AUTOMATIC GENERATION OF INTERFERENCE-FREE GEOMETRIC MODELS OF SPATIAL MECHANISMS

by

Mitchel J. Keil

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:

Dr. A. Myklebust, Chairman

Dr. R. G. Leonard

Dr. J. L. Lytton

Dr. C. F. Reinholtz

Dr. R. L. West

June, 1990

Blacksburg, Virginia

AUTOMATIC GENERATION OF INTERFERENCE-FREE GEOMETRIC MODELS OF SPATIAL MECHANISMS

by

Mitchel J. Keil

Committee Chairman: Dr. A. Myklebust
Mechanical Engineering

(ABSTRACT)

This work presents methods used to obtain geometric models of spatial mechanisms which can be realized in hardware. Each model is created automatically from the kinematic description of a mechanism. The models are tested for interference between joints and links. Models with interfering links or joints are reshaped automatically into an interference-free configuration.

An investigation of the relative efficiency of different interference detection techniques is discussed. A method for determining interferences based on vector loop equations was developed for this work. Other approaches for interference detection include parametric space and a method using parallel coordinates. 2000 line segments were randomly generated to test the three methods. No significant difference between the three techniques was found, but a coarse detection scheme was developed based on observations of intersection conditions in parallel coordinates. The coarse detection technique reduced interference detection times by 48%.

The concept of joint positioning freedoms is presented formally for the first time. Using a unidirectional avoidance strategy along a straight line, these

repositioning freedoms are exploited in a manner which guarantees the elimination of interferences for revolute, prismatic, and cylindric joints.

A unique method for optimal orientation of spheric joint ball–cup pairs is described. Points from an inverse image of the attachment piece for the ball are mapped onto a unit sphere in the reference frame of the cup. The axis of a bounding cone is then used to align the attachment piece for the cup. The method minimizes the chances for collisions between the cup and the ball attachment piece.

Elements which attach the joints are modeled as three segments. This has proven to be an optimal representation. Interferences with these elements are eliminated using the elliptical projection of circular paths onto a plane which is perpendicular to the axis of symmetry for an intruding object.

Several examples are given illustrating the successful generation of interference-free spatial mechanism models. The mechanisms include an RSSR, an RPCS, an RCCC, and an RRRRRRR.

Acknowledgements

I would like to thank Dr. A. Myklebust for his guidance and advice on this work. I would also like to thank the other members of my committee for the time they spent reviewing this work and for their comments. A special thanks goes to Dr. C. F. Reinholtz for the time he spent with me on his specialty the RCCC mechanism. Special thanks also go to Dr. M. P. Deisenroth for his timely last minute assistance.

To David Montgomery, I would like to express thanks for needing a thesis topic when he did. His work was invaluable in helping to debug and verify this work. The extra effort he put forth in capturing images was also a tremendous aid in documentation.

Thanks to Dr. W. R. Winfrey for his work on bit map manipulation, and the time it took out of his busy schedule.

A deep appreciation goes to Dr. A. Inselberg for the time spent on explaining the parallel coordinate method.

I would like to thank my father Wilbur E. Keil for the faith and support he has shown me over the years.

Finally, thanks to my beautiful wife Marla for her prayers and inspiration. I could not have done it without her.

Table of Contents

Appendices

List of Illustrations

Figure Page

Figure Page

List of Tables

Chapter 1

Introduction

The problem of modeling spatial mechanisms is very interesting. The author first became involved in mechanism modeling while working on his master's thesis at Florida Atlantic University [1,2]. The initial objective at that time was to create three dimensional models of planar mechanisms similar to the one shown in figure 1. The program which produced these models, ANIMEC, proved to be useful in visualizing mechanisms by removing the ambiguity of how such mechanisms might appear using real hardware [3]. It was known that synthesis and analysis algorithms had been developed at that time, notably IMP[4], KINSYN III [5], MECSYN [6,7], and LINCAGES [8]. This lead to the concept of a mechanism design system using MECSYN, IMP, and ANIMEC [9]. Further work toward this system in the form of a general mechanism input interface was later completed by Thatch [10,11]. It soon became apparent, however, that there was a piece missing from the design system. Figure 2 illustrates the problem. The mechanism shown here is the same four-bar mechanism shown in Figure 1 but at another position. It was obvious that interference was a problem to be dealt with,

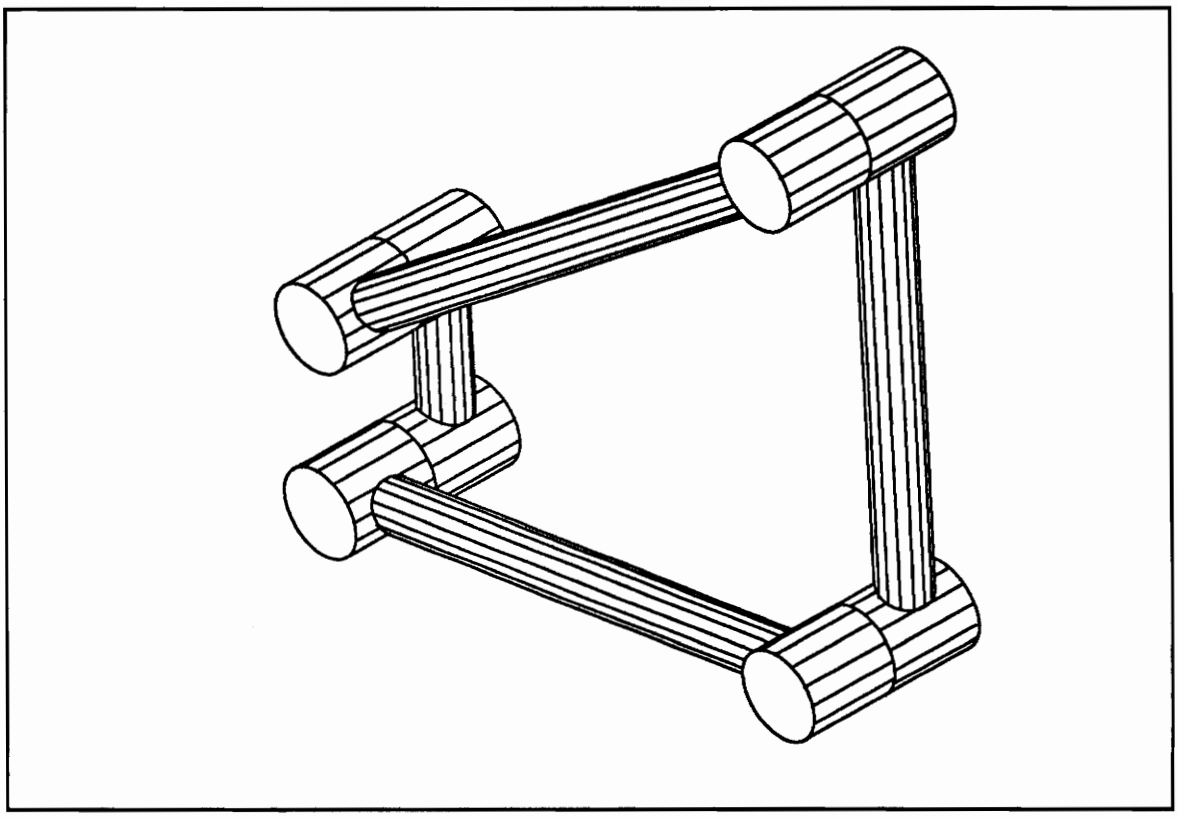

Figure 1. 3-D model of a planar four-bar

even in modeling planar mechanisms. This was the initial impetus toward addressing the interference problem in mechanism modeling.

It was quickly realized that modeling spatial mechanisms involved three distinct procedures. First, there had to be a set of rules for creating the elements of the mechanism. For planar mechanisms with revolute joints the task was fairly simple. The joints of the four-bar mechanism in Figures 1 and 2 were modeled as cylinders. The cylinders of a joint shared a common axis and were positioned along that axis so that their volumes did not intersect. Joints on a common link had their centroids located in a common plane perpendicular to the axes of the joints. It was simple to connect the centroids with a single straight connecting

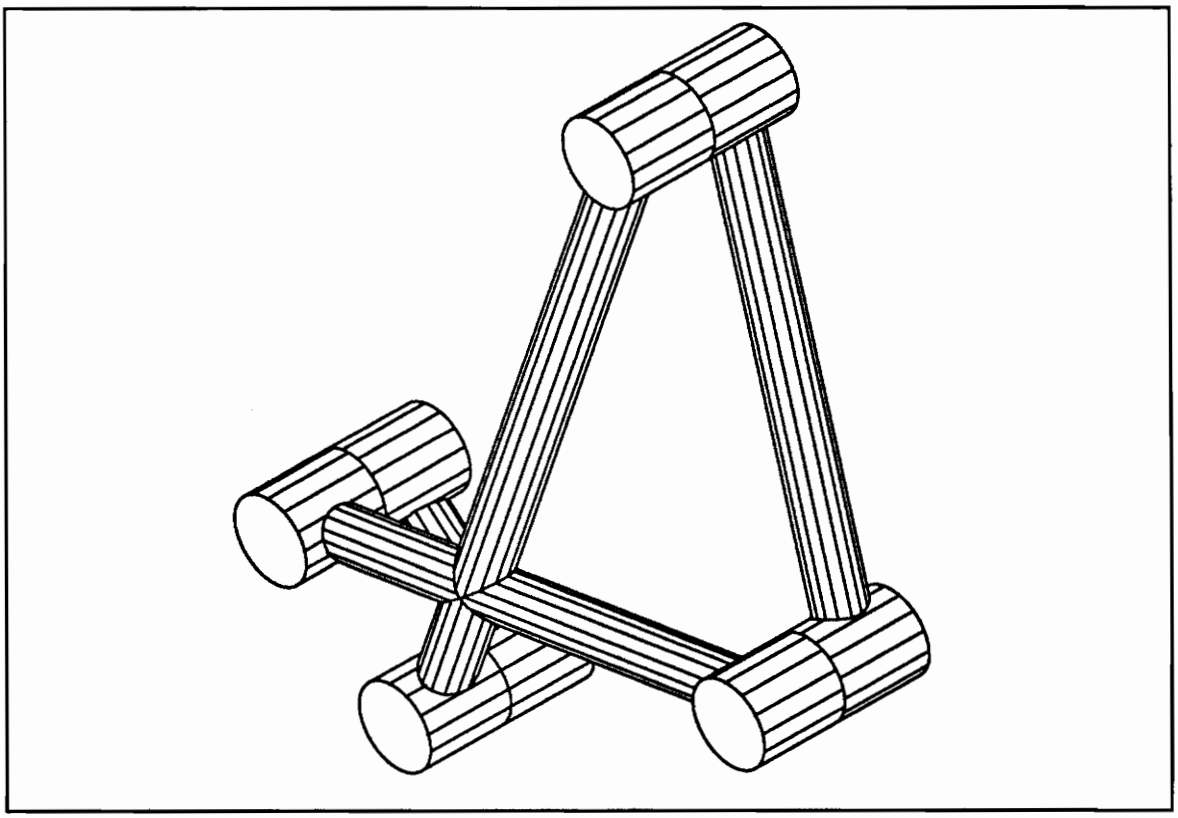

Figure 2. Four-bar with interference

element. Pennington [12,13] made significant advances in modeling other joint types for spatial mechanisms by breaking joints into internal and external parts. The connections for binary links were made through the intersection and trimming of curves. This began by first defining two planes orthogonal to the the local Z axis for each joint. The Z axis represented the axis of translation or rotation for revolute, cylindric, and prismatic joints and was arbitrary for spheric joints. The line of intersection for these two planes was determined, and a minimum distance solution was found for a set of line segments which were constrained to lie within either of the two planes. This resulted in two straight elements connecting the

joints of a link. The new modeling algorithm was more complex but far more general.

The second step in modeling involved the detection of interferences. The author's work in this area involved developing bounding round-ended cylinders or spheres for all elements of a mechanism and developing a vector loop equation similar to that used in mechanism loop analysis [14]. The process of interference detection was later improved by doing simplified coordinate tests [15]. This will be discussed in greater detail in the chapter on interference detection.

The third step was to develop a method for reconfiguring a mechanism once an interference was found. For planar mechanisms, the only method used was shifting links into parallel planes. If an interference was found, one of the colliding links was shuffled into another plane. The proper contiguous relationship was maintained with other links which shared the joints of the repositioned link. In other words, the repositioned link always had to be in a plane adjacent to links which shared joints with that link. Reconfiguring links for spatial mechanisms turned out to be a more complicated process. The links were not constrained to exist in a plane. This forced a solution where the links had to be reshaped, not merely shuffled into another plane. Once a link was reconfigured to miss another link, this created a potential for interference between the reconfigured link and all other links of the mechanism. Again, this will be described in greater detail in the chapters on eliminating interferences in joints and links.

The purpose of this dissertation is to outline the methods which the author used to create interference-free configurations of spatial mechanisms with binary links and binary joints. Binary links are defined as having two joints. Similarly,

binary joints are defined as having attachments to two links. In describing the methods, the author will also briefly describe some of the unsuccessful attempts. Two programs will be presented: SLIDE (Spatial Linkage Interference Detection and Elimination) and IPOST (IMP POSTprocessor). SLIDE is the linkage construction and reconfiguration algorithm which was used to verify the methods for interference detection and reshaping presented here. IPOST is a postprocessor for IMP output which configures the output into a format which can be used by SLIDE. All of the bitmap images presented here were derived from a program called PriSM which serves as a postprocessor for displaying SLIDE output. PriSM will not be discussed in detail since it is described completely in the thesis work of Montgomery [16,17].

Chapter 2

Literature Review

No references were found in the literature relating specifically to the problem of constructing interference-free spatial mechanisms. This does not mean nothing was found relating to interference detection or collision avoidance. To the contrary, there was much information on these two topics concerning robots and robotic manipulators. For completeness, this literature is reviewed here. The format for the review is to take the papers in approximate chronological order of appearance in the literature.

1977

Udupa [18] dealt with the problem of detecting and avoiding collisions. Using minimum bounding cylinder models for two and three-dimensional manipulators, he found collision-free movements through obstacles by iterating through several planning phases. He introduced the concept of characterizing collision-free space in terms of cartesian space and joint variables.

1979

Boyse [19] presented a method for detecting interferences where smooth surfaces were approximated by planar facets. A method for detecting interferences between rotating objects and stationary objects was also presented. Edges of the stationary objects were tested for intersection with surfaces generated by revolving the edges of the moving objects to generate swept conic sections. Boyse went on to mention methods for rapid intersection testing using bounding boxes and spheres. These rapid approximations appear throughout the literature. A form of the bounding box test was incorporated into the detection algorithm of SLIDE.

Lozano-Perez and Wesley [20] discussed a method for generating collision-free paths for objects where the moving objects were reduced to a point and stationary polyhedral objects were enlarged. The modified space was termed configuration space, a term which in later literature was truncated to C-space. This conversion resulted in a navigation problem for a simple point passing among the enlarged objects. The basic algorithm used a line of sight method on obstacle vertices to chart paths through the obstacles.

1980

Ahuja et al [21] described two methods for collision detection. The first method used projections of polyhedral approximations of objects onto the principal coordinate planes. Nonconvex objects were processed by decomposition into convex objects. The other method decomposed the objects into an octree representation. A collision avoidance scheme was discussed for robots where the braking distance necessary to stop the robots was used to expand the objects by

the characteristic braking distance. Intersections between the expanded objects were proposed to serve as warnings for impending collisions.

1981

Lozano-Perez [22] extended the algorithm of [20] to planning manipulator movements. He stated that rotations of three-dimensional objects create a C-space in six dimensions. The six-dimensional space was reduced to a three-dimensional space by using a swept volume based on the rotation of the original object. This new object was used to create the C-space.

Meyer [23] presented an emulation system for robots with interference detection based on intersecting volumes. Sensor functions could be simulated using the intersection tests.

Inselberg [24] began publishing work where intersections were determined using parallel coordinates. The technique was based on theories of projective geometry. One advantage to his technique was that multiple dimensional data could be presented in a planar format. Later work [25,26,27] showed how parallel coordinates could be used to assist in predicting aircraft trajectories, and offered the promise of faster intersection tests.

1982

Myers and Agin [28] stated that the C-space approach worked well for robots only up to three degrees of freedom. A PUMA 600 robot was modeled with six degrees of freedom. A stepped movement approach was used where each element of the robot was checked for intersection with obstacles in the work environment at each step in the robot's motion. In order to speed the process, a hierarchy of enclosing boundaries was employed using bounding boxes, spheres,

and infinite planes. Myers and Agin argued to use bounding spheres for all elements on the robot, because the interference check for spheres was faster than all other primitives. Once a collision was found, alternate motions were considered in a plane perpendicular to the motion at the time of the collision.

1983

Brooks [29] introduced a method for defining the free space of a robot. The free space was modeled as the union of generalized cones. Where many algorithms had used an approach where moving objects barely missed stationary objects, Brooks' intent was to allow a generous clearance by moving along the axes of these cones. The method was implemented for two dimensional workspaces and did not work well in tightly cluttered spaces, but in uncluttered spaces it was shown to be very fast.

Brooks [30] used his generalized cone approach to plan pick-and-place operations for a PUMA robot. Initial attempts at finding paths for the six degree of freedom robot were not successful, but it was found that the pick-and-place operations did not need to take advantage of all the robot's degrees of freedom. By locking selected joints to reduce the degrees of freedom, successful solutions were found.

Lozano-Perez [31] continued developing his configuration space approach by using it in spatial planning. He described two problems: findpath and findspace. Findpath was described as the problem of finding a collision-free path through a set of obstacles. This is a common problem for robots. Findspace was described as the problem of finding a suitable space for a shape (object) to exist without interference with other shapes. This is a common problem with placing

components on printed circuit boards. Also mentioned were problems relating to machining and template layouts on sheet metal to minimize material usage.

Gouzenes [32] continued the development of the C-space approach by characterizing the empty space subset of C-space. He used a method of decomposing the empty space into manageable subsets where straight line motion was possible and established the connectivity between the subsets. A method for taking advantage of the specific kinematic constraints of robots was also mentioned.

Schwartz and Sharir [33] attacked the problem of coordinating the continuous motion of multiple circles moving in a space constrained by polygonal boundaries. The resulting method was two dimensional in its scope.

Hopcroft, Schwartz, and Sharir [34] presented a technique for detecting intersections among pairs of spheres. The technique reduced the time to detect spherical intersections to a period which was proportional to the time for calculating the intersection between the same number of parallelepipeds. This was accomplished by ordering the spheres into a balanced binary tree structure for interference detection.

A swept volume approach was introduced by de Pennington et al [35]. In this approach, elements of a robot arm were placed within bounding spheres. The algorithm described could accommodate motion along a straight line or along a circular arc. Successive positions were used to create solid elements which were comprised of the moving element at the start and end of that increment of motion. The path was then used to create cylinders or toroidal sections which were unioned with the two spherical elements. The resulting swept volumes were checked for

intersection with other swept volumes and objects within the workspace. The algorithm was very conservative in its solution, since intersecting swept volumes did not necessarily represent actual collisions between the moving objects.

1984

Davis and Camacho [36] outlined a method of determining collision-free paths using a minimum cost function and task dependent information on the robot and its goal state. This algorithm was also based on the C-space approach.

Faverjon [37] presented a method for transforming three dimensional objects into joint space objects using the first three joints of a revolute jointed robot. Joint space was defined as the rotational variables controlling the joint positions set up in an orthogonal coordinate system. In this way, a position for the robot could be represented as a point in joint space. An octree approach was used to represent the objects.

Freund and Hoyer [38,39] discussed the importance of coordinating two or more robots that share the same workspace. They established a hierarchical scheme for giving robots priority of movement in cases of potential collisions. The robot highest in the hierarchy was allowed to continue on its optimum path while robots lower in the hierarchy were rerouted around the collision situation.

Refinements of Gouzenes' [40,41] methods were presented with a study on how to develop simple gross motions for general manipulator robots.

A method for avoiding obstacles using visual and other sensory feedback was presented by Grechanovsky and Pinsker [42]. Using the sensory information, elements of a robot would touch the edge of an object and slide along this edge until sliding off.

Hogan [43] introduced the concept of using impedances where velocity and proximity dependent fields are simulated for use in collision avoidance. The method was used to handle manipulators enclosed within spherical boundaries.

Hopcraft et al [44] continued the work on the Warehouseman's Problem (also known as the Piano Mover's problem or findpath problem) by developing a proof that the problem in two dimensions is PSPACE-hard. Essentially, the work proved that a polynomial representation for a findpath problem exists in polynomial space (PSPACE-hard) even though its solution may not be very efficient in the general case.

Larson and Donath [45] discussed an interactive interference checking method. Faceted surface models were used to represent robots and objects in their workspace. Their argument for not using solid models was that they were too slow to work with. Function keys and input knobs were used for manipulation. They expanded upon this method in 1985 [46].

Fine motion was analyzed by Lozano-Perez [47]. He considered the problem of placing a peg into a hole based on subtle changes in geometry such as the presence or absence of chamfers. Surface sliding with friction was one strategy considered. He noted that the subtle geometry changes could greatly alter the strategy for finding the goal.

Park [48] offered an interesting approach to the problem of coordinating multiple manipulators using a multidimensional state-space representation. He ran into several problems involving large computational and data storage requirements. Although he found the results encouraging, convergence was

difficult to achieve and many collisions were not avoidable. The problem of getting caught in local dwell points was also encountered.

Configuration space was also used by Red et al [49,50,51]. A method for slicing the configuration joint space for a three jointed robot was presented. A minimum distance scheme was also described. The slice technique was used to make the problem of finding a suitable path more manageable. The slice contained the two points in joint space representing the start and goal positions. This reduced the find path problem to one of routing a point through polygons. Further developments in this area occurred in 1985 [52] and 1987 [53,54].

1985

A two dimensional study on the findpath problem with translations and rotations was outlined by Brooks and Lozano-Perez [55]. One rotational and two translational degrees of freedom were allowed. Non-convex objects were moved among non-convex objects by decomposing them into convex objects and recursively operating on the convex objects. The solution times were very long.

Cameron [56,57] used a four dimensional approach to find collisions among moving objects. The solutions were limited to objects moving without rotation, but the over conservative boundaries of de Pennington [35] were eliminated for these cases.

Clermont et al [58] applied C-space techniques to a specific robot. Two heuristics were employed to aid in finding a good path. They are referred to as the "bug over the wall" heuristic and the "dog through the door" heuristic. The "bug over the wall" refers to raising the manipulator over objects while the "dog

through the door" applies to rotations about the vertical axis to pass through narrow passages.

Donald [59] discussed possible techniques for solving the C-space problem in six dimensions which allowed for the three translational and rotational degrees of freedom needed to move some bodies. Schemes described involved sliding along five dimensional surfaces, sliding along one to four dimensional intersections, and jumping between six dimensional objects.

Ganter [60] offered a method for using swept solids in interference detection which eliminated the overly conservative solutions presented by de Pennington [35]. Ganter used a kinematic inversion technique where objects were moved relative to an object being tested for interference. Faceted convex hulls were developed over the objects before sweeping to maintain a moderately conservative solution. These techniques were expanded in 1986 [61,62].

Hasegawa [63] discussed a method for characterizing free space for a six degree of freedom robot. The method built a three dimensional C-space in rotation variables for the robot arm while enclosing the manipulator within a sphere.

Kovesi [64] described a method for collision avoidance which was applied to the problem of sheep shearing. The method was sensor based because Kovesi thought obstacle avoidance path planning techniques were not appropriate.

Laugier and Germain [65] presented a method for overconstraining C-space and combined this with a swept volume characterization of a robot's payload.

Nagata et al [66] presented a strategy for coordinating multiple robots of a specific type within a common workspace.

A circuit concept was introduced by Valade [67] by cataloging the edges of polyhedra. Using this method, concavities were identified and a list of constraints was produced.

Wong and Fu [68] offered a technique for characterizing C-space in terms of orthogonal projections. The method brought the promise of characterizing C-space using TV cameras.

1986

A method of generalizing axisymmetric link elements was described by Baldur and Dube [69]. Bounding ellipses were placed over the links and a correction factor was applied to each ellipse. The method dealt with the limitations encountered when computers without floating point arithmetic were used to control robots.

Hayward [70] found that, by characterizing a robot's workspace using an octree and bounding the robot elements with round-ended cylinders, interference detection times could be significantly reduced. This hybrid approach allowed for one collision detection check per second.

Octrees were used again by Herman [71] as a rapid means of characterizing C-space for rapid collision detection.

Khatib [72]used the force field approach for obstacle avoidance using visual sensing. The system was implemented on a PUMA 560 robot

Sensory feedback was the basis for Lumelsky [73] developing a non-heuristic approach to path planning. When the technique was applied to

planar robots in subsequent work [74,75,76], he stated that this reduced the problem to the analysis of simple closed curves on the surface of a two dimensional manifold.

Redundant robots (those with more than six joints) were addressed by Dupont and Derby [77]. They acknowledged that C-space was difficult to work with if more than three degrees of freedom were used. They obtained a fast algorithm by developing partial maps of C-space.

Young and Duffy [78] offered a solution for motion planning for planar robots based on a robot's kinematic relationship with its surroundings.

1987

Erdmann and Lozano-Perez [79] combined some of their techniques to address the problem of moving multiple bodies. One example involved positioning non-convex planar objects without rotations into a very tight fitting assembly. A second example covered positioning three articulated pieces where each piece consisted of two four sided polygons pinned together at a point. In each case, a priority of movement relationship was established to reposition the elements.

Grau et al [80] described a minimum distance algorithm using a four dimensional approach where time was represented in the fourth dimension. The study was applied to lathe elements. Simultaneous rotations and translations were not allowed.

Inertial constraints on motion planning were considered by O'Dunlaing [81]. The problem presented was one dimensional. Quadratic polynomial

functions were applied to determine accelerations necessary to maintain minimum boundaries on cartesian armed robots.

Oommen and Reichstein [82] presented an interesting problem where elliptical objects move through elliptical obstacles. All ellipses maintained the same ratio of major axis to minor axis, all major axes were parallel. With constraints, a transformation was done where a coordinate system was set up with one coordinate axis parallel to the major axes. This coordinate was then modified until the major axis was equal to the minor axis. The problem of dealing with ellipses was thus reduced to one which dealt with circles.

A piecewise linear solution was outlined by Pappadimitriou and Silverberg [83]. The solution tended toward a shortest path for an obstacle moving among polygons. By analyzing the nodes of a visibility graph representing line of sight transitions between vertices, several paths are generated and the shortest one is chosen.

The problem of moving a rod among polygonal objects in two dimensions was discussed by Sifrony and Sharir [84]. Rotations as well as translations were allowed.

Xing et al [85] presented a method of dealing with C-spaces of dimension greater than three by decomposition into lower dimensional subspaces.

1988

Amirouche and Jia [86,87] outlined a method for avoiding collisions in robots. The links in the arms were modeled as line segments and collisions were assumed to occur only if links crossed in the same plane or were collinear. A

parametric line segment approach was used. The theory was discussed, but no examples of implementation were given.

An interactive approach to collision avoidance was described by Choi et al [88]. Using a 6 degree of freedom joystick, a PUMA 560 robot model was maneuvered through its workspace. Warnings occurred in the event of collisions. The path for the model could then be corrected before transferring the path to the actual robot.

The C-space approach was used on a model of an IBM 7565 robot by Fletcher and Goldenberg [89]. The system was integrated into a CATIA/IBM 7565 interface.

Hurteau and Stewart [90] applied Construction Solid Geometry (CSG) techniques to distance calculations to determine imminent collisions. Unions of convex polyhedra and cylinders of ellipsoidal cross section were used as the building blocks for this process.

Another example of the C-space approach was applied to the movement of mobile robots on a shop floor by Palma-Villalon and Dauchez [91].

Standardization efforts toward the find path problem were discussed by Quiocho and Allen [92]. They state that certain techniques are more applicable to specific types of problems. They broke the problem into four subsets: inverse kinematics of the manipulator, obstacle location and identification, path evaluation and generation, and path selection and display.

Zhu and Freeman [93] used spatial sorting techniques to quickly find intersections among objects. Objects were represented in spherical and cylindrical form and projected onto a two dimensional grid. The grid was used to eliminate

pairs which obviously did not interfere before doing more refined intersection tests.

1989

Buchal and Cherchas [94] presented a method for iterating to avoid interfering trajectories using the Newton-Raphson method to minimize a cost function. Planar objects were analyzed. The cost function algorithm was based on incremental sweeps of the objects where the sweeps were checked for penetration by objects in the workspace. When a sweep was penetrated, the path was modified.

Another numerical method based on the penetration of objects into paths and the inverse kinematic solution for a robot was offered by Dai [95]. The method was three dimensional in its scope, and Dai claimed it could accommodate a dynamically changing environment. A hierarchy for collision avoidance was established with the manipulator at top priority in avoidance while links attached to the base of a robot were at the lowest priority.

Jacak [96] proposed using a Finite State Machine (FSM) to model the kinematics of a robot. He stated that the FSM was limited at the time to planar robots and would be difficult to apply to spatial robots.

Ku and Ravani [97] presented an algorithm for decomposing non-convex regions into convex partially bounded regions. The algorithm was used as an aid in negotiating objects through non-convex regions in a planar configuration.

Ozaki et al [98] offered a very interesting method for finding collision-free paths in a plane. Objects were shrunk to assist in finding an initial collision-free

path. When the path was found, the objects were relaxed back to their original size, pushing the curve representing the path ahead of them.

Shin and Bien [99] introduced the notion of virtual coordination space for two planar robots in a common workspace. The virtual coordination space was described as the normalized path arc lengths of two robots placed in a set of orthogonal coordinates. A potential collision region was mapped out in this space.

SUMMARY OF LITERATURE

The configuration space approach to collision avoidance was found to be very popular, but it was very cumbersome when dealing with rotations and translations in three dimensions. Such conditions were common in almost every type of spatial mechanism. For this reason alone, a configuration space approach was never considered.

Other methods presented were limited to planar configurations, or were too unstable, as in the case of Park [48]. This was unacceptable for spatial links with highly complex motions in very close proximity.

A swept volume approach was considered for determining intersections, but Ganter [60] made a statement which steered the author away. He stated that non-convex solids produced irregular objects when their swept volumes intersected with themselves. In other words, the intersecting swept surfaces produced logical conflicts which prevented their resolution into a solid object. All of the objects that the author was considering for use were non-convex, and a high percentage of the of the objects would sweep back into themselves such as cranks on four bar mechanisms. Thus, it appeared at the time that the swept volume approach would

not work. Ganter later retracted that statement [61], but the author's direction was set.

Bounding cylinders and parallelepipeds were popular throughout the literature, and they were also used in this work in approximating objects. Actually, it was not that much of an approximation since the models were generated using cylinders, spheres, and parallelepipeds.

There were several methods for reducing intersection detection times. These methods involved projections of object profiles onto planes, bounding objects with spheres, and bounding objects with parallelepipeds or boxes. The only approximation used here was the bounding parallelepiped method where the faces of the parallelepiped were always parallel to a principal coordinate plane. This will be described further in the chapter on interference detection.

Chapter 3

Research Objectives

One objective of this research was to create a method for modeling spatial mechanisms. Modeling alone was not sufficient, however, since several methods for modeling had been implemented at the time. Two constraints were placed on the method to be developed. The automatically generated models had to be realizable in hardware, and the models could have no interfering structures. This had never been accomplished before.

A method for creation of the models was needed. The decision was made to build on the concepts developed by Pennington and Myklebust [12,13]. The input format would be similar to that of GENMOD.

Interference detection had to be performed before any avoidance strategy could be considered. It was envisioned that models could be composed of many elements, and each of these elements had to be examined for interference. Thus, an investigation of efficient interference detection techniques was necessary. Vector loop equations, commonly used in mechanism analysis, appeared promising. Parametric equations were commonly used in geometric modeling, and

a new technique using parallel coordinates was appearing in the literature. The decision was made to investigate the relative efficiency of the three techniques in determining intersections. The author was to implement methods based on the these techniques and develop a means of testing their relative performance.

Once models were created and interferences were found, methods had to be developed for reshaping the mechanism. The definition of these methods and their implementation was left to the author.

Verification of the methods would require the ability to test many different mechanisms. Because of the general nature of this requirement, a method for extracting data from IMP [4] to generate formatted input data was needed. Verification also required the development of a format for output. This was necessary to support any methods which might be developed for model visualization.

Chapter 4

Interference Detection Investigation

It was impossible to initiate any interference elimination strategy until interferences could be detected. The modeling approach was to use bounded line segments to approximate the elements of a mechanism. This precipitated the investigation of methods to detect line segment intersections.

The author had developed an intersection detection algorithm based on vector loop analysis [14]. An algorithm using parametric equations was documented in [100]. The author also developed an algorithm based on Inselberg's parallel coordinate methods [15]. These three approaches were evaluated to solve the problem of finding the intersection of line segments in euclidean three-space. The objective for the investigation was to determine if the interference detection process could be performed more quickly.

VECTOR ANALYSIS

Figure 3 illustrates the vector analysis approach. By setting up a vector loop, equation 1 can be written.

$$(1) \qquad \underline{a}_0 - \underline{b}_0 + \lambda_1\underline{a} - \lambda_2\underline{b} + \lambda_3 \frac{(\underline{a} \times \underline{b})}{|\underline{a} \times \underline{b}|} = \underline{a}_0 - \underline{b}_0 + \lambda_1\underline{a} - \lambda_2\underline{b} + \lambda_3\underline{n} = 0$$

where:

$\underline{a}$ ≡ the vector along the first line segment or link
$\underline{b}$ ≡ the vector along the second line segment or link
$\underline{a}_0$ ≡ the position vector for the first line segment
$\underline{b}_0$ ≡ the position vector for the second line segment
$\underline{n}$ ≡ the vector normal to $\underline{a}$ and $\underline{b}$
$\lambda_1,\lambda_2,\lambda_3$ ≡ scale factors

When equation 1 is solved for the three scalars, the simultaneous conditions for intersection become:

$$(2) \qquad 0 \leq \lambda_1 \leq 1$$

$$(3) \qquad 0 \leq \lambda_2 \leq 1$$

$$(4) \qquad \lambda_3 = 0$$

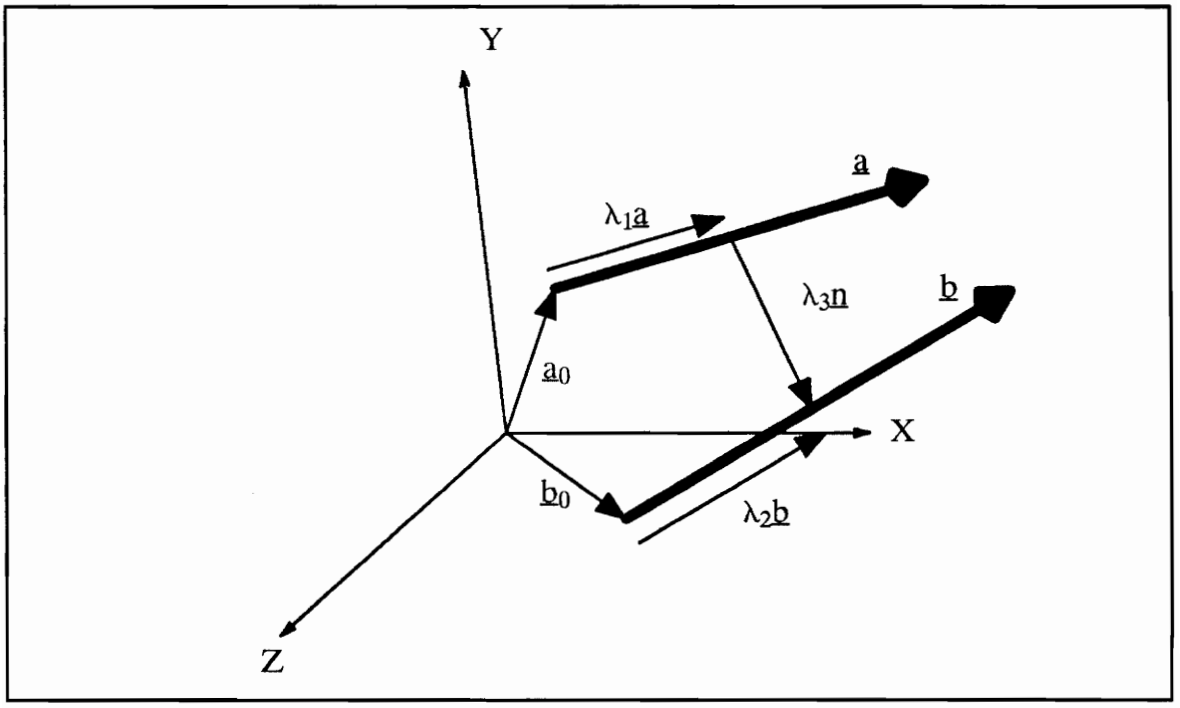

Figure 3. Vector loop for link models

The parallel line segments, shown in Figure 4, must be treated as a special case which requires a different approach. Here, the point at the tail of one vector is projected onto the other vector using the equations

(5) $$\underline{c} = \underline{a}_0 - \underline{b}_0$$

(6) $$\underline{p} = \left[\underline{c} \bullet \frac{\underline{b}}{|\underline{b}|}\right] \frac{\underline{b}}{|\underline{b}|}$$

(7) $$\underline{n} = \underline{c} - \underline{p}$$

where:

$\underline{p}$ $\equiv$ the vector describing the projection onto of $\underline{c}$ onto $\underline{b}$

The conditions for interference in the parallel case become:

(8) $|\underline{n}| = 0$ and either

(9) $|\underline{p}| = 0$ or

(10) $0 < |\underline{p}| \leq |\underline{b}|$ and $\underline{p}$ has the same sense as $\underline{b}$.

