B., Arun, V., and Reinholtz, C. F., 1990, "Closed-Form Inverse Kinematic Analysis of Variable-Geometry Truss Manipulators," *Proceedings of the 1990 ASME Mechanisms Conference*, DE-VOL 26, Cams, Gears, and Mechanisms Design, Chicago, pp. 99-105.

48. Padmanabhan, B., and Reinholtz, C. F., 1989, "Design of a Robotic Manipulator Using Variable-Geometry Trusses as Joints," *Proceedings of the 1st Robotics and Mechanisms Conference*, Cincinnati, OH, Paper 89AMR-8C-7(6 Pages).

49. Padmanabhan, B., 1989, "Design of a Robotic Manipulator Using Variable-Geometry Trusses as Joints," *M. S. Thesis*, Virginia Polytechnic Institute and State University, Blacksburg, VA.

50. Padmanabhan B., and Arun V., 1988, "VGT's - A New Concept for Manipulator Joints," Third Prize Winning Design Project, Student Mechanisms Design Contest (Graduate), ASME Design Automation Conference, Orlando, Florida.

51. Recksi, A., 1984, "A Network Approach to the Rigidity of Skeletal Structures II," *Discrete Applied Mathematics*, Vol. 8, pp. 63-68.

52. Reinholtz, C. F., and Gokhale, D., 1987, "Design and Analysis of Variable-Geometry Truss Robots," *Proceedings of the 10th Applied Mechanisms Conference*, Vol. 1, New Orleans, LA.

53. Rhodes, M. D., and Mikulas, M. M. Jr., 1985, "Deployable Controllable Geometry Truss Beam," *NASA Technical Memorandum 86366*,

54. Rockwell International, 1982, "Development of Deployable Structures for Large Space Platform Systems," Interim Report, VOL 1, SSD 82-0121-1, (Contract NAS8-34677).

55. Robertshaw, H. H., 1985, "Position Control of Flexible Beams with Root Actuation," Final Report of NASA Grant NAG-1-5-70.

56. Robertshaw, H. H., Wynn, R. H. Jr., Kung H. F., Hendricks, S. L., and Clark W. W., 1989, "Dynamics and Control of a Spatial Active Truss Actuator," *Proceedings of the 30th SDM Conference*, AIAA paper no. 89-1328.

57. Salerno, R. J., Dhande, S. G., Reinholtz, C. F., 1991, "Kinematics of Long-Chain Variable-Geometry-Truss Manipulators : An Overview of Solution Techniques," *Advances in Robot Kinematics - with Emphasis on Symbolic Computation*, Springer-Verlag, Wien.

58. Sincarsin, W. G., and Hughes, P. C., 1987, "Trussarm Candidate Geometries," *Dynacon Report* 28-611/0401.

59. Sohmshetty, R. S., and Kramer, S. N., 1989, "Forward and Inverse Kinematics of the Tetrahedron-Tetrahedron Variable-Geometry-Truss Manipulators," *Proceedings of the 1st National Applied Mechanisms and Robotics Conference*, Vol. 1, 89-AMR-4A-2, Cincinnati, Ohio.

60. Stewart, D., 1965, "Platform with Six Degrees-of-Freedom," *Proceedings of the Institute of Mechanical Engineers*, Vol. 180, Part I, No. 15, pp. 371-386.

61. Suh, C. H., and Radcliffe, C. W., 1983, "*Kinematics and Mechanisms Design*," Robert E. Krieger Publishing Company, Malabar, Florida.

62. Takamatsu, K. A., and Onoda, J., 1991, "New deployable Truss Concepts for Large Antenna Structure or Solar Concentrators," *Journal of Spacecraft and Rockets*, Vol. 28, No. 3, pp. 330-338.

63. Tay, T. S., and Whitley, W., 1985, "Generating Isostatic Frameworks," *Structural Topology* 11, pp. 21-68.

64. Tidwell, P. H., 1989, "Design and Construction of a Double-Octahedral Variable-Geometry-Truss Manipulator," *M. S. Thesis*, Virginia Polytechnic Institute and State University, Blacksburg, VA.

65. Tidwell, P. H., Reinholtz, C. F., Robertshaw, H. H., and Horner, C. G., 1990, "Kinematic Analysis of Generalized Adaptive Trusses," *1st Joint US/Japan Conference on Adaptive Trusses*, Maui, Hawaii.

66. Timoshenko, S. P., and Young, D. H., 1965, "*Theory of Structures*," McGraw-Hill Book Company, Second Edition.

67. Tutte, W. T., 1984, "*Graph Theory*," Encyclopedia of Mathematics and its Applications, Addison-Wesley Publishing Company, Advanced Book Program, Menlo Park, California.

68. Wada, B. K., 1990, "Adaptive Structures : An overview," *Journal of Spacecraft and Rockets*, Vol. 27, No. 3, pp. 330-337.

69. Wada, B. K., Fanson, J. I., Crawley, E. F., 1990, "Adaptive Structures," *Mechanical Engineering*, Vol. 112, No. 11, pp. 41-46.

70. Warrington, T. J., 1991, "Experiments and Simulation of Large Angle Flexible Beam Control Using an Adaptive Truss," *M.S. Thesis*, Virginia Polytechnic and State University, Blacksburg, VA.

71. Wynn, R. H. Jr., 1990, "The Control of Flexible Structure Vibrations Using a Cantilevered Adaptive Truss," *Ph. D. Thesis*, Virginia Polytechnic Institute and State University, Blacksburg, VA.

72. Yang, D. C. H., and Lee, T. W., 1984, "Feasibility Study of a Platform Type of Robotic Manipulator from a Kinematic Viewpoint", *Journal of Mechanisms, Transmissions, and Automation in Design*, Vol. 106, No. 2, pp. 191-198.

Appendix 1a Determination of the 6020 Dodecahedral-6 Geometry

```
/*****************************************************************************/
/*      Program     :  D6plot.C
       Description :  A program to solve the geometry of the 6020 Dodecahedral-6
                      framework.  This program uses functions tetra and func. */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>

/* function declaration */

void tetra (float n1[3], float n2[3], float n3[3], float d14, float d24,
                         float d34, float n4[3], int sgn);
float func (float a);

void main ()

{
FILE *f2;
float f; /* f stores the function value */
float a; /* added link eliminates link l18 */

f2 = fopen ("dodec1.dat", "w");

a = 1.0;

while (a <= 15.0)

{
f = func (a);
fprintf (f2, "%4.2f %4.2f \n", a, f);
a = a + 0.01;
}
fclose (f2);
}

/* Function to calculate the value of the unused link */

float func (float a)

{
/* FILE *f1;    */
float p1[3], p2[3], p3[3];
float p4[3], p5[3], p6[3];
```

```
float p7[3], p8[3];                    /* Node coordinates */
float l14, l24, l45, l46, l47, l48;
float l35, l36, l37, l38;
float l25, l56, l67, l78, l18;         /* Link dimensions */
float f ;                              /* function value */
int sgn;

p1[0] = 0.0;
p1[1] = 0.0;
p1[2] = 0.0;
p2[0] = 6.0;
p2[1] = 0.0;
p2[2] = 0.0;
p3[0] = 3.0;
p3[1] = 5.76628;
p3[2] = 0.0;

l14 = 6.5;
l24 = 6.5;
l45 = 6.5;
l46 = 6.5;
l47 = 6.5;
l48 = 6.5;

l35 = 6.5;
l36 = 6.5;
l37 = 6.5;
l38 = 6.5;

l25 = 6.0;
l56 = 6.0;
l67 = 6.0;
l78 = 6.0;
l18 = 6.0;

sgn = 1;
printf ("a = %4.2f \n", a);
tetra (p1, p2, p3, l14, l24, a, p4, sgn);

/* The variable sgn can assume two values +1 or -1. The various combinations of the values
of sgn represents the various possible closures to the geometry */

if (p4 [2] > -999.0)
        {
        sgn = 1;
        tetra (p3, p4, p2, l35, l45, l25, p5, sgn);

        if (p5[2] > -999.0)
                {
                sgn = 1;
```

```
                tetra (p3, p4, p5, l36, l46, l56, p6, sgn);

                if (p6[2] > -999.0)
                        {
                        sgn = 1;
                        tetra (p3, p4, p6, l37, l47, l67, p7, sgn);

                                if (p7[2] > -999.0)
                                {
                                sgn = -1;
                                tetra (p3, p4, p7, l38, l48, l78, p8, sgn);

                                }
                        }
                }
        }

if (p4[2] < -999.0) return 12.0;
if (p5[2] < -999.0) return 12.0;
if (p6[2] < -999.0) return 12.0;
if (p7[2] < -999.0) return 12.0;
if (p8[2] < -999.0) return 12.0;

f =sqrt ((p1[0] - p8[0]) * (p1[0] - p8[0]) + (p1[1] - p8[1])
        * (p1[1] - p8[1]) + (p1[2] - p8[2]) * (p1[2] - p8[2]));

printf ("%4.2f \n", sqrt ((f -l18) * (f - l18)));

return sqrt ((f - l18) * (f - l18));

}

/* Function to solve the coordinate of the tetrahedron */

void tetra (float n1[3], float n2[3], float n3[3], float d14, float d24, float d34,
                        float n4[3], int sgn)
{
float x1, x2, x3, y1, y2, y3, z1, z2, z3;
float a12, a13, b12, b13, c12, c13, d12, d13, e1, e2, f1, f2;
float a, b, c, d;

x1 = n1[0];
y1 = n1[1];
z1 = n1[2];
x2 = n2[0];
y2 = n2[1];
z2 = n2[2];
x3 = n3[0];
y3 = n3[1];
z3 = n3[2];
```

```
a12 = 2 * (x1 - x2);
a13 = 2 * (x1 - x3);
b12 = 2 * (y1 - y2);
b13 = 2 * (y1 - y3);
c12 = 2 * (z1 - z2);
c13 = 2 * (z1 - z3);
d12 = x1 * x1 - x2 * x2 + y1 * y1 - y2 * y2 + z1 * z1 - z2 * z2 - d14 * d14 + d24 * d24;
d13 = x1 * x1 - x3 * x3 + y1 * y1 - y3 * y3 + z1 * z1 - z3 * z3 - d14 * d14 + d34 * d34;
e1 = (d12 * b13 - d13 * b12) / (a12 * b13 - a13 * b12);
f1 = - (c12 * b13 - c13 * b12) / (a12 * b13 - a13 * b12);
e2 = (d12 * a13 - d13 * a12) / (b12 * a13 - b13 * a12);
f2 = - (c12 * a13 - c13 * a12) / (b12 * a13 - b13 * a12);

a = f1 * f1 + f2 * f2 + 1;
b = 2 * (e1 * f1 - x1 * f1 + e2 * f2 - y1 * f2 - z1);
c = x1 * x1 - 2 * e1 * x1 + e1 * e1 + y1 * y1 - 2 * y1 * e2 + e2 * e2 + z1 * z1 - d14 * d14;
d = b * b - 4 * a * c;

if (d >= 0.0)
{

n4[2] = ( - b + sgn * sqrt (d))/ ( 2 * a);
n4[0] = e1 + f1 * n4[2];
n4[1] = e2 + f2 * n4[2];

}
else
{
n4[0] = 0.0;
n4[1] = 0.0;
n4[2] = -1000.0;
}

}
```