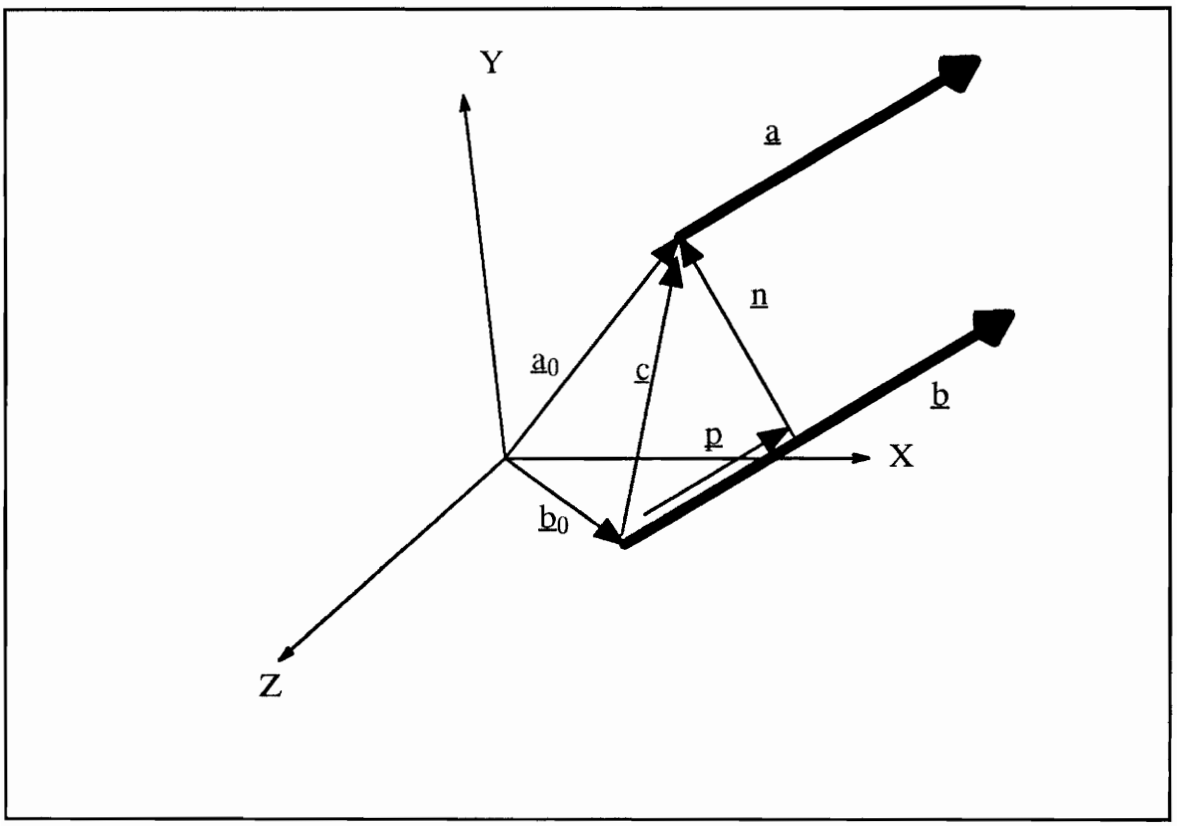

Figure 4. Projections for parallel links

There are two other cases for parallel links. Figure 5 shows the second case where the tail of $\underline{a}$ does not project onto $\underline{b}$, but the head of $\underline{a}$ does. Figure 6

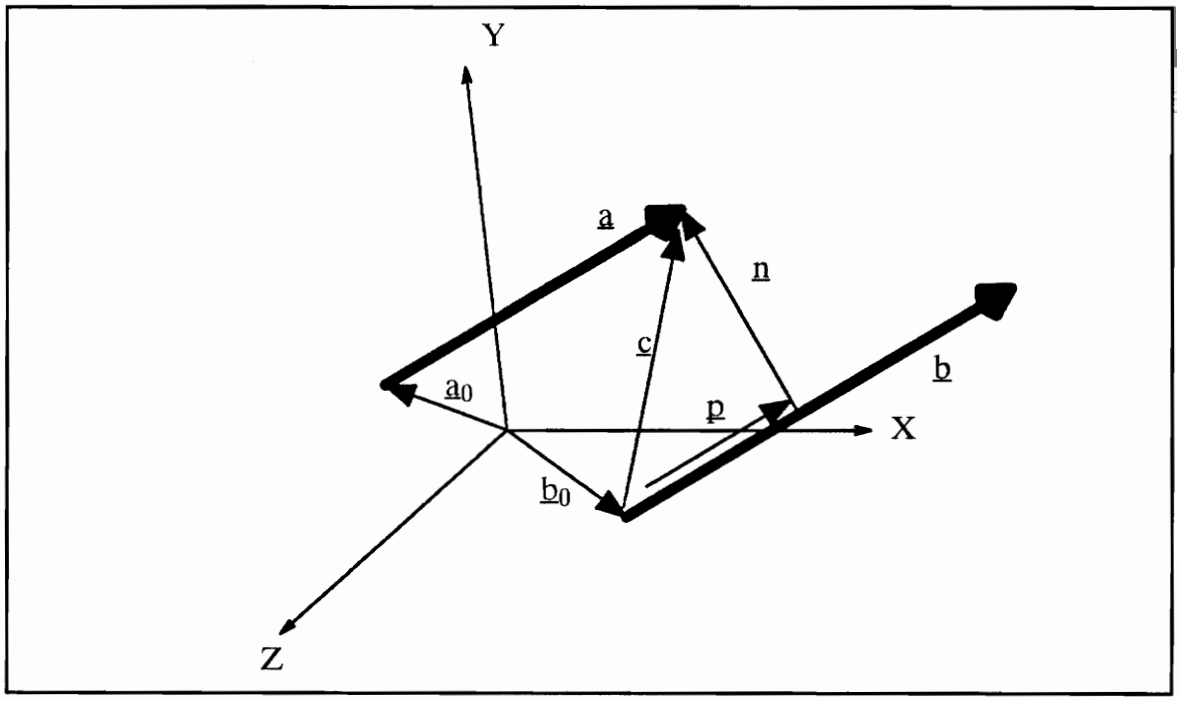

Figure 5. Case 2 for parallel links

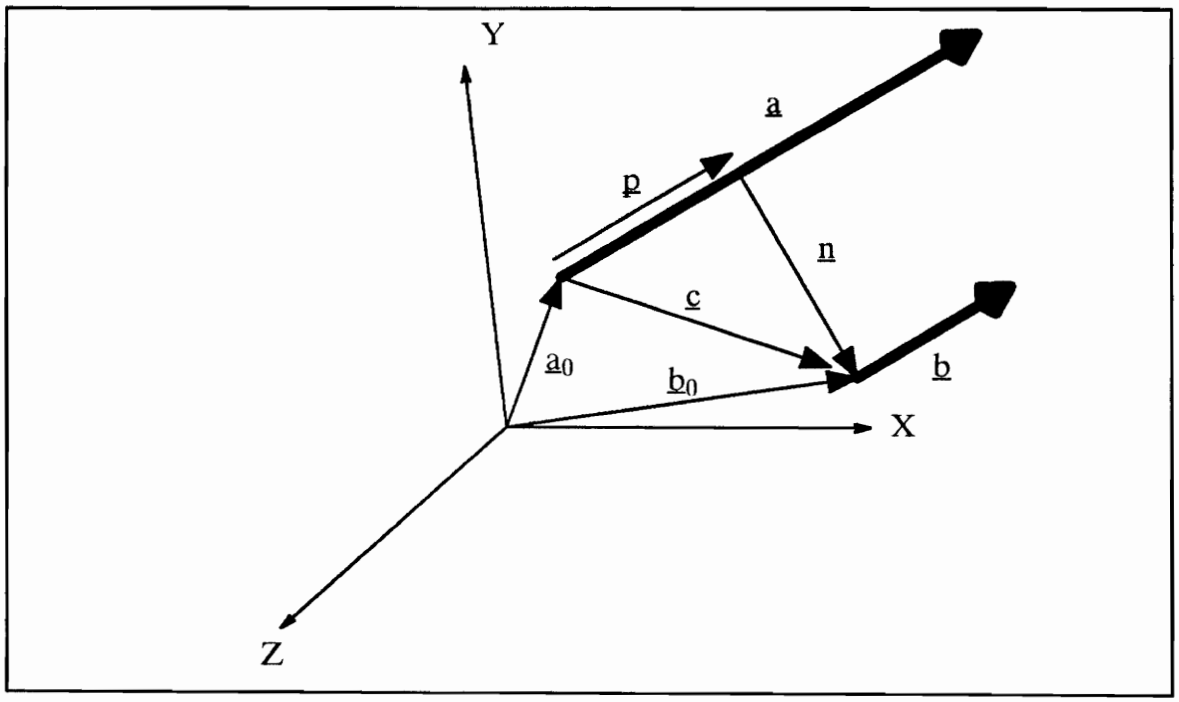

Figure 6. Case 3 for parallel links

shows the case where neither the head nor tail of $\underline{a}$ projects onto $\underline{b}$ but the tail of $\underline{b}$ projects onto $\underline{a}$.

The flow for interference checking using vector analysis is shown in Figure 7.

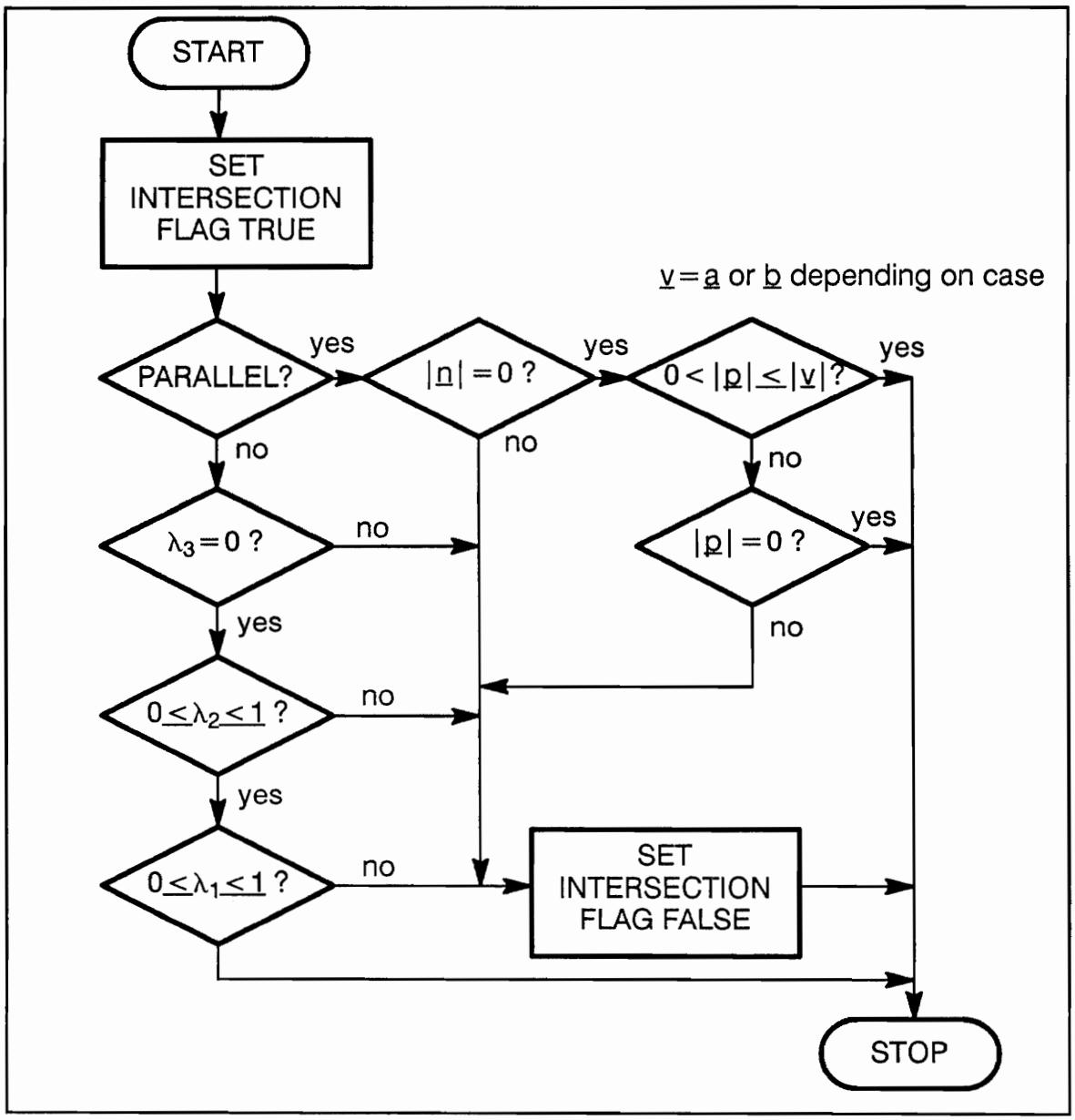

Figure 7. Flow for intersection check using vector analysis

PARAMETRIC EQUATIONS

A variation on the vector analysis approach can be developed through the use of parametric equations. When discussing parametric curves, it can be useful to introduce the concepts of object space and parametric space [100]. Object space is commonly thought of as the space described by Cartesian coordinates. Parametric space provides a means of breaking an N-dimensional object space into a set of two-dimensional spaces where each object space coordinate becomes dependent on one universal parameter. The independent parameter u can be allowed to vary through any range, but it is often convenient to restrict this range to between 0 and 1. An illustration of these spaces is in Figure 8.

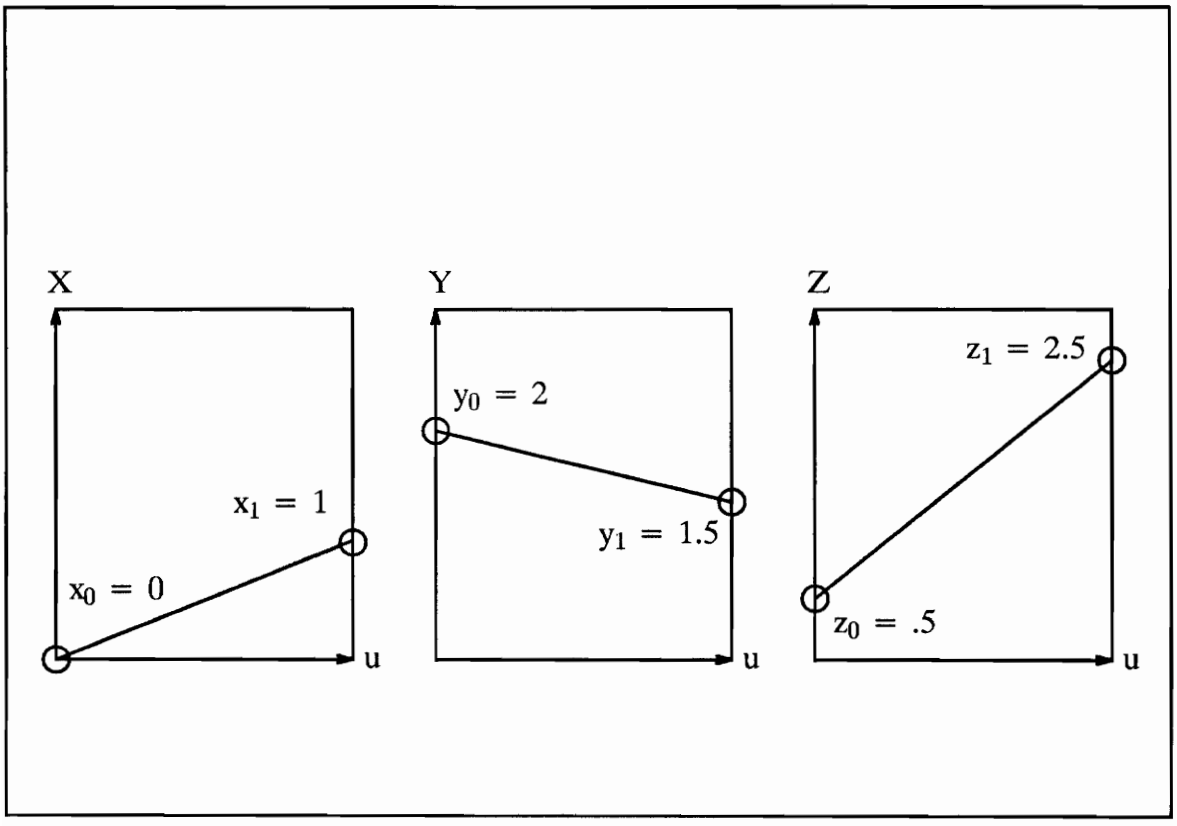

Figure 8. A straight line segment in its 3 parametric spaces

The conditions for intersection, using parametric equations, depend on the value of the independent parameter at points of intersection in each of the two-dimensional spaces. The independent parameter of a curve must be able to take on the same value in all of the subspace intersections. Thus, the problem of lines intersecting in three dimensions is reduced to a problem of determining intersections in two dimensions. The following equations represent the location of points Q and P on two lines whose end points are denoted by the subscripts 0 and 1.

(11) $$\underline{P} = \underline{P}_0 + u\,(\underline{P}_1 - \underline{P}_0)$$

(12) $$\underline{Q} = \underline{Q}_0 + w\,(\underline{Q}_1 - \underline{Q}_0)$$

The solution of these equations results in three pairs of simultaneous equations with the two unknowns, u and w. Intersection requires that u and w must have the same value in all three solutions, and both u and w must lie on the closed interval, $\{0,1\}$. The real power of the parametric approach is realized when it is applied to non-linear space curves; but, in this analysis, only straight lines were used. The flow for the parametric equation algorithm is shown in Figure 9.

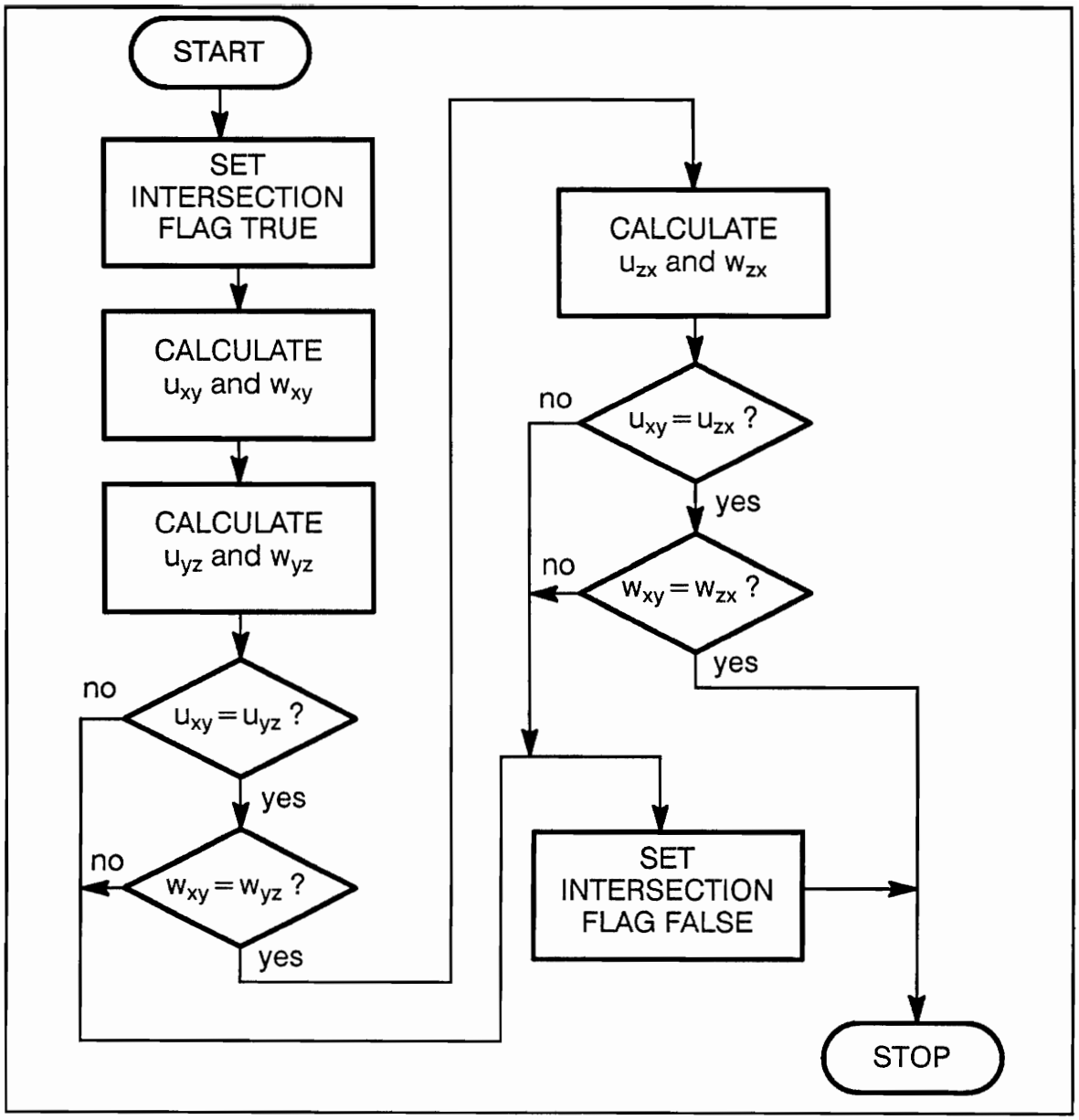

Figure 9. Flow for intersection check using parametric equations

PARALLEL COORDINATES

The method of parallel coordinates was developed by Inselberg [24,25,26] and has been applied to the area of mobility analysis in mechanisms by Cohan

[101,102]. The parallel coordinate method has its roots in projective geometry where the relationship between points and lines can be inverted. Points can become lines and lines can become points. Parallel coordinates provide a means of visualizing this transformation. Figure 10 illustrates this transformation. The

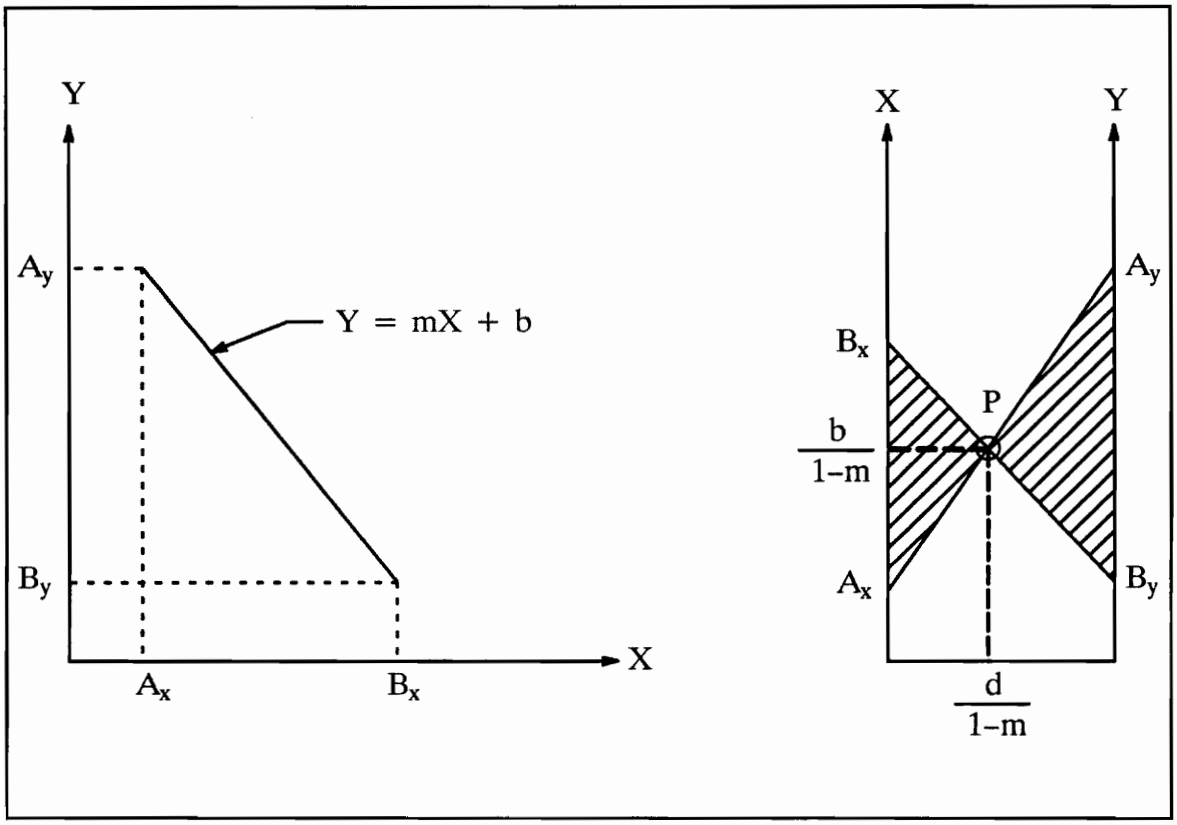

Figure 10. Transformation from orthogonal coordinates to parallel coordinates for a straight line segment

orthogonal coordinates are disassembled and placed parallel to one another. The spacing of the parallel coordinates is arbitrary, but is usually set at unity for convenience. A point is represented as a line connecting the two coordinate values. Once all of the points have been transformed into lines in parallel coordinates they all pass through the same point. This point represents the transformed line. Just as all of the points resided on the line segment (represented

by $y = mx + b$ and the segment limits at A and B), the transformed points (now lines) all pass through the transformation of the line (the point P). By treating the point P as a pivot, the swept envelope (shown shaded) containing all of the transformed points can be created.

The conditions for the intersection of two-dimensional line segments are shown in Figure 11. For two lines to intersect in parallel coordinates, their respective pivot points must be contained in the other's respective sweeps. This is due to the fact that, in parallel coordinates, the pivot represents the line containing the line segment. The line passing between two pivots represents a common point on both lines in cartesian coordinates. The extents of the sweeps in parallel coordinates represent the extents of the line segments in cartesian coordinates. Thus, it is a requirement that a sweep from one extent to another contains the line which represents a point on that line segment in cartesian coordinates. As an illustration of this point, the transformed lines represented by points 1 and 2 intersect. The others do not. This operation can be extended to three or more dimensions by adding the necessary dimensional coordinates. The power of parallel coordinates lies in the fact that the relationship between dependent variables can be visualized for an N dimensional system in a two dimensional format.

A good demonstration of how parallel coordinates can be used to visualize a multidimensional problem is seen in Figure 12. As shown by Inselberg [27], parallel coordinates are used to determine whether or not aircraft are going to collide. The aircraft are represented as three dimensional points with time dependence. The result is the aircraft trajectories are represented as four

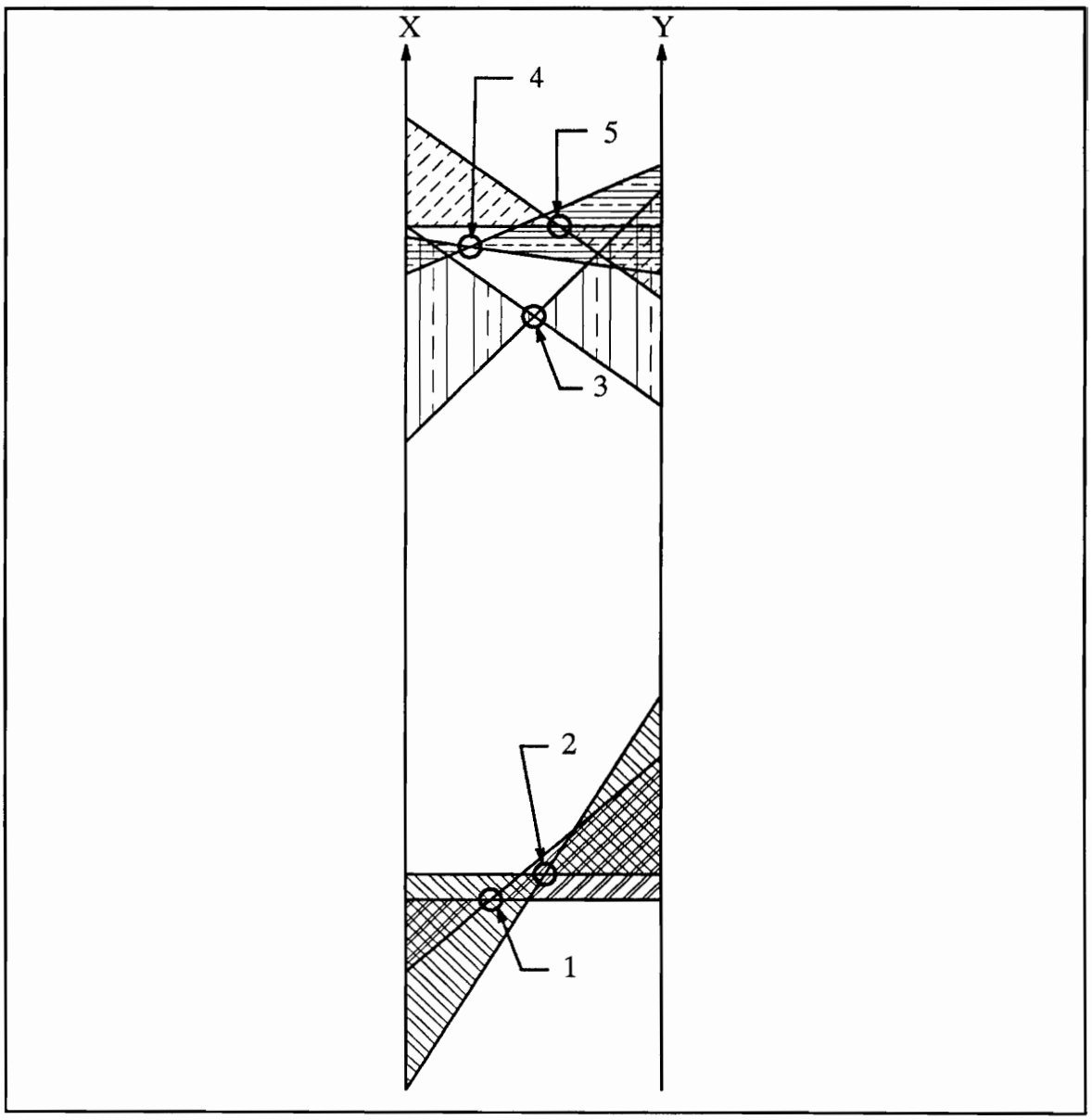

Figure 11. Conditions for intersection in parallel coordinates

dimensional straight lines. Collision results when two aircraft occupy the same location at the same time or when the four-dimensional lines intersect. This sounds intuitively obvious; but, as Figure 13 shows, visualization can be difficult at best. Figure 13 shows aircraft proximity at one instance in time, and it looks as

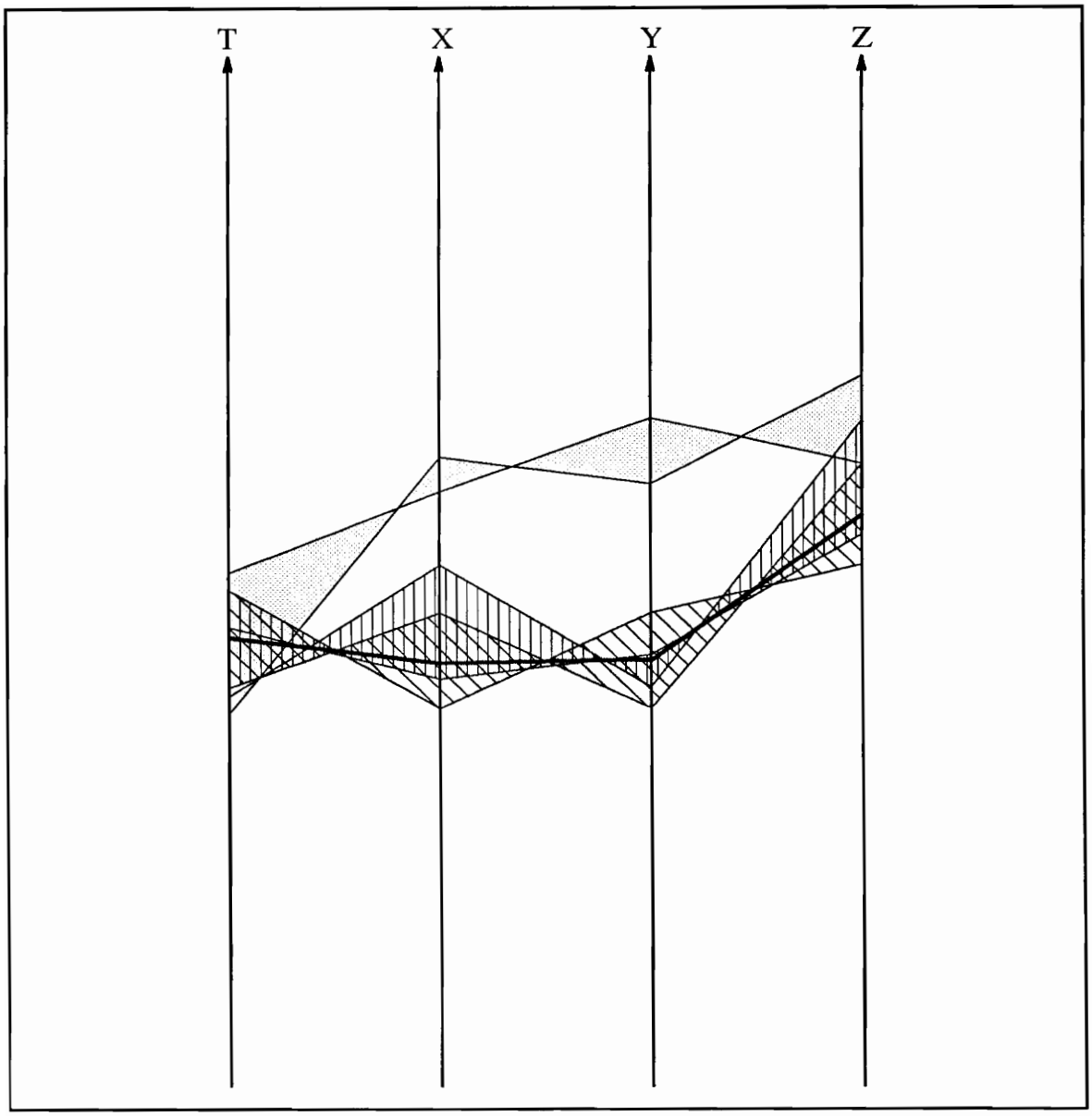

Figure 12. Aircraft trajectories in parallel coordinates showing one collision point

though all aircraft may collide. Figure 12 shows the proximity of all aircraft at all times and shows that only two aircraft will collide and when.

The dark highlight passing through the two lower envelopes shows the intersection. This highlight is a straight line passing from the T (time) coordinate

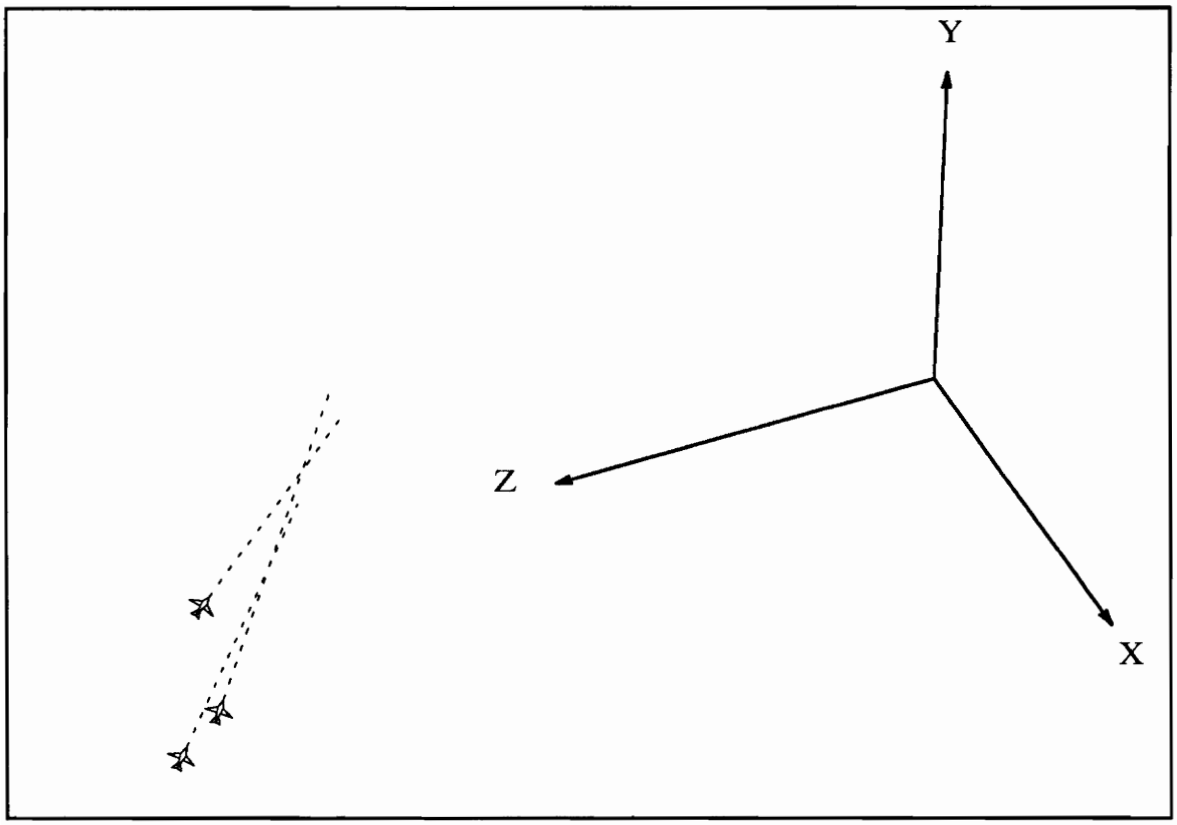

Figure 13. Aircraft trajectories in cartesian coordinates

through the two T–X pivot points to the X coordinate. From this location on the X coordinate, a straight line is drawn through the two X–Y pivots. The process is continued to the Z coordinate. If one line could not have been drawn from the X coordinate through the two X–Y pivots, the intersection requirement would not have been satisfied. This condition would have forced two different values on the Y coordinate.

Inselberg has extended this technique to check for nearness of points. In essence, he uses the time dependence to position the points, and then checks their separation on the three position coordinates. If any position coordinate shows a separation less than a specified bound, an additional proximity test is performed.

The first test requires very little calculation and can be performed much more quickly than a test which determines actual proximity. In a situation where many objects are to be tested, this quick test can be very valuable. Unfortunately, while the test works very well for points moving in time and space, it does not work for line segments moving in time and space.

The flow for intersection checking in parallel coordinates is shown in Figure 14.

INVESTIGATION APPROACH

Algorithms were developed based on vector analysis, parametric space, and parallel coordinates. A random number generator was then used to generate 2,000 line segments confined within an 18 unit cube (18x18x18). The line end points were assigned integer values for their X,Y,Z coordinate values. This increased the likelihood of intersecting line segments. The number of combinations of 2,000 lines taken two at a time is 1,999,000. Of the 1,999,000 possible pairings, 3,238 pairs (0.16%) intersected; two of which were collinear. These randomly generated lines segments were a test group for testing the relative efficiency of the algorithms.

Checking large numbers of line segments for interference detection of a spatial mechanism may take significant computer time. Any measure of performance should not be affected by system load. A system dependent function was used to measure real machine cycles used by each algorithm. This function was invoked at the beginning and end of a run to return the system time used by a given algorithm. There was a significant amount of run-time needed to read and process the data for the line segments. To eliminate this overhead, all of the test

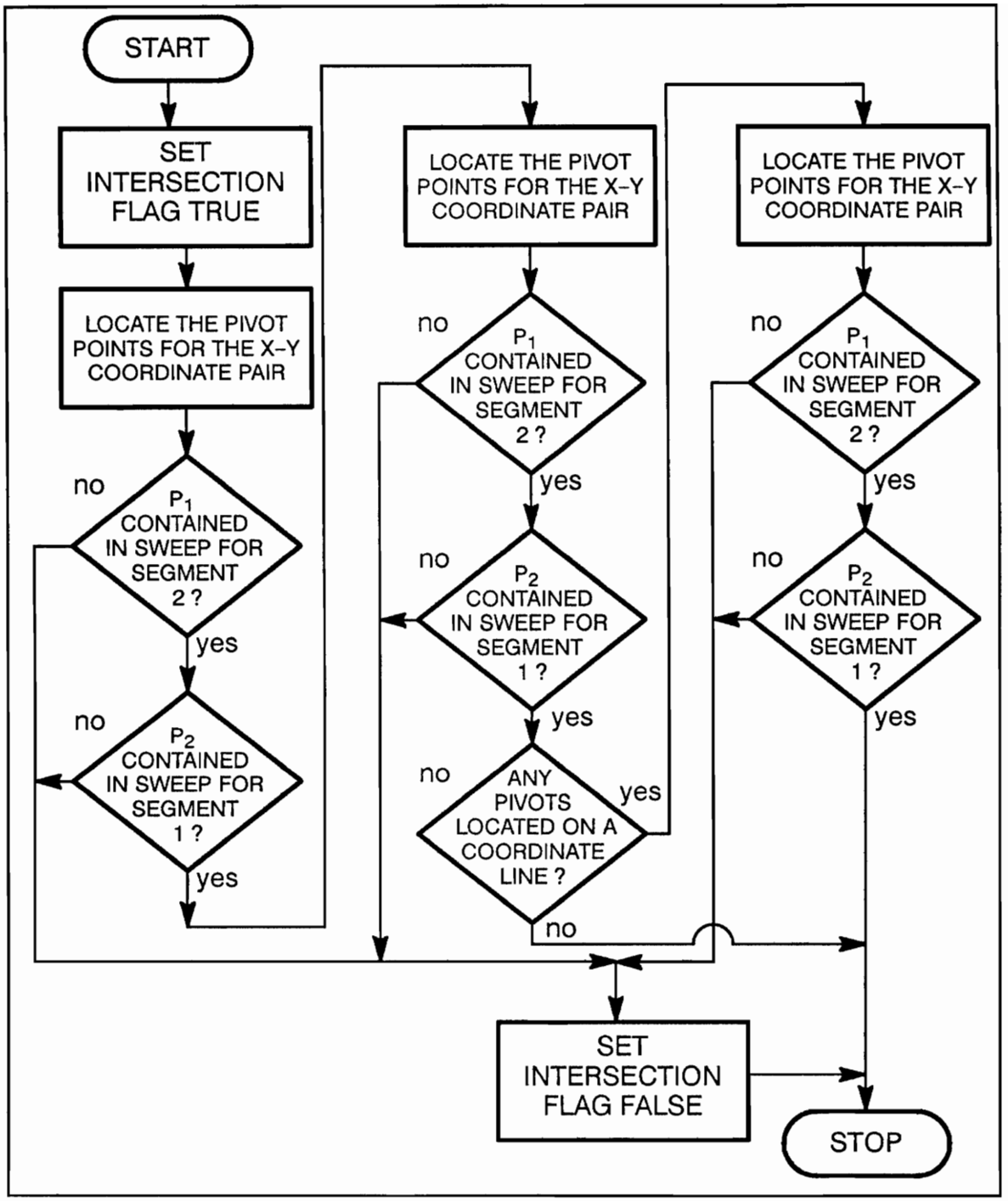

Figure 14. Flow for intersection check using parallel coordinates

algorithms were subroutines called by the main program. A dummy subroutine was created which simply passed control back to the main program. The time to run through the data with the dummy subroutine was subtracted from the run-times of the other algorithms resulting in the interference detection time only.

RESULTS

The results of the testing listed in the table show the relative performance of the three algorithms normalized to that of vector analysis.

Vector Analysis	Parallel Coordinates	Parametric Equations
1.00	1.16	1.13

Relative comparison of run times

The author believes the differences are insignificant. The vector analysis routine existed before this study was started [14] and was streamlined further for this testing. The differences are believed to be more a result of the degree of code optimization than any algorithmic differences. Indeed, the number of operations were approximately the same for all algorithms.

Intersection detection could not be made faster, but the real problem was nearness, not intersection. An algorithm for determining the exact relative location of mechanism elements is reported in [14]. The algorithm, using vector analysis, was developed when interferences were discovered in 3-D models of planar mechanisms as illustrated in Figure 2. The efficiency of the algorithm was improved approximately 20% during this study, but significant improvements were not expected beyond that.

The method of parallel coordinates provided the insight for a more efficient detection scheme. Figure 15 shows two line segments that do not intersect. The

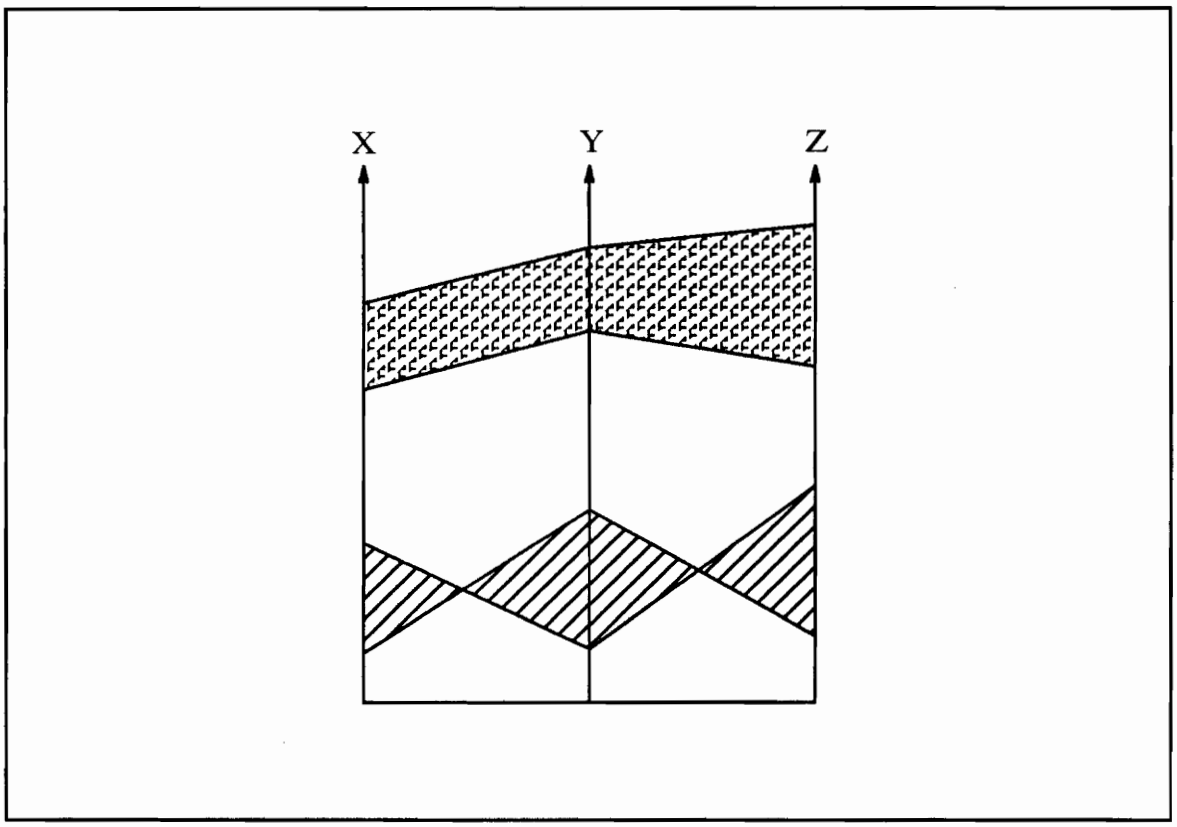

Figure 15. Nonintersecting line segments in parallel coordinates

three dimensional information presented in a planar format shows the conditions that guarantee an entity does not interfere. If the entities do not overlap on any of the coordinates, they do not interfere. This test requires no calculation at all, simply a series of inequality tests. Lines detected by this test were processed 5 to 20 times faster than the vector analysis algorithm. The differences noted here were dependent on how deep into the conditional test a pair of lines had to go. Lines that were caught by the first condition of the test were processed 20 times

faster. Lines that were not caught until the last condition of the test were processed 5 times faster. The flow chart in Figure 16 illustrates this

It was established that this conditional test could rule out the intersection of certain lines very quickly, but it was not known how significant this might be. The test was then used on the 2,000 randomly generated line segments. Of the 1,995,762 non-intersecting pairings, 1,310,403, or 65%, of the pairings were caught by this test. When the test was incorporated into the vector analysis algorithm, the time to process the 1,999,000 pairings was reduced by 48%.

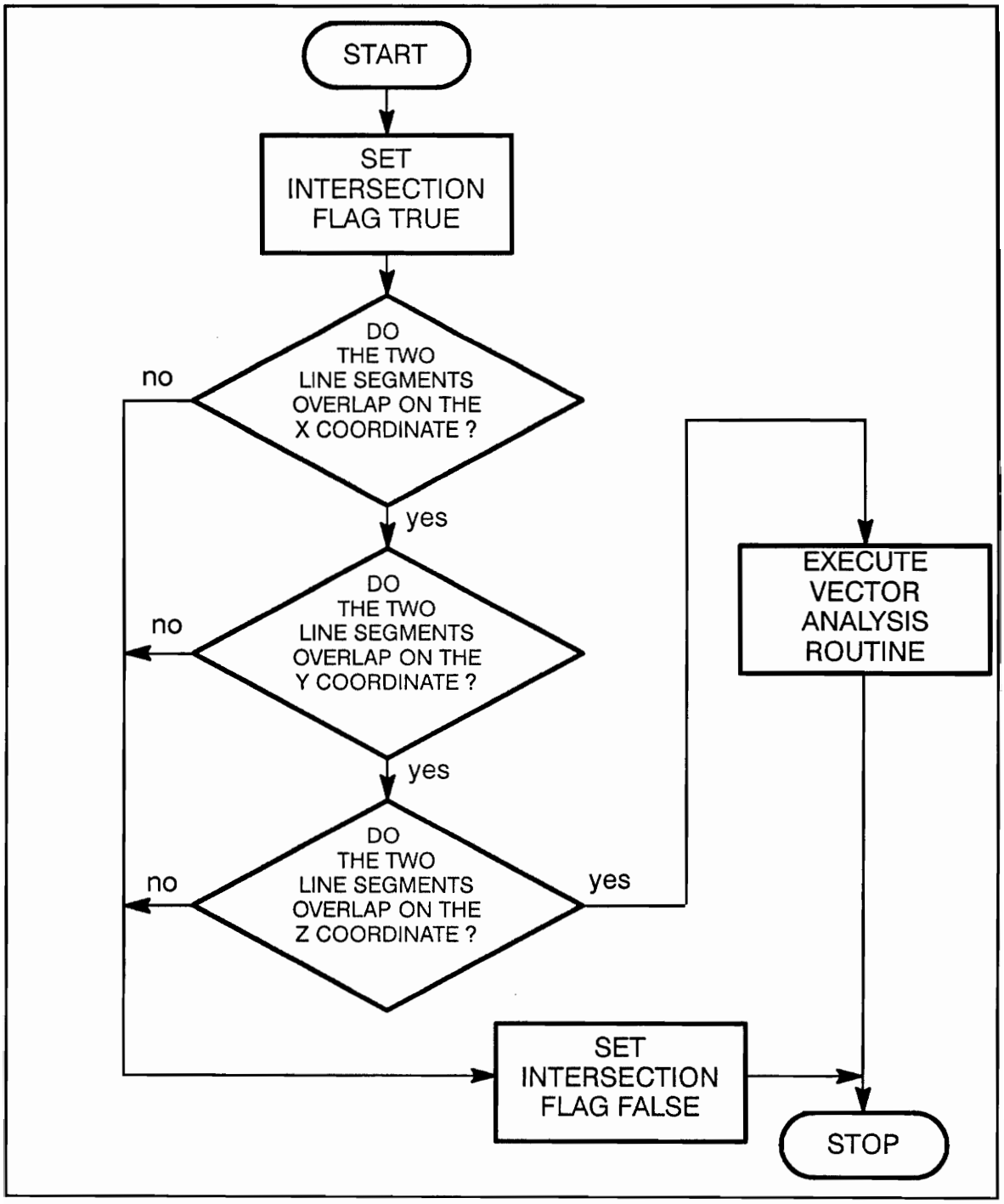

Figure 16. Flow for the coordinate overlap test

Chapter 5

Initial Considerations in Interference Elimination

The problem of generating interference-free spatial mechanisms was an open ended one. The only constraints were those contained in the kinematic description of the mechanism and the characteristic sizes of joints. These constraints were to be supplied in the input files. The kinematic description determined the location for each joint in space. The relative movement which each joint allowed was also supplied in the kinematic description. For example, if a joint was to be cylindric it would allow relative rotations about an axis and translations along that axis.

The input files also contained information on the size of the joints. Thus, it was easy to generate bounding round-ended cylinder and sphere models of the joints. Figure 17 illustrates the elements used to model joints for interference detection. The elements could be sized arbitrarily based on a user's entries into the input files. A complete description of the sizing variables is given in the section on Input File Formats.

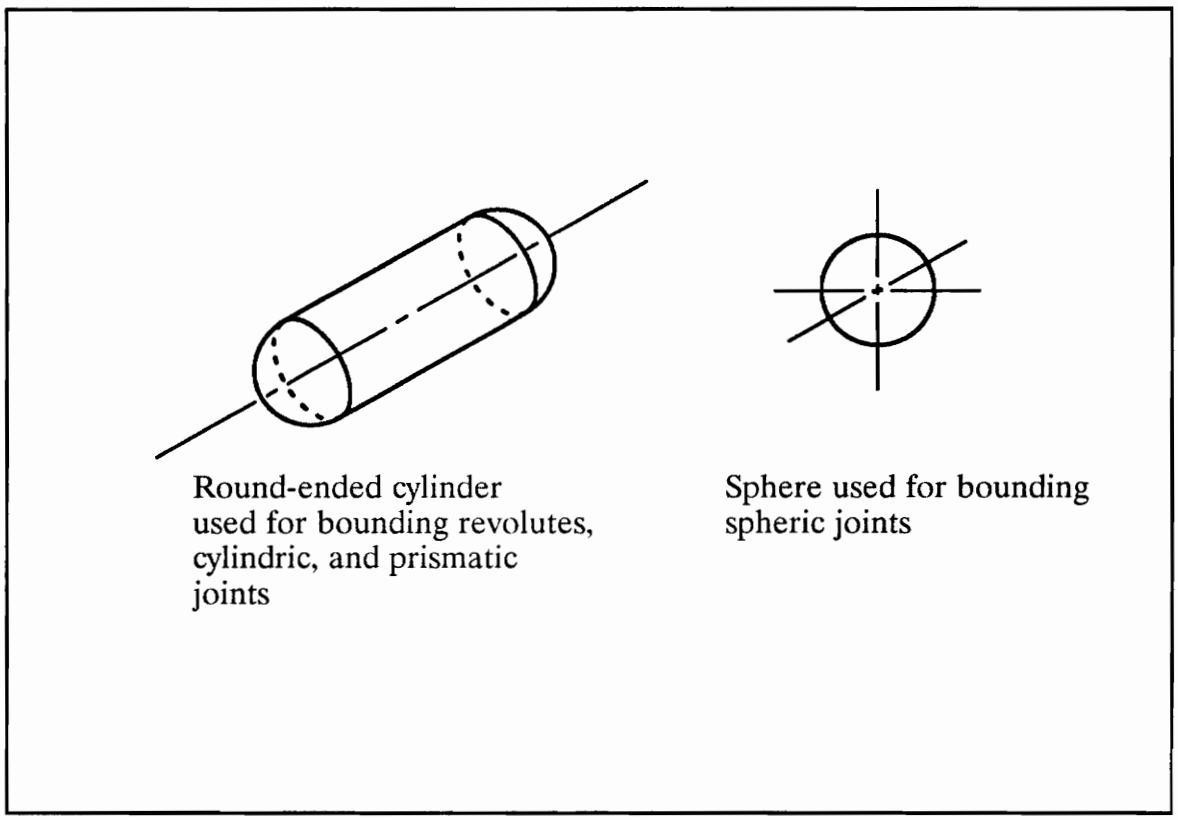

Figure 17. Bounding elements for joint models

Joint Connectivity

Positioning and sizing the joints was fairly simple, but connecting them was not. Pennington [12] established a constraint of orthogonal attachment for elements which connected to joints. This proved to be an important constraint for the external parts of a joint pair. An external piece of a revolute joint pair, for example, could not have an attachment which lined up with its axis of revolution since the internal portion of the joint had to reside there.

Associated with connectivity, another problem developed. This related to a body sweeping around a joint in a manner which contained the joint. Figure 18

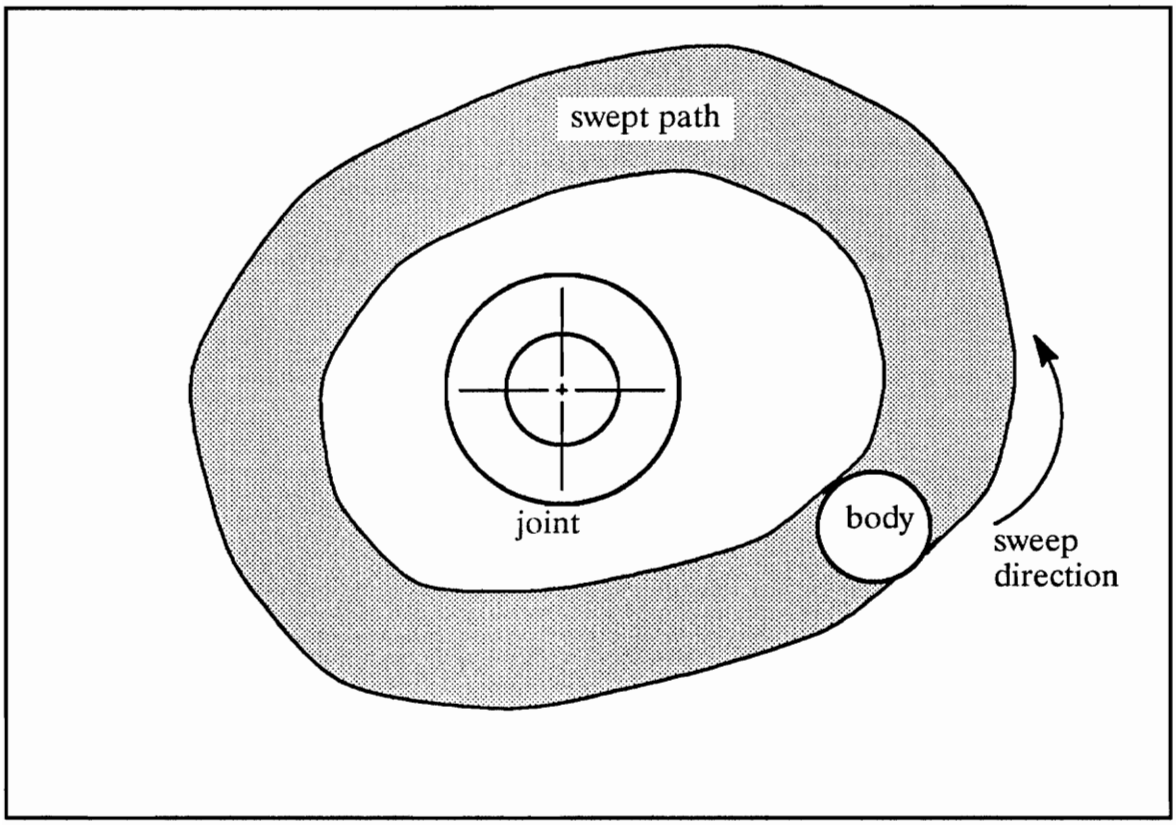

Figure 18. A joint being contained by a sweeping body

shows a planar example. Any element connecting to the joint from outside the swept path would collide with the sweeping body at some position.

The orthogonal attachment constraint was part of the connectivity solution necessary to join two joints. Pennington proceeded from here to establish a plane orthogonal to a joint's axis at the midpoint of the joint. A line of intersection between two joint planes was then determined. An algebraic solution produced the shortest two attachment elements which connected the two joints with these constraints, but this method was not always satisfactory. The line of intersection could be a great distance from the joints; or, in the case where the joint axes were parallel, it did not exist at all. In these cases, a third element was created. Figure

19 shows the third element for attaching joints with parallel axes. The two

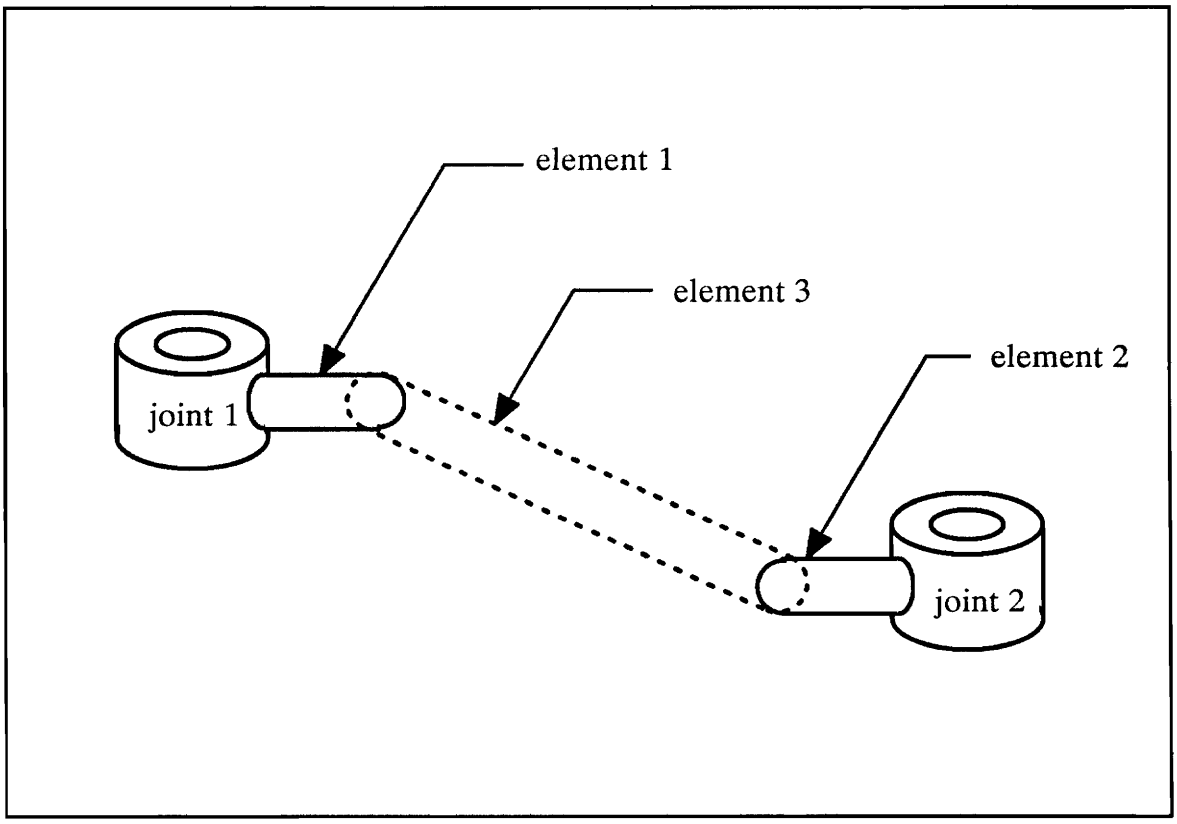

Figure 19. Creating a third connecting element for parallel joints

element method was too restrictive for interference elimination for these and other reasons. Another significant case was that of joints existing on the line of intersection. In this case, the two elements were constrained to lie along the line of intersection and repositioning the elements was impossible.

Two connecting elements were insufficient, but there was a question on whether breaking the third element into more elements was beneficial. A smooth curve between elements 1 and 2 would have represented the most extreme case. The method of avoidance made this difficult to deal with. The method of avoidance involved moving the connecting elements when an intruder came into

contact with a connecting element. There were cases where a smooth curve would start wrapping around the intruder. Given a finite number of positions it may have been possible to find a configuration which did not interfere but was completely impractical. Figure 20 illustrates this point. Positions of a possible intruder are

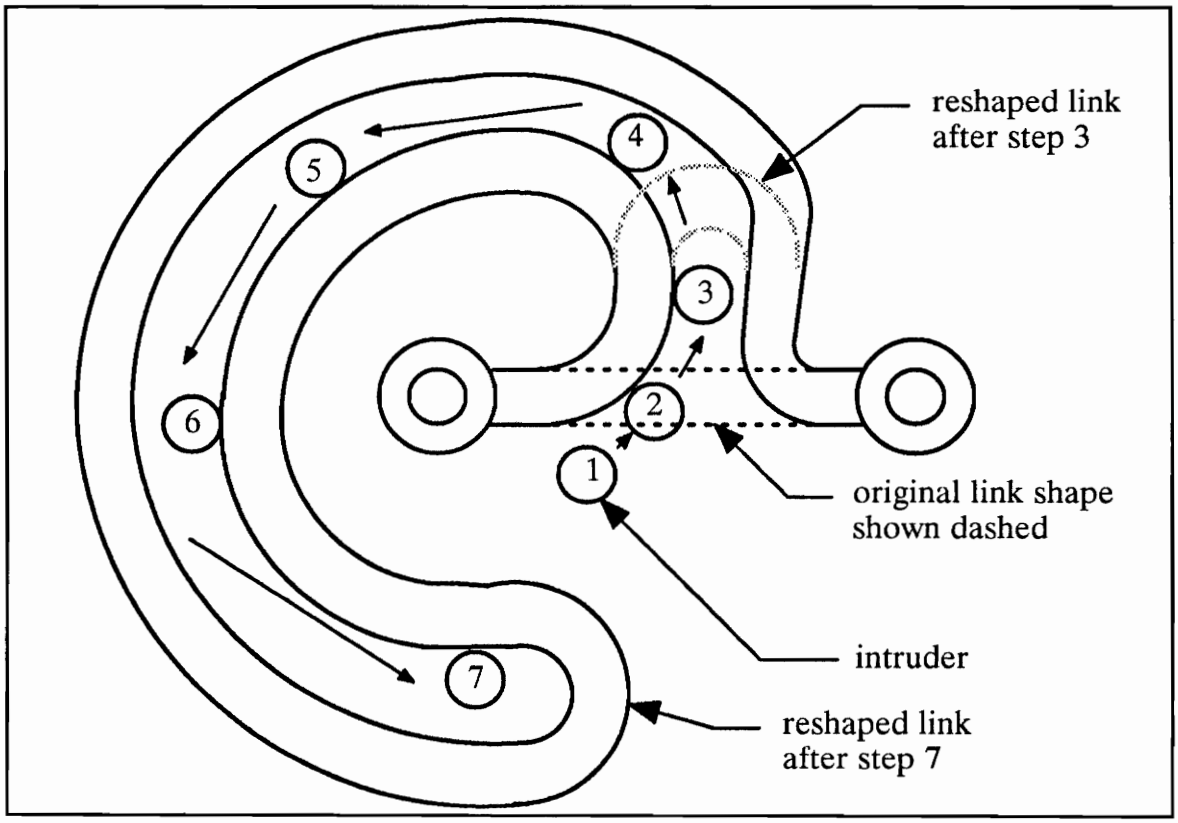

Figure 20. Intruder causing a smooth curve to wrap

numbered 1 through 7. This was not considered to be an uncommon occurrence since any crank element could have been expected to see a intruder with such motion relative to its local frame of reference. A smooth curve was not acceptable either. Three connecting elements were sufficient while four began to introduce problems with the method selected for avoidance.

The avoidance method is outlined in the flow chart in Figure 21. The element numbering was shown in Figure 19. The order in which elements 1 and

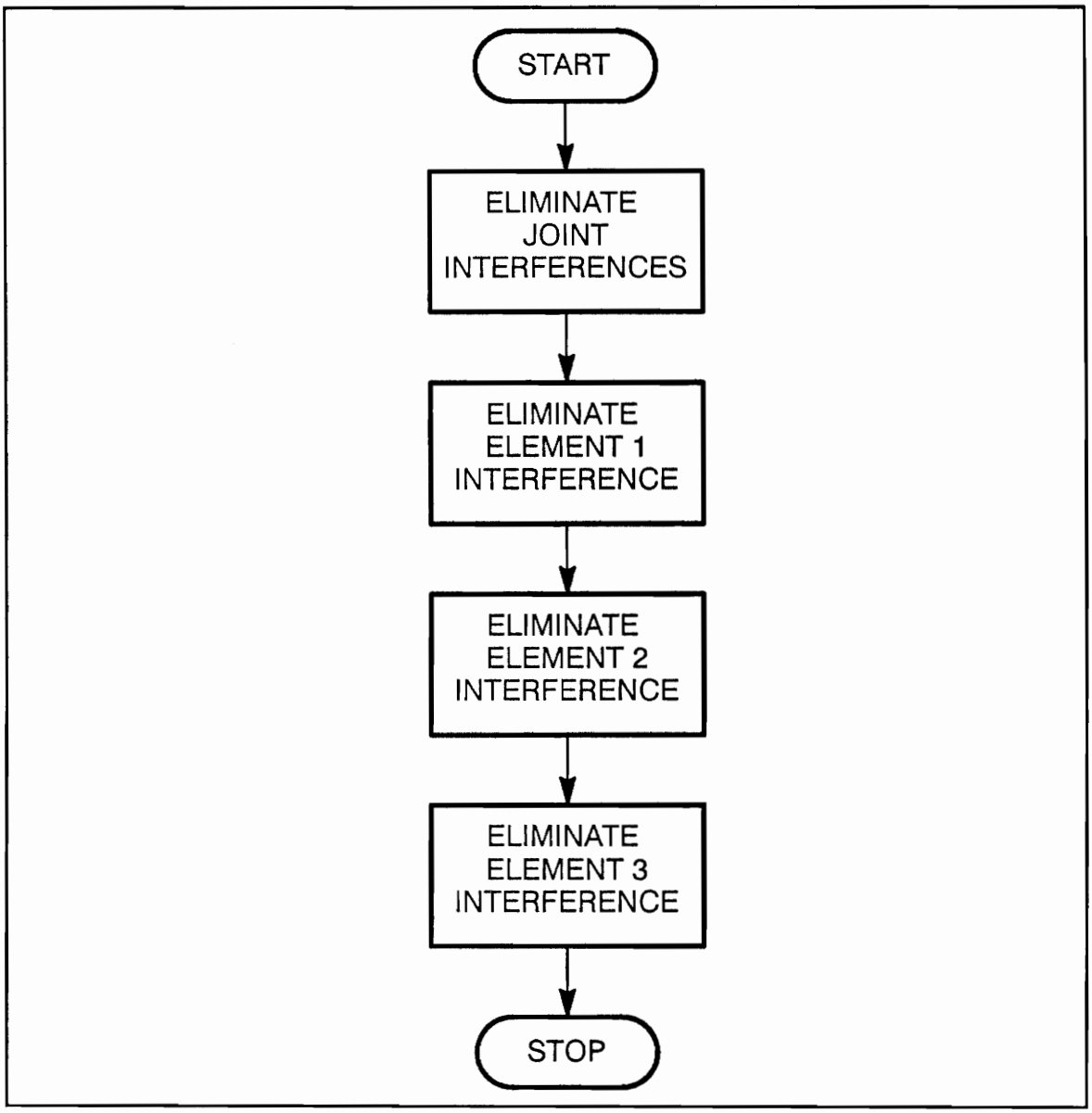

Figure 21. Interference avoidance flow

were addressed was totally arbitrary, but the order of the joints and element 3 in this flow was not. This flow avoided conditions where repositioning would be

impossible. Figure 22 helps to illustrate this. Elements 1 and 2 are assumed to

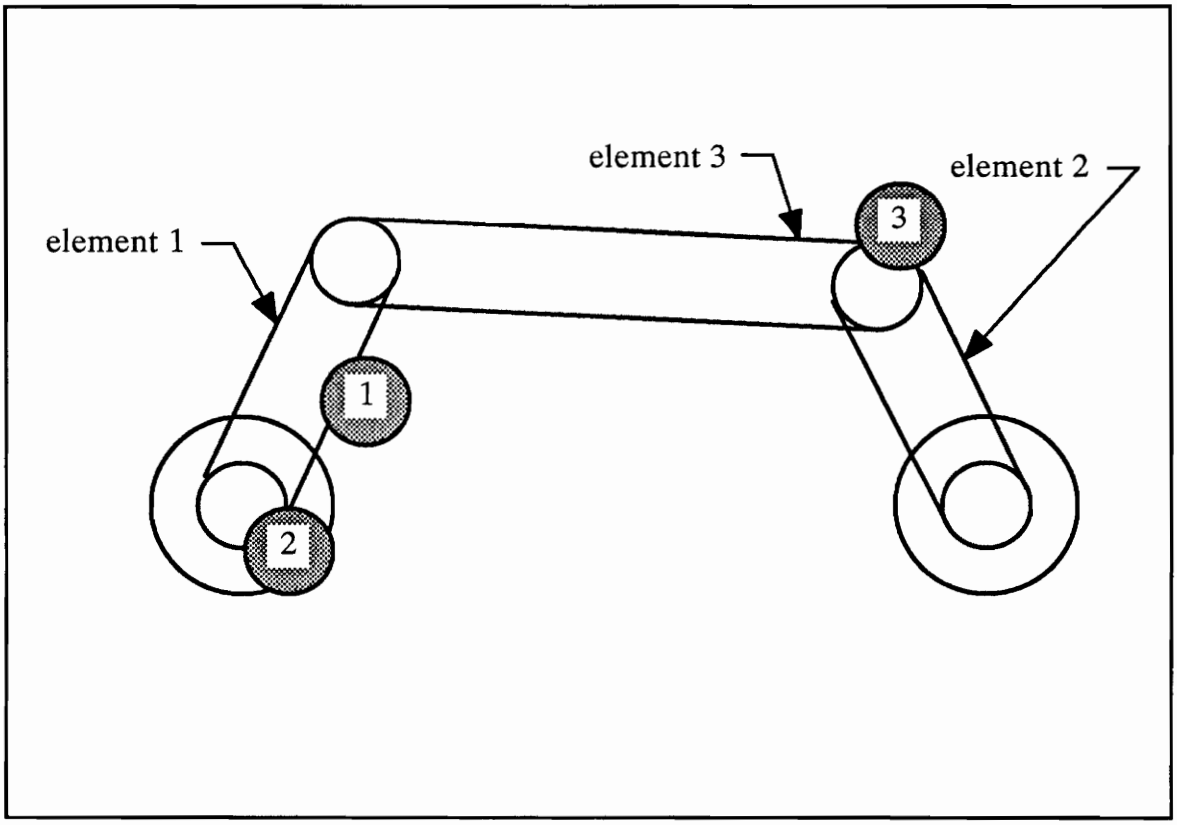

Figure 22. Possible intruder cases

penetrate to the center of the joints to which they are attached. Element 3 is assumed to share spherical regions at the ends of elements 1 and 2. Three intruders are shown. The intruder numbered 1 is shown in collision with link element 1. The joint to which element 1 is attached is assumed to be a revolute joint with its axis perpendicular to the page. To avoid intruder 1, element 1 is rotated about the joint axis in either a clockwise or counterclockwise direction. Intruder 2 also interferes with element 1, but it is impossible to rotate to a position where element 1 does not interfere. By requiring the joint to be

repositioned first, intruder 2 can never interfere as shown. Similarly, repositioning elements 1 and 2 precludes intruder 3 from interfering with element 3 as shown.

Element 3 avoids collisions by causing element 1 or 2 to lengthen. If the condition to lengthen element 2 was invoked, the collision could be avoided; but, if the condition to lengthen element 1 was invoked, there would be no solution. Causing elements 1 and 2 to avoid collisions before checking element 3 increases the likelihood that element 3 can be repositioned successfully. This still does not guarantee that element 3 can be repositioned successfully. Figure 23 helps to clarify this point. The intruder in this case causes element 3 to swing to an angle

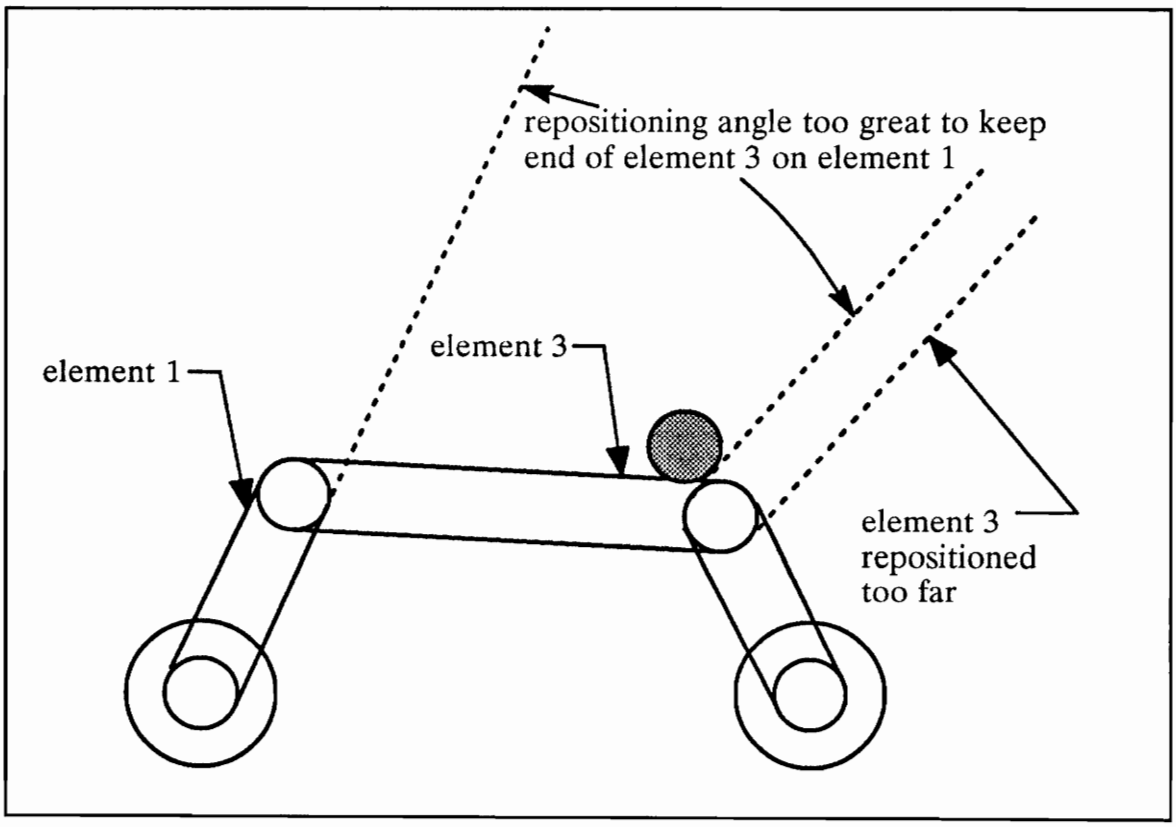

Figure 23. Repositioning case which fails for element 3

which does not allow the length of element 1 to grow in a positive sense. Adding

a fourth element only magnifies this problem and also makes it difficult to keep the intruder condition 3 in Figure 22 from happening.

Joint Repositioning Freedoms

Interfering joints were repositioned before interfering links were repositioned. This was necessary to avoid collisions at instantaneous rotation centers for the link elements. This prompted a study of how joints could be repositioned. It was curious to note that, while this problem is always addressed when mechanisms are manufactured, it is never discussed in any design text. The study began by stating the function of a joint. The function of a joint according to Shigley and Uicker is to constrain the relative motion of two links in a mechanism [103]. Thus any change in the position of a joint which did not alter that relative motion was allowed. An application of this concept to a revolute joint is shown in Figure 24. Cylindric joints have the same freedom in repositioning as the revolute. One joint which cannot be repositioned is the spheric joint. Repositioning a spheric would change the constraints imposed by that joint.

A prismatic joint has no limits on where it can be repositioned, but it cannot be rotated relative to its local reference frame. This would alter its kinematic constraint. As long as the axis of translation remains parallel to the original axis of translation, the kinematic constraint for that joint remains the same. This does not imply that it is advantageous to use every degree of freedom in repositioning a joint.

A successful avoidance strategy guarantees that, once an object is moved from a position where interference is possible, it is never moved back to that position. If the object is made to move in one direction along a straight line it will

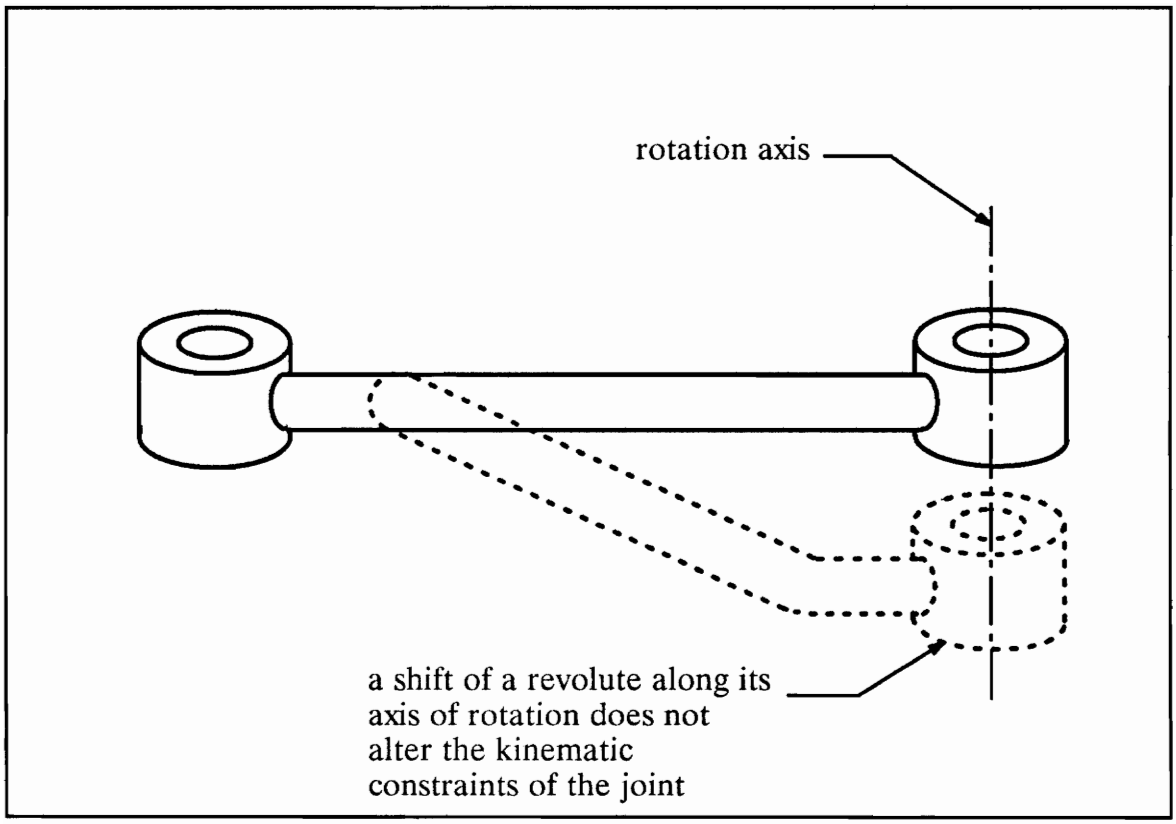

Figure 24. Repositioning a revolute while maintaining its kinematic constraints

always be able to avoid interferences at a finite number of locations. The object never retraces a bad path. In two dimensions, the problem becomes more difficult. The object can move in a circle. For movement less than 360°, there is no problem; but, for any movement greater than 360°, the object is retracing a bad path. An avoidance scheme based on potential depth of penetration of an intruder into the object could allow this to happen.