Appendix 1b Determination of the 4400 Dodecahedral-7 Geometry

```
/*****************************************************************************/
/*      Program     :  d7plot.C
       Description :  A program to solve the geometry of the 4400 Dodecahedral-7
                      framework.  This program uses functions tetra and func. */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>

/* function declaration */

void tetra (float n1[3], float n2[3], float n3[3], float d14, float d24,
                        float d34, float n4[3], int sgn);
float func (float a, float b);

void main ()
{
FILE *f2;
float f; /* f stores the function value */
float a, b; /* added links eliminate link l17 and l68 */

f2 = fopen ("dodec2.dat", "w");

b = 0.5;

while (b <= 15.0)
{
fprintf (f2, "next\n");
a = 0.5;
        while (a <= 15.0)
        {
        f = func (a, b);
        fprintf (f2, "%6.4f %6.4f %6.4f \n", a, b, f);
        a = a + 0.5;
        }
b = b + 0.5;
}
fclose (f2);
}

/* Function to calculate the value of the unused links */

float func (float a, float b)
```

```
{
/* FILE *f1;    */
float p1[3], p2[3], p3[3];
float p4[3], p5[3], p6[3];
float p7[3], p8[3];                    /* Node coordinates */
float l14, l24, l45, l46, l47;
float l25, l56, l58;
float l36, l37, l38;
float l28, l68, l67, l17;              /* Link dimensions */
float f1, f2, f ;                      /* function value */
int sgn;

p1[0] = 0.0;
p1[1] = 0.0;
p1[2] = 0.0;
p2[0] = 6.0;
p2[1] = 0.0;
p2[2] = 0.0;
p3[0] = 3.0;
p3[1] = 5.76628;
p3[2] = 0.0;

l14 = 6.5;
l24 = 6.5;
l45 = 6.5;
l46 = 6.5;
l47 = 6.5;

l25 = 6.5;
l58 = 6.5;
l56 = 6.5;

l38 = 6.5;
l36 = 6.5;
l37 = 6.5;

l28 = 6.0;
l68 = 9.2315;
l67 = 6.0;
l17 = 6.8522;

sgn = 1;
printf ("a = %4.2f \n", a);
tetra (p1, p2, p3, l14, l24, a, p4, sgn);

if (p4 [2] > -999.0)
        {
        sgn = 1;
        tetra (p3, p4, p2, b, l45, l25, p5, sgn);
```

```
        if (p5[2] > -999.0)
                {
                sgn = 1;
                tetra (p3, p4, p5, l36, l46, l56, p6, sgn);

                if (p6[2] > -999.0)
                        {
                        sgn = 1;
                        tetra (p3, p4, p6, l37, l47, l67, p7, sgn);

                                if (p7[2] > -999.0)
                                {
                                sgn = -1;
                                tetra (p3, p5, p2, l38, l58, l28, p8, sgn);
                                }
                        }
                }
        }

if (p4[2] < -999.0) return 12.0;
if (p5[2] < -999.0) return 12.0;
if (p6[2] < -999.0) return 12.0;
if (p7[2] < -999.0) return 12.0;
if (p8[2] < -999.0) return 12.0;

f1 =sqrt ((p1[0] - p7[0]) * (p1[0] - p7[0]) + (p1[1] - p7[1])
        * (p1[1] - p7[1]) + (p1[2] - p7[2]) * (p1[2] - p7[2]));
f2 =sqrt ((p6[0] - p8[0]) * (p6[0] - p8[0]) + (p6[1] - p8[1])
        * (p6[1] - p8[1]) + (p6[2] - p8[2]) * (p6[2] - p8[2]));

f = sqrt ((f1 - l17) * (f1 - l17) + (f2 - l68) * (f2 - l68));
printf ("%4.2f \n", f);

return f;

}
/* A function to solve the coordinate of the tetrahedron */
void tetra (float n1[3], float n2[3], float n3[3], float d14, float d24, float d34,
                        float n4[3], int sgn)
{
float x1, x2, x3, y1, y2, y3, z1, z2, z3;
float a12, a13, b12, b13, c12, c13, d12, d13, e1, e2, f1, f2;
float a, b, c, d;

x1 = n1[0];
y1 = n1[1];
z1 = n1[2];
x2 = n2[0];
y2 = n2[1];
z2 = n2[2];
```

```
x3 = n3[0];
y3 = n3[1];
z3 = n3[2];

a12 = 2 * (x1 - x2);
a13 = 2 * (x1 - x3);
b12 = 2 * (y1 - y2);
b13 = 2 * (y1 - y3);
c12 = 2 * (z1 - z2);
c13 = 2 * (z1 - z3);
d12 = x1 * x1 - x2 * x2 + y1 * y1 - y2 * y2 + z1 * z1 - z2 * z2 - d14 * d14 + d24 * d24;
d13 = x1 * x1 - x3 * x3 + y1 * y1 - y3 * y3 + z1 * z1 - z3 * z3 - d14 * d14 + d34 * d34;
e1 = (d12 * b13 - d13 * b12) / (a12 * b13 - a13 * b12);
f1 = - (c12 * b13 - c13 * b12) / (a12 * b13 - a13 * b12);
e2 = (d12 * a13 - d13 * a12) / (b12 * a13 - b13 * a12);
f2 = - (c12 * a13 - c13 * a12) / (b12 * a13 - b13 * a12);

a = f1 * f1 + f2 * f2 + 1;
b = 2 * (e1 * f1 - x1 * f1 + e2 * f2 - y1 * f2 - z1);
c = x1 * x1 - 2 * e1 * x1 + e1 * e1 + y1 * y1 - 2 * y1 * e2 + e2 * e2 + z1 * z1 - d14 * d14;
d = b * b - 4 * a * c;

if (d >= 0.0)
{
n4[2] = ( - b + sgn * sqrt (d))/ ( 2 * a);
n4[0] = e1 + f1 * n4[2];
n4[1] = e2 + f2 * n4[2];
}
else
{
n4[0] = 0.0;
n4[1] = 0.0;
n4[2] = -1000.0;
}

}
```

Appendix 2a Determination of the 60 Octahedral-4 Geometry

```
/*****************************************************************************/
/*      Program     :  oplot.C
       Description :  A program to solve the geometry of the 60 Octahedral-4
                      framework.  This program uses functions tetra and func. */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>

/* function declaration */
void tetra (float n1[3], float n2[3], float n3[3], float d14, float d24,
                      float d34, float n4[3], int sgn);
float func (float a);

void main ()
{
FILE *f2;
float f; /* f stores the function value */
float a;  /* added link eliminates link l34 */

f2 = fopen ("octa.dat", "w");

a = 0.5;

while (a < 15.00)
{
f = func (a);
fprintf (f2, "%5.4f %5.4f 0.0 X\n", a, f);
a = a + 0.05;
}
fclose (f2);
}

/* Function Definitions */

float func (float a)
{
/* FILE *f1; */
float p1[3], p2[3], p3[3], p4[3], p5[3], p6[3];          /* Node coordinates */
float l14, l24, l25, l35, l36, l16, l45, l46, l56, f;    /* Link dimensions */
int sgn;

p1[0] = 0.0;
p1[1] = 0.0;
```

```
p1[2] = 0.0;
p2[0] = 6.0;
p2[1] = 0.0;
p2[2] = 0.0;
p3[0] = 3.0;
p3[1] = 5.1962;
p3[2] = 0.0;

l14 = 6.0;
l45 = 6.0;
l35 = 6.0;

l24 = 6.0;
l25 = 6.0;
l16 = 6.0;
l46 = 6.0;
l36 = 6.0;
l56 = 6.0;

sgn = 1;
tetra (p1, p2, p3, l14, l24, a, p4, sgn);

if (p4 [2] > -999.0)
        {
        sgn = 1;
        tetra (p3, p4, p2, l35, l45, l25, p5, sgn);

        if (p5[2] > -999.0)
                {
                sgn = 1;
                tetra (p3, p4, p5, l36, l46, l56, p6, sgn);
                }
        }

if (p4[2] < -999.0) return 10.0;
if (p5[2] < -999.0) return 10.0;
if (p6[2] < -999.0) return 10.0;

f =sqrt ((p1[0] - p6[0]) * (p1[0] - p6[0]) + (p1[1] - p6[1])
        * (p1[1] - p6[1]) + (p1[2] - p6[2]) * (p1[2] - p6[2])) ;

printf ("%f \n", f);

return sqrt ((f - l16) * (f - l16));

}

void tetra (float n1[3], float n2[3], float n3[3], float d14, float d24, float d34,
                        float n4[3], int sgn)
{
```

```
float x1, x2, x3, y1, y2, y3, z1, z2, z3;
float a12, a13, b12, b13, c12, c13, d12, d13, e1, e2, f1, f2;
float a, b, c, d;

x1 = n1[0];
y1 = n1[1];
z1 = n1[2];
x2 = n2[0];
y2 = n2[1];
z2 = n2[2];
x3 = n3[0];
y3 = n3[1];
z3 = n3[2];

a12 = 2 * (x1 - x2);
a13 = 2 * (x1 - x3);
b12 = 2 * (y1 - y2);
b13 = 2 * (y1 - y3);
c12 = 2 * (z1 - z2);
c13 = 2 * (z1 - z3);
d12 = x1 * x1 - x2 * x2 + y1 * y1 - y2 * y2 + z1 * z1 - z2 * z2 - d14 * d14 + d24 * d24;
d13 = x1 * x1 - x3 * x3 + y1 * y1 - y3 * y3 + z1 * z1 - z3 * z3 - d14 * d14 + d34 * d34;
e1 = (d12 * b13 - d13 * b12) / (a12 * b13 - a13 * b12);
f1 = - (c12 * b13 - c13 * b12) / (a12 * b13 - a13 * b12);
e2 = (d12 * a13 - d13 * a12) / (b12 * a13 - b13 * a12);
f2 = - (c12 * a13 - c13 * a12) / (b12 * a13 - b13 * a12);

a = f1 * f1 + f2 * f2 + 1;
b = 2 * (e1 * f1 - x1 * f1 + e2 * f2 - y1 * f2 - z1);
c = x1 * x1 - 2 * e1 * x1 + e1 * e1 + y1 * y1 - 2 * y1 * e2 + e2 * e2 + z1 * z1 - d14 * d14;
d = b * b - 4 * a * c;

if (d >= 0.0)
{

n4[2] = ( - b + sgn * sqrt (d))/ ( 2 * a);
n4[0] = e1 + f1 * n4[2];
n4[1] = e2 + f2 * n4[2];

}
else
{
n4[0] = 0.0;
n4[1] = 0.0;
n4[2] = -1000.0;
}

}
```