The requirement of never retracing a bad path lead to a reduction in the degrees of freedom allowed in repositioning a prismatic joint. Prismatic joints were constrained to be repositioned along their translation axis only. This reduction in freedom reduced repositioning of all movable joints considered in this

work to the same problem. One more constraint on repositioning was necessary to guarantee a solution.

There were two options for movement along an axis: movement in a positive direction, or movement in a negative direction. One possible method for avoiding a collision is to consider the minimum movement distance needed to avoid the collision. This strategy was employed by Dai [95] and Buchal and Chercas [94] as well as others in dealing with their faceted models of robots. This method was abandoned, because it was possible to encounter collisions from either direction at successive positions of the mechanism. The approach adopted was to establish one direction for avoidance. Any collision could be avoided by requiring the joint to move in one direction. The reasoning behind this is that a finite number of objects of finite volume occupy a finite volume, and moving the objects to a finite number of positions to increase this original volume creates another finite volume. It is then a simple matter to move an object of interest along a straight line a finite distance until the following equation is satisfied:

(13) $$\forall_t \cap \forall_o = 0$$

where:

$\forall_t \equiv$ the set of object volumes positioned with respect to object o

$\forall_o \equiv$ the volume of object o

The only criteria for failure imposed on this scheme was if the separation between two joints on a link exceeded a distance of 30 times the shortest link length. If a failure was encountered, the reverse direction was considered, but it motion in the

reverse direction was not allowed until it was determined that a given repositioning option was exhausted.

These initial considerations determined much of the structure for dealing with interferences. The three element approach was adopted for modeling connecting links. The flow for interference avoidance always begins with joints. Joints are checked against joints, then links are checked against joints, and finally links are checked against links. The approach of not allowing retrograde motion in avoiding collisions was used not only on joints but also as the strategy in avoiding collisions in link elements.

Chapter 6

Input File Formats

The input file formats for SLIDE are similar to those used for ANIMEC [1] and GENMOD [12]. In fact, they represent an evolution of the formats. Two files are required as input to SLIDE: an attribute file, and a position file. Figure 25 illustrates the structure for these files. The FORTRAN formats for writing each line are given in the square braces next to the fields. The variables in these files are described as follows:

ID–FILENAME – an 8 character identifier

TYPE – an 8 character field (used by GENMOD)

MECHANISM NAME – an 80 character name for a mechanism

R,G,B – red, green, blue color values

RO – the outside radius for a non-spheric joint

RI – the inside radius for a non-spheric joint

RL – the half length for a non-spheric joint

RS – the outside radius for a spheric joint

ICODE – used to specify a mechanism or a link (1 or 2) in GENMOD

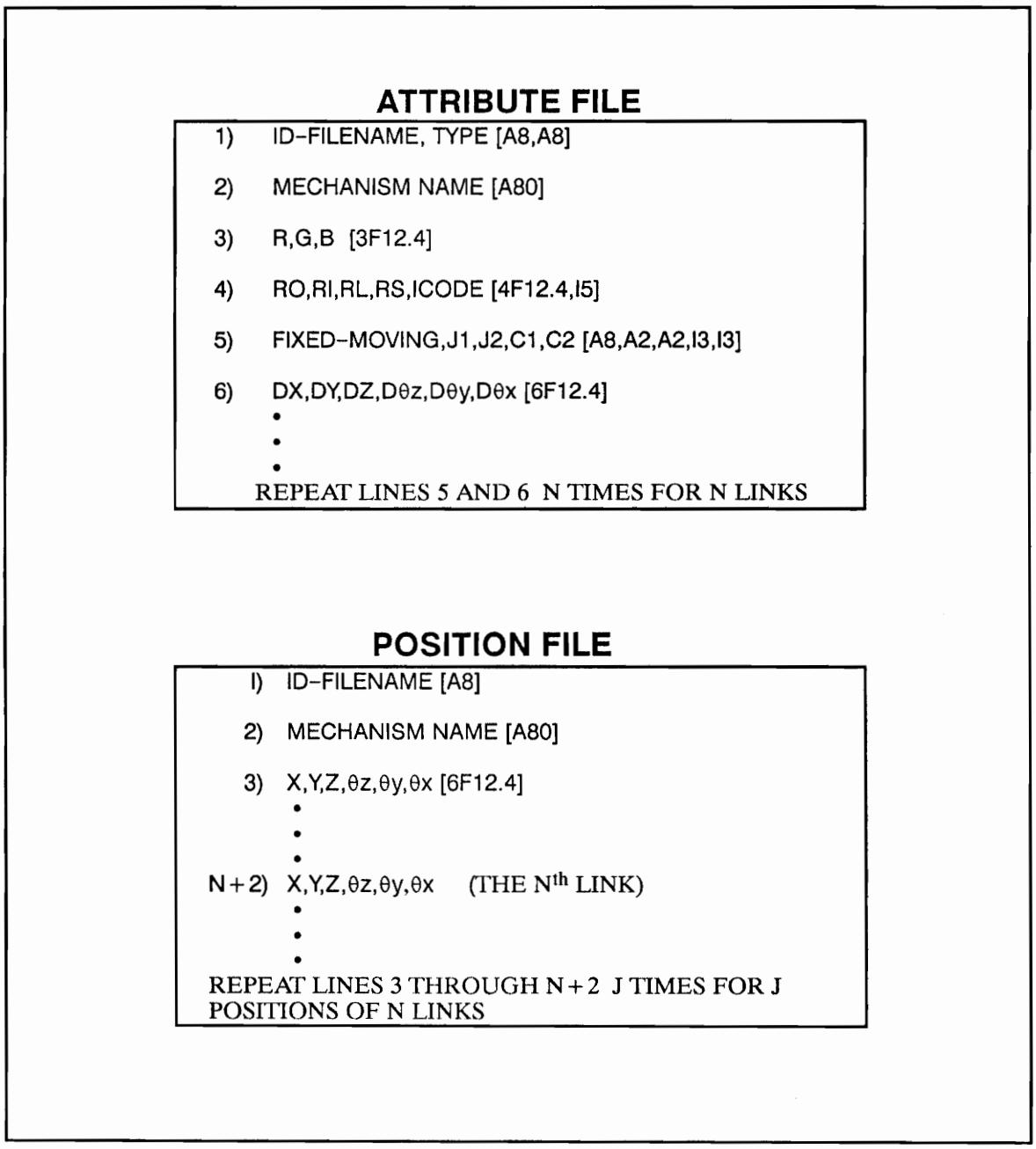

Figure 25. SLIDE input file structures

FIXED–MOVING – used to specify a moving or fixed link

J1,J2 – describes joint type (C for cylindric, P for prismatic, R for revolute, and S for spheric) and configuration (I for internal, and E for external) a typical entry would be RE for revolute-external

C1,C2 – gives joint connectivity, as an integer, for end 1 and end 2 of a link

DX,DY,DZ – relative displacement coordinates for the second joint in the link's local reference frame

Dθz,Dθy,Dθx – successive rotations applied to the second joint in the link's local reference frame

X,Y,Z – coordinates used to position a link in a mechanism

θz,θy,θx – successive rotations used to orient a link in a mechanism.

The first joint in a link is always assumed to be at the link's origin with its axis of rotation collinear with the link's local Z axis. The second joint is then rotated and positioned in this local frame of reference as illustrated in Figure 26. These positions may be modified later by SLIDE, but this is convention for input. For the user who wishes to model a synthesized mechanism (or just one link), it is a simple matter to identify the location vector of a joint and its Z axis as the primary reference point in a link. The Z axis is the axis of rotation for revolutes and cylindrics, the axis of translation for prismatic joints, and the Z axis is totally arbitrary for spheric joints. Unit vectors orthogonal to the joint's local Z axis can be constructed to represent the joint's local X and Y axes. The position vector to the second joint can be determined along with the X, Y, and Z axes for the second joint. These vectors can be used directly to fill the offset values required in the

attribute file. Once these vectors are found, a transformation is applied to align the local origin of the first joint with the global origin. This transformation is applied to all points on the link to achieve a rigid body transformation to the new position and location. Equation 14 represents this transformation.

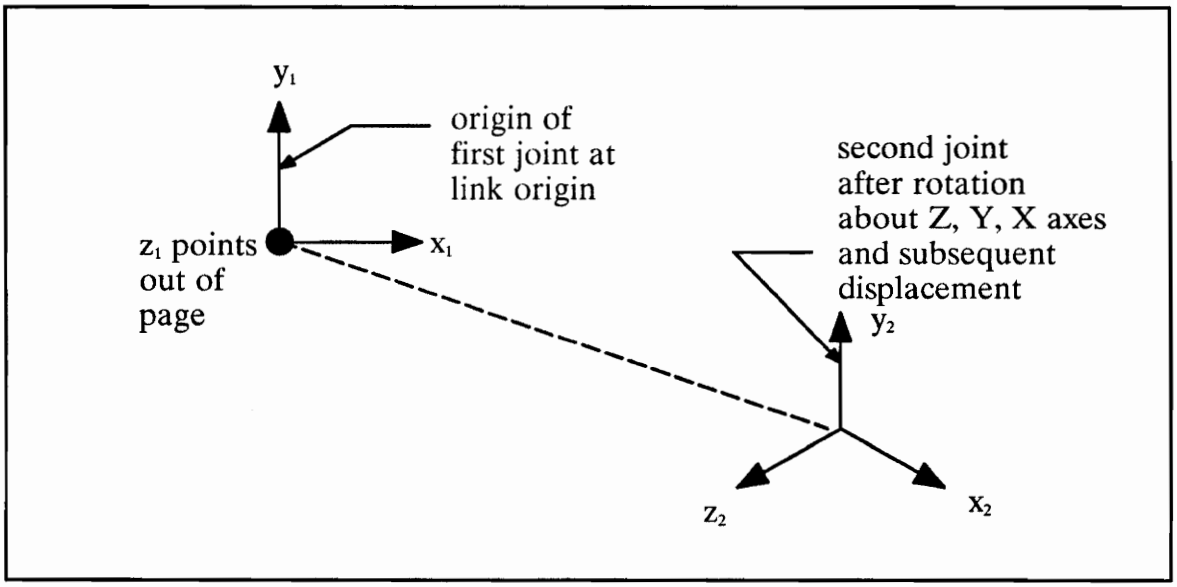

Figure 26. Positioning of origin triads for joints in a link

$$(14) \qquad \begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix} = [R\theta_z][R\theta_y][R\theta_x] \begin{bmatrix} X_p - X_1 \\ Y_p - Y_1 \\ Z_p - Z_1 \end{bmatrix}$$

where:

$X_i, Y_i, Z_i \equiv$ the cartesian coordinates of point i (the subscript 1 represents the location of the first joint, the subscript p represents the point to be transformed, and the subscript 2 represents the transformed point)

$[R\theta_i] \equiv$ the rotation matrix for rotation about axis i

The cartesian coordinates of the first joint and the inverse of the rotations at each position of the link can be used to fill the the position file. Another option is to use IMP to help generate these files.

Any model entered into IMP can be processed into SLIDE input with the addition of reference points representing each link's local coordinate triad. IPOST then processes the IMP output to create the attribute and position files shown in Figure 25. IPOST assumes that this output file was created by IMP75, the last public domain release of IMP. A sample of the input file with the triad points for each link is listed in appendix A for an RCCC mechanism. Four points are created for each of the four links: one for the link origin, one for the links unit X axis, one for the links unit Y axis, and one for the links unit Z axis. These points are included in the PRINT/POSITN statement near the end of the file.

The IPOST program listed in Appendix B was created as a general means of entering any mechanism into SLIDE. IPOST creates the necessary input files and can be used to verify any programs the user may want to write to create the attribute and position files for SLIDE directly from custom mechanism analysis.

Chapter 7

Methods of Solution in Interference Elimination

The methods of solution described here have been incorporated into a program which is presently called SLIDE. For clarity, the methods are presented in the order in which they appear in the program. The program flow for SLIDE is shown in Figure 27. Appendix C contains a listing of the program. The flow through the main program is direct with the only opportunity for branching due to the value of the interference detection option flag. If this flag is set to false no interference avoidance is attempted. This is useful since it allows the user to perform some visual error checking. In addition, there are some configurations which can be modified slightly to achieve reshapable configurations. This will be explained further in a later section.

Reading the geometry and position files involves a two step process for each file. First, each file is read with a blank read statement to count the number of lines. The number of lines in the attribute file is used to determine the number of links in a mechanism. The number of lines in the position file is used to

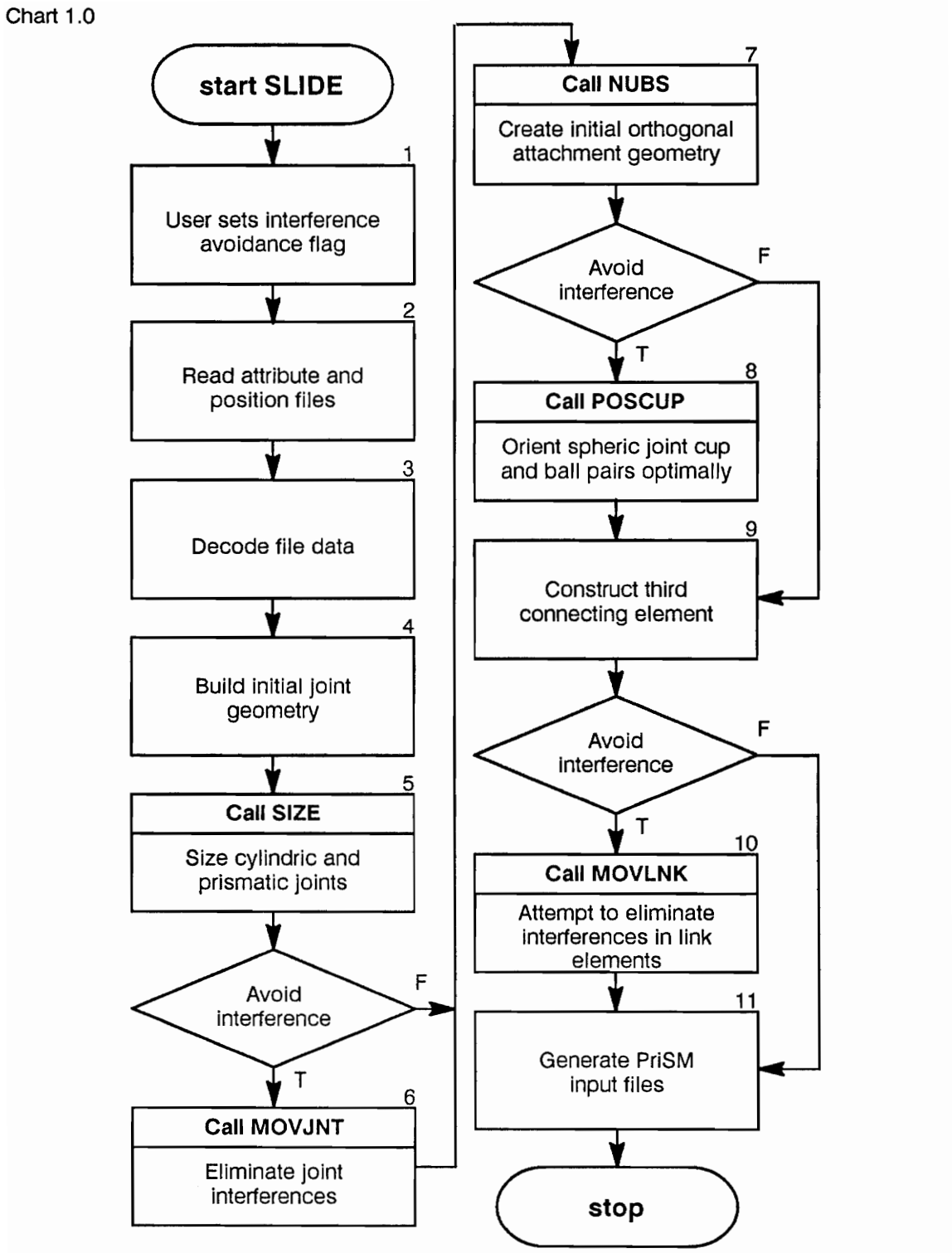

Figure 27. SLIDE program flow

determine the number of positions used to assemble the links of a mechanism. Once the number of links and positions have been determined, the pointer for each file is set to the beginning of each file and the data is extracted.

Decoding the data involves establishing a connectivity matrix for both joints and links. The connectivity of joints to links and links to joints is determined here. The connectivity information is especially important in repositioning joints, since moving a joint drags the links to which it is connected along with it. Connectivity is also used to eliminate spurious intersection tests. For example, a link element which attaches to a joint is not considered to be in interference with that joint. Similarly, link elements on the same link are not considered in an interference check. Unit axis vectors for each joint are also created in this stage.

Other information gleaned from the decoding stage involves the creation of joint axis vectors, cataloging links, and cataloging joints. Joint axis vectors are created in the local coordinate system of the link and are used for orientation information in joint sizing operations. Links are cataloged by the types of joints at each end and as reshapable or nonreshapable. At this writing, the only nonreshapable link is an S–S link. The reason it is nonreshapable is due to the idle degree of freedom associated with this link. The idle degree of freedom allows the link to spin around the axis determined by the two centers of the two spheric joints. Any reshaping would only enlarge the potential collision envelope for this type of link.. Joints are cataloged by type, configuration (internal or external), and movable or not movable. The only joint which is not movable is the spheric joint: this will be explained in a later section.

Initial Joint Geometry

Initial joint geometry is based on the joint size variables from line 4 of the attribute file. RS is used as the radius of the cup in a spheric joint. The ball of a spheric is assumed to be 0.9(RS). RO is used as the radius of outer joint pieces while RI is used as the radius of inner pieces. RL is the half length of an outer joint piece. The length of inner joint pieces is built in proportion to the size of the outer joint. Figure 28 shows the relationship of the joint parameters. Until the final output to PriSM, the geometry for prismatic, revolute, and cylindric joints is identical. The dashed boundaries show how the prismatic profile is contained within these geometric bounds. The reference point for the internal and external joint pieces is identical at this point and is based on the first position of the mechanism.

Sizing Joints

Revolute and spheric joints are properly sized during initial construction, but cylindrics and prismatics must be sized based on motion data. The subroutine SIZE does this by using the internal joint's local origin as the reference. An image of the initial reference point, noted in Figure 28, is then created on each joint piece. Deviations of the outer piece with respect to the inner piece are then recorded. Both positive and negative deviations are recorded. Once the mechanism has been stepped through all positions, the maximum absolute value for the positive and negative movement is found and used to lengthen the internal link as shown in Figure 29. Equations 15 and 16 on page 66 are used to determine positive and negative shifts.

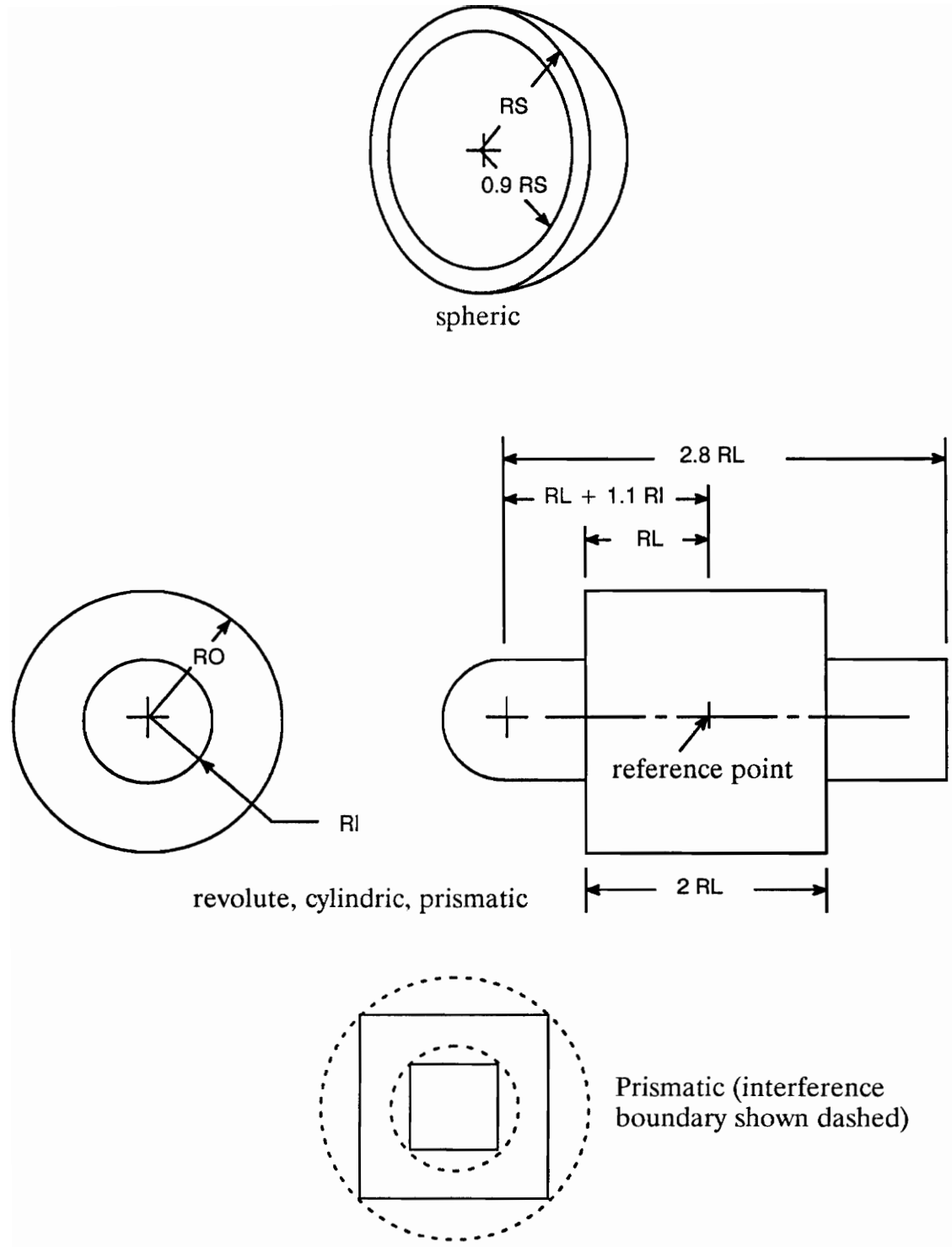

Figure 28. Initial construction of joints

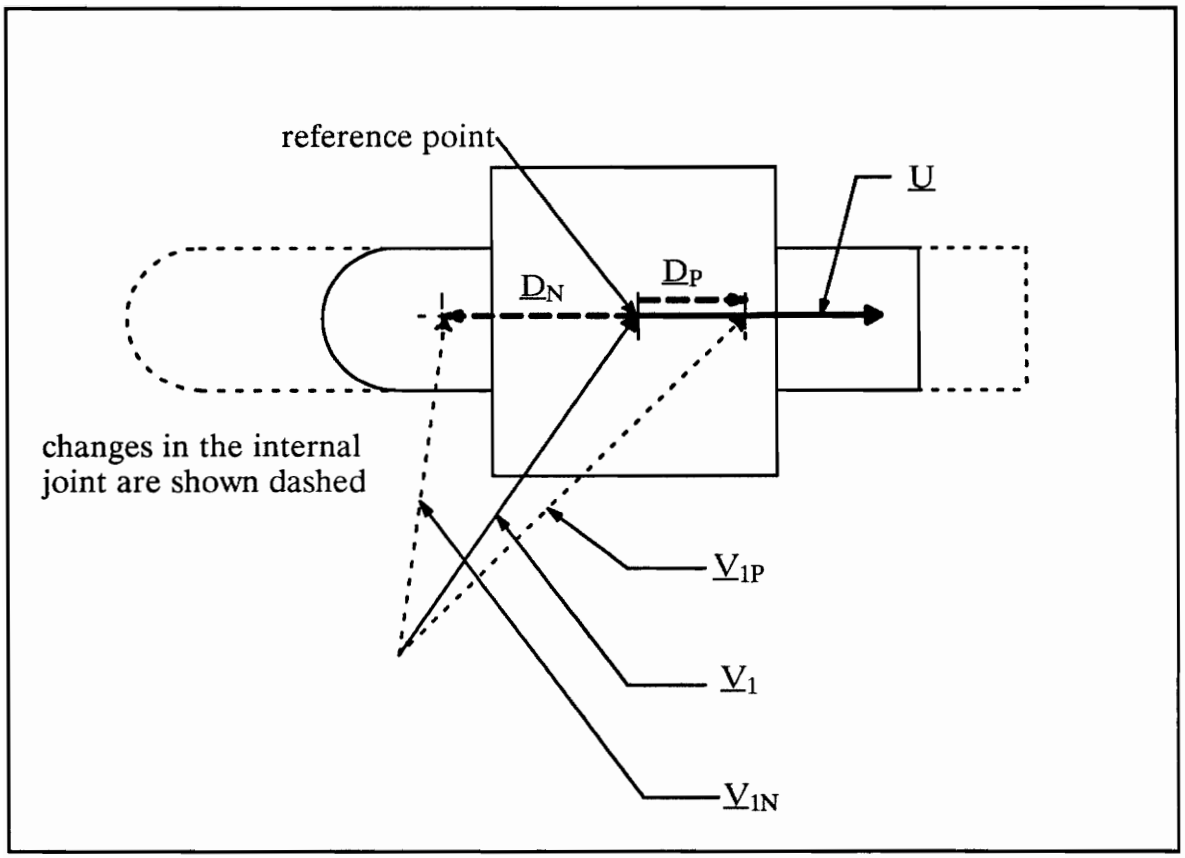

Figure 29. Sizing joints

(15) $|\underline{D}_P| = (\underline{V}_{1P} - \underline{V}_1) \bullet \underline{U}$

(16) $|\underline{D}_N| = (\underline{V}_{1N} - \underline{V}_1) \bullet \underline{U}$

where:

$\underline{D}_P$ ≡ the vector used to size the positive portion of the internal joint

$\underline{D}_N$ ≡ the vector used to size the negative portion of the internal joint

$\underline{V}_1$ ≡ the position vector to the reference point on the internal joint

$\underline{V}_{1P}$ ≡ the external joint position vector with a positive dot product on $\underline{U}$

$\underline{V}_{1N}$ ≡ the external joint position vector with a negative dot product on $\underline{U}$

$\underline{U}$ ≡ the unit axis vector for the internal joint

Moving Interfering Joints

Joints that can be repositioned are repositioned along a characteristic axis as noted in the section on initial considerations by the subroutine MOVJNT. If two spheric joints collide, then repositioning is not possible and no interference-free configuration exists for the mechanism as it is currently defined. The only possible solution at that point is to reduce the characteristic size of joints in the mechanism. Of course, this will not work if two spheric joints share the same center.

Joints that can be repositioned are organized into the first M positions of a one by N matrix, where N is the number of joints and M is the number of joints which are candidates for repositioning. N can be as large as 20. This structure is illustrated in Figure 30. If one of the joints violates the maximum distance constraint, the order in this matrix is indexed to the next permutation of the M joints until avoidance is successful or all permutations of order have been exhausted. There are practical limits to the number of permutations which can be performed. 12! is the largest number of order permutations currently allowed by SLIDE.

For interference avoidance purposes, only the internal joint piece is used. The envelope for the internal joint piece is grown to the proportions of a round-ended cylinder with the radius of the external joint piece. Figure 31 shows the relationship of this envelope to the internal joint piece of a cylindric joint. This envelope provides for a conservative interference check on both the external and internal joint pieces.

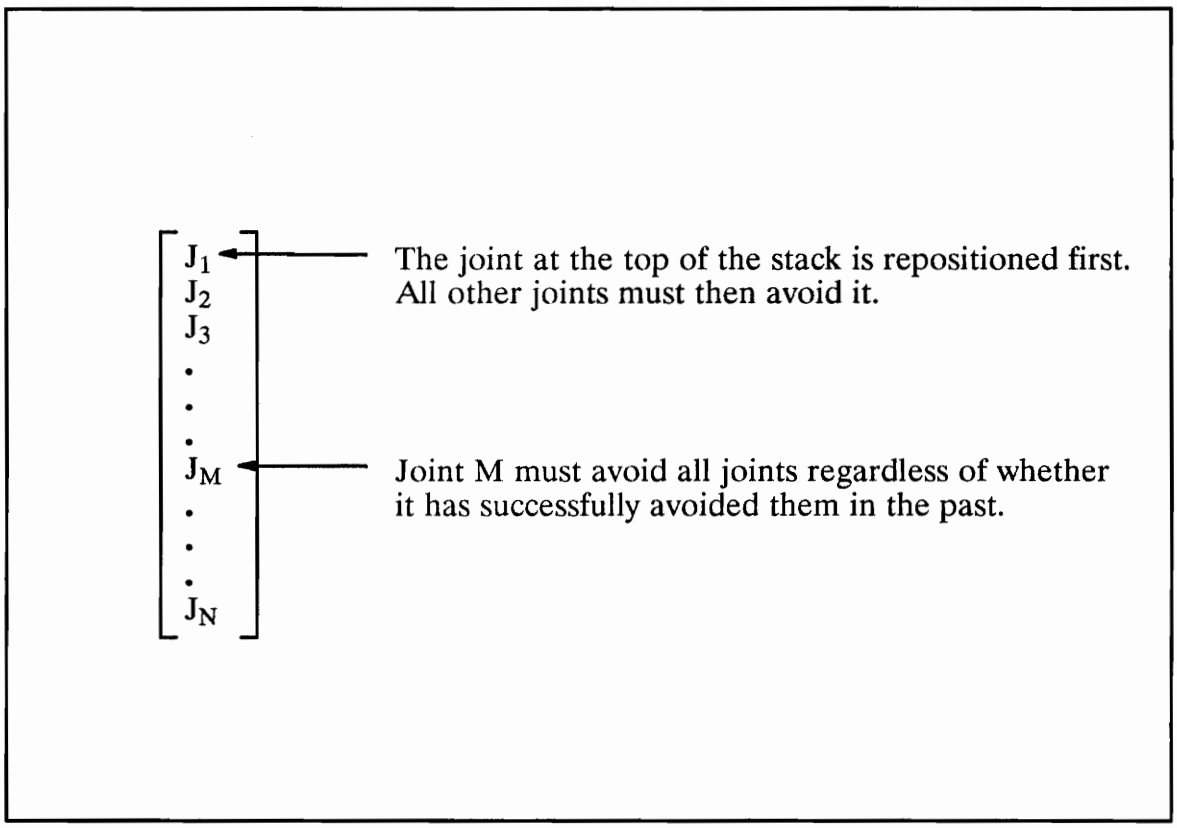

Figure 30. The avoidance order matrix

Creating the Initial Attachment Elements

Once all interference has been eliminated among all joints, the initial attachment elements are created. This process is best described by referring to Figure 32. The reference point described in the Figure is the location of the joint triad in the link's local frame of reference (see Figure 26). If the projection of a joint used to orient the attachment piece of a revolute, prismatic, or cylindric (R,P, or C) joint coincides with its reference point of the R,P, or C joint, then the attachment piece is aligned with the local X axis for the joint.

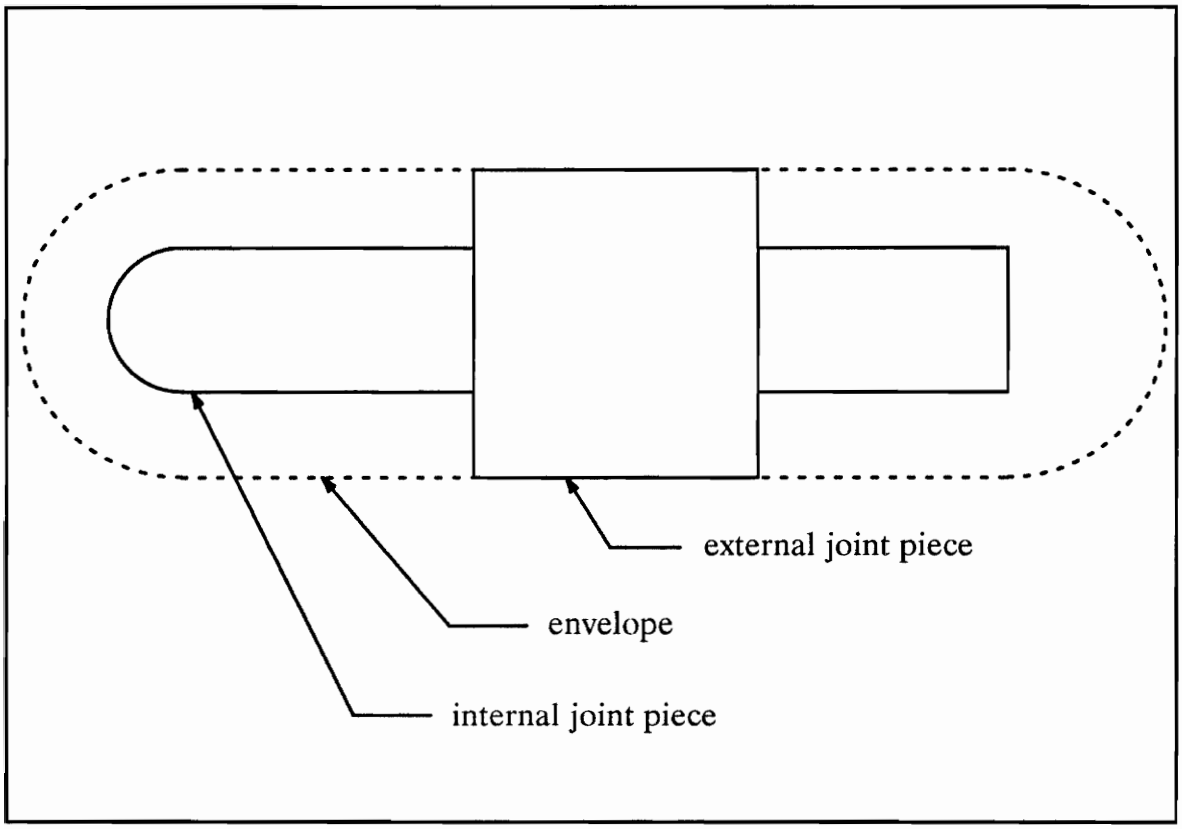

Figure 31. Interference checking envelope for a cylindric joint

Optimal Positioning of Cups to Balls

Joint sizing eliminates collisions between internal and external joint pieces for most joint types, but it does not for spheric joints. Figure 33 shows how the cup of a spheric joint might collide. The Figure also shows how the cup can be optimally oriented for one position, but the cup needs to be oriented optimally for all positions.

The solution of optimally orienting cups begins by creating an inverse image of the vector representing the ball attachment element in the cup's frame of reference. The vector is normalized to a unit length. Points are then created on a unit sphere in the cup's frame of reference by applying the relative rotation

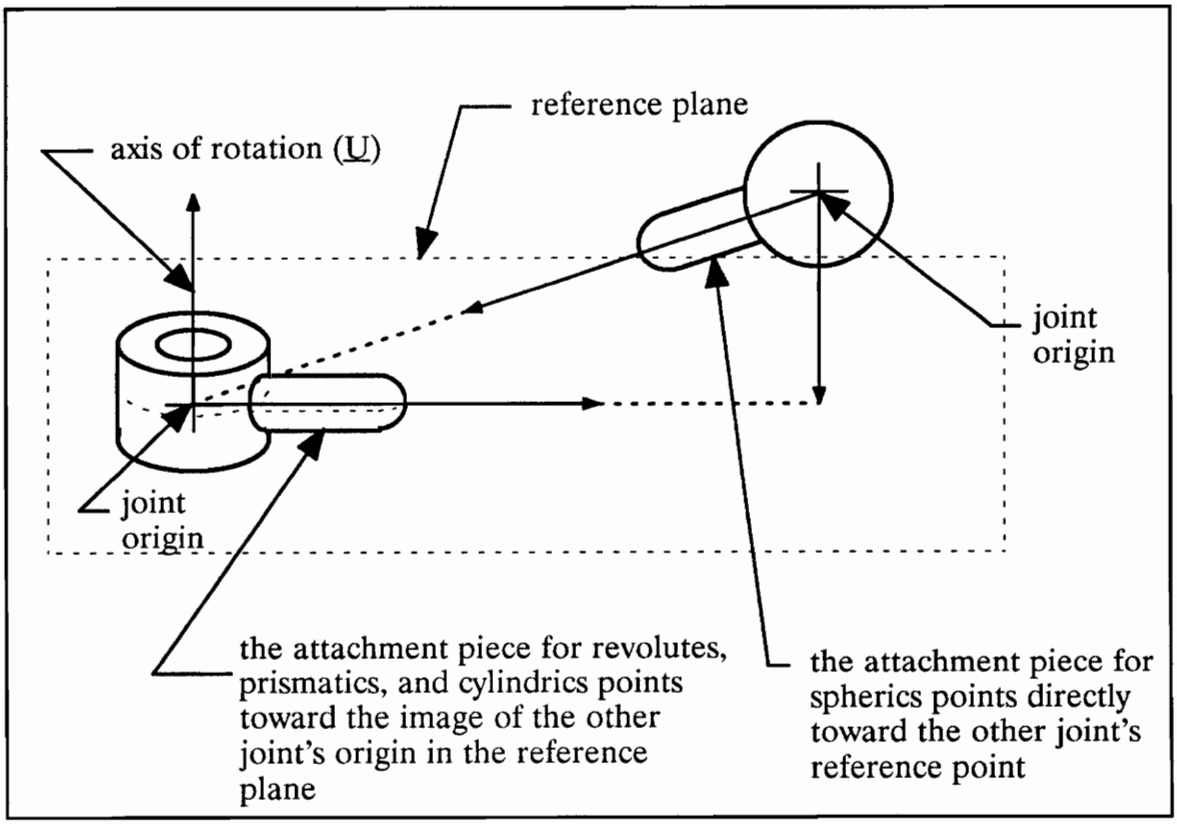

Figure 32. Creating initial attachment elements

transformations on the unit vector as shown in equation 17.

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = [R\theta_{xc}][R\theta_{yc}][R\theta_{zc}][R\theta_{zb}][R\theta_{yb}][R\theta_{xb}]\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \tag{17}$$

where:

x_i, y_i, z_i ≡ the cartesian coordinates of point i (the subscript 1 represents the ball's frame of reference, and the subscript 2 represents the point transformed into the cup's frame of reference)

$[R\theta_{ij}]$ ≡ the rotation matrix for rotation about axis i (subscript j set to b indicates that this is the ball's transformation, c indicates the cup)

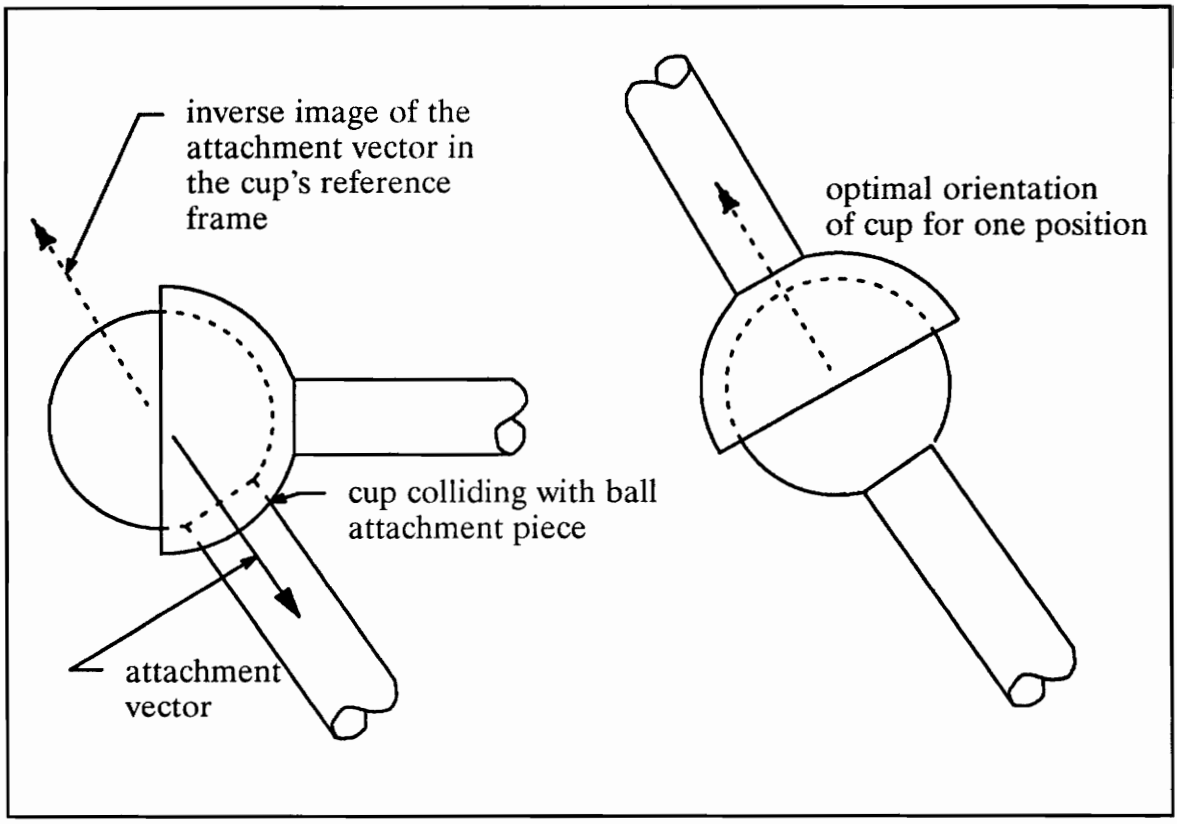

Figure 33. Orienting cups in spheric joints

The equation shows the transformation for one position of the mechanism. With these points resident in the cup's reference frame, the objective becomes to place the cup's attachment element some where in the middle of the points. Three methods were studied to accomplish this.

The first method was an averaging technique. The x, y, and z values of all vectors were each summed and averaged and the resulting vector was then normalized to a unit length. This method was highly susceptible to position biasing from dwells in the mechanism. Figure 34 shows an extreme example of this in two dimensions. This was clearly unacceptable.

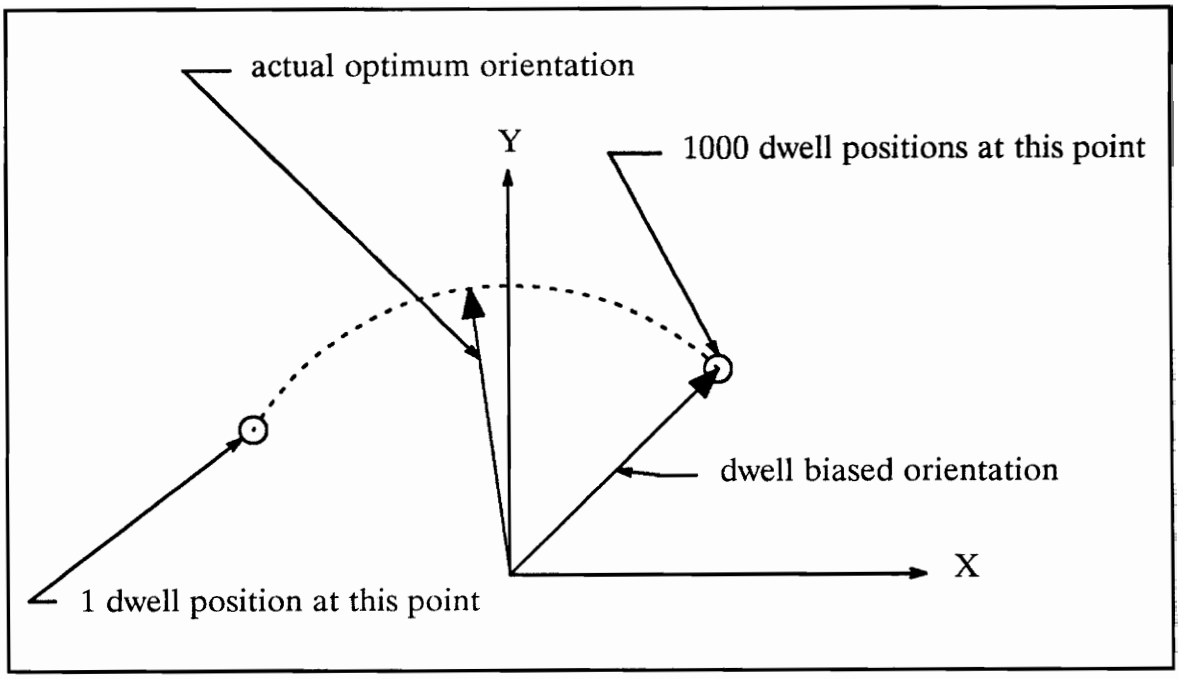

Figure 34. Dwell biasing in the position averaging technique

The next technique used a settling plane. This method started with a plane which was tangent to the unit sphere at the first point. The plane was then settled toward the center of the sphere to the second point as shown in Figure 35. Subsequent points on the sphere center side of the plane where to be used to settle the plane farther. The objective was to settle the plane toward the center of the sphere until no more points existed on the center side of the plane, and then orient the cup perpendicular to the settled plane. Adding a third point coplanar with the first two and the sphere center proved to be a disaster. The settling plane would immediately collapse into the plane which contained the first two points and the sphere center, and no further settling would occur.

The technique finally used was to generate a bounding cone for the points. The the first estimate of the cone was determined by finding the two points on the

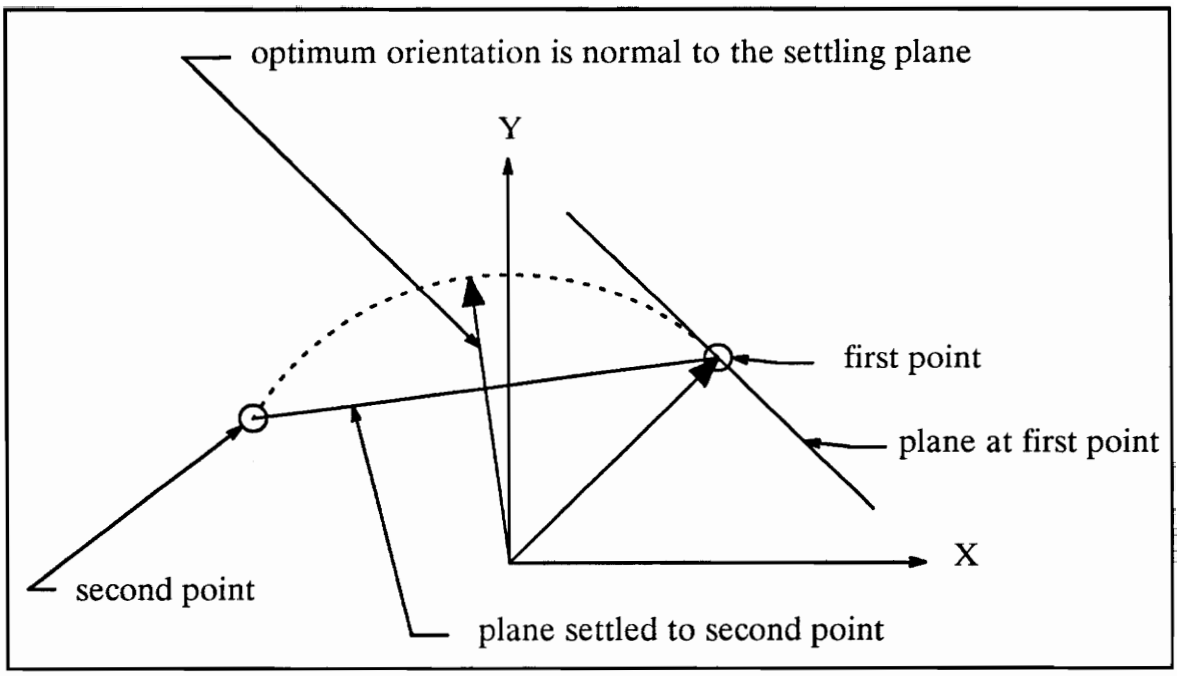

Figure 35. Using a settling plane for optimal orientation

unit sphere with the greatest separation. This creates the bounds of a cross sectional slice of the cone shown in dark highlight in Figure 36. The intersection

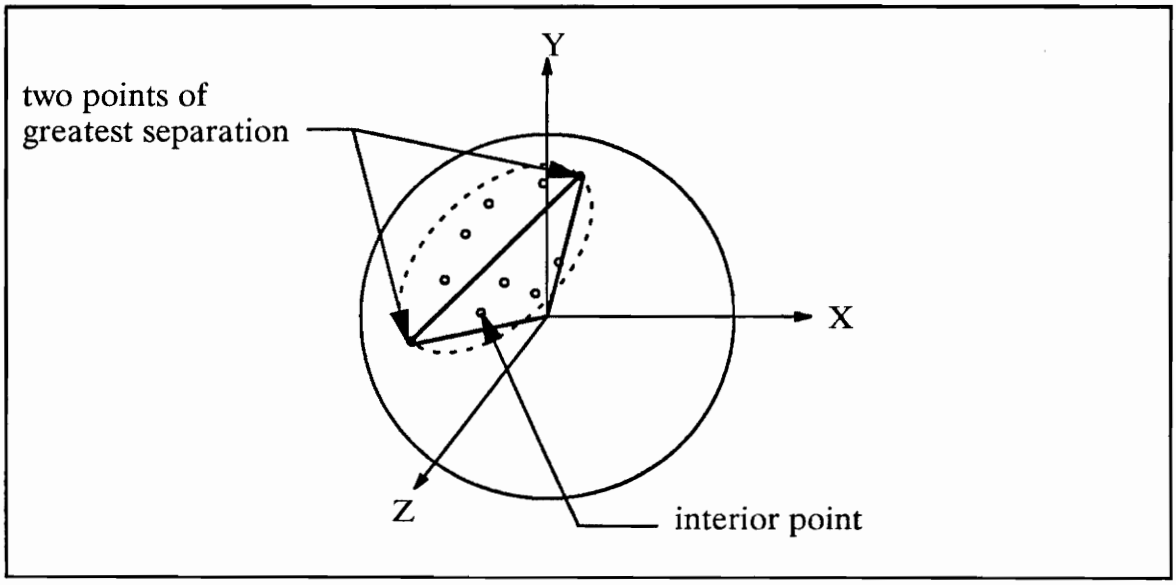

Figure 36. Creating a bounding cone from the two points of greatest separation

of the cone with the sphere is shown as a dashed circle on the sphere's surface. If all other points are contained within the bounds of this dashed circle, then the process is complete. However, this is not always the case.

this last statement can be clarified by considering the case of three equidistant points. Using the chord between any two of the three points to define the cone leaves one point outside the cone. In this case, the cone is defined by the three points. A generalization of this concept is presented in Figure 37. Here it is

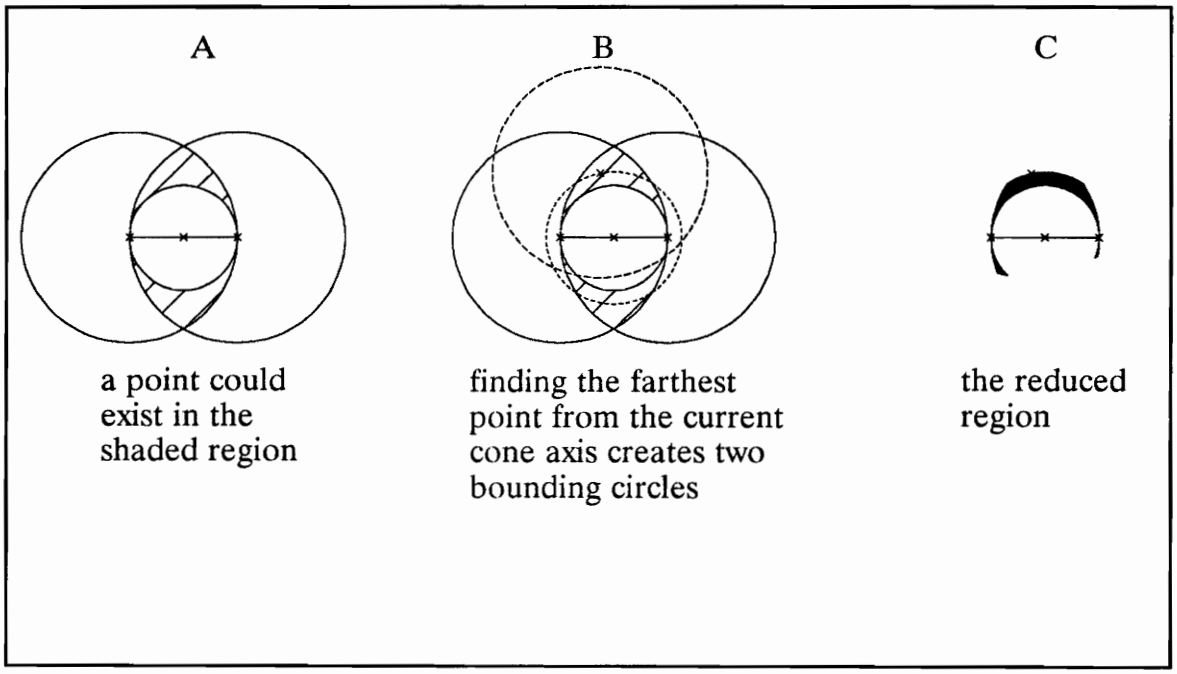

Figure 37. Points which can exist outside the original bounding cone

assumed that the points exist in a very small region on the surface of the sphere so that the region is nearly planar. The initial cone in Figure 37–A is defined by two points whose separation is the maximum for any pair in the set. The line connecting these two points can be used to sweep circles around the two points. The shaded region shows the area shared by the two circles where other points

could exist and still be at a distance less than or equal to the distance separating the two points. Finding the point which is farthest away from the cone axis defined by the two points creates the smaller dashed circle. The larger dashed circle represents the maximum distance constraint already established by the two original points. This reduces the region where additional points could exist to that shown in Figure 37–C. This serves as the justification for searching for points outside the initial cone.

The subroutine POSCUP enlarges the bounding cone by finding the points on either side of the line of maximum separation. POSCUP searches for those points are located farthest (toward the center of the sphere) from the plane defined by the intersection of the sphere with the original cone. The two points farthest to either side are used to establish a second line. The two lines are then used to create a plane which is parallel to both lines. The new cone axis is perpendicular to this plane.

Once the cone axis has been defined, the attachment piece for the cup is aligned with the axis. This rule applies to all cases except those where the the cup is attached to an S–S link. In those cases, the role of the cup and ball are inverted.

Creating the Third Connecting Element

The construction of the third connecting element is trivial. As Figure 38 illustrates, it simply involves connecting the two ends of the two attachment elements.

Eliminating Link Interference

Eliminating link interferences is similar to eliminating joint interferences with the exception that rotations are used instead of translations. It is similar in

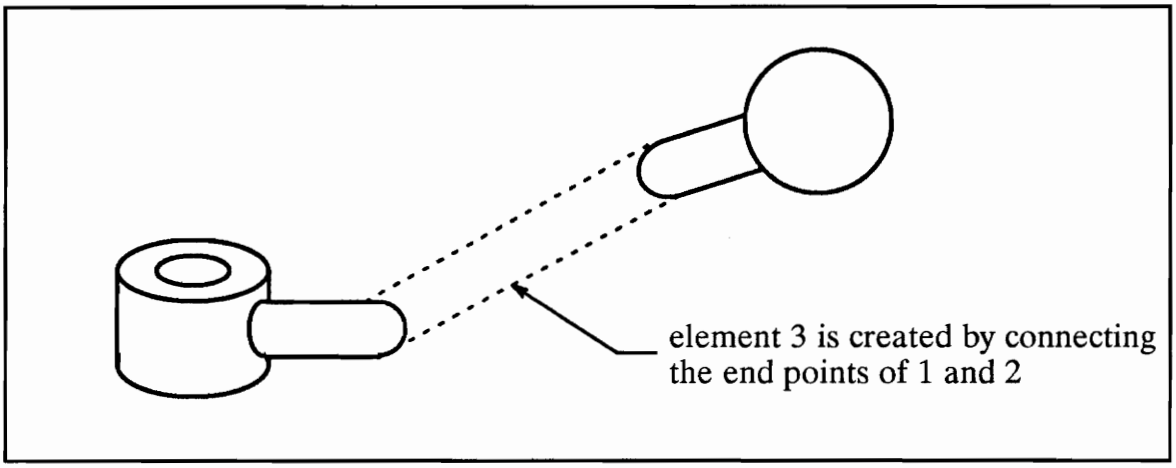

Figure 38. Creation of the third connecting element

that a direction for rotation is enforced until it it is found that a successful solution does not exist. The limiting factor is the number of positions for the mechanism. Only N rotations corresponding to N positions are allowed before the avoidance attempt is declared a failure. At that point, rotation is performed in the other direction. For element 3, three options are considered. First, rotations which lengthen element 1 are performed. If that is not successful, rotations which lengthen element 2 are performed. A third option involves alternately lengthening elements 1 and 2.

The first attempt at eliminating interferences treated the elements as infinite cylinders avoiding either other infinite cylinders or spheres. Three methods were considered to determine the rotation angle necessary to avoid link collisions between infinite cylinders. One involved iterating by a set increment until the interference was eliminated. This was judged to be too time consuming. A second method involved modeling interfering elements as a mechanism. It turned that the constraint equations of an RCCC mechanism were appropriate, but branching would have to be considered. The method finally used involved

projection of the interfering elements onto a plane perpendicular to the intruding element's axis of symmetry.

The projection resulted in a view of the intruder and movable element as shown in Figure 39. In this projection, a circle in the rotation plane and at the

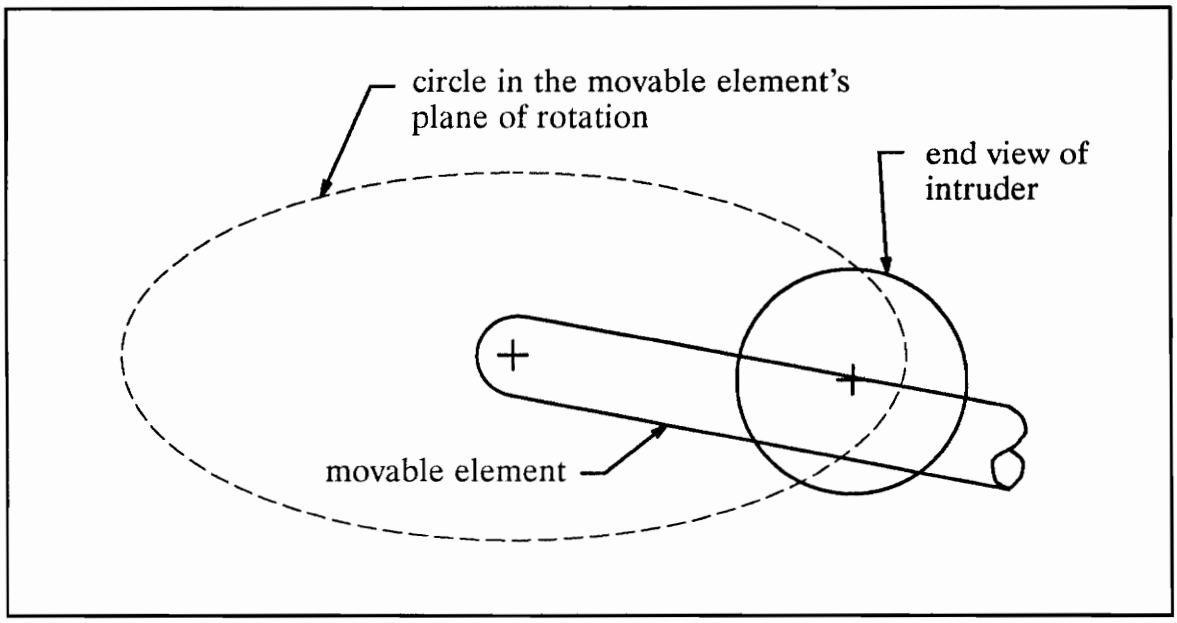

Figure 39. Projection of interfering elements onto plane perpendicular to intruder

rotation center for the movable link element appears as an ellipse. By striking a line which passes through the center of rotation for the movable element and the center of the projected intruder (Figure 40), the angle ϕ can be found from the equation:

(18) $$\phi = \sin^{-1}(B/D)$$

where:

B $\equiv$ the sum of the radii for the the two interfering elements

D $\equiv$ the distance from the rotation center to the intruder axis of symmetry

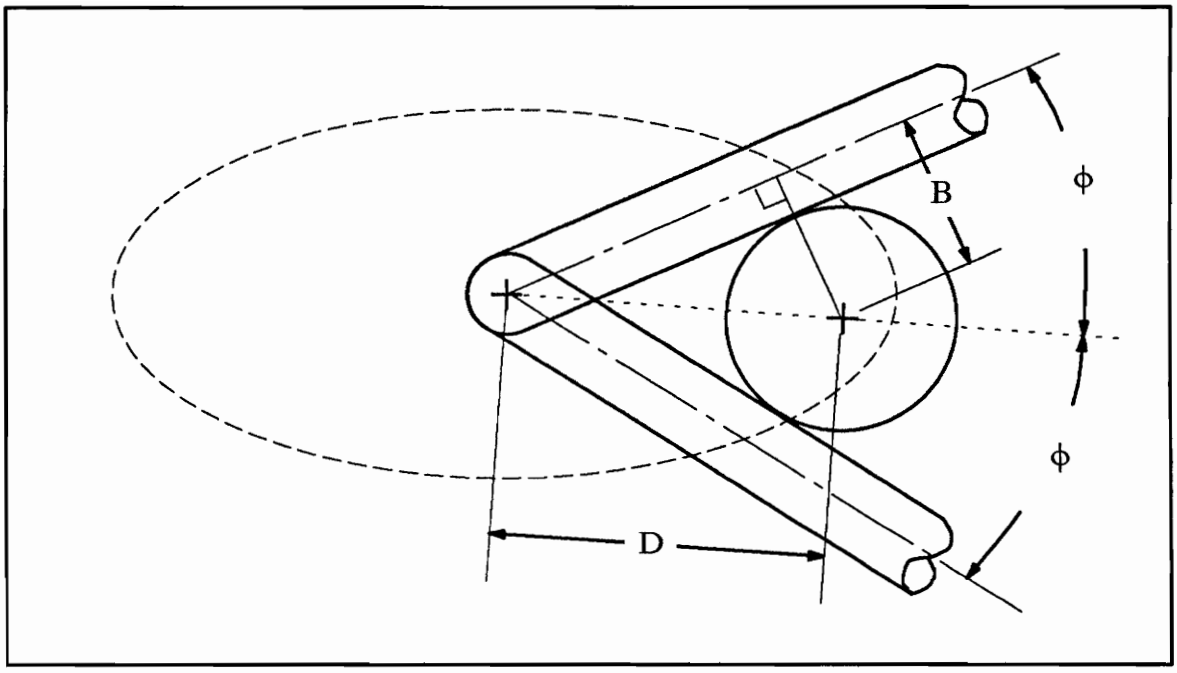

Figure 40. Minimum avoidance conditions in the elliptical projection

A rotation of ϕ in either direction from this line will eliminate the interference.

At this point it is useful to identify the major axis and the projected angle, γ, between the major axis and the line passing through the two centers. This is done in Figure 41.

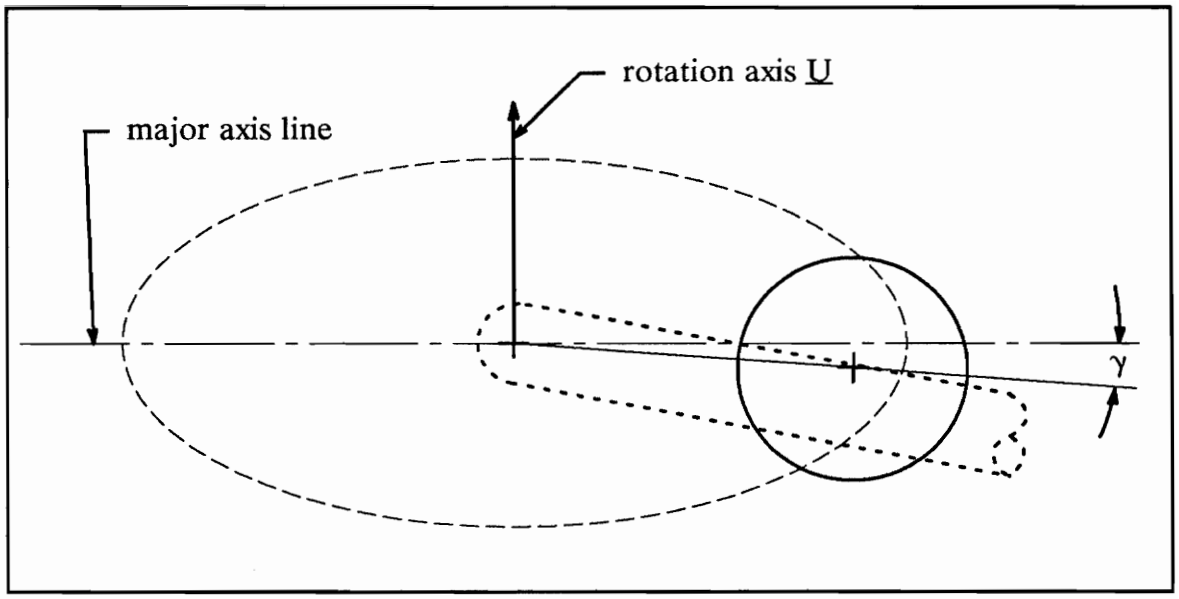

Figure 41. Projected angular location of intruder center

The major axis direction is found from the equation:

(19) $$\underline{A_M} = \underline{U} \times \underline{L_2}$$

where:

$\underline{A_M}$ ≡ the vector describing the major axis

$\underline{L_2}$ ≡ the vector describing the intruder axis of symmetry.

Subtracting γ from ϕ then adding γ to ϕ gives the two angles, θ_1 and θ_2, shown in Figure 42 along with the angle β. β is the angle between the major axis and the projection of the movable element's center line. With these angles, there is enough information to determine the actual rotation angles in the plane of rotation perpendicular to $\underline{U}$.

Finding these angles in the plane of rotation is a matter of determining how much the sine component of the angles has been foreshortened in the projection.

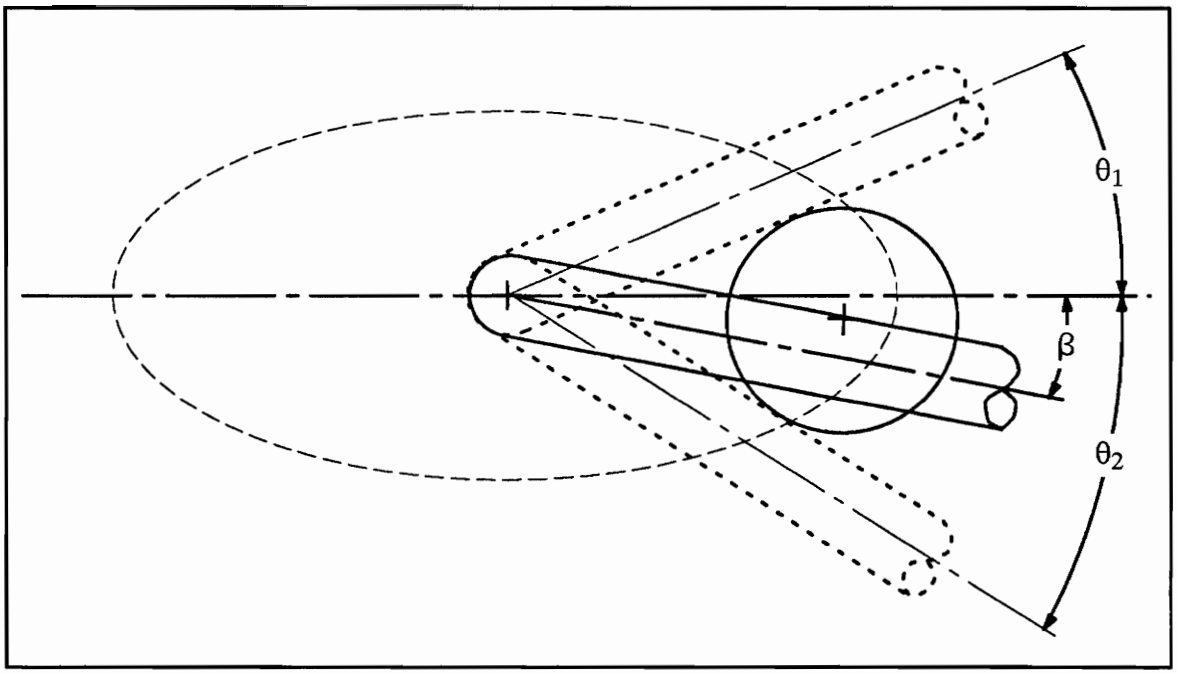

Figure 42. Angles referenced to the major axis of the ellipse

This is done by calculating the angle, α, between the unit vector, $\underline{U}$, and $\underline{L_2}$ by means of the equation:

(20)
$$\alpha = \cos^{-1}\left[\frac{\underline{L_2} \bullet \underline{U}}{|\underline{L_2}|}\right]$$

The true angles are then found from the following equation:

$$\psi' = \text{atan2}\left[\frac{\sin^{-1}(\psi)}{\cos^{-1}(\alpha)}\ ,\ \cos^{-1}(\psi)\right] \tag{21}$$

where:

$\psi \equiv$ the angle to be transformed

$\psi' \equiv$ the transformed angle

atan2 $\equiv$ the arctangent function with quadrant sensitivity

The positive and negative rotation angles about $\underline{U}$ (θ_p and θ_n) are then:

$$\theta_p = \theta_1 + \beta \quad \text{and} \quad \theta_n = \theta_2 - \beta \tag{22}$$

Avoiding spheres is accomplished in a similar fashion except that an elliptical projection is never needed, and the movable element's finite length is always used. Udupa's [18] method for simplifying the boundaries is employed. This method involves shrinking the bounded cylinder to a line and growing the sphere to compensate. This is illustrated in Figure 43. The expanded spheres are again projected onto the plane of rotation and two cases are considered. In the first case, the center of the sphere is within the bounds of the circle swept by the shrunken cylinder. In the second case, the sphere center is outside this circle. The two cases are shown in Figure 44 with the modified boundary representations.

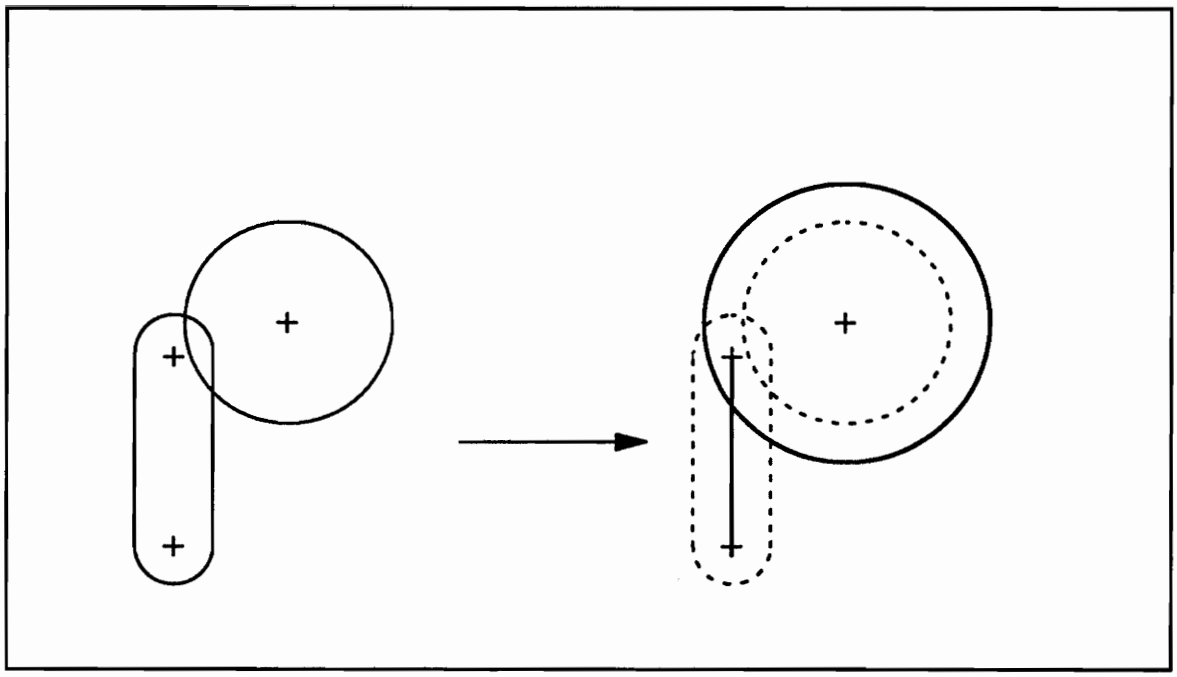

Figure 43. Udupa's process of shrinking the cylinder and growing the sphere

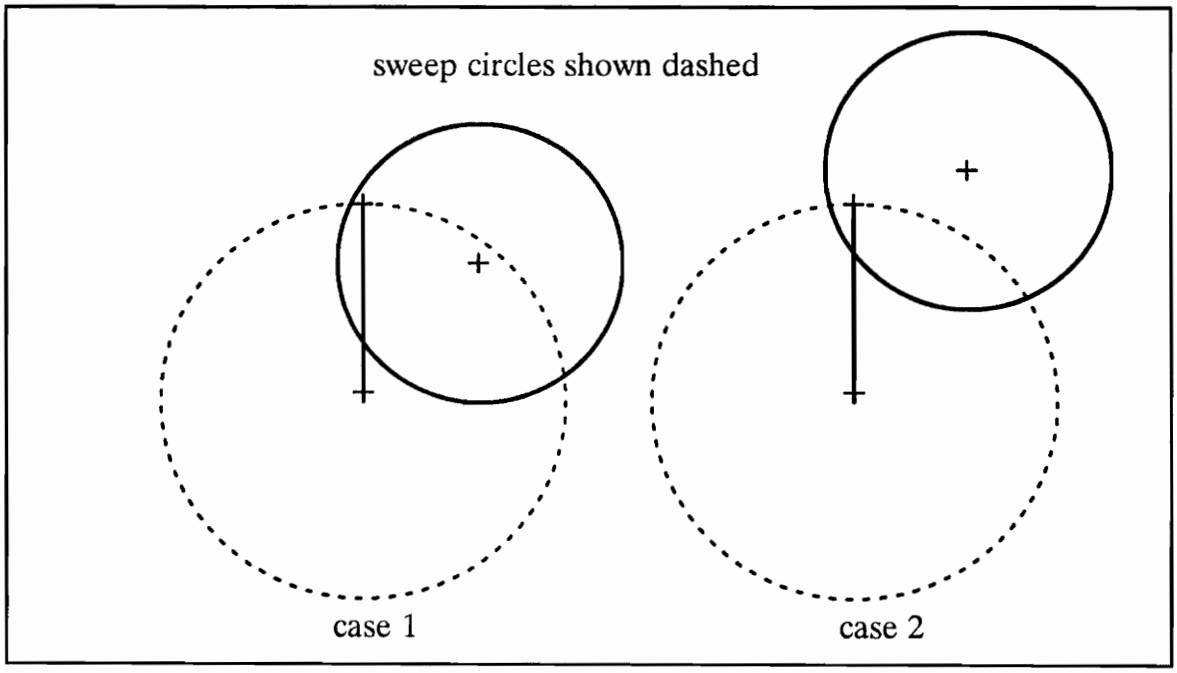

Figure 44. The two projection cases for dealing with interfering spheres

The distance of the sphere from the plane of rotation must also be considered. Here the boundary representing the sphere is reduced based on the distance from the plane as follows in equation 23:

(23) $$R_s = R_{so} \sin[\cos^{-1}(D_s/R_{so})]$$

where:

$R_s \equiv$ the distance modified radius

$R_{so} \equiv$ the initial radius

$D_s \equiv$ the distance to the plane

The process of dealing with spheres has greater utility than merely avoiding spherical joints. In many cases, the technique of dealing with infinite cylinders in elliptical projection causes the movable element to move too far. One example of this is shown in Figure 45. This condition occurs when the closest point, on the infinite axis lines for the cylinders, projects off the end of finite line segments which model the finite cylinders. In this extreme case, the projected intersection is at infinity. The infinite cylinder model causes the movable element to rotate until it is parallel, but this is clearly too far. When such a case is discovered, the avoidance is recalculated using an end of the intruder as a sphere. Initial interference with the sphere at the end of the intruder is not necessary. Once a collision has been detected and the spherical avoidance routine is called, the avoidance angle is determined by calculating how far the movable element has to move beyond the sphere in the current rotation direction.

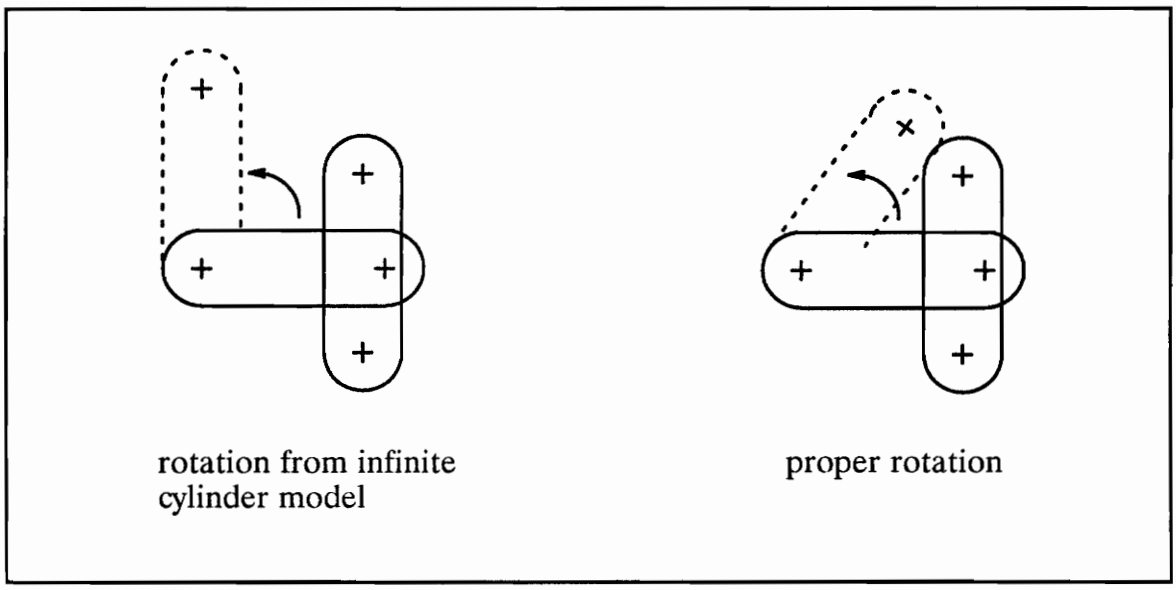

Figure 45. Repositioning a movable link that has been rotated too far

Creating the PriSM Input Files

The last step in SLIDE is to create the files used by PriSM for geometric visualization and animation. The formats for these files appear here through the courtesy of Montgomery [16]. The model file format shown here in Figure 46 allows for twice as many joint types as the attribute file in Figure 25, because internal and external joint parts exist here where they did not exist before. The variables Tx, Ty, and Tz correspond to rotations necessary to orient joints in their local link frames. Similarly, Px, Py, and Pz correspond to positions in the local link frames. Len is the length of external joints and Z1 and Z2 correspond to an internal link's end locations along the local link Z axis before being positioned in the local link frame. Ro and Ri retain their original meaning from the attribute file used as input to SLIDE. while the R variable for internal joints is equivalent to Ri. A pillow block element is included in PriSM which is not supported in SLIDE.

1	Character string description of the model					
2	Element Type	Element ID	Element Description			
3	Parameter 1	Parameter 2	Parameter 3	Parameter 4	. . .	Parameter M
.						
.						
.	Element Type	Element ID	Element Description			
N	Parameter 1	Parameter 2	Parameter 3	Parameter 4	. . .	