Appendix 2b Determination of Forces using the Method of Joints

```
/*****************************************************************************/
/*      Program     :  static.C
       Description :  A program to solve the forces in the members of a framework
                      using the method of joints */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>

/* function declaration */

void main()
{
FILE *f1;
float p, px, py, pz;
float a, ax, ay, az;
float b, bx, by, bz;
float c, cx, cy, cz;
float d, da, db, dc;
float norm;

f1 = fopen ("static.dat", "w");

/* Input Parameters of the truss */
printf ("Key-in the Direction of force (unit vector) \n");
printf ("x = ");
scanf ("%f",&px);
printf ("y = ");
scanf ("%f",&py);
printf ("z = ");
scanf ("%f",&pz);
norm = sqrt (px * px + py * py + pz * pz);
px = - px / norm;
py = - py / norm;
pz = - pz / norm;
printf ("Key-in Magnitude of force \n");
printf ("p = ");
scanf ("%f",&p);


fprintf (f1, "P = %6.4f (%6.4f i + %6.4f j + %6.4f k) \n", p, px, py, pz);

printf ("Key-in the Direction of link 1  \n");
printf ("x = ");
```

```
scanf ("%f",&ax);
printf ("y = ");
scanf ("%f",&ay);
printf ("z = ");
scanf ("%f",&az);
norm = sqrt (ax * ax + ay * ay + az * az);
ax = ax / norm;
ay = ay / norm;
az = az / norm;

fprintf (f1, "A = a (%6.4f i + %6.4f j + %6.4f k) \n", ax, ay, az);

printf ("Key-in the Direction of link 2 \n");
printf ("x = ");
scanf ("%f",&bx);
printf ("y = ");
scanf ("%f",&by);
printf ("z = ");
scanf ("%f",&bz);
norm = sqrt (bx * bx + by * by + bz * bz);
bx = bx / norm;
by = by / norm;
bz = bz / norm;

fprintf (f1, "B = b (%6.4f i + %6.4f j + %6.4f k) \n", bx, by, bz);

printf ("Key-in the Direction of link 2 \n");
printf ("x = ");
scanf ("%f",&cx);
printf ("y = ");
scanf ("%f",&cy);
printf ("z = ");
scanf ("%f",&cz);

norm = sqrt (cx * cx + cy * cy + cz * cz);
cx = cx / norm;
cy = cy / norm;
cz = cz / norm;

fprintf (f1, "C = c (%6.4f i + %6.4f j + %6.4f k) \n", cx, cy, cz);

printf ("P = %6.4f (%6.4f i + %6.4f j + %6.4f k) \n", p, px, py, pz);
printf ("A = a (%6.4f i + %6.4f j + %6.4f k) \n", ax, ay, az);
printf ("B = b (%6.4f i + %6.4f j + %6.4f k) \n", bx, by, bz);
printf ("C = c (%6.4f i + %6.4f j + %6.4f k) \n", cx, cy, cz);

d =  ax*(by*cz - bz*cy) -  bx*(ay*cz - az*cy) +  cx*(ay*bz - az*by);
da = p*px*(by*cz - bz*cy) - p*bx*(py*cz - pz*cy) + p*cx*(py*bz - pz*by);
db = p*ax*(py*cz - pz*cy) - p*px*(ay*cz - az*cy) + p*cx*(ay*pz - az*py);
dc = p*ax*(by*pz - bz*py) - p*bx*(ay*pz - az*py) + p*px*(ay*bz - az*by);
```

```
a = da/d;
b = db/d;
c = dc/d;

fprintf (f1, "a = %6.4f \n", a);
fprintf (f1, "b = %6.4f \n", b);
fprintf (f1, "c = %6.4f \n", c);

printf ("a = %6.4f \n", a);
printf ("b = %6.4f \n", b);
printf ("c = %6.4f \n", c);

fclose (f1);
}
```

Appendix 3a Forward Solution to the Double-Octahedral Manipulator with Positioning Specification

```
/*****************************************************************************/
/*      Program     :  Position.C
       Description :  A program to solve the forward kinematics of the double-octahedral
                      manipulator with the positioning specification. This program uses
                      functions fdvalue, fvalue, and trans. */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>

/* function declaration */
void fdvalue (float theta[], float f[], float fd[][3], float nodv[][3],
              float l1, float l2, float l3);

float fvalue (float theta[], float f[], float nodv[][3],
              float l1, float l2, float l3);

void trans (float rot[][4]);

float k1[3], k2[3], k3[3], k4[3], k5[3], k6[3], k7[3], k8[3], k9[3];

                        /* Geometric constants */
void main()
{
float xm, ym, zm;       /* End-effector location wrt moving frame */
float xe, ye, ze;       /* End-effector location wrt fixed frame */
float tran[4][4];       /* Transformation matrix */

printf ("Key-in the X location of end-effector     : ");
scanf ("%f",&xm);
printf ("Key-in the Y location of end-effector     : ");
scanf ("%f",&ym);
printf ("Key-in the Z location of end-effector     : ");
scanf ("%f",&zm);

trans (tran);
xe = tran[0][0] * xm + tran[0][1] * ym + tran[0][2] * zm + tran[0][3];
ye = tran[1][0] * xm + tran[1][1] * ym + tran[1][2] * zm + tran[1][3];
ze = tran[2][0] * xm + tran[2][1] * ym + tran[2][2] * zm + tran[2][3];
printf ("The end-effector location wrt fixed frame is {%6.4f %6.4f %6.4f} \n", xe, ye, ze);
printf ("The transformation matrix from the moving to fixed frame is : \n ");
printf ("[%6.4f %6.4f %6.4f %6.4f] \n", tran[0][0], tran[0][1], tran[0][2], tran[0][3]);
printf ("[%6.4f %6.4f %6.4f %6.4f] \n", tran[1][0], tran[1][1], tran[1][2], tran[1][3]);
printf ("[%6.4f %6.4f %6.4f %6.4f] \n", tran[2][0], tran[2][1], tran[2][2], tran[2][3]);
```

```
printf ("[%6.4f %6.4f %6.4f %6.4f] \n", tran[3][0], tran[3][1], tran[3][2], tran[3][3]);
}
```

Appendix 3b Forward Solution to the Double-Octahedral Manipulator with Gimbal Specification

```
/*******************************************************************************/
/*      Program     :  Gimbal.C
       Description :  A program to solve the forward kinematics of the double-octahedral
                      manipulator with the gimbal specification.  This program uses
                      functions fdvalue, fvalue, and trans. */
/*******************************************************************************/
#include <stdio.h>
#include <math.h>

/* function declaration */
void fdvalue (float theta[], float f[], float fd[][3], float nodv[][3],
              float l1, float l2, float l3);

float fvalue (float theta[], float f[], float nodv[][3],
              float l1, float l2, float l3);

void trans (float rot[][4]);

float k1[3], k2[3], k3[3], k4[3], k5[3], k6[3], k7[3], k8[3], k9[3];

                        /* Geometric constants */
void main()
{
float beta, theta, d;   /* Gimbal parameters */
float ux2, uy2, uz2;    /* Unit normal to the moving plane */
float x, y, z;          /* Origin of the moving frame */
float tran[4][4];       /* Transformation matrix */


trans (tran);
ux2 = tran[0][2];
uy2 = tran[1][2];
uz2 = tran[2][2];
x = tran[0][3];
y = tran[1][3];
z = tran[2][3];

beta = atan2 (sqrt (ux2 * ux2 + uy2 * uy2), uz2);
theta = atan2 (uy2, ux2);

beta = beta * 180.0 / 3.1415927;
theta = theta * 180.0 / 3.1415927;
d = sqrt (x * x + y * y + z * z);
```

```
printf ("Gimbal parameter 'beta'  in degrees = %6.4f \n", beta);
printf ("Gimbal parameter 'theta' in degrees = %6.4f \n", theta);
printf ("Gimbal parameter 'd'                = %6.4f \n", d);

printf ("The transformation matrix from the moving to fixed frame is : \n ");
printf ("[%6.4f %6.4f %6.4f %6.4f] \n", tran[0][0], tran[0][1], tran[0][2], tran[0][3]);
printf ("[%6.4f %6.4f %6.4f %6.4f] \n", tran[1][0], tran[1][1], tran[1][2], tran[1][3]);
printf ("[%6.4f %6.4f %6.4f %6.4f] \n", tran[2][0], tran[2][1], tran[2][2], tran[2][3]);
printf ("[%6.4f %6.4f %6.4f %6.4f] \n", tran[3][0], tran[3][1], tran[3][2], tran[3][3]);
}
```

Appendix 3c Forward Solution to the Quadruple-Octahedral Manipulator

```
/*****************************************************************************/
/*      Program     :  Docking.C
        Description :  A program to solve the forward kinematics of the quadruple-
                       octahedral manipulator with the docking specification.  This
                       program uses functions fdvalue, fvalue, and trans. */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>

/* function declaration */
void fdvalue (float theta[], float f[], float fd[][3], float nodv[][3],
              float l1, float l2, float l3);

float fvalue (float theta[], float f[], float nodv[][3],
              float l1, float l2, float l3);

void trans (float rot[][4]);

float k1[3], k2[3], k3[3], k4[3], k5[3], k6[3], k7[3], k8[3], k9[3];
                        /* Geometric constants */
void main()
{
float tran[4][4];       /* Final Transformation matrix */
float tran1[4][4];      /* First Transformation matrix */
float tran2[4][4];      /* Second Transformation matrix */

int i, j;               /* Counters */

printf ("Key-in dimensions for the first double-octahedral manipulator\n");
trans (tran1);
printf ("Key-in dimensions for the second double-octahedral manipulator\n");
trans (tran2);

for (i = 0 ; i <= 3 ; ++i)
        for (j = 0 ; j <= 3 ; ++j)
        tran[i][j] = tran1[i][0] * tran2[0][j] + tran1[i][1] * tran2[1][j] +
                     tran1[i][2] * tran2[2][j] + tran1[i][3] * tran2[3][j];
printf ("The transformation matrix from the moving to fixed frame is : \n ");
printf ("[%6.4f %6.4f %6.4f %6.4f] \n", tran[0][0], tran[0][1], tran[0][2], tran[0][3]);
printf ("[%6.4f %6.4f %6.4f %6.4f] \n", tran[1][0], tran[1][1], tran[1][2], tran[1][3]);
printf ("[%6.4f %6.4f %6.4f %6.4f] \n", tran[2][0], tran[2][1], tran[2][2], tran[2][3]);
printf ("[%6.4f %6.4f %6.4f %6.4f] \n", tran[3][0], tran[3][1], tran[3][2], tran[3][3]);
}
```