Supported Element Types

1	ID #	External Revolute Joint							
Ro	Ri	Len	Tz	Ty	Tx	Px	Py	Pz	
2	ID #	Internal Revolute Joint							
R	Z1	Z2	Tz	Ty	Tx	Px	Py	Pz	
3	ID #	Connecting Element							
R	Num	P1x	P1y	P1z	P2x	P2y	P2z	. . .	
4	ID #	Link (Joint, Joint, Connector,...)							
Num	ID 1	ID 2	ID 3	. . .					
5	ID #	External Prismatic Joint							
Ro	Ri	Len	Tz	Ty	Tx	Px	Py	Pz	
6	ID #	Internal Prismatic Joint							
R	Z1	Z2	Tz	Ty	Tx	Px	Py	Pz	
7	ID #	External Cylindric Joint							
Ro	Ri	Len	Tz	Ty	Tx	Px	Py	Pz	
8	ID #	Internal Cylindric Joint							
R	Z1	Z2	Tz	Ty	Tx	Px	Py	Pz	
9	ID #	External Spheric Joint							
Ro	Ri	Tz	Ty	Tx	Px	Py	Pz		
10	ID #	Internal Spheric Joint							
R	Px	Py	Pz						
11	ID #	Ground Element (Pillow Block)							
Ro	Ri	Z	Ly	Lz	Tz	Ty	Tx	Px	Py	Pz

Figure 46. PriSM model file format

The position file in Figure 47 is nearly identical to the position file of Figure 25 with the exception that orientation data is listed before position data. This was done to facilitate the packing of transformation arrays in PHIGS+ more easily.

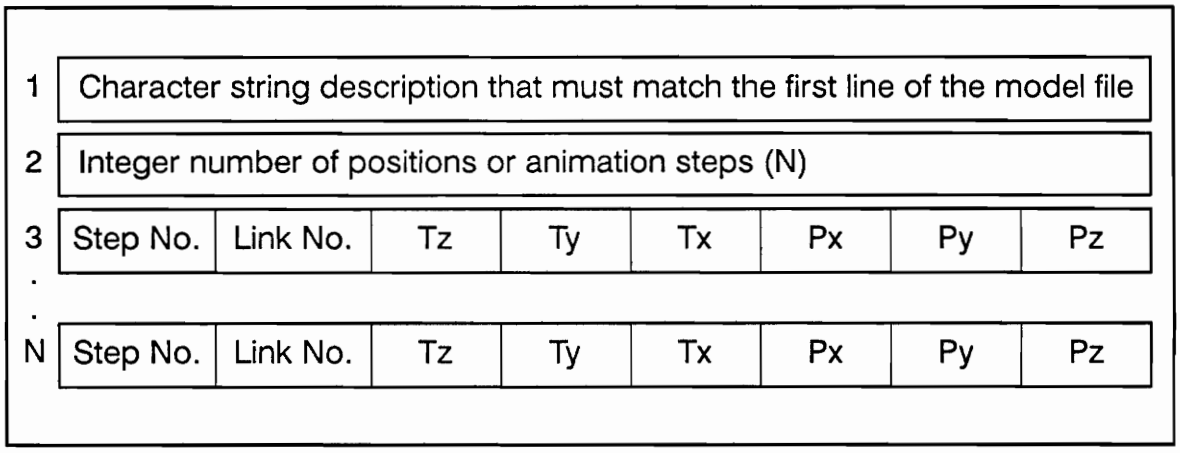

Figure 47. PriSM position file format

Chapter 8

Results

The methods described in this work have been successfully implemented to eliminate interferences in several mechanisms. There are conditions, however, where interferences cannot be eliminated. These conditions and their resolution are discussed in the mechanism case studies described below.

The RSSR mechanism in Figure 48 was created from analysis data derived from the work of R. L. Williams [104] using a custom postprocessor to create the SLIDE attribute and position files. This is to be differentiated from the other

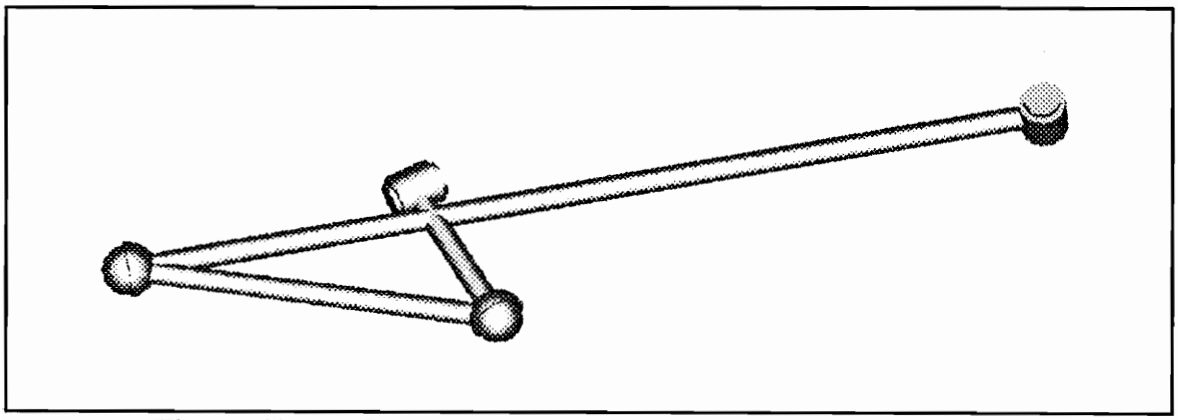

Figure 48. RSSR mechanism as synthesized with interference

mechanisms which were processed through IMP75. The mechanism is shown as it

was synthesized using a minimum connecting distance approach for all links. The ground link is not shown for this mechanism. Figure 49 shows the result of optimally orienting the cups on the spheric joints. Figure 50 shows the result when

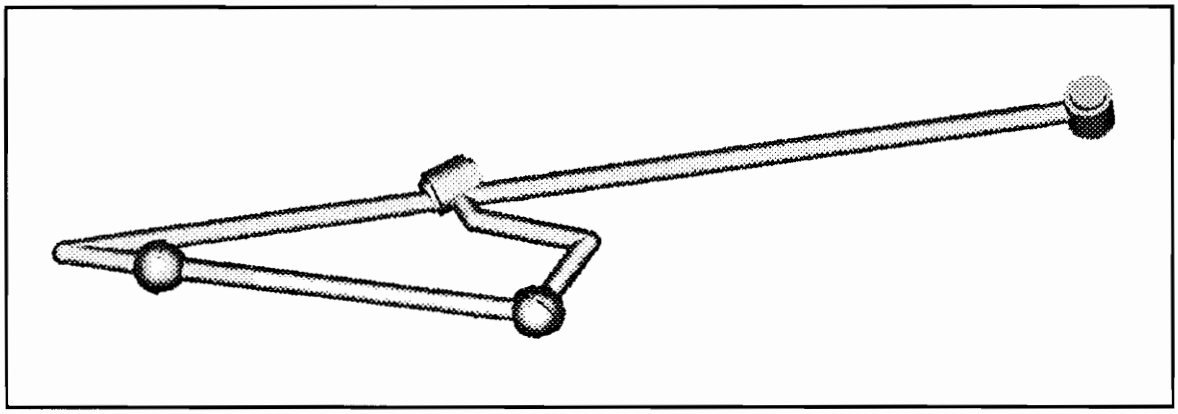

Figure 49. RSSR with oriented cups

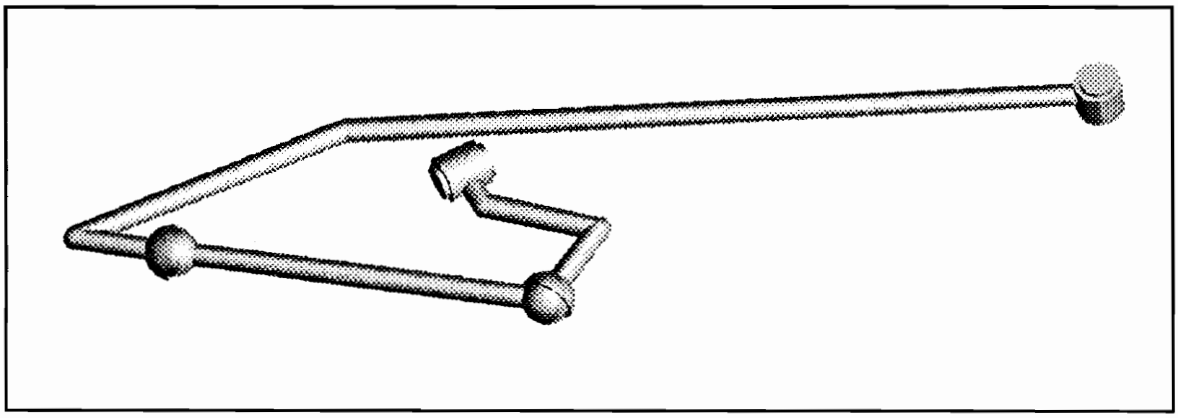

Figure 50. RSSR reshaped to eliminate interference

when the mechanism has been successfully reshaped. This interference-free configuration is not unique. To illustrate, Figure 51 shows another successful reshaping. The difference here is due to the initial ordering of the links in the input files.

The RPCS of Figures 52 and 53 further illustrates the point that a reshaping solution is not unique. In this case, the differences are the result of changing the

direction of the joint axis for the prismatic joint.

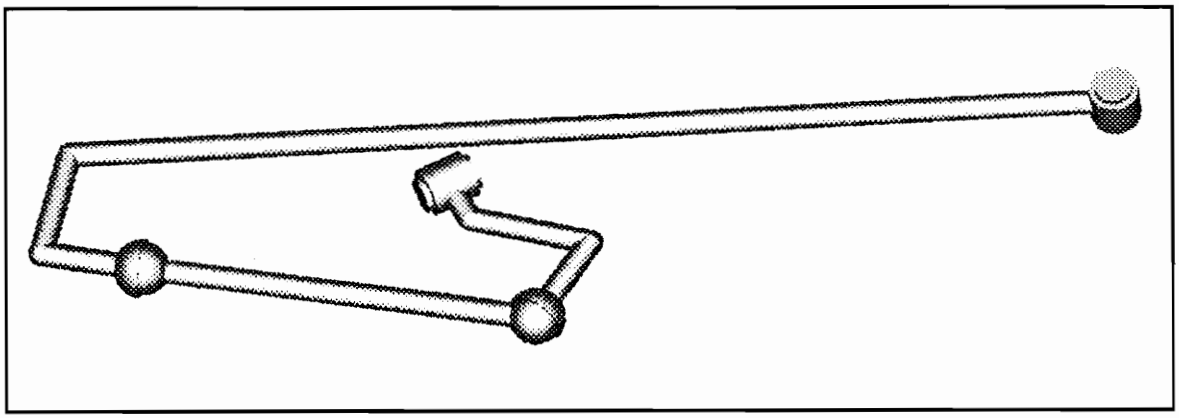

Figure 51. Alternate form of a reshaped RSSR

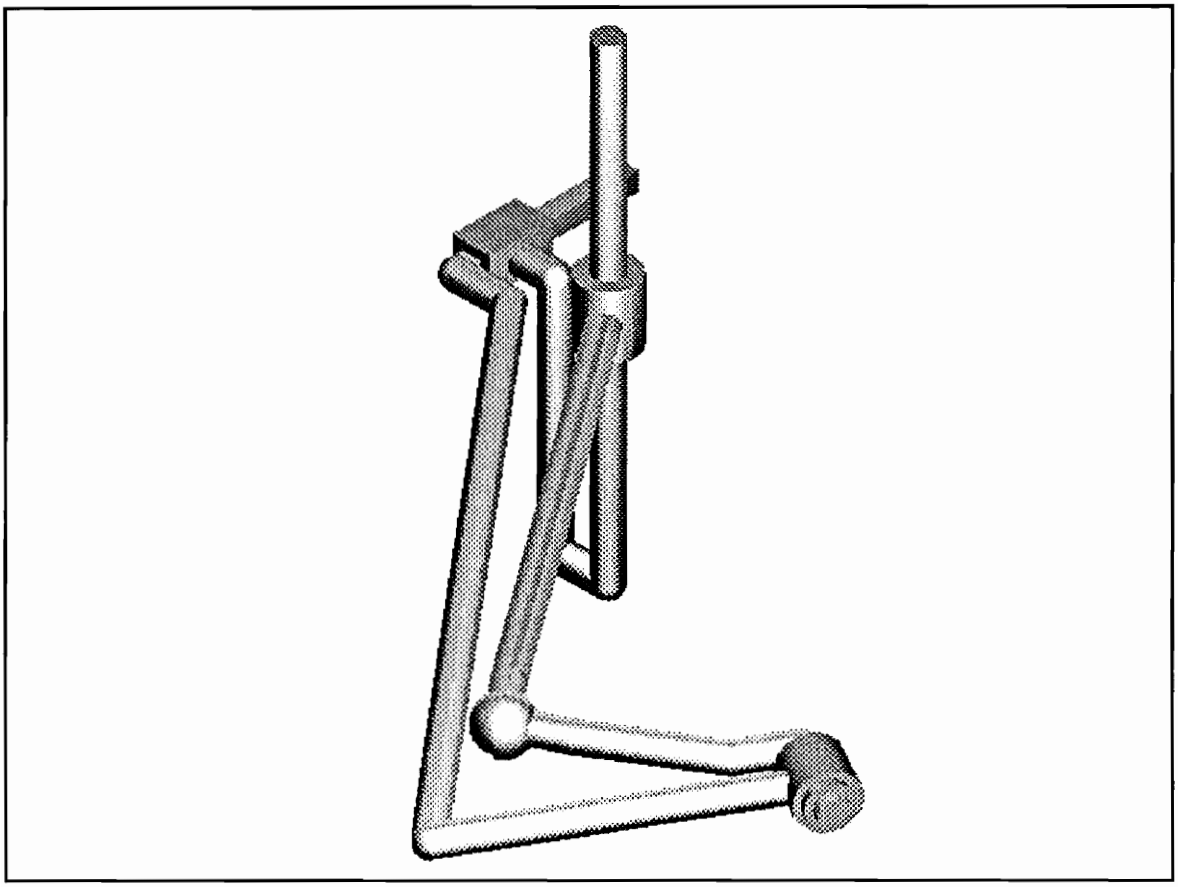

Figure 52. RPCS mechanism

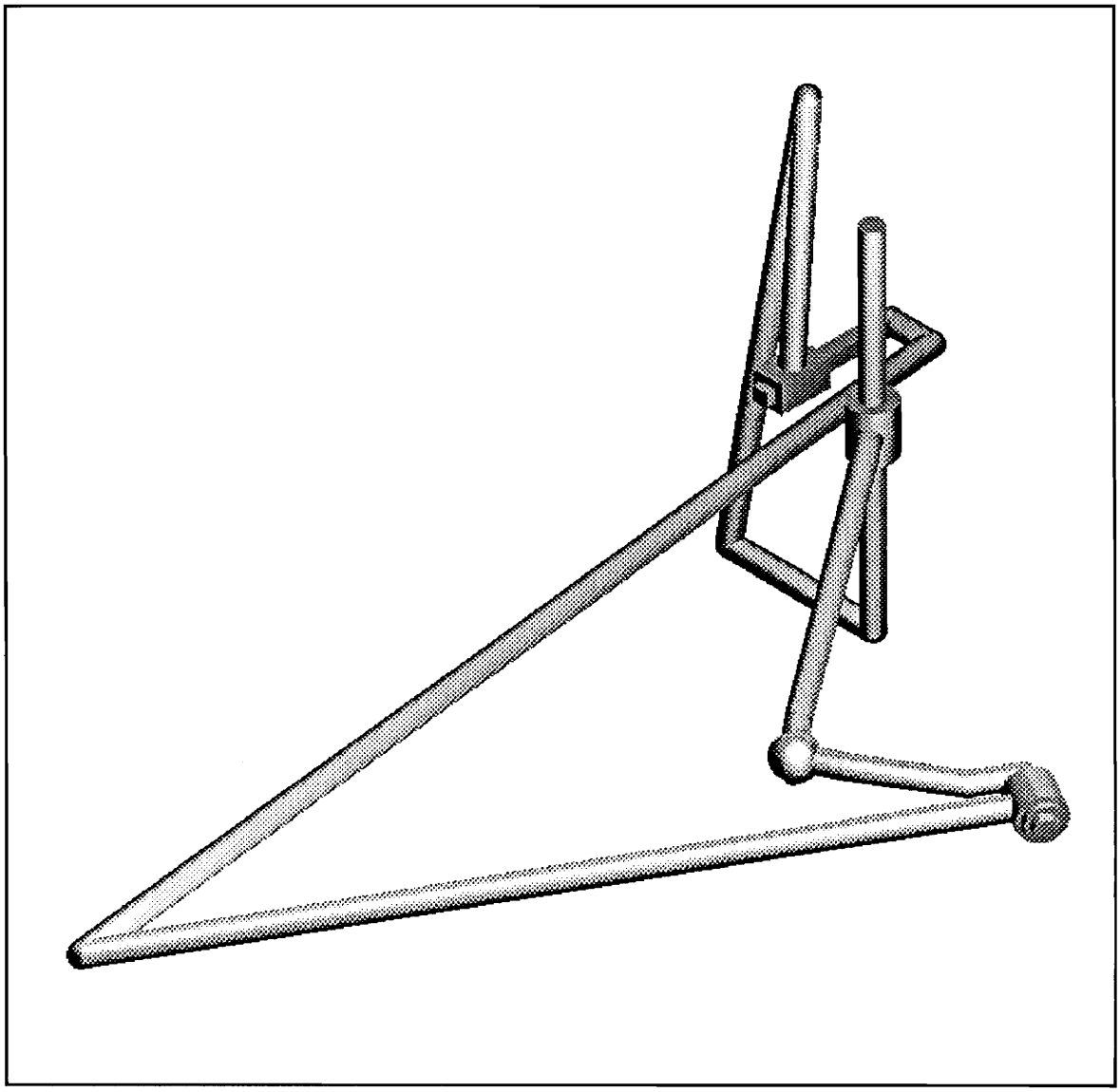

Figure 53. Alternate form of the RPCS mechanism

The RCCC shown in Figure 54 was created from data developed by Veeraraghavan [105], and serves to show one condition where reshaping fails. The

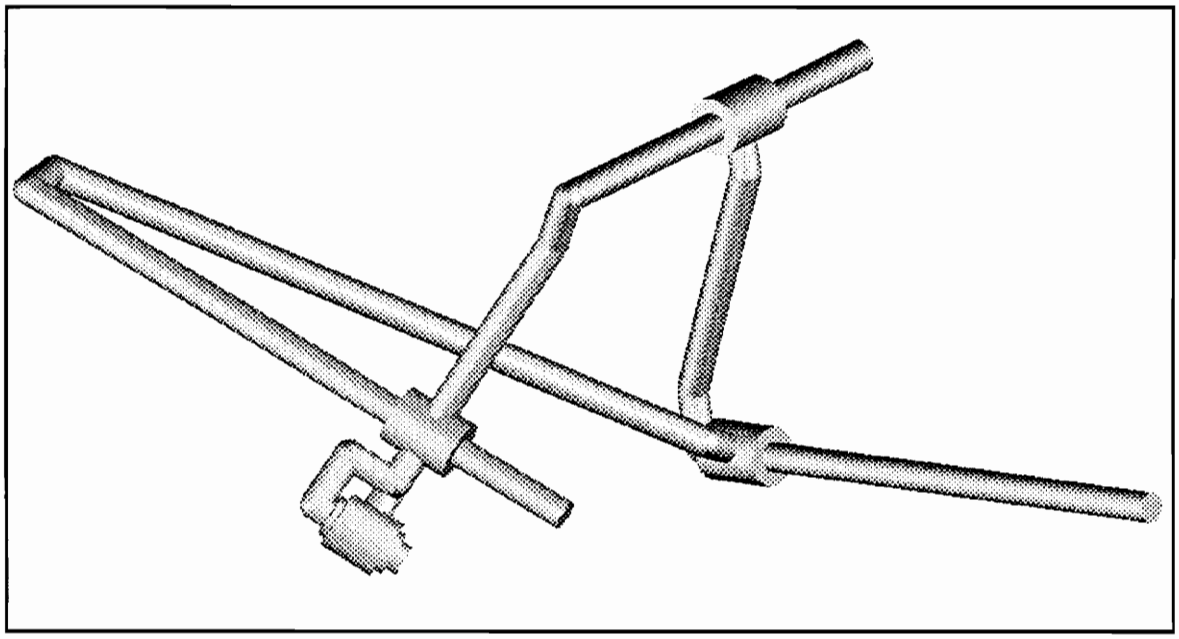

Figure 54. RCCC mechanism with interference

short crank is shown interfering with the internal joint of a cylindric link. The reshaping options allow for configurations which effectively wrap around the the cylindric joint, but no configuration is ever found where interference is eliminated at all positions. A joint shift is the only solution which will remove the interference. Such a joint shift was used to obtain the configuration of the RCCC shown in Figure 55. This configuration can be reshaped to eliminate all interference, but SLIDE only repositions joints to avoid other joints. This means that a manual edit of the IMP75 input file is necessary. The final configuration of the RCCC, after reshaping, is shown in Figure 56.

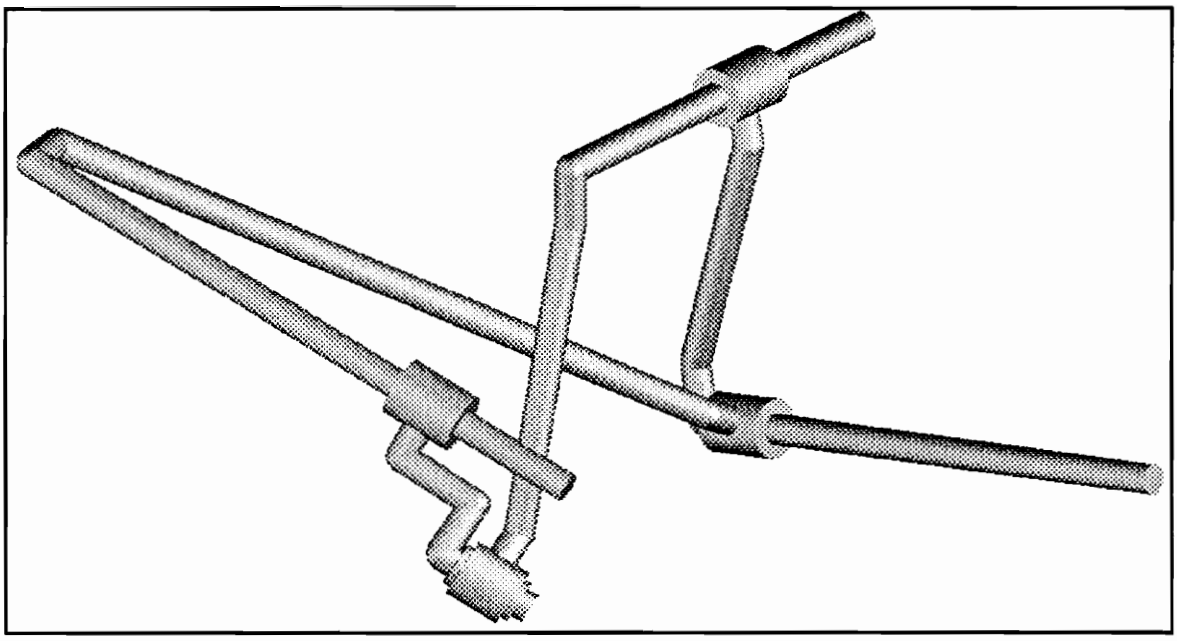

Figure 55. RCCC mechanism with joint shifted

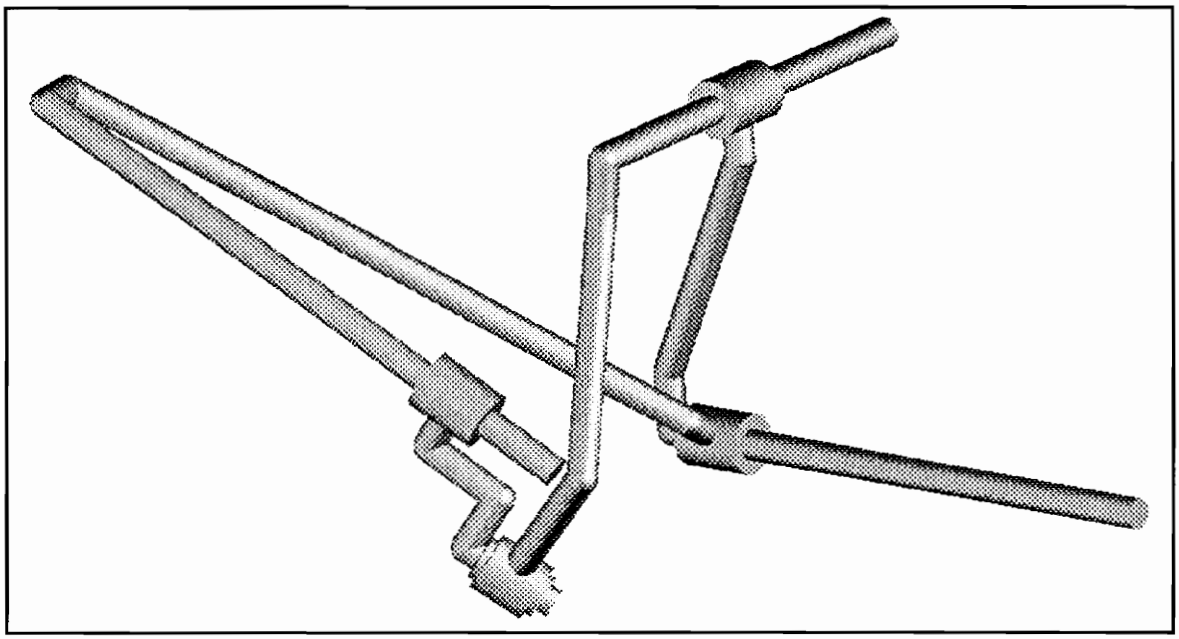

Figure 56. RCCC mechanism with interference eliminated

The mechanism in Figure 57 is a 7R mechanism which appeared in a paper by Sandor, Xu, and Weng [106]. Figure 58 shows the 7R after the joints have

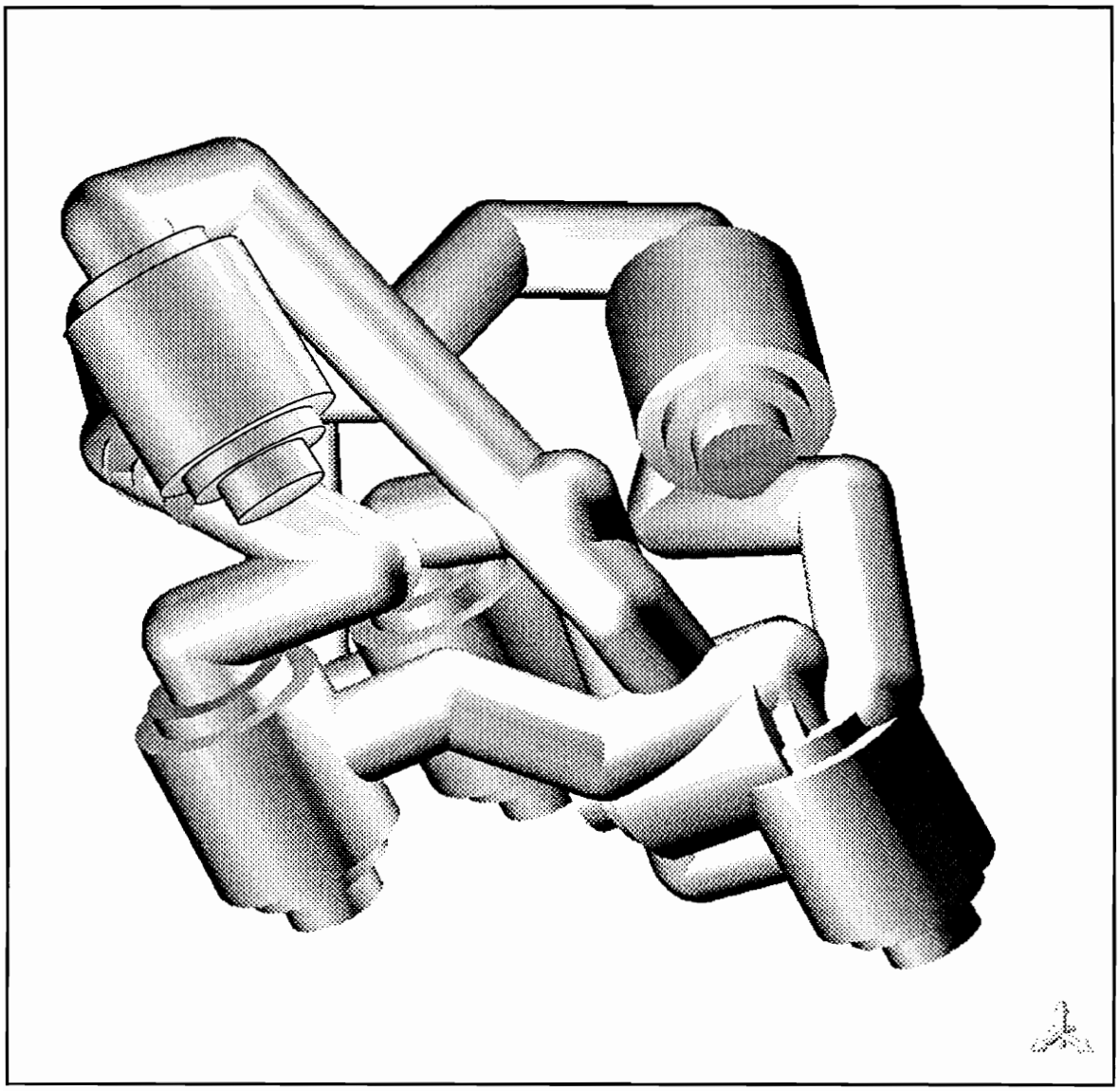

Figure 57. 7R mechanism before reshaping

been successfully repositioned. The reshaped model appears in Figure 59. The most notable feature of the reshaped mechanism is the length of some of the elements in the reshaped link. Reshaping is not necessarily optimal. To prove this

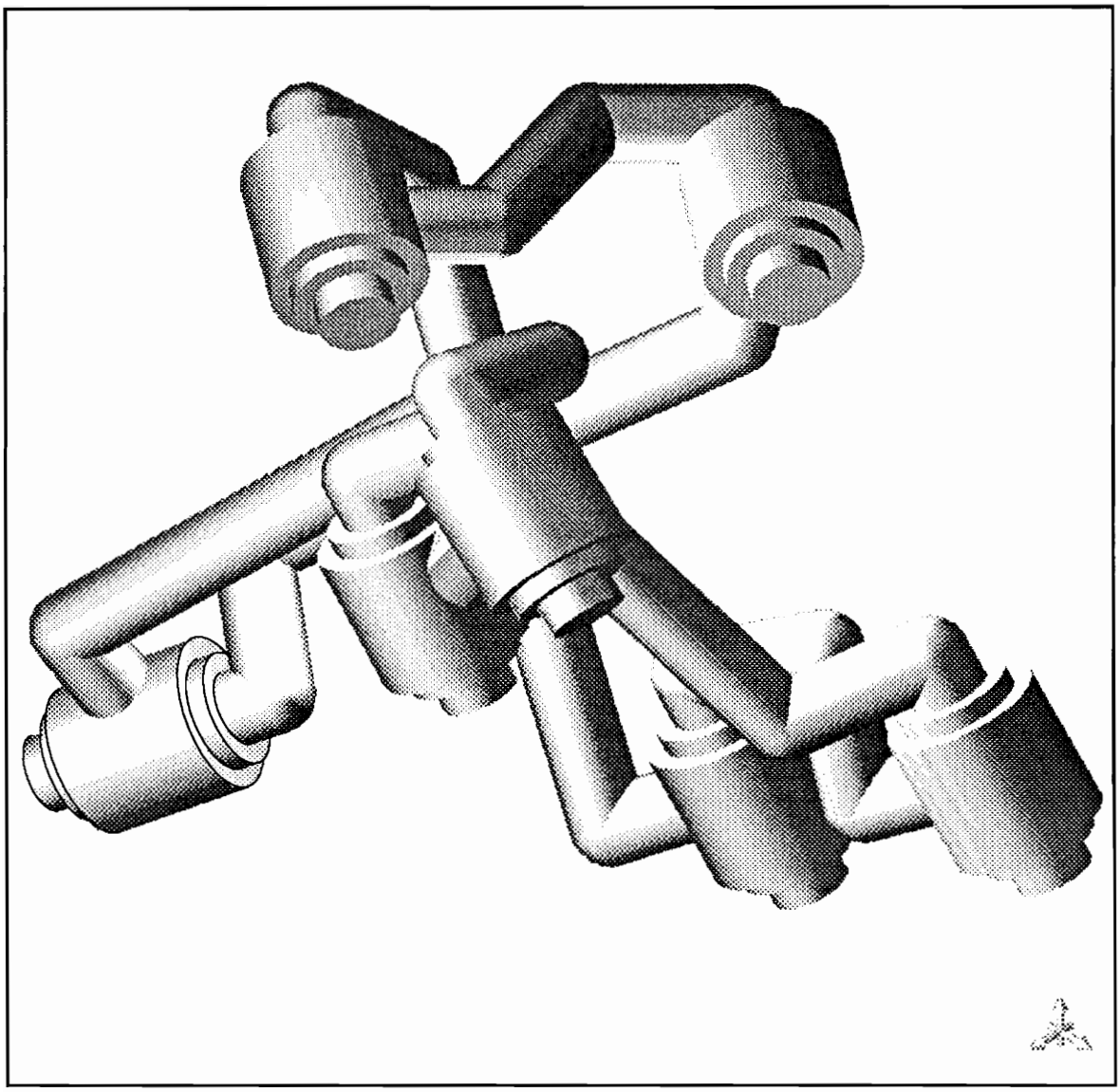

Figure 58. 7R mechanism with joint interference eliminated

last statement, the excessively long link was modified by editing the PriSM file. The result is shown in Figure 60. This configuration also has no interference. Of course, it is much easier to modify one link of a noninterfering mechanism and change it so there is still no interference than it is to find an interference-free configuration from a condition as in Figure 57.

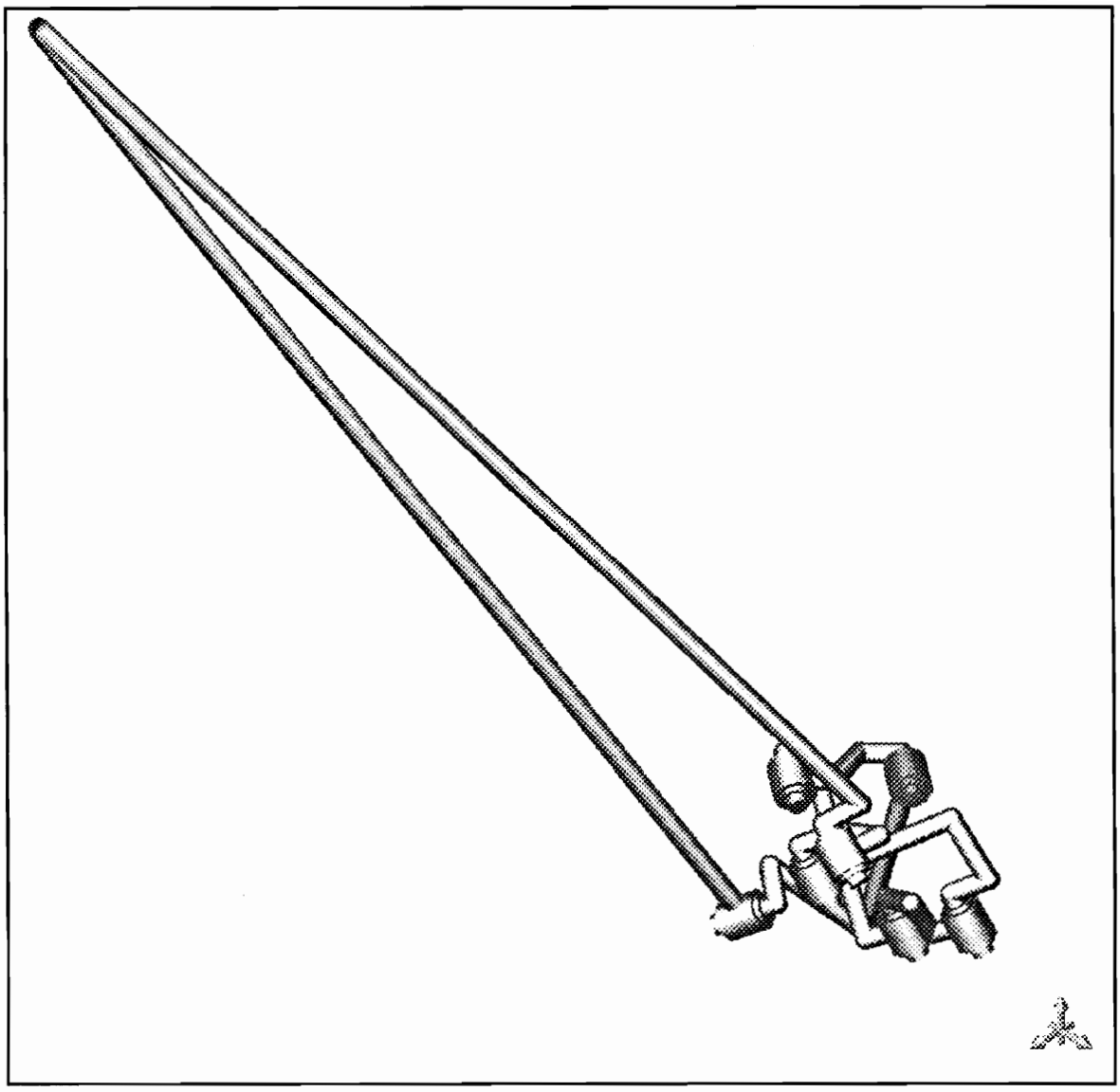

Figure 59. 7R mechanism with link interference eliminated

There is one warning to the user concerning the number of links. SLIDE will try to exhaust all permutations of order in reshaping the links of a mechanism. For a four link mechanism, there are 4! permutations of this order or 24 different orderings based on an ordering matrix as shown in Figure 30. For a seven link mechanism, the number of orderings is 5040. The full impact of these numbers is

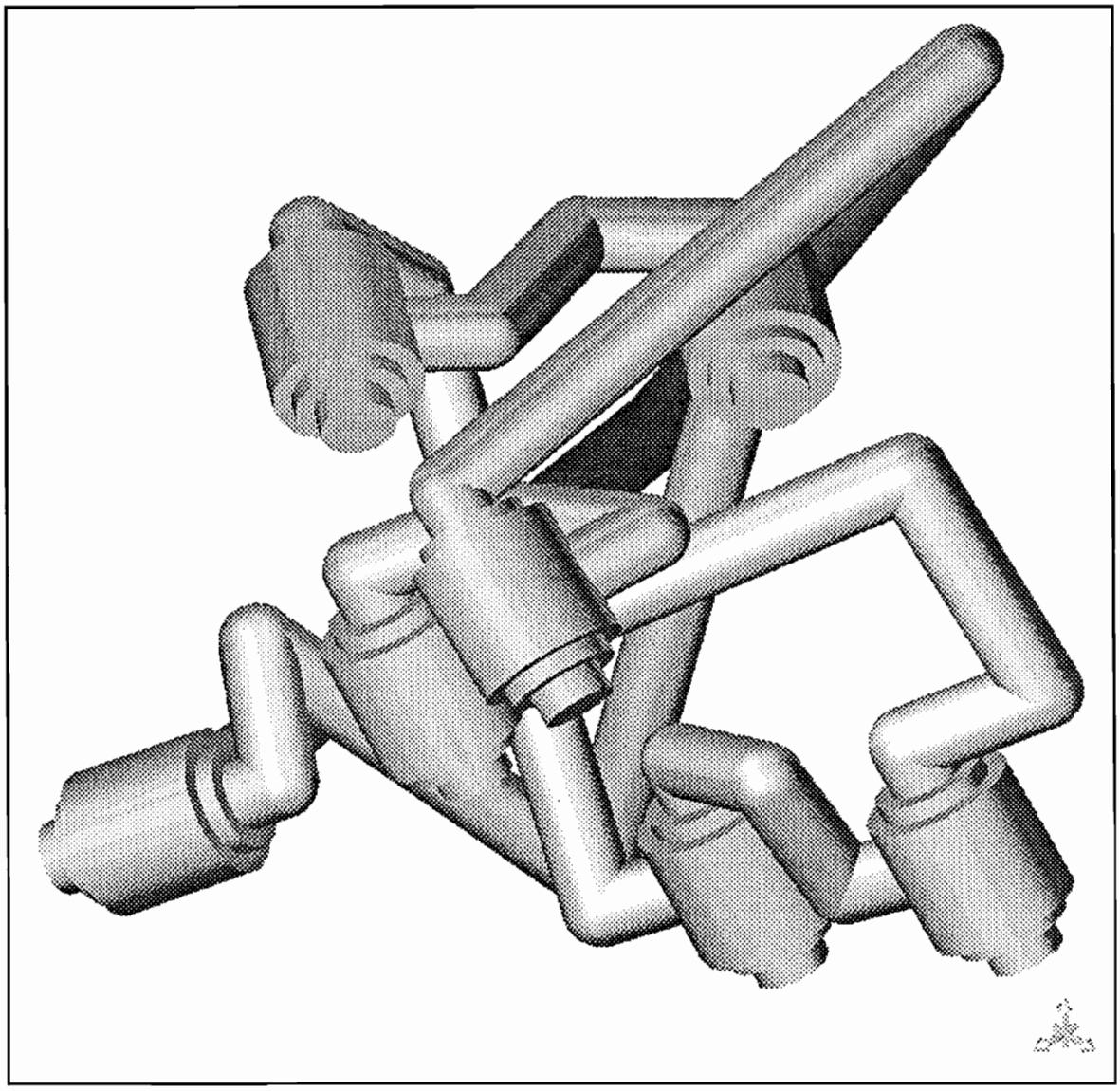

Figure 60. 7R mechanism with interference eliminated after editing

not appreciated until actual time is considered. A four link mechanism can take 2 hours to exhaust all options on an Apollo DN3000 workstation.