Appendix 3d Forward Solution to the Double-Octahedral Manipulator

```
/*****************************************************************************/
        Function      : trans
        Description   : A function to determine the transformation matrix from the moving
                        frame to the fixed frame */
/*****************************************************************************/

void trans (float rot[][4])
{
FILE *fl;
float nodf[3][3] ;        /* Nodes of the fixed triangle */
float nodm[3][3] ;        /* Nodes of the moving triangle */
float nodv[3][3] ;        /* Nodes of the offset plane */
float orig[3][3] ;        /* Revolute joint origins */
float refv[3][3] ;        /* Reference vectors (R-S pair on the fixed triangle) */
float urev[3][3] ;        /* Unit vectors along the revolute joints */
float l1, l2, l3;         /* Variable-member lengths */
float lb, ll ;            /* Batten length, longeron length */
float lrs ;               /* Length of the R-S pair */
float f[3], fd[3][3];     /* Function and its derivative wrt theta */
float fv;                 /* Function value */
float nfv;                /* New function value */
float theta [3];          /* Rotation angle of the RS pair */
float ctheta [3];         /* Change in theta value */
float thetan [3];         /* New theta value */
float x, y, z;            /* Origin of the moving frame */
float xe, ye, ze;         /* Output parameter (end effector location) */
float ux2, uy2, uz2;      /* Unit normal to the moving plane */
float ux1, uy1, uz1;      /* Unit normal to the mid-plane */
float ux0, uy0, uz0;      /* Unit normal to the fixed plane */
float un1, un2;           /* normalizing factor */
float v1[3], v2[3];       /* vectors on the offset plane */
float s;                  /* Joint Offset */
float n, r, d;            /* Scalar distances */
float rod;                /* Quasi-Newton method ratio parameter */

int i, j, ierror;         /* Indexes and switches */


fl = fopen ("position.dat", "w");

/* Input Parameters of the truss */
printf ("Key-in the Batten Length 'lb'      : ");
scanf ("%f",&lb);
fprintf (fl, "Batten Length  'lb'   = %4.2f \n", lb);
```

```
printf ("Key-in the Longeron Length 'll'     : ");
scanf ("%f",&ll);
fprintf (f1, "Longeron Length 'll'   = %4.2f \n", ll);
printf ("Key-in the Joint Offset 's'         : ");
scanf ("%f",&s);
fprintf (f1, "Longeron Length 's'    = %4.2f \n", s);


/* Definition of the fixed triangle nodes */
nodf[0][0] = -lb/2.0 ;
nodf[0][1] = -lb/3.4641016 ;
nodf[0][2] = 0.0 ;
nodf[1][0] = lb/2.0 ;
nodf[1][1] = -lb/3.4641016 ;
nodf[1][2] = 0.0 ;
nodf[2][0] = 0.0 ;
nodf[2][1] = lb/1.7320508 ;
nodf[2][2] = 0.0 ;

/* Definition of revolute joint origins */
for (i = 0 ; i <= 2 ; ++i)
orig[0][i] = (nodf[0][i] + nodf[1][i])/2.0 ;
for (i = 0 ; i <= 2 ; ++i)
orig[1][i] = (nodf[1][i] + nodf[2][i])/2.0 ;
for (i = 0 ; i <= 2 ; ++i)
orig[2][i] = (nodf[2][i] + nodf[0][i])/2.0 ;

/* Definition of unit vectors along the revolute joints */
for (i = 0 ; i <= 2 ; ++i)
urev[0][i] = (nodf[0][i] - nodf[1][i]) / lb ;
for (i = 0 ; i <= 2 ; ++i)
urev[1][i] = (nodf[1][i] - nodf[2][i]) / lb ;
for (i = 0 ; i <= 2 ; ++i)
urev[2][i] = (nodf[2][i] - nodf[0][i]) / lb ;

/* Length of the R-S pair */
lrs = sqrt (4 * ll * ll - lb * lb) / 2.0 ;

/* Reference Vectors */
refv[0][0] = 0.0 ;
refv[0][1] = -lrs ;
refv[0][2] = 0.0 ;
refv[1][0] = 0.8660254 * lrs ;
refv[1][1] = 0.5 * lrs ;
refv[1][2] = 0.0 ;
refv[2][0] = -0.8660254 * lrs ;
refv[2][1] = 0.5 * lrs ;
refv[2][2] = 0.0 ;

/* Unit normal to the fixed plane */
```

```
ux0 = 0.0;
uy0 = 0.0;
uz0 = 1.0;

for (i = 0 ; i <= 2 ; ++i)
{
k7[i] = refv[i][0] * urev[i][0] * urev[i][0] + refv[i][1] * urev[i][1] *
                    urev[i][0] + refv[i][2] * urev[i][2] * urev[i][0] + orig[i][0];
k8[i] = refv[i][1] * urev[i][1] * urev[i][1] + refv[i][0] * urev[i][0] *
                    urev[i][1] + refv[i][2] * urev[i][2] * urev[i][1] + orig[i][1];
k9[i] = refv[i][2] * urev[i][2] * urev[i][2] + refv[i][0] * urev[i][0] *
                    urev[i][2] + refv[i][1] * urev[i][1] * urev[i][2] + orig[i][2];
k1[i] = refv[i][0] + orig[i][0] - k7[i];
k2[i] = refv[i][1] + orig[i][1] - k8[i];
k3[i] = refv[i][2] + orig[i][2] - k9[i];
k4[i] = refv[i][2] * urev[i][1] - refv[i][1] * urev[i][2];
k5[i] = refv[i][0] * urev[i][2] - refv[i][2] * urev[i][0];
k6[i] = refv[i][1] * urev[i][0] - refv[i][0] * urev[i][1];
}


/* Input parameters to the positioning problem */
printf ("Key-in the length of actuator 1 'l1': ");
scanf ("%f",&l1);
fprintf (f1, "Actuator Length l1     = %4.2f \n", l1);
printf ("Key-in the length of actuator 2 'l2': ");
scanf ("%f",&l2);
fprintf (f1, "Actuator Length l2     = %4.2f \n", l2);
printf ("Key-in the length of actuator 3 'l3': ");
scanf ("%f",&l3);
fprintf (f1, "Actuator Length l3     = %4.2f \n", l3);

/* Assume theta */

theta [0] = 0.785398;
theta [1] = 0.785398;
theta [2] = 0.785398;

/* Compute the function and its derivative */
/* Loop to find the solution */
fv = fvalue (theta, f, nodv, l1, l2, l3);
while (fv > 0.0001)
        {
        fdvalue (theta, f, fd, nodv, l1, l2, l3);

        /* solve for change in theta */
        ctheta [2] = ((-f[1] * fd[2][0] + f[2] * fd[1][0]) * fd[0][1] +
                f[0] * fd[1][0]) / (fd[1][2] * fd[2][0] * fd[0][2] +
                fd[0][2] * fd[1][0]);
        ctheta [1] = (-f[0] - fd[0][2] * ctheta [2]) / fd[0][1];
```

```
        ctheta [0] = (-f[2] - fd[2][1] * ctheta [1]) / fd[2][0];
/* Do line search */

        rod = 1.0;
        for (i = 0; i <= 2; ++i)
        thetan[i] = theta[i] + rod * ctheta[i];
        nfv = fvalue (thetan, f, nodv, l1, l2, l3);
        while ((fv < nfv) && (rod > 0.2))
                {
                rod = rod / 2.0;
                for (i = 0; i <= 2; ++i)
                thetan[i] = theta[i] + rod * ctheta[i];
                nfv = fvalue (thetan, f, nodv, l1, l2, l3);
                }

        for (i = 0; i <= 2; ++i)
        {
        theta [i] = thetan[i];
        if ((theta[i] < 0.0) || (theta [i] > 3.1415927))
        theta [i] = 0.785398;
        }
        fv = nfv;

/* without line search  */
/*
        for (i = 0; i <= 2; ++i)
        theta [i] = theta [i] + ctheta [i];
        fv = fvalue (theta, f, nodv, l1, l2, l3);
*/
        }

fv = fvalue (theta, f, nodv, l1, l2, l3);

for (i = 0; i <= 2; ++i)
{
v1[i] = nodv[0][i] - nodv[1][i];
v2[i] = nodv[0][i] - nodv[2][i];
}

ux1 = v1[1] * v2[2] - v2[1] * v1[2];
uy1 = v2[0] * v1[2] - v1[0] * v2[2];
uz1 = v1[0] * v2[1] - v2[0] * v1[1];
un1 = sqrt (ux1 * ux1 + uy1 * uy1 + uz1 * uz1);
ux1 = ux1 / un1;
uy1 = uy1 / un1;
uz1 = uz1 / un1;

n = 2.0 * (nodv[0][0] * ux1 + nodv[0][1] * uy1 + nodv[0][2] * uz1);
d = n + s;
r = d / ( 2.0 * (ux0 * ux1 + uy0 * uy1 + uz0 * uz1));
```

```
x = d * ux1;
y = d * uy1;
z = d * uz1;

ux2 = d * ux1 / r - ux0;
uy2 = d * uy1 / r - uy0;
uz2 = d * uz1 / r - uz0;
un2 = sqrt (ux2 * ux2 + uy2 * uy2 + uz2 * uz2);
ux2 = ux2 / un2;
uy2 = uy2 / un2;
uz2 = uz2 / un2;

printf ("Origin of moving plane  %6.4f %6.4f %6.4f \n", x, y, z);
fprintf (f1, "Origin of moving plane  %6.4f %6.4f %6.4f \n", x, y, z);

/* compute the transformation matrix */
rot[0][0] = 1 - ux2 * ux2 / ( 1.0 + uz2);
rot[0][1] = - ux2 * uy2 / (1.0 + uz2);
rot[0][2] = ux2;
rot[0][3] = x;
rot[1][0] = - ux2 * uy2 / (1.0 + uz2);
rot[1][1] = 1 - uy2 * uy2 / ( 1.0 + uz2);
rot[1][2] = uy2;
rot[1][3] = y;
rot[2][0] = - ux2;
rot[2][1] = - uy2;
rot[2][2] = uz2;
rot[2][3] = z;
rot[3][0] = 0.0;
rot[3][1] = 0.0;
rot[3][2] = 0.0;
rot[3][3] = 1.0;

fclose (f1);
}

/* Function to calculate the Jacobian matrix */

void fdvalue (float theta [], float f[], float fd[][3], float nodv[][3],
              float l1, float l2, float l3)

{
float fv;
int i;


fv = fvalue (theta, f, nodv, l1, l2, l3);

fd[0][0] = 0.0;
fd[0][1] = ( (nodv[1][0] - nodv[2][0]) * (-k1[1] * sin (theta[1])
```

```
            + k4[1] * cos (theta [1]))
           + (nodv[1][1] - nodv[2][1]) * (-k2[1] * sin (theta[1])
            + k5[1] * cos (theta [1]))
           + (nodv[1][2] - nodv[2][2]) * (-k3[1] * sin (theta[1])
            + k6[1] * cos (theta [1]))) / (f[0] + l1);
fd[0][2] = ( (nodv[1][0] - nodv[2][0]) * (k1[2] * sin (theta[2])
            - k4[2] * cos (theta [2]))
           + (nodv[1][1] - nodv[2][1]) * (k2[2] * sin (theta[2])
            - k5[2] * cos (theta [2]))
           + (nodv[1][2] - nodv[2][2]) * (k3[2] * sin (theta[2])
            - k6[2] * cos (theta [2]))) / (f[0] + l1);

fd[1][0] = ( (nodv[0][0] - nodv[2][0]) * (-k1[0] * sin (theta[0])
            + k4[0] * cos (theta [0]))
           + (nodv[0][1] - nodv[2][1]) * (-k2[0] * sin (theta[0])
            + k5[0] * cos (theta [0]))
           + (nodv[0][2] - nodv[2][2]) * (-k3[0] * sin (theta[0])
            + k6[0] * cos (theta [0]))) / (f[1] + l1);
fd[1][1] = 0.0;
fd[1][2] = ( (nodv[0][0] - nodv[2][0]) * (k1[2] * sin (theta[2])
            - k4[2] * cos (theta [2]))
           + (nodv[0][1] - nodv[2][1]) * (k2[2] * sin (theta[2])
            - k5[2] * cos (theta [2]))
           + (nodv[0][2] - nodv[2][2]) * (k3[2] * sin (theta[2])
            - k6[2] * cos (theta [2]))) / (f[1] + l1);

fd[2][0] = ( (nodv[1][0] - nodv[0][0]) * (k1[0] * sin (theta[0])
            - k4[2] * cos (theta [2]))
           + (nodv[1][1] - nodv[0][1]) * (k2[0] * sin (theta[0])
            - k5[0] * cos (theta [0]))
           + (nodv[1][2] - nodv[0][2]) * (k3[0] * sin (theta[0])
            - k6[0] * cos (theta [0]))) / (f[2] + l1);
fd[2][1] = ( (nodv[1][0] - nodv[0][0]) * (-k1[1] * sin (theta[1])
            + k4[1] * cos (theta [1]))
           + (nodv[1][1] - nodv[0][1]) * (-k2[1] * sin (theta[1])
            + k5[1] * cos (theta [1]))
           + (nodv[1][2] - nodv[0][2]) * (-k3[1] * sin (theta[1])
            + k6[1] * cos (theta [1]))) / (f[2] + l1);
fd[2][2] = 0.0;
}

/* A function to generate the function values */

float fvalue (float theta [], float f[], float nodv[][3],
              float l1, float l2, float l3)

{
int i;

for (i = 0 ; i <= 2 ; ++i)
```

```
        {
        nodv[i][0] = k1[i] * cos (theta [i]) + k4[i] * sin (theta [i]) + k7[i];
        nodv[i][1] = k2[i] * cos (theta [i]) + k5[i] * sin (theta [i]) + k8[i];
        nodv[i][2] = k3[i] * cos (theta [i]) + k6[i] * sin (theta [i]) + k9[i];
        }

/* compute link lengths as function values */
f[0] = sqrt ((nodv [1][0] - nodv [2][0]) * (nodv [1][0] - nodv [2][0]) +
            (nodv [1][1] - nodv [2][1]) * (nodv [1][1] - nodv [2][1]) +
            (nodv [1][2] - nodv [2][2]) * (nodv [1][2] - nodv [2][2])) - l1;
f[1] = sqrt ((nodv [0][0] - nodv [2][0]) * (nodv [0][0] - nodv [2][0]) +
            (nodv [0][1] - nodv [2][1]) * (nodv [0][1] - nodv [2][1]) +
            (nodv [0][2] - nodv [2][2]) * (nodv [0][2] - nodv [2][2])) - l2;
f[2] = sqrt ((nodv [1][0] - nodv [0][0]) * (nodv [1][0] - nodv [0][0]) +
            (nodv [1][1] - nodv [0][1]) * (nodv [1][1] - nodv [0][1]) +
            (nodv [1][2] - nodv [0][2]) * (nodv [1][2] - nodv [0][2])) - l3;
return sqrt (f[0] * f[0] + f[1] * f[1] + f[2] * f[2]);
}
```

Appendix 4a Inverse Solution to the Positioning Problem

```
/*****************************************************************************/
/*      Program     :  Position.C
       Description :  A program to solve the Inverse kinematics of the double-
                      octahedral manipulator with the position specification.  This
                      program uses funcation solve */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>

/* function declaration */

void solve (float x, float y, float z, float lb, float ll, float s,
          float l[]);

float k1[3], k2[3], k3[3], k4[3], k5[3], k6[3], k7[3], k8[3], k9[3];
                        /* Geometric constants */

void main()
{
FILE *f1;

float x, y, z;          /* origin of the moving frame */
float xe, ye, ze;       /* Location of the end-effector */
float lb, ll, s;        /* Batten length, longeron length, joint offset */
float l[3];             /* Actuator lengths */
float vr[3];            /* Reflected vector */
float fact;             /* Multiplication factor */

f1 = fopen ("position.dat", "w");

/* The end-effector is assumed to be attached normal to the moving plane
and 10 units long, therefore the reflected vector is */
vr[0] = 0.0;
vr[1] = 0.0;
vr[2] = -10.0;

printf ("Key-in x value          : ");
scanf ("%f", &xe);
fprintf (f1, "X value          = %6.4f \n",xe);

printf ("Key-in y value          : ");
scanf ("%f", &ye);
fprintf (f1, "Y value          = %6.4f \n",ye);
```

```
printf ("Key-in z value              : ");
scanf ("%f", &ze);
fprintf (fl, "Z value              =  %6.4f \n",ze);

printf ("Key-in batten length 'lb'   : ");
scanf ("%f", &lb);
fprintf (fl, "Batten length 'lb'   =  %6.4f \n",lb);

printf ("Key-in longeron length 'll' : ");
scanf ("%f", &ll);
fprintf (fl, "Longeron length 'll' =  %6.4f \n",ll);

printf ("Key-in joint offset 's'     : ");
scanf ("%f", &s);
fprintf (fl, "Joint offset 's'     =  %6.4f \n",s);

fact = ((xe + vr[0]) * (xe - vr[0]) + (ye + vr[1]) * (ye - vr[1]) +
      (ze + vr[2]) * (ze - vr[2])) /
      ((xe - vr[0]) * (xe - vr[0]) + (ye - vr[1]) * (ye - vr[1]) +
      (ze - vr[2]) * (ze - vr[2]));

x = (xe - vr[0]) * fact;
y = (ye - vr[1]) * fact;
z = (ze - vr[2]) * fact;

solve (x, y, z, lb, ll, s, l);

fprintf (fl, "link lengths = %6.4f %6.4f %6.4f \n", l[0], l[1], l[2]);
fclose (fl);
}
```

Appendix 4b Inverse Solution to the Gimbal Problem

```
/*****************************************************************************/
/*      Program     :  Gimbal.C
       Description :  A program to solve the Inverse kinematics of the double-
                      octahedral manipulator with the gimbal specification.  This
                      program uses the function solve */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>

/* function declaration */

void solve (float x, float y, float z, float lb, float ll, float s,
            float l[]);

float k1[3], k2[3], k3[3], k4[3], k5[3], k6[3], k7[3], k8[3], k9[3];
                        /* Geometric constants */

void main()
{
FILE *f1;

float x, y, z;          /* origin of the moving frame */
float beta, theta, d, r; /* Gimbal Parameters */
float lb, ll, s;        /* Batten length, longeron length, joint offset */
float l[3];             /* Actuator lengths */
float u2[3];            /* unit normal to the moving plane */
float u0[3];            /* unit normal ot the fixed plane */

f1 = fopen ("position.dat", "w");


printf ("Key-in beta (in degrees)    : ");
scanf ("%f", &beta);
beta = beta * 3.1415927 / 180.0;
fprintf (f1, "beta              = %6.4f \n",beta);

printf ("Key-in theta (in degrees)   : ");
scanf ("%f", &theta);
theta = theta * 3.1415927 / 180.0;
fprintf (f1, "theta             = %6.4f \n",theta);

printf ("Key-in d value              : ");
scanf ("%f", &d);
fprintf (f1, "d value           = %6.4f \n",d);
```

```
printf ("Key-in batten length 'lb'   : ");
scanf ("%f", &lb);
fprintf (f1, "Batten length 'lb'   =  %6.4f \n",lb);

printf ("Key-in longeron length 'll' : ");
scanf ("%f", &ll);
fprintf (f1, "Longeron length 'll' =  %6.4f \n",ll);

printf ("Key-in joint offset 's'     : ");
scanf ("%f", &s);
fprintf (f1, "Joint offset 's'     =  %6.4f \n",s);

u2[0] = cos (theta) * sin (beta);
u2[1] = sin (theta) * sin (beta);
u2[2] = cos (beta);

/* By definition of the fixed frame */
u0[0] = 0.0;
u0[1] = 0.0;
u0[2] = 1.0;

r = d / sqrt (2 * (1 + u2[0] * u0[0] + u2[1] * u0[1] + u2[2] * u0[2]));

x = r * (u2[0] + u0[0]);
y = r * (u2[1] + u0[1]);
z = r * (u2[2] + u0[2]);

solve (x, y, z, lb, ll, s, l);

fprintf (f1, "link lengths = %6.4f %6.4f %6.4f \n", l[0], l[1], l[2]);
fclose (f1);
}
```