Chapter 9

Conclusions

This work proves the feasibility of generating interference-free spatial mechanisms. As the PriSM output of Figures 48 through 60 shows, these are not simple kinematic sketches. The models can be used to create actual hardware, and hardware built to a model's specifications will have no interferences. The solution presented here does not find the minimum link length solution, but it does serve to prove the feasibility of constructing a mechanism. Combined with the methods for preprocessing, and visualization, the techniques for interference elimination provide powerful tools for mechanism development. The preprocessor serves not only as a powerful generalized input generator, but also serves as a means of checking any custom interfaces from other analysis programs. In addition to providing the tools for spatial mechanism design, there were several accomplishments pertaining to interference detection and elimination techniques.

The interference detection study began by investigating the relative performance of three different techniques for the detection of line segment intersections. The author developed algorithms based on these techniques and a

method for testing their relative performance. Finding no significant differences in the real machine cycles used by the three algorithms, the focus of the investigation was directed toward detecting nearness. Viewing the three-dimensional problem with parallel coordinates allowed for the development of an algorithm to do gross intersection checks, which nearly halved the time to check for intersections in 1,999,000 combinations of randomly generated lines.

The coarse interference detection algorithm which resulted from the parallel coordinate observations was actually a dynamically changing bounding parallelepiped method. The method effectively reduced the coarse interference checking to one where objects are projected onto one-dimensional spaces. In these spaces it was sufficient to look for simple overlaps in projections. The vector loop method [14], which the author developed for absolute determination of interferences, has proven to be very robust for axisymmetric models. The method is unique in that it will determine intersections as well as minimum distance between line segments without resorting to iteration methods.

This represents the first time that the concept of joint freedoms has been formally expressed as a means of eliminating joint interference. A straight line unidirectional avoidance strategy, based on these freedoms, is outlined which guarantees successful interference elimination for revolute, cylindric, and prismatic joint types.

The method for optimal orientation of ball–cup pairs for spheric joints is unique. The method is based on mapping the projection of an inverse image of the ball's attachment piece onto a unit sphere in the cup's local frame of reference. The axis of a bounding cone is then used to align the cup and its attachment piece.

This serves to minimize the cup's chances for collision with the ball's attachment piece.

The avoidance strategy, based on a three segment scheme for attaching joints of a link, has proven to be adequate for the mechanisms tested. The elliptical projection technique has proven to be very useful in simplifying the problem of interference avoidance for the link elements. These methods eliminated the need for time consuming iterative methods. In fact, no iterative methods were used for avoidance or detection.

More work is needed concerning the positioning of joints. The current method, which freezes joints once they no longer interfere with other joints, forces a manual edit of input files to obtain an interference-free solution in some cases. This was seen in the RCCC example. A method for reversing the direction of joint axes could prove helpful in finding more compact forms of mechanisms, as was illustrated in the RPCS example. It is anticipated, however, that these enhancements will greatly increase program run times.

Future work might involve swept volume techniques to optimize link element lengths. The methods employed here make no attempt at such an optimization. Using swept volumes, joint attachments could be made by finding the shortest path around volumes swept relative to the to a link's local reference frame.

References

1. Keil, M. J., *A Method for Automatically Generating and Animating 3-D Models of Planar Linkages*, Master's Thesis, Florida Atlantic University, 1984.

2. Keil, M. J., Myklebust, A., "Automatic 3-D Geometric Modeling and Animation of Planar Mechanisms", *Proceedings of the JSPE International Symposium on Design and Synthesis*, Tokyo, July 1984, pp. 395–399.

3. Myklebust, A., Reinholtz, C. F., Francis, W. H., and Keil, M. J., "Design of a Radar Guidance Mechanism Using MECSYN-ANIMEC", ASME Paper No. 84–DET–139.

4. Sheth, P. N., and Uicker, J. J., Jr., "IMP (Integrated Mechanisms Program), A Computer-Aided Design Analysis System for Mechanisms and Linkages", *Transactions of the ASME, Journal of Engineering for Industry*, Vol. 94, May 1972, pp. 454–464.

5. Rubel, A. J., and Kaufman, R. E., "KINSYN III: A New Human-Engineered System for Interactive Computer-Aided Design of Planar Linkages", *Transactions of the ASME, Journal of Engineering for Industry*, Vol. 99, No.2, May 1977, pp. 440–448.

6. Myklebust, A., and Tesar, D., "The Analytical Synthesis of Complex Mechanisms for Combinations of Specified Geometric or Time Derivatives up to the Fourth Order," *Transactions of the ASME, Journal of Engineering for Industry*, Vol. 97, May 1975, pp. 714–722.

7. Sivertsen, O., and Myklebust, A., "MECSYN: An Interactive Computer Graphics System for Mechanism Synthesis by Algebraic Means," ASME Design Engineering Technical Conference, Beverly Hills, California, Paper No. 80–DET–68, September 1980.

8. Erdman, A. G., Gustafson, J. E., "LINCAGES: Linkage INteractive Computer Analysis and Graphically Enhanced Synthesis Package", ASME Design Engineering Technical Conference, Paper No. 77-DET-5, September 1977.

9. Myklebust, A., M. J. Keil, M. J., and Reinholtz, C. F., "MECSYN-IMP-ANIMEC: Foundation for a New Computer-Aided Spatial Mechanism Design System", *Journal of Mechanism and Machine Theory*, Vol. 20, No. 4, 1985, pp. 257-269.

10. Thatch, B. R., *A PHIGS Based Interactive Graphical Preprocessor for Spatial Mechanism Analysis and Synthesis*, Master's Thesis, Virginia Polytechnic Institute and State University, 1987.

11. Thatch, B. R., and Myklebust, A., "A PHIGS Based Graphics Input Interface for Spatial-Mechanism Design," *IEEE Computer Graphics and Applications*, Vol. 8, No. 2, March 1988, pp. 26-38.

12. Pennington, S. L., *Automatic Geometric Modeling of Spatial Mechanism Links*, Master's Thesis, Virginia Polytechnic Institute and State University, 1986.

13. Pennington, S. L., Keil, M. J., and Myklebust, A., "Automatic Generation of Spatial Mechanisms Geometric Models to Avoid Link Interference", *Proceedings of the Seventh World Congress on the Theory of Machines and Mechanisms*, Vol 2., Sevilla, Spain, 1987, pp. 285-288.

14. Keil, M. J., Myklebust, A., Reinholtz, C. F., "Prediction of Link Interference in Planar Mechanisms", (awards paper) 9th Applied Mechanisms Conference, Kansas City, MO, October 1985.

15. Keil, M. J., Myklebust, A., "An Improved Interference Detection Method for Spatial Mechanism Geometric Models", *Proceedings of The 1988 ASME Design Technology Conferences 20th Biennial Mechanisms Conference*, Kissimmee, FL, DE-Vol. 15-2, September 1988, pp. 83-89.

16. Montgomery, D. E., *An Interactive PHIGS+ Model Rendering System Applied to Postprocessing of Spatial Mechanisms*, Master's Thesis, Virginia Polytechnic Institute and State University, 1990.

17. Montgomery, D. E., Keil, M. J., Myklebust, A., "A PHIGS+ Model Rendering System for Simulation of Spatial Mechanisms", submitted to the *2nd International Workshop on Advances in Robot Kinematics*, Linz, Austria, September 1990.

18. Udupa, S. M., "Collision Detection and Avoidance in Computer Controlled Manipulators", *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, Cambridge, MA, 1977, pp. 737–748.

19. Boyse, J. W., "Interference Among Solids and Surfaces", *Communications of the Association for Computing Machinery*, Vol. 2, No. 1, January 1979, pp. 3–9.

20. Lozano-Perez, T., and Wesley, M. A., "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles", *Communications of the ACM*, Vol. 22, No. 1, October 1979, pp. 560–570.

21. Ahuja, N., Chien, R. T., Yen, R., and Bridwell, N., "Interference Detection and Collision Avoidance Among Three Dimensional Objects", *Proceedings of the First Annual Conference on AI*, September 1980, pp. 44–48.

22. Lozano-Perez, T., "Automatic Planning of Manipulator Transfer Movements", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC–11, No. 10, October 1981, pp. 681–698.

23. Meyer, J., "An Emulation System for Programmable Sensory Robots", *IBM Journal of Research and Development*, Vol. 25, No. 6, November 1981, pp. 955–961.

24. Inselberg, A., "N-dimensional Graphics Part 1 – Lines and Hyperplanes", IBM Los Angeles Scientific Center Report no. G320–2711, July 1981.

25. Inselberg, A., "Intelligent Instrumentation and Process Control", Proceedings of the IEEE Conference on Artificial Intelligence Applications, December 1985.

26. Inselberg, A., "The Plane with Parallel Coordinates", Special Issue on Computational Geometry The Visual Computer, 1985, 1:69–91.

27. Inselberg, A., "Parallel Coordinates for Visualizing Multi-dimensional Geometry", 5th International Conference on Computer Graphics, Japan, May 1987.

28. Myers, J. K., and Agin, G. J., "A Supervisory Collision Avoidance System for Robot Controllers", *Proceedings of the ASME Robotics Research and Advanced Applications Symposium*, New York, NY, 1982, pp. 225–232.

29. Brooks, R. A., "Solving the Find-Path Problem by Good Representation of Free Space", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC–13, No. 3, March/April 1983, pp. 190–197.

30. Brooks, R. A., "Planning Collision-Free Motions for Pick-and-Place Operations", *The International Journal of Robotics Research*, Vol. 2, No. 4, Winter 1983, pp. 19–44.

31. Lozano-Perez, T., "Spatial Planning: A Configuration Space Approach", *IEEE Transactions on Computers*, Vol. C–32, No. 2, February 1983, pp. 108–120.

32. Gouzenes, L., "Generation of Collision-Free Trajectories for Mobile and Manipulator Robots", *Artificial Intelligence Proceedings of the IFAC Symposium*, Leningrad, USSR, October 1983, pp. 265–272.

33. Schwartz, J. T., Sharir, M., "On the Piano Movers' Problem: III. Coordinating the Motion of Several Independent Bodies: The Special Cases of Circular Bodies Moving Amidst Polygonal Barriers", *The International Journal of Robotics Research*, Vol. 2, No. 3, Fall 1983, pp. 46–74.

34. Hopcroft, J. E., Schwartz, J. T., and Sharir, M., "Efficient Detection of Intersections among Spheres", *The International Journal of Robotics Research*, Vol. 3, No. 4, Winter 1983, pp. 77–80.

35. de Pennington, A., Bloor, M. S., and Blila, M., "Geometric Modelling: A Contribution Towards Intelligent Robots", SME Technical Paper MS83–339.

36. Davis, R. H., and Camacho, M., "The Application of Logic Programming to the Generation of Paths for Robots", *Robotica*, Vol. 2, 1984, pp. 93–103.

37. Faverjon, B., "Obstacle Avoidance Using an Octree in the Configuration Space of a Manipulator", *IEEE International Conference on Robotics and Automation*, Atlanta, GA., March 1984, pp. 504–512.

38. Freund, E., and Hoyer, H., "Collision Avoidance in Multi-Robot Systems", *Second International Symposium on Robotics Research*, Kyoto, Japan, August 1984, pp. 135–146.

39. Freund, E., and Hoyer, H., "Collision Avoidance for Industrial Robots with Arbitrary Motion", *Journal of Robotic Systems*, 1(4), 1984, pp. 317–329.

40. Gouzenes, L., "Collision Avoidance for Robots in an Experimental Flexible Assembly Cell", *IEEE International Conference on Robotics and Automation*, Atlanta, GA., March 1984, pp. 474–476.

41. Gouzenes, L., "Strategies for Solving Collision-free Trajectories Problems for Mobile and Manipulator Robots", *The International Journal of Robotics Research*, Vol. 3, No. 4, Winter 1984, pp. 51–65.

42. Grechanovsky, E. and Pinsker, I. S., "An Algorithm for a Computer-Controlled Manipulator While Avoiding Obstacles", *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, 1984, pp. 807–813.

43. Hogan, N., "Impedance Control: An Approach to Manipulation", *Proceedings of the 1984 American Control Conference*, June 1984, San Diego, CA, pp. 304–313.

44. Hopcroft, J. E., Schwartz, J. T., and Sharir, M., "On the Complexity of Motion Planning for Multiple Independent Objects; PSPACE-Hardness of the 'Warehouseman's Problem'", *The International Journal of Robotics Research*, Vol. 3, No. 4, Winter 1984, pp. 77–80.

45. Larson, G., and Donath, M., "Interactive Interference Checking for Robot Workcell Emulation", *Proceedings of the Computers in Engineering Conference 1984*, Vol. 1, Las Vegas, NV, August 1984, pp. 107–110.

46. Larson, G., and Donath, M., "Animated Simulation of Intelligent Robot Workcells", *ROBOTS9 Conference Proceedings*, Vol. 2, June 1985, pp. 19–54.

47. Lozano-Perez, T., "Automatic Synthesis of Fine-Motion Strategies for Robots", *The International Journal of Robotics Research*, Vol. 3, No. 1, Spring 1984, pp. 3–23.

48. Park, W. T., "State-space Representations for Coordination of Multiple Manipulators", *14th International Symposium on Industrial Robotics*, Gothenburg, Sweden, October 1984, pp. 397–405.

49. Red, W. E., "Configuration Maps for Robot Task Planning in 3-D", *Proceedings of the 1984 International Computers in Engineering Conference*, Las Vegas, NV, August 1984.

50. Red, W. E., "Minimum Distances for Robot Task Simulation", *Robotica*, Vol. 1, 1984, pp. 231–238.

51. Red, W. E., and Truong-Cao, H. V., "The Configuration Space Approach to Robot Path Planning", *Proceedings of the 1984 American Control Conference*, June 1984, San Diego, CA, pp. 115–121.

52. Red, W. E., "Configuration Maps for Robot Task Planning in Two Dimensions", *Transactions of the ASME, Journal of Dynamic Systems, Measurement, and Control*, Vol. 107, December 1985, pp. 292–298.

53. Red, W. E., Truong-Cao, H. V., and Kim, K. H., "Robot Path Planning in Three-Dimensions Using the Direct Subspace", *Transactions of the ASME, Journal of Dynamic Systems, Measurement, and Control*, Vol. 109, September, 1987, pp. 238–244.

54. Red, W. E., and Kim, K. H., "Dynamic Direct Subspaces for Robot Path Planning", *Robotica*, Vol. 5, 1987, pp. 29–36.

55. Brooks, R. A., and Lozano-Perez, T., "A Subdivision Algorithm in Configuration Space for Findpath with Rotation", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 15, No. 2, 1985, pp. 224–233.

56. Cameron, S., "An Implementation of Clash Detection by Four-Dimensional Intersection Tests", *SIAM Conference on Geometric Modelling and Robotics*, Albany NY, 1985.

57. Cameron, S., "A Study of the Clash Detection Problem in Robotics", *Proceedings of the IEEE International Conference on Robotics and Automation*, St. Louis, MO, July 1985, pp. 488–493.

58. Clermont, G., Gaspart, P., and Lecoco, P., "Generating 3D-Collision-Free Paths for a Computer Controlled Robot", *Proceedings of '85 International Conference on Advanced Robotics*, Tokyo, Japan, September 1985, pp. 59–67.

59. Donald, B. R., "On Motion Planning with Six Degrees of Freedom: Solving the Intersection Problems in Configuration Space", *Proceedings of the IEEE International Conference on Robotics and Automation*, St. Louis, MO., July 1985, pp. 536–540.

60. Ganter, M. A., *Dynamic Collision Detection Using Kinematics and Solid Modelling Techniques*, Dissertation, University of Wisconsin-Madison, 1985.

61. Ganter, M. A., and Uicker, J. J., "Dynamic Collision Detection Using Swept Solids," *Transactions of the ASME, Journal of Mechanisms, Transmissions, and Automation in Design*, December 1986, Vol. 180, pp. 549–555.

62. Ganter, M. A., and Uicker, J. J., "Generating Swept Solids for Convex Objects", ASME paper 86-DET-110.

63. Hasegawa, T., "Collision Avoidance Using Characterized Description of Free Space", *Proceedings of '85 International Conference on Advanced Robotics*, Tokyo, Japan, September 1985, pp. 69–76.

64. Kovesi, P. D., "Collision Avoidance", *Proceedings of '85 International Conference on Advanced Robotics*, Tokyo, Japan, September 1985, pp. 51–58.

65. Laugier, C., and Germain, F., "An Adaptive Collision-Free Trajectory Planner", *Proceedings of '85 International Conference on Advanced Robotics*, Tokyo, Japan, September 1985, pp. 33–41.

66. Nagata, T., Honda, K., and Teramoto, Y., "Multi-Robot Plan Generation in a Continuous Domain", *Proceedings of '85 International Conference on Advanced Robotics*, Tokyo, Japan, September 1985, pp. 119–126.

67. Valade, J. "Geometric Reasoning and Automatic Synthesis of Assembly Trajectory", *Proceedings of '85 International Conference on Advanced Robotics*, Tokyo, Japan, September 1985, pp. 43–50.

68. Wong, E. K., and Fu, K. S., "A Heirarchical-Orthogonal Approach to Collision-Free Path Planning", *Proceedings of the IEEE International Conference on Robotics and Automation*, St. Louis, MO, July 1985, pp. 506–510.

69. Baldur, R., and Dube, M., "Interference Between Generalized Solid Links", *Proceedings of the 1986 ASME International Computers in Engineering Conference and Exhibition*, Chicago, IL, July 1986, pp. 79–86.

70. Hayward, V., "Fast Collision Detection Scheme by Recursive Decomposition of a Manipulator Workspace", *IEEE International Conference on Robotics and Automation*, Vol. 2, 1986, pp. 1044–1047.

71. Herman, M., "Fast, Three-Dimensional, Collision-Free Motion Planning", *IEEE International Conference on Robotics and Automation*, Vol. 2, 1986, pp. 1056–1063.

72. Khatib, O., "Real-Time Obstacle Avoidance for Manipulators and Robots", *The International Journal of Robotics Research*, Vol. 5, No. 1, Spring 1986.

73. Lumelsky, V. J., "Continuous Motion Planning in Unknown Environment for a 3D Cartesian Robot Arm", *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, CA, April 1986, pp. 1050–1055.

74. Lumelsky, V. J., "Dynamic Path Planning for a Planar Articulated Robot Arm Moving Amidst Unknown Objects", *Automatica*, Vol. 23, No. 5, 1987, pp. 551–570.

75. Lumelsky, V. J., "Path _Planning Strategies for a Point Mobile Automaton Moving Amidst Unknown Obstacles of Arbitrary Shape", *Algorithmica*, Vol. 2, 1987, pp. 403–430.

76. Sun, K., and Lumelsky, V., "Computer Simulation of Sensor-Based Robot Collision Avoidance in an Unknown Environment", *Robotica*, Vol. 5, 1987, pp. 291–302.

77. Dupont, P. E., and Derby, S., "Planning Collision Free Paths for Redundant Robots Using a Selective Search of Configuration Space", ASME Paper 86–DET–145.

78. Young, L., and Duffy, J., "A Theory for the Articulation of Planar Robots: Part II – Motion Planning Procedure for Interference Avoidance", ASME Paper 86–DET–76.

79. Erdmann, M., and Lozano-Perez, T., "On Multiple Moving Objects", *Algorithmica*, Vol. 2, 1987, pp. 476–521.

80. Grau, S., Collingwood, M. C., and Blount, G. N., "A Minimum Distance Algorithm for Moving Objects", *Proceedings of the Seventh World Congress on the Theory of Machines and Mechanisms*, Vol. 2, Sevilla, Spain, 1987, pp. 1011–1014.

81. O'Dunlaing, C., "Motion Planning with Inertial Constraints", *Algorithmica*, Vol. 2, 1987, pp. 431–475.

82. Oommen, B. J., and Reichstein, I., "On the Problem of Translating an Elliptic Object Through a Workspace of Elliptic Obstacles", *Robotica*, Vol. 5, 1987, pp. 187–196.

83. Pappadimitriou, C. H., and Silverberg, E. B., "Optimal Piecewise Linear Motion of an Object Among Obstacles", *Algorithmica*, Vol. 2, 1987, pp. 523–539.

84. Sifrony, S., and Sharir, M., "A New Efficient Motion-Planning Algorithm for a Rod in Two-Dimensional Polygonal Space", *Algorithmica*, Vol. 2, 1987, pp. 367–402.

85. Xing, D. M., Barraco, A., and Cuny, B., "A Hierarchical Scheme for Trajectory Generation of a General Robot Manipulator", *Proceedings of the Seventh World Congress on the Theory of Machines and Mechanisms*, Vol 2., Sevilla, Spain, 1987, pp. 1103–1109.

86. Amirouche, F. M. L., and Jia, T., "A Look at the Collision Avoidance Conditions in the Motion Coordination of Multi-Arm Robot Systems", *Proceedings of the 1988 ASME International Computers in Engineering Conference and Exhibition*, San Francisco, CA, July 1988, pp. 411–417.

87. Amirouche, F. M. L., and Jia, T., "Spatial Coordination of a Two Arm Robot System Subjected to Kinematical and Geometric Constraints", *Proceedings of the 1988 ASME International Computers in Engineering Conference and Exhibition*, San Francisco, CA, July 1988, pp. 419–424.

88. Choi, Y. J., Crane, C. D., and Matthew, G. K., "Interactive Off-Line Robot Path Processor", *Proceedings of the 1988 ASME International Computers in Engineering Conference and Exhibition*, San Francisco, CA, July 1988, pp. 411–417.

89. Fletcher, R. W., and Goldenberg, A. A., "Collision Avoidance for Robot Manipulators: Application to CATIA/IBM 7565 Interface", *Journal of Robotic Systems*, 5 (2), 1988, pp. 125–146.

90. Hurteau, G., and Stewart, N. F., "Distance Calculation for Imminent Collision Indication in a Robot System Simulation", *Robotica*, Vol. 6, 1988, pp. 47–51.

91. Palma-Villalon, E., and Dauchez, P., "World Representation and Path Planning for a Mobile Robot", *Robotica*, Vol. 6, 1988, pp. 33–40.

92. Quiocho, L. J., and Allen, R., H., "RCAPS: Progress Toward Standardizing the Find-Path Problem", *Proceedings of the 1988 ASME International Computers in Engineering Conference and Exhibition*, San Francisco, CA, July 1988, pp. 377–381.

93. Zhu Q., and Freeman, H., "Combined Spatial-Sorting and Model-Reference Approach for Fast Interference Detection", *Proceedings of the 1988 ASME International Computers in Engineering Conference and Exhibition*, San Francisco, CA, July 1988, pp. 403–409.

94. Buchal, R. O., and Cherchas, D. B., "An Iterative Method for Generating Kinematically Feasible Interference-Free Robot Trajectories", *Robotica*, Vol. 7, 1989, pp. 119–127.

95. Dai, F., "Collision-Free Motion of an Articulated Kinematic Chain in a Dynamic Environment", *IEEE Computer Graphics and Applications,* Vol. 9, No. 1, January 1989, pp. 70–74.

96. Jacak, W., "Strategies of Searching for Collision-free Manipulator Motions: Automata Theory Approach", *Robotica*, Vol. 7, 1989, pp. 129–138.

97. Ku, T. S., and Ravani, B., "A Separating Channel Decomposition Algorithm for Nonconvex Polygons With Application in Interference Detection", *Transactions of the ASME, Journal of Mechanisms, Transmissions, and Automation in Design*, Vol. 111, December 1989, pp. 270–277.

98. Ozaki, H., Shimadzu, T., and Mohri, A., "Collision-free Path Generation for a Mobile Robot by an Artificial Transformation of Obstacle Spaces", *Robotica*, Vol. 7, 1989, pp. 139–142.

99. Shin, Y., and Bien, Z., "Collision-free Trajectory Planning for Two Robot Arms", *Robotica*, Vol. 7, 1989, pp. 205–212.

100. Mortenson, M. E., *Geometric Modeling*, John J. Wiley and Sons, Inc., 1985.

101. Cohan, S. M., *Mobility Analysis of Mechanisms Through the Parallel Coordinate System*, Master's Thesis, University of California, Los Angeles, 1984.

102. Cohan, S. M., and Yang, D. C. H., "Mobility Analysis of Planar Four-Bar Mechanisms Through the Parallel Coordinate System", *Mechanism and Machine Theory*, Vol. 21, No. 1, 1986, pp. 63–71.

103. Shigley, J. E., and Uicker, J. J., *Theory of Machines and Mechanisms*, McGraw-Hill, Inc., 1980.

104. Williams, R. L., *Synthesis and Analysis of the RSSR Spatial Mechanism for Function Generation*, Master's Thesis, Virginia Polytechnic Institute and State University, 1985.

105. Veeraraghavan, A., *Computer Aided Design of the RCCC Spatial Mechanism*, Master's Thesis, Virginia Polytechnic Institute and State University, 1986.

106. Sandor, G. N., Xu, and Y. Weng, T., "On the Elimination of Branching and the Synthesis of 7-R Spatial Motion Generators", 9th Applied Mechanisms Conference, Kansas City, MO, October 1985.

Appendix A.
Sample Input for IMP75

```
GROUND=L4
REVLUT(L4,L1)=A
CYLNDR(L1,L2)=B
CYLNDR(L2,L3)=C
CYLNDR(L3,L4)=D
DATA/LINK (L4,A)=-21.06,10.92,-5.33/-20.746,10.378,-4.55/$
-21.219,10.386,-5.637/
DATA/LINK (L4,D)=-12.19,29.8,3.52/-11.321,30.136,3.883/$
-11.731,29.236,2.943231/
DATA/LINK(L1,B)=-18.6,14.85,-10.01/-18.319,14.487,-9.121/$
-18.759,14.316,-10.17779/
DATA/LINK (L1,A)=-21.06,10.92,-5.33/-20.746,10.378,-4.55/$
-21.219,10.386,-5.637/
 DATA/LINK(L2,C)=-7.23,17.98,2.59/-7.537,17.779,3.52/$
-7.615,18.9,2.6617473/
 DATA/LINK(L2,B)=-18.6,14.85,-10.01/-18.319,14.487,-9.121/$
-18.759,14.316,-10.17779/
 DATA/LINK(L3,D)=-12.19,29.8,3.52/-11.321,30.136,3.883/$
-11.731,29.236,2.943231/
 DATA/LINK(L3,C)=-7.23,17.98,2.59/-7.537,17.779,3.52/$
-7.615,18.9,2.6617473/
POINT(L4)=L4O
```

```
DATA/POINT(L4O,A)=0,0,0
POINT(L4)=L4X
DATA/POINT(L4X,A)=1,0,0
POINT(L4)=L4Y
DATA/POINT(L4Y,A)=0,1,0
POINT(L4)=L4Z
DATA/POINT(L4Z,A)=0,0,1
POINT(L1)=L1O
DATA/POINT(L1O,B)=0,0,0
POINT(L1)=L1X
DATA/POINT(L1X,B)=1,0,0
POINT(L1)=L1Y
DATA/POINT(L1Y,B)=0,1,0
POINT(L1)=L1Z
DATA/POINT(L1Z,B)=0,0,1
POINT(L2)=L2O
DATA/POINT(L2O,C)=0,0,0
POINT(L2)=L2X
DATA/POINT(L2X,C)=1,0,0
POINT(L2)=L2Y
DATA/POINT(L2Y,C)=0,1,0
POINT(L2)=L2Z
DATA/POINT(L2Z,C)=0,0,1
POINT(L3)=L3O
DATA/POINT(L3O,D)=0,0,0
POINT(L3)=L3X
DATA/POINT(L3X,D)=1,0,0
POINT(L3)=L3Y
DATA/POINT(L3Y,D)=0,1,0
POINT(L3)=L3Z
DATA/POINT(L3Z,D)=0,0,1
DATA/POSITN(A)=0,10,36
DATA/VELO(A)=1
PRINT/POSITN(L4O,L4X,L4Y,L4Z,L1O,L1X,L1Y,L1Z,L2O,L2X,L2Y,L2Z,$
L3O,L3X,L3Y,L3Z)
EXECUTE
FINISH
```