Appendix 4c Inverse Solution to the Docking Problem

```
/*****************************************************************************/
/*      Program     :  Docking.C
       Description :  A program to solve the Inverse kinematics of the quadruple-
                      octahedral manipulator with the docking specification. This
                      program uses the function solve */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>

/* function declaration */

void solve (float x, float y, float z, float lb, float ll, float s,
            float l[]);

float k1[3], k2[3], k3[3], k4[3], k5[3], k6[3], k7[3], k8[3], k9[3];
                        /* Geometric constants */

void main()
{
FILE *f1;

float x, y, z;          /* Origin of the moving frame (double-octahedron)*/
float xe, ye, ze;       /* Specified End-point (centroid of the top plane)*/
float beta, theta;      /* Orientation of the moving plane */
float c;                /* ratio of heights (free parameter) */
float lb, ll, s;        /* Batten length, longeron length, joint offset */
float l[3];             /* Actuator lengths */
float k1, k2, k3, k4;   /* Computational variables */
float u0[3];            /* Unit normal to plane 0 */
float u2[3];            /* Unit normal to plane 2 */
float u4[3];            /* Unit normal to plane 4 */
float u42[3];           /* Unit normal to plane 4 wrt 2 */
float r1, r2;           /* Unit cell heights in the two bays */
float rot[3][3];        /* Rotational part of the transformation matrix */


f1 = fopen ("docking.dat", "w");


printf ("Key-in x value          : ");
scanf ("%f", &xe);
fprintf (f1, "X value          =  %6.4f \n",xe);

printf ("Key-in y value          : ");
```

```
scanf ("%f", &ye);
fprintf (f1, "Y value            = %6.4f \n",ye);

printf ("Key-in z value            : ");
scanf ("%f", &ze);
fprintf (f1, "Z value            = %6.4f \n",ze);

printf ("Key-in beta (in degrees)    : ");
scanf ("%f", &beta);
beta = beta / 180.0 * 3.1415927;
fprintf (f1, "beta               = %6.4f \n",beta);

printf ("Key-in theta (in degrees)   : ");
scanf ("%f", &theta);
theta = theta / 180.0 * 3.1415927;
fprintf (f1, "theta              = %6.4f \n",theta);

printf ("Key-in c value            : ");
scanf ("%f", &c);
fprintf (f1, "c value            = %6.4f \n",c);

printf ("Key-in batten length 'lb'   : ");
scanf ("%f", &lb);
fprintf (f1, "Batten length 'lb'   = %6.4f \n",lb);

printf ("Key-in longeron length 'll' : ");
scanf ("%f", &ll);
fprintf (f1, "Longeron length 'll' =  %6.4f \n",ll);

printf ("Key-in joint offset 's'     : ");
scanf ("%f", &s);
fprintf (f1, "Joint offset 's'     = %6.4f \n",s);

u0[0] = 0.0;
u0[1] = 0.0;
u0[2] = 1.0;

u4[0] = cos (theta) * sin (beta);
u4[1] = sin (theta) * sin (beta);
u4[2] = cos (beta);

k1 = 2.0 * c * (u0[0] * u4[0] + u0[1] * u4[1] + u0[2] * u4[2] - 1.0);
k2 = -2.0 * ((c * u0[0] + u4[0]) * xe + (c * u0[1] + u4[1]) * ye +
            (c * u0[2] + u4[2]) * ze);
k3 = xe * xe + ye * ye + ze * ze;
k4 = k2 * k2 - 4.0 * k1 * k3;

if (k4 >= 0)
{
if (k1 == 0)
```

```
r2 = -k3 / k2;
else
r2 = (-k2 - sqrt (k4)) / (2.0 * k1);

r1 = c * r2;

u2[0] = xe / (r2 * (c + 1.0)) - (c * u0[0] + u4[0]) / (c + 1.0);
u2[1] = ye / (r2 * (c + 1.0)) - (c * u0[1] + u4[1]) / (c + 1.0);
u2[2] = ze / (r2 * (c + 1.0)) - (c * u0[2] + u4[2]) / (c + 1.0);

x = r1 * (u0[0] + u2[0]);
y = r1 * (u0[1] + u2[1]);
z = r1 * (u0[2] + u2[2]);

solve (x, y, z, lb, ll, s, l);

fprintf (f1, "l1 = %6.4f l2 = %6.4f l3 = %6.4f \n", l[0], l[1], l[2]);

rot[0][0] = 1.0 - u2[0] * u2[0] / (1.0 + u2[2]);
rot[0][1] = - u2[0] * u2[1] / (1.0 + u2[2]);
rot[0][2] = u2[0];

rot[1][0] = - u2[0] * u2[1] / (1.0 + u2[2]);
rot[1][1] = 1.0 - u2[1] * u2[1] / (1.0 + u2[2]);
rot[1][2] = u2[1];

rot[2][0] = -u2[0];
rot[2][1] = -u2[1];
rot[2][2] = u2[2];

/* Multiplying u4 with the transpose of this matrix yields the required
vector */
u42[0] = rot[0][0] * u4[0] + rot[1][0] * u4[1] + rot[2][0] * u4[2];
u42[1] = rot[0][1] * u4[0] + rot[1][1] * u4[1] + rot[2][1] * u4[2];
u42[2] = rot[0][2] * u4[0] + rot[1][2] * u4[1] + rot[2][2] * u4[2];

x = r2 * (u0[0] + u42[0]);
y = r2 * (u0[1] + u42[1]);
z = r2 * (u0[2] + u42[2]);

solve (x, y, z, lb, ll, s, l);

fprintf (f1, "l4 = %6.4f l5 = %6.4f l6 = %6.4f \n", l[0], l[1], l[2]);
}
else
printf ("The problem has no solution \n");
fclose (f1);
}
```

Appendix 4d Inverse Solution to the Moving Frame Origin Specification

```
/*****************************************************************************/
/*      Function    : solve
       Description  : A function to solve the inverse problem of the double-octahedral
                      manipulator given the origin of the moving frame. */
/*****************************************************************************/

void solve (float x, float y, float z, float lb, float ll, float s,
          float l[])

{
float nodf[3][3] ;        /* Nodes of the fixed triangle */
float nodv[3][3] ;        /* Offset plane nodes */
float orig[3][3] ;        /* Revolute joint origins */
float refv[3][3] ;        /* Reference vectors (R-S pair on the fixed triangle) */
float urev[3][3] ;        /* Unit vectors along the revolute joints */
float lrs ;               /* Length of the R-S pair */
float d;                  /* Distance between the origins */
int i, ierror;            /* Counter, Error switch */
float a1[3], a2[3], a3[3], a4[3];
                          /* Other computational variables */
float ct[3], st[3];       /* Cosine and sine of the RS pair angle */

/* Definition of the fixed triangle nodes */

nodf[0][0] = -lb/2.0 ;
nodf[0][1] = -lb/3.4641016 ;
nodf[0][2] = 0.0 ;
nodf[1][0] = lb/2.0 ;
nodf[1][1] = -lb/3.4641016 ;
nodf[1][2] = 0.0 ;
nodf[2][0] = 0.0 ;
nodf[2][1] = lb/1.7320508 ;
nodf[2][2] = 0.0 ;

/* Definition of revolute joint origins */

for (i = 0 ; i <= 2 ; ++i)
orig[0][i] = (nodf[0][i] + nodf[1][i])/2.0 ;
for (i = 0 ; i <= 2 ; ++i)
orig[1][i] = (nodf[1][i] + nodf[2][i])/2.0 ;
for (i = 0 ; i <= 2 ; ++i)
orig[2][i] = (nodf[2][i] + nodf[0][i])/2.0 ;

/* Definition of unit vectors along the revolute joints */
```

```
for (i = 0 ; i <= 2 ; ++i)
urev[0][i] = (nodf[0][i] - nodf[1][i]) / lb ;
for (i = 0 ; i <= 2 ; ++i)
urev[1][i] = (nodf[1][i] - nodf[2][i]) / lb ;
for (i = 0 ; i <= 2 ; ++i)
urev[2][i] = (nodf[2][i] - nodf[0][i]) / lb ;

/* Length of the R-S pair */

lrs = sqrt (4 * ll * ll - lb * lb) / 2.0 ;

/* Reference Vectors */

refv[0][0] = 0.0 ;
refv[0][1] = -lrs ;
refv[0][2] = 0.0 ;
refv[1][0] = 0.8660254 * lrs ;
refv[1][1] = 0.5 * lrs ;
refv[1][2] = 0.0 ;
refv[2][0] = -0.8660254 * lrs ;
refv[2][1] = 0.5 * lrs ;
refv[2][2] = 0.0 ;

for (i = 0 ; i <= 2 ; ++i)
{
k7[i] = refv[i][0] * urev[i][0] * urev[i][0] + refv[i][1] * urev[i][1] *
                    urev[i][0] + refv[i][2] * urev[i][2] * urev[i][0] + orig[i][0];
k8[i] = refv[i][1] * urev[i][1] * urev[i][1] + refv[i][0] * urev[i][0] *
                    urev[i][1] + refv[i][2] * urev[i][2] * urev[i][1] + orig[i][1];
k9[i] = refv[i][2] * urev[i][2] * urev[i][2] + refv[i][0] * urev[i][0] *
                    urev[i][2] + refv[i][1] * urev[i][1] * urev[i][2] + orig[i][2];
k1[i] = refv[i][0] + orig[i][0] - k7[i];
k2[i] = refv[i][1] + orig[i][1] - k8[i];
k3[i] = refv[i][2] + orig[i][2] - k9[i];
k4[i] = refv[i][2] * urev[i][1] - refv[i][1] * urev[i][2];
k5[i] = refv[i][0] * urev[i][2] - refv[i][2] * urev[i][0];
k6[i] = refv[i][1] * urev[i][0] - refv[i][0] * urev[i][1];
}

d = sqrt (x * x + y * y + z * z);
for (i = 0 ; i <= 2 ; ++i)
        {
        a1[i] = k1[i] * x + k2[i] * y + k3[i] * z;
        a2[i] = k4[i] * x + k5[i] * y + k6[i] * z;
        a3[i] = k7[i] * x + k8[i] * y + k9[i] * z - d * d / 2.0 + s * d / 2.0;
        a4[i] = a1[i] * a1[i] + a2[i] * a2[i] - a3[i] * a3[i];
        }
ierror = 0;
for (i = 0; i<= 2; ++i)
        if (a4[i] >= 0)
```

```
        {
        ct[i] = (- a1[i]* a3[i] + a2[i] * sqrt (a4[i]))/(a1[i]*a1[i] + a2[i]*a2[i]);
        st[i] = (- a2[i]* a3[i] - a1[i] * sqrt (a4[i]))/(a1[i]*a1[i] + a2[i]*a2[i]);
        nodv[i][0] = k1[i] * ct[i] + k4[i] * st[i] + k7[i];
        nodv[i][1] = k2[i] * ct[i] + k5[i] * st[i] + k8[i];
        nodv[i][2] = k3[i] * ct[i] + k6[i] * st[i] + k9[i];

        }
        else
        ierror = 1;
if (ierror == 0)
        {
        l[0] = sqrt ((nodv[2][0] - nodv[1][0]) * (nodv[2][0] - nodv[1][0])
                + (nodv[2][1] - nodv[1][1]) * (nodv[2][1] - nodv[1][1])
                + (nodv[2][2] - nodv[1][2]) * (nodv[2][2] - nodv[1][2]));

        l[1] = sqrt ((nodv[0][0] - nodv[2][0]) * (nodv[0][0] - nodv[2][0])
                + (nodv[0][1] - nodv[2][1]) * (nodv[0][1] - nodv[2][1])
                + (nodv[0][2] - nodv[2][2]) * (nodv[0][2] - nodv[2][2]));

        l[2] = sqrt ((nodv[1][0] - nodv[0][0]) * (nodv[1][0] - nodv[0][0])
                + (nodv[1][1] - nodv[0][1]) * (nodv[1][1] - nodv[0][1])
                + (nodv[1][2] - nodv[0][2]) * (nodv[1][2] - nodv[0][2]));

        printf ("link lengths = %6.4f %6.4f %6.4f \n", l[0], l[1], l[2]);
        }
else
        printf ("There is no solution. \n");

}
```

Appendix 5a Jacobian Matrix Determination for the Positioning Problem

```
/*****************************************************************************/
/*      Program     : Position
        Description : A Program to obtain the Velocity Jacobian to the positioning
                      specification for the double-octahedral manipulator in closed
                      -form.  Function includes dot and solve given in Appendix 4d.
/*****************************************************************************/

#include <stdio.h>
#include <math.h>

/* function declaration */

float dot (float v1[], float v2[]);

void solve (float x, float y, float z, float lb, float ll, float s,
            float l[]);

float k1[3], k2[3], k3[3], k4[3], k5[3], k6[3], k7[3], k8[3], k9[3];
                                        /* Geometric constants */
float a1[3], a2[3], a3[3], a4[3];       /* Other computational variables */
float b[3][3];                          /* Offset Plane nodes */

void main()
{
FILE *f1;

float p[3];             /* origin of the moving frame */
float pe[3];            /* Location of the end-effector */
float lb, ll, s;        /* Batten length, longeron length, joint offset */
float l[3];             /* Actuator lengths */
float vr[3];            /* Reflected vector */
float pemvr[3], pepvr[3];  /* Vectors used for intermediate calculations */
float fact, d;          /* Multiplication factor */

float dpe[3][3];        /* Partial derivatives of the end-effector position vector */
float dp[3][3];         /* Partial derivatives of the origin of the moving plane */
float da1[3][3], da2[3][3], da3[3][3];
                        /* Partial derivatives of the variable a */
float db1[3][3], db2[3][3], db3[3][3];
                        /* Partial derivatives of the mid-plane nodes */
float bn[3][3];         /* Mid-plane vectors */
float jacoba[3][3];     /* Jacobian [A] matrix */
float jacobb[3][3];     /* Jacobian [B] matrix */
float dcos[3][3], dsin[3][3];
```

```
                    /* Partial derivative to the cosine and sine of the angles */

int i, j;

fl = fopen ("position.dat", "w");

/* The end-effector is assumed to be attached normal to the moving plane
and 10 units long, therefore the reflected vector is */
vr[0] = 0.0;
vr[1] = 0.0;
vr[2] = -10.0;

fprintf (fl, "vr %6.4f %6.4f %6.4f \n", vr[0], vr[1], vr[2]);

printf ("Key-in x value          : ");
scanf ("%f", &pe[0]);
fprintf (fl, "X value          = %6.4f \n",pe[0]);

printf ("Key-in y value          : ");
scanf ("%f", &pe[1]);
fprintf (fl, "Y value          = %6.4f \n",pe[1]);

printf ("Key-in z value          : ");
scanf ("%f", &pe[2]);
fprintf (fl, "Z value          = %6.4f \n",pe[2]);

printf ("Key-in batten length 'lb'  : ");
scanf ("%f", &lb);
fprintf (fl, "Batten length 'lb'  = %6.4f \n",lb);

printf ("Key-in longeron length 'll' : ");
scanf ("%f", &ll);
fprintf (fl, "Longeron length 'll' = %6.4f \n",ll);

printf ("Key-in joint offset 's'    : ");
scanf ("%f", &s);
fprintf (fl, "Joint offset 's'    = %6.4f \n",s);

fprintf (fl, "pe %6.4f %6.4f %6.4f \n", pe[0], pe[1], pe[2]);

for (i = 0 ; i <= 2 ; ++i)
{
pemvr[i] = pe[i] - vr[i];
pepvr[i] = pe[i] + vr[i];
}

fact = dot (pepvr, pemvr) / dot (pemvr, pemvr);

for (i = 0 ; i <= 2 ; ++i)
p[i] = (pe[i] - vr[i]) * fact;
```

```
solve (p[0], p[1], p[2], lb, ll, s, l);

fprintf (f1, "link lengths = %6.4f %6.4f %6.4f \n", l[0], l[1], l[2]);

jacobb[0][0] = l[0];
jacobb[0][1] = 0.0;
jacobb[0][2] = 0.0;
jacobb[1][0] = 0.0;
jacobb[1][1] = l[1];
jacobb[1][2] = 0.0;
jacobb[2][0] = 0.0;
jacobb[2][1] = 0.0;
jacobb[2][2] = l[2];

dpe[0][0] = 1.0;
dpe[0][1] = 0.0;
dpe[0][2] = 0.0;
dpe[1][0] = 0.0;
dpe[1][1] = 1.0;
dpe[1][2] = 0.0;
dpe[2][0] = 0.0;
dpe[2][1] = 0.0;
dpe[2][2] = 1.0;

d = sqrt (p[0] * p[0] + p[1] * p[1] + p[2] * p[2]);

for (i = 0 ; i <= 2 ; ++i)
for (j = 0 ; j <= 2 ; ++j)
dp[j][i] = (dot (pepvr, pemvr) * dpe[i][j] + 2 * dot (pe, dpe[i]) * (pe[j] - vr[j])) /
                    dot (pemvr, pemvr)
                  - (2 * dot (pepvr, pemvr) * dot (pemvr, dpe[i]) * (pe[j] - vr[j])) /
                    (dot (pemvr, pemvr) * dot (pemvr, pemvr));

for (i = 0 ; i <= 2 ; ++i)
for (j = 0 ; j <= 2 ; ++j)
{
da1[i][j] = k1[i] * dp[0][j] + k2[i] * dp[1][j] + k3[i] * dp[2][j];
da2[i][j] = k4[i] * dp[0][j] + k5[i] * dp[1][j] + k6[i] * dp[2][j];
da3[i][j] = k7[i] * dp[0][j] + k8[i] * dp[1][j] + k9[i] * dp[2][j]
          - (1.0-s/d)*(p[0]*dp[0][j]+p[1]*dp[1][j]+p[2]*dp[2][j]);
}

for (i = 0 ; i <= 2 ; ++i)
for (j = 0 ; j <= 2 ; ++j)
{
dcos[i][j] = (- a1[i] * da3[i][j] - a3[i] * da1[i][j] + sqrt (a4[i])
             * da2[i][j] + (a2[i] * (a1[i] * da1[i][j] + a2[i] * da2[i][j]
             - a3[i] * da3[i][j])) / sqrt (a4[i]))
             / (a1[i] * a1[i] + a2[i] * a2[i])
             - (2 * (-a1[i] * a3[i] + a2[i] * sqrt (a4[i]))
```

```
            * (a1[i] * da1[i][j] + a2[i] * da2[i][j]))
            / ((a1[i] * a1[i] + a2[i] * a2[i]) *
              (a1[i] * a1[i] + a2[i] * a2[i])) ;

dsin[i][j] = (- a2[i] * da3[i][j] - a3[i] * da2[i][j] - sqrt (a4[i])
            * da1[i][j] - (a1[i] * (a1[i] * da1[i][j] + a2[i] * da2[i][j]
            - a3[i] * da3[i][j])) / sqrt (a4[i]))
            / (a1[i] * a1[i] + a2[i] * a2[i])
            - (2 * (-a2[i] * a3[i] - a1[i] * sqrt (a4[i]))
            * (a1[i] * da1[i][j] + a2[i] * da2[i][j]))
            / ((a1[i] * a1[i] + a2[i] * a2[i]) *
              (a1[i] * a1[i] + a2[i] * a2[i])) ;

}
for (j = 0 ; j <= 2 ; ++j)
{
db1[j][0] = k1[1] * dcos[1][j] + k4[1] * dsin[1][j] - k1[2] * dcos[2][j]
                        - k4[2] * dsin[2][j];
db1[j][1] = k2[1] * dcos[1][j] + k5[1] * dsin[1][j] - k2[2] * dcos[2][j]
                        - k5[2] * dsin[2][j];
db1[j][2] = k3[1] * dcos[1][j] + k6[1] * dsin[1][j] - k3[2] * dcos[2][j]
                        - k6[2] * dsin[2][j];

db2[j][0] = k1[2] * dcos[2][j] + k4[2] * dsin[2][j] - k1[0] * dcos[0][j]
                        - k4[0] * dsin[0][j];
db2[j][1] = k2[2] * dcos[2][j] + k5[2] * dsin[2][j] - k2[0] * dcos[0][j]
                        - k5[0] * dsin[0][j];
db2[j][2] = k3[2] * dcos[2][j] + k6[2] * dsin[2][j] - k3[0] * dcos[0][j]
                        - k6[0] * dsin[0][j];

db3[j][0] = k1[0] * dcos[0][j] + k4[0] * dsin[0][j] - k1[1] * dcos[1][j]
                        - k4[1] * dsin[1][j];
db3[j][1] = k2[0] * dcos[0][j] + k5[0] * dsin[0][j] - k2[1] * dcos[1][j]
                        - k5[1] * dsin[1][j];
db3[j][2] = k3[0] * dcos[0][j] + k6[0] * dsin[0][j] - k3[1] * dcos[1][j]
                        - k6[1] * dsin[1][j];
}

for (j = 0 ; j <= 2 ; ++j)
{
bn[0][j] = b[1][j] - b[2][j];
bn[1][j] = b[2][j] - b[0][j];
bn[2][j] = b[0][j] - b[1][j];
}

for (i = 0 ; i <= 2 ; ++i)
{
jacoba[0][i] = dot (bn[0], db1[i]);
jacoba[1][i] = dot (bn[1], db2[i]);
jacoba[2][i] = dot (bn[2], db3[i]);
```

```
}

fprintf ("Jacobian Matrix A \n");
fprintf (fl, "%6.4f %6.4f %6.4f \n", jacoba[0][0], jacoba[0][1], jacoba[0][2]);
fprintf (fl, "%6.4f %6.4f %6.4f \n", jacoba[1][0], jacoba[1][1], jacoba[1][2]);
fprintf (fl, "%6.4f %6.4f %6.4f \n", jacoba[2][0], jacoba[2][1], jacoba[2][2]);

fprintf ("Jacobian Matrix B \n");
fprintf (fl, "%6.4f %6.4f %6.4f \n", jacobb[0][0], jacobb[0][1], jacobb[0][2]);
fprintf (fl, "%6.4f %6.4f %6.4f \n", jacobb[1][0], jacobb[1][1], jacobb[1][2]);
fprintf (fl, "%6.4f %6.4f %6.4f \n", jacobb[2][0], jacobb[2][1], jacobb[2][2]);

fclose (fl);
}

/* A function to find the dot product of two vectors */

float dot (float v1[], float v2[])

{
return (v1[0] * v2[0] + v1[1] * v2[1] + v1[2] * v2[2]);
}
```

Appendix 5b Jacobian Matrix Determination for the Gimbal Problem

```
/*****************************************************************************/
/*      Program     : Gimbal
       Description : A Program to obtain the Velocity Jacobian to the gimbal
                     specification for the double-octahedral manipulator in closed
                     -form.  Function includes dot and solve (given in Appendix 4d.)
/*****************************************************************************/

#include <stdio.h>
#include <math.h>

/* function declaration */

float dot (float v1[], float v2[]);

void solve (float x, float y, float z, float lb, float ll, float s,
            float l[]);

float k1[3], k2[3], k3[3], k4[3], k5[3], k6[3], k7[3], k8[3], k9[3];
                                        /* Geometric constants */
float a1[3], a2[3], a3[3], a4[3];       /* Other computational variables */
float b[3][3];                          /* Offset Plane nodes */

void main()
{
FILE *f1;

float p[3];             /* origin of the moving frame */
float beta, theta, r;   /* Gimbal Parameters */
float u2[3];            /* Unit vector normal to the moving plane */
float u0[3];            /* Unit vector normal to the fixed plane */
float lb, ll, s;        /* Batten length, longeron length, joint offset */
float l[3];             /* Actuator lengths */
float d;                /* Multiplication factor */

float da1[3][3], da2[3][3], da3[3][3];
                        /* Partial derivatives of variable a */
float db1[3][3], db2[3][3], db3[3][3];
                        /* Partial derivatives of the mid-plane nodes */
float bn[3][3];         /* Mid-plane vectors */
float jacoba[3][3];     /* Jacobian [A] matirx */
float jacobb[3][3];     /* Jacobian [B] matrix */
float dcos[3][3], dsin[3][3];
                        /* Partial derivatives to the sine and cosine angles */
float dxdthe, dydthe, dzdthe;
```

```
float dxdbet, dydbet, dzdbet;
float dxdr, dydr, dzdr;
                        /* Partial derivatives of the origin of the moving plane */
int i, j;

fl = fopen ("position.dat", "w");

printf ("Key-in beta (in degrees)    : ");
scanf ("%f", &beta);
beta = beta * 3.1415927 / 180.0;
fprintf (fl, "beta                = %6.4f \n", beta);

printf ("Key-in theta (in degrees)   : ");
scanf ("%f", &theta);
theta = theta * 3.1415927 / 180.0;
fprintf (fl, "theta               = %6.4f \n", theta);

printf ("Key-in r value              : ");
scanf ("%f", &r);
fprintf (fl, "r value             = %6.4f \n",r);

printf ("Key-in batten length 'lb'   : ");
scanf ("%f", &lb);
fprintf (fl, "Batten length 'lb'  = %6.4f \n",lb);

printf ("Key-in longeron length 'll' : ");
scanf ("%f", &ll);
fprintf (fl, "Longeron length 'll' =  %6.4f \n",ll);

printf ("Key-in joint offset 's'     : ");
scanf ("%f", &s);
fprintf (fl, "Joint offset 's'    = %6.4f \n",s);


u2[0] = cos (theta) * sin (beta);
u2[1] = sin (theta) * sin (beta);
u2[2] = cos (beta);

/* Unit normal to the Fixed frame */

u0[0] = 0.0;
u0[1] = 0.0;
u0[2] = 1.0;

p[0] = r * (u2[0] + u0[0]);
p[1] = r * (u2[1] + u0[1]);
p[2] = r * (u2[2] + u0[2]);

solve (p[0], p[1], p[2], lb, ll, s, l);
```

```
fprintf (f1, "link lengths = %6.4f %6.4f %6.4f \n", l[0], l[1], l[2]);

jacobb[0][0] = l[0];
jacobb[0][1] = 0.0;
jacobb[0][2] = 0.0;
jacobb[1][0] = 0.0;
jacobb[1][1] = l[1];
jacobb[1][2] = 0.0;
jacobb[2][0] = 0.0;
jacobb[2][1] = 0.0;
jacobb[2][2] = l[2];

dxdthe = -r * sin (theta) * sin (beta);
dydthe = r * cos (theta) * sin (beta);
dzdthe = 0.0;

dxdbet = r * cos (beta) * cos (theta);
dydbet = r * cos (beta) * sin (theta);
dzdbet = -r * sin(beta);

dxdr = u0[0] + cos(theta) * sin (beta);
dydr = u0[1] + sin (theta) * sin (beta);
dzdr = u0[2] + cos (beta);

d = sqrt (p[0] * p[0] + p[1] * p[1] + p[2] * p[2]);

for (i = 0 ; i <= 2 ; ++i)
{
da1[i][0] = k1[i] * dxdthe + k2[i] * dydthe + k3[i] * dzdthe;
da2[i][0] = k4[i] * dxdthe + k5[i] * dydthe + k6[i] * dzdthe;
da3[i][0] = k7[i] * dxdthe + k8[i] * dydthe + k9[i] * dzdthe - (1 - s/d) *
            (p[0] * dxdthe + p[1] * dydthe + p[2] * dzdthe);

da1[i][1] = k1[i] * dxdbet + k2[i] * dydbet + k3[i] * dzdbet;
da2[i][1] = k4[i] * dxdbet + k5[i] * dydbet + k6[i] * dzdbet;
da3[i][1] = k7[i] * dxdbet + k8[i] * dydbet + k9[i] * dzdbet - (1 - s/d) *
            (p[0] * dxdbet + p[1] * dydbet + p[2] * dzdbet);

da1[i][2] = k1[i] * dxdr + k2[i] * dydr + k3[i] * dzdr;
da2[i][2] = k4[i] * dxdr + k5[i] * dydr + k6[i] * dzdr;
da3[i][2] = k7[i] * dxdr + k8[i] * dydr + k9[i] * dzdr - (1 - s/d) *
            (p[0] * dxdr + p[1] * dydr + p[2] * dzdr);
}

for (i = 0 ; i <= 2 ; ++i)
for (j = 0 ; j <= 2 ; ++j)
{
dcos[i][j] = (- a1[i] * da3[i][j] - a3[i] * da1[i][j] + sqrt (a4[i])
             * da2[i][j] + (a2[i] * (a1[i] * da1[i][j] + a2[i] * da2[i][j]
             - a3[i] * da3[i][j])) / sqrt (a4[i]))
```

```
                / (a1[i] * a1[i] + a2[i] * a2[i])
                - (2 * (-a1[i] * a3[i] + a2[i] * sqrt (a4[i]))
                * (a1[i] * da1[i][j] + a2[i] * da2[i][j]))
                / ((a1[i] * a1[i] + a2[i] * a2[i]) *
                  (a1[i] * a1[i] + a2[i] * a2[i])) ;

dsin[i][j] = (- a2[i] * da3[i][j] - a3[i] * da2[i][j] - sqrt (a4[i])
                * da1[i][j] - (a1[i] * (a1[i] * da1[i][j] + a2[i] * da2[i][j]
                - a3[i] * da3[i][j])) / sqrt (a4[i]))
                / (a1[i] * a1[i] + a2[i] * a2[i])
                - (2 * (-a2[i] * a3[i] - a1[i] * sqrt (a4[i]))
                * (a1[i] * da1[i][j] + a2[i] * da2[i][j]))
                / ((a1[i] * a1[i] + a2[i] * a2[i]) *
                  (a1[i] * a1[i] + a2[i] * a2[i])) ;

}
for (j = 0 ; j <= 2 ; ++j)
{
db1[j][0] = k1[1] * dcos[1][j] + k4[1] * dsin[1][j] - k1[2] * dcos[2][j]
                        - k4[2] * dsin[2][j];
db1[j][1] = k2[1] * dcos[1][j] + k5[1] * dsin[1][j] - k2[2] * dcos[2][j]
                        - k5[2] * dsin[2][j];
db1[j][2] = k3[1] * dcos[1][j] + k6[1] * dsin[1][j] - k3[2] * dcos[2][j]
                        - k6[2] * dsin[2][j];

db2[j][0] = k1[2] * dcos[2][j] + k4[2] * dsin[2][j] - k1[0] * dcos[0][j]
                        - k4[0] * dsin[0][j];
db2[j][1] = k2[2] * dcos[2][j] + k5[2] * dsin[2][j] - k2[0] * dcos[0][j]
                        - k5[0] * dsin[0][j];
db2[j][2] = k3[2] * dcos[2][j] + k6[2] * dsin[2][j] - k3[0] * dcos[0][j]
                        - k6[0] * dsin[0][j];

db3[j][0] = k1[0] * dcos[0][j] + k4[0] * dsin[0][j] - k1[1] * dcos[1][j]
                        - k4[1] * dsin[1][j];
db3[j][1] = k2[0] * dcos[0][j] + k5[0] * dsin[0][j] - k2[1] * dcos[1][j]
                        - k5[1] * dsin[1][j];
db3[j][2] = k3[0] * dcos[0][j] + k6[0] * dsin[0][j] - k3[1] * dcos[1][j]
                        - k6[1] * dsin[1][j];
}

for (j = 0 ; j <= 2 ; ++j)
{
bn[0][j] = b[1][j] - b[2][j];
bn[1][j] = b[2][j] - b[0][j];
bn[2][j] = b[0][j] - b[1][j];
}

for (i = 0 ; i <= 2 ; ++i)
{
jacoba[0][i] = dot (bn[0], db1[i]);
```

```
jacoba[1][i] = dot (bn[1], db2[i]);
jacoba[2][i] = dot (bn[2], db3[i]);
}

fprintf (fl, "Jacobian Matrix A \n");
fprintf (fl, "%6.4f %6.4f %6.4f \n", jacoba[0][0], jacoba[0][1], jacoba[0][2]);
fprintf (fl, "%6.4f %6.4f %6.4f \n", jacoba[1][0], jacoba[1][1], jacoba[1][2]);
fprintf (fl, "%6.4f %6.4f %6.4f \n", jacoba[2][0], jacoba[2][1], jacoba[2][2]);

fprintf (fl, "Jacobian Matrix B \n");
fprintf (fl, "%6.4f %6.4f %6.4f \n", jacobb[0][0], jacobb[0][1], jacobb[0][2]);
fprintf (fl, "%6.4f %6.4f %6.4f \n", jacobb[1][0], jacobb[1][1], jacobb[1][2]);
fprintf (fl, "%6.4f %6.4f %6.4f \n", jacobb[2][0], jacobb[2][1], jacobb[2][2]);

fclose (fl);
}

/* A function to find the dot product of two vectors */

float dot (float v1[], float v2[])

{
return (v1[0] * v2[0] + v1[1] * v2[1] + v1[2] * v2[2]);
}
```

Vita

Babu was born in Coimbatore, India on November 2, 1967. In 1987, he received his Bachelor's Degree in Production Engineering from PSG College of Technology. With Dr. Reinholtz as his advisor, he graduated with a Master's Degree in December 1988 while qualifying for his Ph.D. studies. Before continuing this research area that was planned from his Master's thesis, he worked on a project on extruder screw manufacturing for a year. Upon completion, he plans to return to India and start an engineering industry in his hometown.