Appendix B.
IPOST Program Listing

```
PROGRAM IPOST

REAL*8 END1(20,3),END2(20,3),AXIS(20,3),MAGAX
REAL*8 O(20,3),X(20,3),Y(20,3),Z(20,3),TEMPX,TEMPY,TEMPZ
REAL*8 OFX(20),OFY(20),OFZ(20),OFTZ(20),OFTY(20),OFTX(20)
REAL*8 MAGOX(20),MAGOY(20),MAGOZ(20),MAGX,MAGY,MAGZ
REAL*8 TUX(3),UX(3),UY(3),UZ(3),RJ(3),PHI,TX,TY,TZ,PI
REAL*8 SMALL,SMALL2,RO,RI,RL,RS,RGB1,RGB2,RGB3,VZ(3),VX(3)
INTEGER I,IC,POS1,POS2,POSB,REV,CYL,PRI,SPH,UJO,DAT,PAR,COM,J,K
INTEGER LC(20,2),NJ,JCON(20,2),JID(20),IN,LEN1,LEN2,N,N1,N2,NT
INTEGER IORD,ORD(20),CAT(20,2),JLEN(20),DATI1,DATI2,LNKI1,LNKI2
INTEGER NPOS,ICODE,IP
CHARACTER *80 GRND,JOINT(20),TLINK1,TLINK2,NDAT(40),JNAME(20)
CHARACTER *80 JDUM
CHARACTER *80 TRIADO,TRIADX,TRIADY,TRIADZ
CHARACTER *2 LINK(20,2),TL
CHARACTER *8 MOBIL
CHARACTER *20 MECHN,MECH2
LOGICAL F2
```

```
      JDUM=' '
      PI=DACOS(-1.D0)
      IC=0
      IN=8
      WRITE(6,*)'WHAT MECHANISM?'
      READ(5,4)MECHN
4     FORMAT(A20)
      IP=INDEX(MECHN,' ')-1
      IF(IP.EQ.0)THEN
         WRITE(6,*)'ERROR IN MECHANISM NAME'
         STOP
      END IF
      MECH2=MECHN(1:IP)//'.OUT'
      OPEN(8,FILE=MECH2)
5     READ(IN,20)GRND
      IC=IC+1
20    FORMAT(A80)
      POS1=INDEX(GRND,'GROUND=')
      IF(IC.GT.10)THEN
         WRITE(6,*)'ERROR*** GROUND NOT SPECIFIED'
         STOP
      END IF
      IF(POS1.EQ.0) GO TO 5
      POS2=INDEX(GRND,'=')+1
      GRND=GRND(POS2:80)
      DO 30 I=1,80
         POSB=INDEX(GRND,' ')
         IF(POSB.EQ.1)THEN
            GRND=GRND(2:80)
         ELSE
            GO TO 31
         END IF
30    CONTINUE
      WRITE(6,*)'ERROR*** GROUND NOT SPECIFIED'
      STOP
31    CONTINUE
      DO 35 I=1,20
         LINK(I,1)='  '
         LINK(I,2)='  '
         LC(I,1)=0
         LC(I,2)=0
35    CONTINUE
      N=0
* GROUND LINK IS KNOWN
* IDENTIFY THE JOINTS AND THEIR LINK CONNECTIVITY
      DO 40 I=1,20
         READ(IN,20)
         READ(IN,20)JOINT(I)
         JNIND=INDEX(JOINT(I),'=')+1
         JNAME(I)=JOINT(I)(JNIND:80)
45       CONTINUE
         IF(INDEX(JNAME(I),' ').EQ.1)THEN
            IF(JNAME(I).EQ.JDUM)THEN
               STOP' ERROR IN JNAME '
            END IF
            JNAME(I)=JNAME(I)(2:80)
            GO TO 45
         END IF
         JLEN(I)=INDEX(JNAME(I),' ')-1
```

```
      DAT=INDEX(JOINT(I),'DAT')
      NJ=I
      IF(DAT.GT.0)THEN
         NDAT(1)=JOINT(I)
         IF(INDEX(NDAT(1),'$').GT.0)THEN
            READ(IN,20)
         END IF
         NJ=I-1
         GO TO 41
      END IF
      JCON(I,1)=I
40    CONTINUE
41    CONTINUE

      DO 46 I=1,NJ
         PAR=INDEX(JOINT(I),'(')+1
         COM=INDEX(JOINT(I),',')-1
         TLINK1=JOINT(I)(PAR:COM)
47       CONTINUE
         IF(INDEX(TLINK1,' ').EQ.1)THEN
            TLINK1=TLINK1(2:80)
            IF(TLINK1.EQ.JDUM)THEN
               STOP' ERROR IN TLINK1A '
            END IF
            GO TO 47
         END IF
         LEN1=INDEX(TLINK1,' ')-1
         F2=.FALSE.
         DO 52 J=1,NJ
            PAR=INDEX(JOINT(J),')')-1
            COM=INDEX(JOINT(J),',')+1
            TLINK2=JOINT(J)(COM:PAR)
48          CONTINUE
            IF(INDEX(TLINK2,' ').EQ.1)THEN
               TLINK2=TLINK2(2:80)
               IF(TLINK2.EQ.JDUM)THEN
                  STOP' ERROR IN TLINK2A '
               END IF
               GO TO 48
            END IF
            LEN2=INDEX(TLINK2,' ')-1
            IF(TLINK1(1:LEN1).EQ.TLINK2(1:LEN2))THEN
               N=N+1
               JCON(J,2)=I
               F2=.TRUE.
            END IF
52       CONTINUE
53       CONTINUE
         IF(.NOT.F2)THEN
            WRITE(6,*)'ERROR IN FINDING N2'
            STOP
         END IF
46    CONTINUE
      DO 70 I=1,NJ
         REV=INDEX(JOINT(I),'REV')
         CYL=INDEX(JOINT(I),'CYL')
         PRI=INDEX(JOINT(I),'PRI')
         SPH=INDEX(JOINT(I),'SPH')
         UJO=INDEX(JOINT(I),'UJO')
```

```
         N1=JCON(I,1)
         N2=JCON(I,2)
         IF(REV.GT.0)THEN
            JID(I)=1
            LINK(N1,1)='RE'
            LINK(N2,2)='RI'
            LC(N1,1)=I
            LC(N2,2)=I
         ELSE IF(CYL.GT.0)THEN
            JID(I)=2
            LINK(N1,1)='CE'
            LINK(N2,2)='CI'
            LC(N1,1)=I
            LC(N2,2)=I
         ELSE IF(PRI.GT.0)THEN
            JID(I)=3
            LINK(N1,1)='PE'
            LINK(N2,2)='PI'
            LC(N1,1)=I
            LC(N2,2)=I
         ELSE IF(SPH.GT.0)THEN
            JID(I)=4
            LINK(N1,1)='SE'
            LINK(N2,2)='SI'
            LC(N1,1)=I
            LC(N2,2)=I
         ELSE IF(UJO.GT.0)THEN
            JID(I)=5
            LINK(N1,1)='UE'
            LINK(N2,2)='UI'
            LC(N1,1)=I
            LC(N2,2)=I
         ELSE
            WRITE(6,*)'MAJOR FOUL UP IN SETTING JOINTS'
            STOP
         END IF
70    CONTINUE

* FIND THE GROUND LINK
      DO 80 I=1,N
         PAR=INDEX(JOINT(I),'(')+1
         COM=INDEX(JOINT(I),',')-1
         TLINK1=JOINT(I)(PAR:COM)
         LEN1=INDEX(TLINK1,' ')-1
         IF(INDEX(GRND,TLINK1(1:LEN1)).GT.0)THEN
            K=I
            ORD(1)=I
            GO TO 81
         END IF
80    CONTINUE
81    CONTINUE
* WITH THE GROUND LINK SET IN ORDER 1, ORDER THE REST USING MODULO
* ARITHMETIC
      DO 90 I=2,N
         J=I+K-1
         IORD=MOD(J,N)
         IF(IORD.EQ.0) IORD=N
         ORD(I)=IORD
90    CONTINUE
```

```
*  THE LINK TO JOINT AND JOINT TO LINK CONNECTIVITY IS KNOWN
*  THE ORDERING OF LINKS WITH RESPECT TO GROUND IS ALSO KNOWN IN ORD
*  NOW, PICK UP THE REST OF THE DATA STATEMENTS
      N2=2*N
      DO 100 I=2,N2
         READ(IN,20)
         READ(IN,20)NDAT(I)
         IF(INDEX(NDAT(I),'$').GT.0)THEN
            READ(IN,20)
         END IF
100   CONTINUE
* NOW, ASSOCIATE THE DATA STATEMENTS WITH THEIR LINKS IN PROPER ORDER
* AS TO INTERNAL PORTION AND TO EXTERNAL PORTION
      DO 110 I=1,N
         DO 120 J=1,N2
* LOCATE THE JOINT NAME
            LNKI1=INDEX(NDAT(J),'(')+1
            DATI1=INDEX(NDAT(J),',')+1
            LNKI2=DATI1-2
            DATI2=INDEX(NDAT(J),')')-1
            PAR=INDEX(JOINT(LC(I,1)),'(')+1
            COM=INDEX(JOINT(LC(I,1)),',')-1
            TLINK1=JOINT(LC(I,1))(PAR:COM)
105         CONTINUE
            IF(INDEX(TLINK1,' ').EQ.1)THEN
               IF(TLINK1.EQ.JDUM)THEN
                  STOP' ERROR IN TLINK1 '
               END IF
               TLINK1=TLINK1(2:80)
               GO TO 105
            END IF
            LEN1=INDEX(TLINK1,' ')-1
* CAT(I,1) IS THE DATA STATEMENT SEQUENCE NUMBER ASSOCIATED WITH END 1.
* CAT(I,2) IS END 2. CAT(4,1)=5 MEANS THAT DATA STATEMENT 5 GIVES THE
* INFORMATION ASSOCIATED WITH END 1 OF LINK 4. END 1 IS USED AS THE LOCAL
* ORIGIN LATER.
            IF(INDEX(NDAT(J)(LNKI1:LNKI2),TLINK1(1:LEN1)).GT.0)THEN
               IF(INDEX(NDAT(J)(DATI1:DATI2),JNAME(LC(I,1))(1:JLEN
     >         (LC(I,1)))).GT.0)THEN
                  CAT(I,1)=J
               END IF
               IF(INDEX(NDAT(J)(DATI1:DATI2),JNAME(LC(I,2))(1:JLEN
     >         (LC(I,2)))).GT.0)THEN
                  CAT(I,2)=J
               END IF
            END IF
120      CONTINUE
110   CONTINUE
* EVERYTHING IS CATALOGED NOW FIND THE LINK END LOCATIONS
      DO 130 I=1,N
         TLINK1=NDAT(CAT(I,1))
         LNKI1=INDEX(TLINK1,'=')+1
         TLINK1=TLINK1(LNKI1:80)
         LNKI1=INDEX(TLINK1,'/')-1
         TLINK1=TLINK1(1:LNKI1)
         TLINK2=NDAT(CAT(I,2))
         LNKI2=INDEX(TLINK2,'=')+1
         TLINK2=TLINK2(LNKI2:80)
         LNKI2=INDEX(TLINK2,'/')-1
```

```
         TLINK2=TLINK2(1:LNKI2)
         READ(TLINK1,140)END1(I,1),END1(I,2),END1(I,3)
140      FORMAT(3F10.0)
         READ(TLINK2,140)END2(I,1),END2(I,2),END2(I,3)
130   CONTINUE
* PICKUP THE OFFSET AXIS (AT END 2) ORIENTATION IN ABSOLUTE COORDINATES
      DO 150 I=1,N
         TLINK2=NDAT(CAT(I,2))
         LNKI2=INDEX(TLINK2,'/')+1
         TLINK2=TLINK2(LNKI2:80)
         LNKI2=INDEX(TLINK2,'/')+1
         TLINK2=TLINK2(LNKI2:80)
         LNKI2=INDEX(TLINK2,'/')-1
         TLINK2=TLINK2(1:LNKI2)
         READ(TLINK2,140)AXIS(I,1),AXIS(I,2),AXIS(I,3)
* DETERMINE THE AXIS VECTOR
         AXIS(I,1)=AXIS(I,1)-END2(I,1)
         AXIS(I,2)=AXIS(I,2)-END2(I,2)
         AXIS(I,3)=AXIS(I,3)-END2(I,3)
* NORMALIZE THE AXIS TO UNITY MAGNITUDE
         MAGAX=DSQRT(AXIS(I,1)*AXIS(I,1)+AXIS(I,2)*AXIS(I,2)+
     >               AXIS(I,3)*AXIS(I,3))
         AXIS(I,1)=AXIS(I,1)/MAGAX
         AXIS(I,2)=AXIS(I,2)/MAGAX
         AXIS(I,3)=AXIS(I,3)/MAGAX
150   CONTINUE
* THAT IS EVERYTHING THAT CAN BE GOTTEN FROM THE DATA/LINK STATEMENTS.
* NOW FIND THE FIRST POSITION OUTPUT STATEMENT
155   READ(IN,20,END=160)GRND
      POS1=INDEX(GRND,'0POSITION RESULTS')
      GO TO 161
160   STOP'AN ERROR IN FINDING THE POSITION START POINTER'
161   CONTINUE
      IF(POS1.EQ.0) GO TO 155
* THE NEXT READ IS FOR THE FIRST POSITION OF ALL LINKS
      NPOS=0
165   CONTINUE
      NPOS=NPOS+1
      DO 170 I=1,N
         READ(IN,20)TRIADO
         TRIADO=TRIADO(28:80)
         READ(TRIADO,175)O(I,1),O(I,2),O(I,3)
         READ(IN,20)TRIADX
         TRIADX=TRIADX(28:80)
         READ(TRIADX,175)X(I,1),X(I,2),X(I,3)
         READ(IN,20)TRIADY
         TRIADY=TRIADY(28:80)
         READ(TRIADY,175)Y(I,1),Y(I,2),Y(I,3)
         READ(IN,20)TRIADZ
         TRIADZ=TRIADZ(28:80)
         READ(TRIADZ,175)Z(I,1),Z(I,2),Z(I,3)
175      FORMAT(3F15.3)
         MAGOX(I)=DSQRT((X(I,1)-O(I,1))**2+(X(I,2)-O(I,2))**2+
     >                  (X(I,3)-O(I,3))**2)
         MAGOY(I)=DSQRT((Y(I,1)-O(I,1))**2+(Y(I,2)-O(I,2))**2+
     >                  (Y(I,3)-O(I,3))**2)
         MAGOZ(I)=DSQRT((Z(I,1)-O(I,1))**2+(Z(I,2)-O(I,2))**2+
     >                  (Z(I,3)-O(I,3))**2)
170   CONTINUE
```

```
* IF THIS IS THE FIRST POSITION, THERE IS NOW ENOUGH INFORMATION TO
* WRITE THE ATTRIBUTE FILE AND THE HEADER INFO FOR THE POSITION FILE
      UX(1)=1.D0
      UX(2)=0.D0
      UX(3)=0.D0
      UY(1)=0.D0
      UY(2)=1.D0
      UY(3)=0.D0
      UZ(1)=0.D0
      UZ(2)=0.D0
      UZ(3)=1.D0
      IF(NPOS.EQ.1)THEN
         DO 180 I=1,N
* FIND THE LOCAL COORDINATE OFFSET BY DOT PRODUCTS ON THE TRIAD
* COMPONENTS
            TEMPX=END2(I,1)-END1(I,1)
            TEMPY=END2(I,2)-END1(I,2)
            TEMPZ=END2(I,3)-END1(I,3)
            OFX(I)=(TEMPX*(X(I,1)-O(I,1))+TEMPY*(X(I,2)-O(I,2))+
     >              TEMPZ*(X(I,3)-O(I,3)))/MAGOX(I)
            OFY(I)=(TEMPX*(Y(I,1)-O(I,1))+TEMPY*(Y(I,2)-O(I,2))+
     >              TEMPZ*(Y(I,3)-O(I,3)))/MAGOY(I)
            OFZ(I)=(TEMPX*(Z(I,1)-O(I,1))+TEMPY*(Z(I,2)-O(I,2))+
     >              TEMPZ*(Z(I,3)-O(I,3)))/MAGOZ(I)
            IF(I.EQ.1)THEN
               SMALL=DSQRT(OFX(I)**2+OFY(I)**2+OFZ(I)**2)
            ELSE
               SMALL2=DSQRT(OFX(I)**2+OFY(I)**2+OFZ(I)**2)
               SMALL=DMIN1(SMALL,SMALL2)
               IF(SMALL.EQ.0.D0)THEN
                  STOP'ONE OF THE LINK LENGTHS IS ZERO'
               END IF
            END IF
            IF(LINK(I,2)(1:1).EQ.'U'.OR.LINK(I,2)(1:1).EQ.'S')THEN
               OFTX(I)=0.D0
               OFTY(I)=0.D0
               OFTZ(I)=0.D0
            ELSE
               RJ(1)=AXIS(I,1)
               RJ(2)=AXIS(I,2)
               RJ(3)=AXIS(I,3)
               MAGY=AXIS(I,1)*(Y(I,1)-O(I,1))+AXIS(I,2)*(Y(I,2)-O(I,2))+
     >              AXIS(I,3)*(Y(I,3)-O(I,3))
               MAGZ=AXIS(I,1)*(Z(I,1)-O(I,1))+AXIS(I,2)*(Z(I,2)-O(I,2))+
     >              AXIS(I,3)*(Z(I,3)-O(I,3))
               IF(MAGY.EQ.0.D0.AND.MAGZ.EQ.0.D0)THEN
                  OFTX(I)=0.D0
               ELSE
                  OFTX(I)=DATAN2(MAGZ,MAGY)-PI/2.D0
               END IF
               PHI=-OFTX(I)
               OFTX(I)=OFTX(I)*180.D0/PI
               TUX(1)=(X(I,1)-O(I,1))/MAGOX(I)
               TUX(2)=(X(I,2)-O(I,2))/MAGOX(I)
               TUX(3)=(X(I,3)-O(I,3))/MAGOX(I)
               CALL ROTATE(TUX,PHI,RJ)
               MAGZ=RJ(1)*(Z(I,1)-O(I,1))+RJ(2)*(Z(I,2)-O(I,2))+
     >              RJ(3)*(Z(I,3)-O(I,3))
               MAGX=RJ(1)*(X(I,1)-O(I,1))+RJ(2)*(X(I,2)-O(I,2))+
```

```
     >              RJ(3)*(X(I,3)-O(I,3))
                 IF(MAGZ.EQ.0.DO.AND.MAGX.EQ.0.DO)THEN
                    STOP 'ERROR IN ONE OF THE MAGNITUDES'
                 ELSE
                    OFTY(I)=DATAN2(MAGX,MAGZ)*180.DO/PI
                 END IF
                 OFTZ(I)=0.
              END IF
180        CONTINUE
* OK, WRITE IT OUT
           RO=SMALL/4.DO
           RI=RO/2.DO
           RS=RO
           RL=1.2DO*RO
           MECH2=MECHN(1:IP)//'.IATR'
           OPEN(9,FILE=MECH2)
           MECH2=MECHN(1:IP)//'.IPOS'
           OPEN(10,FILE=MECH2)
           WRITE(9,185)
           WRITE(10,185)
185        FORMAT('IMP OUTPUT')
           JDUM=' '
           DO 190 I=1,N
              REV=INDEX(JOINT(I),'REV')
              CYL=INDEX(JOINT(I),'CYL')
              PRI=INDEX(JOINT(I),'PRI')
              SPH=INDEX(JOINT(I),'SPH')
              UJO=INDEX(JOINT(I),'UJO')
              IF(REV.GT.0)THEN
                 JDUM(I:I)='R'
              ELSE IF(CYL.GT.0)THEN
                 JDUM(I:I)='C'
              ELSE IF(PRI.GT.0)THEN
                 JDUM(I:I)='P'
              ELSE IF(SPH.GT.0.OR.UJO.GT.0)THEN
                 JDUM(I:I)='S'
              ELSE
                 STOP'ERROR CREATING MECHANISM TEXT'
              END IF
190        CONTINUE
           WRITE(9,195)JDUM
           WRITE(10,195)JDUM
* THE LAST WRITE BEFORE POSITION OUTPUT FOR LOGICAL UNIT 10
195        FORMAT(A80)
           RGB1=1.0
           RGB2=0.
           RGB3=0.
           WRITE(9,196)RGB1,RGB2,RGB3
196        FORMAT(3F12.4)
           ICODE=11
           WRITE(9,197)RO,RI,RL,RS,ICODE
197        FORMAT(4F12.4,I5)
           DO 200 I=1,N
              IF(I.EQ.K)THEN
                 MOBIL='FIXED'
              ELSE
                 MOBIL='MOVING'
              END IF
              IF(LINK(I,1)(1:1).EQ.'U')THEN
```

```
               LINK(I,1)(1:1)='S'
            END IF
            IF(LINK(I,2)(1:1).EQ.'U')THEN
               LINK(I,2)(1:1)='S'
            END IF
            WRITE(9,205)MOBIL,LINK(I,1),LINK(I,2),LC(I,1),LC(I,2)
205         FORMAT(A8,2A2,2I3)
            WRITE(9,206)OFX(I),OFY(I),OFZ(I),OFTZ(I),OFTY(I),OFTX(I)
206         FORMAT(6F12.4)
200      CONTINUE
* THAT TAKES CARE OF THE HEADERS AND THE ATTRIBUTE FILE
      END IF
* WRITE THE POSITION RECORD
      DO 220 I=1,N
         VZ(1)=Z(I,1)-O(I,1)
         VZ(2)=Z(I,2)-O(I,2)
         VZ(3)=Z(I,3)-O(I,3)
         VX(1)=X(I,1)-O(I,1)
         VX(2)=X(I,2)-O(I,2)
         VX(3)=X(I,3)-O(I,3)
* ROTATE THE Z AND X LOCAL COORDINATES ABOUT GLOBAL X SUCH THAT LOCAL
* Z ENDS UP IN THE GLOBAL ZX PLANE WITH A NON-NEGATIVE DOT PRODUCT
* WITH A POSITIVE Z VECTOR.
         IF(VZ(3).EQ.0.D0.AND.VZ(2).EQ.0.D0)THEN
            TX=0.D0
         ELSE
            PHI=DATAN2(VZ(3),VZ(2))-PI/2.D0
            TX=PHI*180.D0/PI
            PHI=-PHI
            CALL ROTATE(UX,PHI,VZ)
            CALL ROTATE(UX,PHI,VX)
         END IF
         IF(VZ(1).EQ.0.D0.AND.VZ(3).EQ.0.D0)THEN
            TY=0.D0
         ELSE
            PHI=DATAN2(VZ(1),VZ(3))
            TY=PHI*180.D0/PI
            PHI=-PHI
            CALL ROTATE(UY,PHI,VZ)
            CALL ROTATE(UY,PHI,VX)
         END IF
         IF(VX(1).EQ.0.D0.AND.VX(2).EQ.0.D0)THEN
            STOP'THE X COMPONENTS ARE ZERO FOR ROTATION'
         ELSE
            PHI=DATAN2(VX(2),VX(1))
            TZ=PHI*180.D0/PI
         END IF
         WRITE(10,225)O(I,1),O(I,2),O(I,3),TZ,TY,TX
225      FORMAT(6F12.4)
220   CONTINUE
* READ THE NEXT TWO RECORDS TO FIND OUT IF THERE IS ANOTHER POSITION
      READ(8,*)
      READ(8,20)JDUM
      IF(INDEX(JDUM,'POSITION').GT.0)THEN
         READ(8,*)
         READ(8,*)
         READ(8,*)
         READ(8,*)
         READ(8,*)
```

```
         READ(8,*)
         GO TO 165
       END IF
       CLOSE(8)
       CLOSE(9)
       CLOSE(10)
       STOP
       END
C    SUBROUTINE ROTATE
C    COMPUTES ROTATION MATRIX ELEMENTS IN TERMS OF ANGLE PHI (RADIANS)
C    ABOUT U AND ROTATES VECTOR RJ TO RJ=(RM)*RJ
C    RJ CAN BE A VECTOR OF ANY MAGNITUDE, BUT U MUST BE A UNIT VECTOR
       SUBROUTINE ROTATE(U,PHI,RJ)
       IMPLICIT REAL*8(A-H,O-Z)
       DIMENSION RM(3,3),U(3),RJ(3),TJ(3)
       C=DCOS(PHI)
       S=DSIN(PHI)
       V=1.-C
       RM(1,1)=U(1)*U(1)*V+C
       RM(1,2)=U(1)*U(2)*V-U(3)*S
       RM(1,3)=U(1)*U(3)*V+U(2)*S
       RM(2,1)=U(1)*U(2)*V+U(3)*S
       RM(2,2)=U(2)*U(2)*V+C
       RM(2,3)=U(2)*U(3)*V-U(1)*S
       RM(3,1)=U(1)*U(3)*V-U(2)*S
       RM(3,2)=U(2)*U(3)*V+U(1)*S
       RM(3,3)=U(3)*U(3)*V+C
       DO 5 I=1,3
     5  TJ(I)=RJ(I)
       DO 10 I=1,3
    10 RJ(I)=RM(I,1)*TJ(1)+RM(I,2)*TJ(2)+RM(I,3)*TJ(3)
       RETURN
       END
```

Appendix C.
SLIDE Program Listing

```
      PROGRAM SLIDE

C DECLARE THE VARIABLES FOR COLLECTING THE DATA FROM THE ATTRIBUTE AND
C POSITION FILES.  ALLOW FOR 20 LINKS AND 360 POSITIONS.
* THE FOLLOWING VARIABLES STORE DATA FROM THE ATTRIBUTE AND POSITION
*FILES.
**********************************************************************
**       OF*(N) - OFFSET OF A JOINT OTHER THAN THE ONE SET AT THE LOCAL
**                LINK(N) ORIGIN. {OFX(20),OFY(20),OFZ(20)}
**       OFT*(N) - ROTATIONAL OFFSET OF A JOINT OTHER THAN THE ONE AT
**                 THE LOCAL LINK(N) ORIGIN {OFTZ(20),OFTY(20),OFTX(20)}
**       POS*(N,M) - POSITION(M) OF LINK(N) IN ABSOLUTE COORDINATES
**                 {POSX(20,360),POSY(20,360),POSZ(20,360)}
**       T*(N,M) - ROTATION(M) OF LINK(N) IN ABSOLUTE COORDINATES. NOT
**                 CUMULATIVE. {TZ(20,360),TY(20,360),TX(20,360)}
**       CON(N,L) - GIVES JOINT CONNECTIVITY OF ENDS 1 AND 2 OF LINK(N)
**                  FOR EXAMPLE: CON(3,1)= 2 MEANS THE FIRST END OF LINK
**                  3 IS ATTACHED TO JOINT 2
**       RO,RI,RL,RS - OUTSIDE JOINT RADIUS, INSIDE JOINT RADIUS, HALF
```

```
**                    JOINT LENGTH, AND SPHERIC JOINT RADIUS
**       JCON(N,L) - GIVES JOINT TYPE AS INTERNAL OR EXTERNAL FOR ENDS
**                   1 AND 2, AND THE JOINT TYPE I.E. REVOLUTE, SPHERIC,
**                   ETC.
**       MOVM - IDENTIFIES A LINK AS MOVING OR FIXED
************************************************************************

      COMMON /LUN/ NIN,NOUT
      REAL *8 OFX(20),OFY(20),OFZ(20),OFTZ(20),OFTY(20),OFTX(20)
      REAL *8 POSX(20,360),POSY(20,360),POSZ(20,360),
     >TZ(20,360),TY(20,360),TX(20,360),RO,RI,RL,RS
      INTEGER CON(20,2)
      CHARACTER*2 JCON(20,2)
      CHARACTER*8 MOVM
C DECLARE THE NECESSARY VARIABLES FOR INTERFERENCE CHECKING.
************************************************************************
**    VECT(N,M,L) -STORE THE LINK(N) DESCRIPTION AS AN M VECTOR SET
**                 WITH L=X,Y,Z AS THE DESCRIPTION OF AN INDIVIDUAL
**                 VECTOR. THIS DOES NOT INCLUDE THE EXTERNAL JOINT
**                 NORMAL DESCRIPTIONS.
**    TVECT(N,M,L) -STORE THE RESHAPED VALUES OF VECT(N,M,L)
**    JSV(N,A,B)- STORE THE POSITION VECTOR FOR JOINT(A) OF THE LINK(N)
**                 AS B=X,Y,Z  RELATIVE TO THE JOINT'S LOCAL SYSTEM
**    JNT(N,A,B)- STORE THE AXIS DIRECTIONAL DESCRIPTION FOR JOINT(A) OF
**                 LINK(N) AS B=X,Y,Z. RELATIVE TO THE JOINT'S LOCAL
**                 SYSTEM
**    ADU- POSITION OF A SPHERIC CENTER DOTTED ONTO A UNIT REVOLUTE AXIS
**          VECTOR (USED TO DETERMINE DISTANCE)
**    IVCNT(N)-CURRENT VECTOR COUNT FOR LINK N (CHANGES AS RESHAPING
**              DICTATES)
**    NIN,NOUT- LUNS FOR INTERACTIVE INPUT AND OUTPUT
**    IPIN,ILIN- LUNS FOR DISC INPUT OF LINKS AND POSITIONS (FILE NAMES
**                ARE DEFINED IN SLIPE EXEC WHICH STARTS THE PROGRAM)
**    ICOUNT- COUNTER USED TO DETERMINE THE NUMBER OF LINKS
**    LNK- NUMBER OF LINKS (ICOUNT/2)
**    RLAM1,2,3- SCALAR VARIABLES USED IN THE VECTOR LOOP EQUATIONS
**    SOL(N,B)- SOLUTION VECTOR FOR LINK N WITH DIRECTION B=X,Y,Z (THIS
**                VECTOR IS USUALLY THE RESULT OF A CROSS PRODUCT
**                OPERATION INVOLVING JOINT AXIS VECTORS)
**    RSHP(N)- LOGICAL VARIABLE WHICH TAGS A LINK AS RESHAPABLE OR NOT
**    RJ(B)- ROTATED ORIENTATION OF A VECTOR AFTER BEING OPERATED ON BY
**            SUBROUTINE ROTATE
**    U(B)-  ROTATION AXIS USED BY SUBROUTINE ROTATE
**    PHI- ROTATION ANGLE USED BY SUBROUTINE ROTATE (RADIANS)
**    PI- 3.14159...
**    IPOS- USED TO DETERMINE THE NUMBER OF POSITIONS
**    IP- NUMBER OF POSITIONS FOR A LINK (IPOS/LNK)
**    FOUND- USED TO FIND AND INDEX THE COUNT OF SPHERIC JOINTS
**    NSPHER- USED TO KEEP TRACK OF THE NUMBER OF SPHERIC JOINTS
**    INDSPH(N,E)- USED TO INDEX THE SPHERIC JOINTS. INDSPH(4,2)=3 MEANS
**                 THAT THE EXTERIOR PORTION OF SPHERIC JOINT 4 IS
**                 ATTACHED TO LINK 3 (IF LINK(I) IS AN SS LINK, THEN
**                 IT IS STORED IN INDSPH(N,2) REGARDLESS IF IT IS
**                 INTERIOR OR EXTERIOR)
**    SPHTRK(N)- USED TO KEEP TRACK OF SPHERIC JOINTS AS ANY PART OF THE
**                JOINT IS FOUND. SPHTRK(2)=4 MEANS THAT THE SECOND
**                SPHERIC JOINT FOUND WAS LINKAGE JOINT NUMBER 4.
**    UNIT(I,J,K)- RELATIVE UNIT VECTOR COMPONENT K OF JOINT I AT
**                 POSITION J. TAKEN RELATIVE TO THE VECTOR FEEDING
```

```
**                INTO THE BALL. (SUBROUTINE POSCUP)
**    SPAN(N)- USED TO HOLD THE VECTOR OF MAXIMUM SEPARATION IN THE
**             GAUSSIAN POINT ANALYSIS FOR SPHERIC JOINTS (SUBROUTINE
**             POSCUP)
**    BISECT(N)- USED TO HOLD THE BISECTING VECTOR FOR THE TWO VECTORS
**              USED TO DESCRIBE SPAN(N). (SUBROUTINE POSCUP)
**    NORM(N)- USED TO PERFORM A LAST DITCH ATTEMPT IN ORIENTING THE
**             SPHERIC CUPS.  THE CROSS PRODUCT OF SPAN AND BISECT.
**             (SUBROUTINE POSCUP)
**    JOINT(N,I)- STORES THE LINK CONNECTIVITY FOR JOINT N.
**                JOINT(2,1)=3 MEANS THE INTERNAL (1) PORTION OF
**                JOINT 2 IS ATTACHED TO LINK 3
**    JTRK(N)- USED TO KEEP TRACK OF ALL JOINT TYPES (I.E. R,P,S,C)
**    JCOUNT- NUMBER OF JOINTS
**    JLEN(N,E)- STORES JOINT LENGTH [JLEN(4,2) LINK4, SECOND JOINT]
**    JPOS(N,E)- STORES JOINT POSITION ON LOCAL Z AXIS
**    GLEN(N)- STORES JOINT LENGTH OF GROUND ELEMENT OF LINK N (IF NEC)
**    GPOS(N)- STORES GROUND JOINT POSITION ALONG JOINT'S Z AXIS
**               (IF NECESSARY)
**    GCOUNT- COUNTER FOR GROUND ELEMENTS
**    OFMAX- MAXIMUM OFFSET (LINK LENGTH)
**    OFTM- TEMPORARY VARIABLE USED TO CATCH OFMAX
*************************************************************************
      REAL*8 VECT(20,3,3),TVECT(20,3,3),JSV(20,2,3),JNT(20,2,3),RJ(3)
      REAL*8 RLAM1(20),RLAM2(20),RLAM3(20),SOL(20,3),PHI,PI,U(3)
      REAL*8 JLEN(20,2),JPOS(20,2),GLEN(20),GPOS(20),OFMAX,OFTM
      REAL*8 SHFT1,SHFT2,V(3),RX,RY,RZ,X,Y,Z,UX(3),UY(3),UZ(3),R,T1,T2
      REAL*8 TRX,P(20,3),E1,E2
      INTEGER IVCNT(20),LNK,ICOUNT,NIN,NOUT,ILIN,IPIN,IJMP
      INTEGER INDSPH(20,2),NSPHER,SPHTRK(20),JOINT(20,2),JCOUNT,GCOUNT
      INTEGER MODTY,MODCNT,LNKCNT,M1,M2,M3,N,ITHEMC
      CHARACTER *1 JTRK(20),ANSWER
      CHARACTER *16 IDFILE
      CHARACTER *80 MECHNM,MODNM
      CHARACTER *20 THEMEC,THEMC2
      LOGICAL RSHP(20),FOUND
      DATA JOINT/40*0.0D0/
      PI=DACOS(-1.D0)
      NIN=5
      NOUT=6
      ILIN=8
      IPIN=10
      WRITE(NOUT,*)'WHAT MECHANSIM?'
      READ(NIN,2222)THEMEC
2222  FORMAT(A20)
      ITHEMC=INDEX(THEMEC,' ')-1
      IF(ITHEMC.LE.0)THEN
         WRITE(NOUT,*)'ERROR IN THE MECHANISM NAME'
         STOP
      END IF
      WRITE(NOUT,*)'DO YOU WANT TO INVOKE COLLISION AVOIDANCE? (Y/N)'
      READ(NIN,2)ANSWER
2     FORMAT(A1)
      IF(ANSWER.EQ.'N')THEN
         IJMP=1
      END IF
      THEMC2=THEMEC(1:ITHEMC)//'.IATR'
      OPEN(8,FILE=THEMC2)
      THEMC2=THEMEC(1:ITHEMC)//'.IPOS'
```

```
      OPEN(10,FILE=THEMC2)
C  FIND OUT HOW MANY LINKS THERE ARE.
      ICOUNT=0
* READ PAST THE FIRST FOUR HEADER LINES THEN START COUNT LINK LINES
      READ(ILIN,*)
      READ(ILIN,*)
      READ(ILIN,*)
      READ(ILIN,*)
4     CONTINUE
      READ(ILIN,*,END=5)
      ICOUNT=ICOUNT+1
      GO TO 4
5     CONTINUE
      REWIND(ILIN)
C  FIND OUT HOW MANY POSITIONS THERE ARE.
      READ(IPIN,*)
      READ(IPIN,*)
      IPOS=0
6     CONTINUE
      READ(IPIN,*,END=7)
      IPOS=IPOS+1
      GO TO 6
7     CONTINUE
      REWIND(IPIN)
      LNK=ICOUNT/2
      IP=IPOS/LNK

C GET THE DATA FROM THESE FILES

      READ(ILIN,101)IDFILE
      READ(ILIN,102)MECHNM
101   FORMAT(A16)
102   FORMAT(A80)
      READ(ILIN,*)
      READ(ILIN,100)RO,RI,RL,RS,ICODE
      NSPHER=0
      JCOUNT=0
      OFMAX=0.DO
      DO 19 I=1,LNK
         READ(ILIN,110)MOVM,JCON(I,1),JCON(I,2),CON(I,1),CON(I,2)
         READ(ILIN,120)OFX(I),OFY(I),OFZ(I),OFTZ(I),OFTY(I),OFTX(I)
         OFTM=DSQRT(OFX(I)**2+OFY(I)**2+OFZ(I)**2)
         IF(OFTM.GT.OFMAX) OFMAX=OFTM
*
* CATALOG THE JOINTS, ZERO IS A GROUND ATTACHMENT
*    FIND WHAT JOINT AND TYPE END 1 OF LINK I IS CONNECTED TO.
         IF(JCON(I,1)(2:2).EQ.'I')THEN
            JOINT(IABS(CON(I,1)),1)=I
            JTRK(IABS(CON(I,1)))=JCON(I,1)(1:1)
            IF(IABS(CON(I,1)).GT.JCOUNT)THEN
               JCOUNT=IABS(CON(I,1))
            END IF
         ELSE IF(JCON(I,1)(2:2).EQ.'E')THEN
            JOINT(IABS(CON(I,1)),2)=I
            JTRK(IABS(CON(I,1)))=JCON(I,1)(1:1)
            IF(IABS(CON(I,1)).GT.JCOUNT)THEN
               JCOUNT=IABS(CON(I,1))
            END IF
         END IF
```

```
*     FIND WHAT JOINT AND TYPE END 2 OF LINK I IS CONNECTED TO.
          IF(JCON(I,2)(2:2).EQ.'I')THEN
             JOINT(IABS(CON(I,2)),1)=I
             JTRK(IABS(CON(I,2)))=JCON(I,2)(1:1)
             IF(IABS(CON(I,2)).GT.JCOUNT)THEN
                JCOUNT=IABS(CON(I,2))
             END IF
          ELSE IF(JCON(I,2)(2:2).EQ.'E')THEN
             JOINT(IABS(CON(I,2)),2)=I
             JTRK(IABS(CON(I,2)))=JCON(I,2)(1:1)
             IF(IABS(CON(I,2)).GT.JCOUNT)THEN
                JCOUNT=IABS(CON(I,2))
             END IF
          END IF

* CONVERT THE DEGREE DATA TO RADIANS
          OFTZ(I)=OFTZ(I)*PI/180.
          OFTY(I)=OFTY(I)*PI/180.
          OFTX(I)=OFTX(I)*PI/180.
*
*         COUNT AND INDEX ALL THE SPHERIC JOINTS
*
          FOUND=.FALSE.
          IF(JCON(I,1).EQ.'SE')THEN
             IF(NSPHER.EQ.0)THEN
                NSPHER=1
                INDSPH(1,2)=I
                SPHTRK(1)=CON(I,1)
             ELSE
                DO 11 J=1,NSPHER
                   IF(SPHTRK(J).EQ.CON(I,1))THEN
                      FOUND=.TRUE.
                      GO TO 12
                   END IF
11                 CONTINUE
12                 CONTINUE
                   IF(.NOT.FOUND)THEN
                      NSPHER=NSPHER+1
                      INDSPH(NSPHER,2)=I
                      SPHTRK(NSPHER)=CON(I,1)
                   ELSE
                      INDSPH(J,2)=I
                   END IF
             END IF
          ELSE IF(JCON(I,1).EQ.'SI')THEN
             IF(NSPHER.EQ.0)THEN
                NSPHER=1
                INDSPH(1,1)=I
                SPHTRK(1)=CON(I,1)
             ELSE
                DO 13 J=1,NSPHER
                   IF(SPHTRK(J).EQ.CON(I,1))THEN
                      FOUND=.TRUE.
                      GO TO 14
                   END IF
13                 CONTINUE
14                 CONTINUE
                   IF(.NOT.FOUND)THEN
                      NSPHER=NSPHER+1
```

```
                  INDSPH(NSPHER,1)=I
                  SPHTRK(NSPHER)=CON(I,1)
               ELSE
                  INDSPH(J,1)=I
               END IF
            END IF
         END IF
         FOUND=.FALSE.
         IF(JCON(I,2).EQ.'SE')THEN
            IF(NSPHER.EQ.0)THEN
               NSPHER=1
               INDSPH(1,2)=I
               SPHTRK(1)=CON(I,2)
            ELSE
               DO 15 J=1,NSPHER
                  IF(SPHTRK(J).EQ.CON(I,2))THEN
                     FOUND=.TRUE.
                     GO TO 16
                  END IF
15                CONTINUE
16                CONTINUE
                  IF(.NOT.FOUND)THEN
                     NSPHER=NSPHER+1
                     INDSPH(NSPHER,2)=I
                     SPHTRK(NSPHER)=CON(I,2)
                  ELSE
                     INDSPH(J,2)=I
                  END IF
            END IF
         ELSE IF(JCON(I,2).EQ.'SI')THEN
            IF(NSPHER.EQ.0)THEN
               NSPHER=1
               INDSPH(1,1)=I
               SPHTRK(1)=CON(I,2)
            ELSE
               DO 17 J=1,NSPHER
                  IF(SPHTRK(J).EQ.CON(I,2))THEN
                     FOUND=.TRUE.
                     GO TO 18
                  END IF
17                CONTINUE
18                CONTINUE
                  IF(.NOT.FOUND)THEN
                     NSPHER=NSPHER+1
                     INDSPH(NSPHER,1)=I
                     SPHTRK(NSPHER)=CON(I,2)
                  ELSE
                     INDSPH(J,1)=I
                  END IF
            END IF
         END IF
         JSV(I,1,1)=0.
         JSV(I,1,2)=0.
         JSV(I,1,3)=0.
         JSV(I,2,1)=OFX(I)
         JSV(I,2,2)=OFY(I)
         JSV(I,2,3)=OFZ(I)
         JNT(I,1,1)=0.
         JNT(I,1,2)=0.
```

```
         JNT(I,1,3)=1.
C NOTE THAT NO ROTATION IS REQUIRED ABOUT THE Z-AXIS SINCE ROTATION AXIS
C IS ORIGINALLY ORIENTED ALONG THE Z-AXIS. ORDER IS Z,Y,X.
         U(1)=0.
         U(2)=1.
         U(3)=0.
         RJ(1)=0.
         RJ(2)=0.
         RJ(3)=1.
         PHI=OFTY(I)
         CALL ROTATE(U,PHI,RJ)
         U(1)=1.
         U(2)=0.
         U(3)=0.
         PHI=OFTX(I)
         CALL ROTATE(U,PHI,RJ)
         JNT(I,2,1)=RJ(1)
         JNT(I,2,2)=RJ(2)
         JNT(I,2,3)=RJ(3)
C ALL THE JOINT AXES HAVE BEEN COMPUTED RELATIVE TO THEIR LOCAL ORIGINS
19     CONTINUE
*       FIND OUT IF ANY OF THE INTERNAL SPHERIC JOINTS ARE CONNECTED TO
*       AN SS LINK IF SO, EXCHANGE THE VALUES OF INDSPH(I,1) AND
*       INDSPH(I,2) SO THAT ORIENTING THE JOINT PIECES WILL HAVE A
*       GREATER CHANCE FOR SUCCESS.  LINK{INDSHP(I,1)} IS USED AS THE
*       REFERENCE IN ORIENTING THE PIECES.
       IF(NSPHER.GT.0)THEN
          DO 20 I=1,NSPHER
             IQUIZ=INDSPH(I,1)
             T1=INDEX(JCON(IQUIZ,1),'S')
             T2=INDEX(JCON(IQUIZ,2),'S')
             IF(T1.GT.0.AND.T2.GT.0)THEN
                IQUIZ=INDSPH(I,2)
                INDSPH(I,2)=INDSPH(I,1)
                INDSPH(I,1)=IQUIZ
             END IF
20        CONTINUE
       END IF
       READ(IPIN,*)
       READ(IPIN,*)
       DO 21 I=1,IP
          DO 30 J=1,LNK
             READ(IPIN,130)POSX(J,I),POSY(J,I),POSZ(J,I),TZ(J,I),
     >       TY(J,I),TX(J,I)
             TZ(J,I)=TZ(J,I)*PI/180.
             TY(J,I)=TY(J,I)*PI/180.
             TX(J,I)=TX(J,I)*PI/180.
30        CONTINUE
21     CONTINUE
100    FORMAT(4F12.4,I5)
110    FORMAT(A8,2A2,2I3)
120    FORMAT(6F12.4)
130    FORMAT(6F12.4)
C ALL THE DATA HAS BEEN READ IN.
* SET INITIAL JLEN BEFORE CALLING SIZE. AND START COUNTING GROUND
* ELEMENTS.
       DO 140 I=1,JCOUNT
          IF(JTRK(I).EQ.'R'.OR.JTRK(I).EQ.'P'.OR.JTRK(I).EQ.'C')THEN
* TEST INTERNAL PART TO SEE IF IT IS ATTACHED TO END1 OR END2 OF A LINK
```

```
* IF NOT, THE INTERNAL PART IS GROUND.
            IF(JOINT(I,1).NE.0)THEN
               IF(CON(JOINT(I,1),1).EQ.I)THEN
                  JLEN(JOINT(I,1),1)=1.4D0*2.D0*RL
                  JPOS(JOINT(I,1),1)=-(RL+1.1D0*RI)
               ELSE IF(CON(JOINT(I,1),2).EQ.I)THEN
                  JLEN(JOINT(I,1),2)=1.4D0*2.D0*RL
                  JPOS(JOINT(I,1),2)=-(RL+1.1D0*RI)
               ELSE
                  WRITE(6,*)'BOTCHED 1'
               END IF
            ELSE
               GCOUNT=GCOUNT+1
               GLEN(I)=1.4D0*2.D0*RL
               GPOS(I)=-(RL+1.1D0*RI)
            END IF
* TEST EXTERNAL PART IN SAME MANNER AS INTERNAL.
            IF(JOINT(I,2).NE.0)THEN
               IF(CON(JOINT(I,2),1).EQ.I)THEN
                  JLEN(JOINT(I,2),1)=2.D0*RL
                  JPOS(JOINT(I,2),1)=-RL
               ELSE IF(CON(JOINT(I,2),2).EQ.I)THEN
                  JLEN(JOINT(I,2),2)=2.D0*RL
                  JPOS(JOINT(I,2),2)=-RL
               ELSE
                  WRITE(6,*)'BOTCHED 2'
               END IF
            ELSE
               GCOUNT=GCOUNT+1
               GLEN(I)=2.D0*RL
               GPOS(I)=-RL
            END IF
* TEST THE INTERNAL SPHERIC
         ELSE IF(JTRK(I).EQ.'S')THEN
            IF(JOINT(I,1).NE.0)THEN
               IF(CON(JOINT(I,1),1).EQ.I)THEN
                  JLEN(JOINT(I,1),1)=0.D0
               ELSE IF(CON(JOINT(I,1),2).EQ.I)THEN
                  JLEN(JOINT(I,1),2)=0.D0
               ELSE
                  WRITE(6,*)'BOTCHED 3'
               END IF
            ELSE
               GCOUNT=GCOUNT+1
               GLEN(I)=0.D0
               GPOS(I)=0.D0
            END IF
* TEST EXTERNAL PART IN SAME MANNER AS INTERNAL.
            IF(JOINT(I,2).NE.0)THEN
               IF(CON(JOINT(I,2),1).EQ.I)THEN
                  JLEN(JOINT(I,2),1)=0.D0
               ELSE IF(CON(JOINT(I,2),2).EQ.I)THEN
                  JLEN(JOINT(I,2),2)=0.D0
               ELSE
                  WRITE(6,*)'BOTCHED 4'
               END IF
            ELSE
               GCOUNT=GCOUNT+1
               GLEN(I)=0.D0
```

```
              GPOS(I)=0.DO
            END IF
          END IF
140   CONTINUE
* SIZE THE INTERNAL JOINT ELEMENTS FOR PRISMATICS AND CYLINDRICS
      CALL        SIZE(JSV,JLEN,JPOS,GLEN,GPOS,JNT,POSX,POSY,POSZ,TZ,
     >                 TY,TX,JOINT,JTRK,JCOUNT,OFX,OFY,OFZ,
     >                 OFTZ,OFTY,OFTX,IP,CON)


      CALL        MOVJNT(JSV,JLEN,JPOS,GLEN,GPOS,JNT,POSX,POSY,
     >           POSZ,TZ,TY,TX,JOINT,JTRK,JCOUNT,OFX,OFY,OFZ,
     >               OFTZ,OFTY,OFTX,IP,RO,RI,RL,RS,NIN,NOUT,OFMAX,CON)
* THE JOINTS HAVE BEEN MOVED SO THAT THEY DO NOT INTERFERE AT ANY
* POSITION

      CALL        NUBS(JNT,OFX,OFY,OFZ,JPOS,OFTZ,OFTY,OFTX,VECT,TVECT
     >                 ,LNK,RO,RS,JTRK,CON)

*  WITH ALL OF THE INITIAL DESCRIPTION,  NOW IS THE TIME TO POSITION ANY
*  SPHERIC CUPS.  THEY ARE TO BE ALIGNED SUCH THAT THEIR EXTREMES OF
*  ANGULAR MOVEMENT ARE CENTERED ABOUT A VECTOR EMANATING FROM THE BALL
*  (RI) LINK.
*      IF(IJMP.NE.1) THEN
      CALL        POSCUP(VECT,IVCNT,OFTX,OFTY,OFTZ,TX,TY,TZ,NSPHER,INDSPH
     >                  ,SPHTRK,IP,CON,RS)
*      END IF

* ALL THE PRELIMINARY STUFF IS DONE.  NOW, RESHAPE THE LINKS.
      UX(1)=1.DO
      UX(2)=0.DO
      UX(3)=0.DO
      UY(1)=0.DO
      UY(2)=1.DO
      UY(3)=0.DO
      UZ(1)=0.DO
      UZ(2)=0.DO
      UZ(3)=1.DO
      DO 150 I=1,LNK
         IF(JCON(I,1)(2:2).EQ.'I'.OR.JCON(I,1)(1:1).EQ.'S')THEN
            SHFT1=JPOS(I,1)
         ELSE
            SHFT1=JPOS(I,1)+RL
         END IF
         IF(JCON(I,2)(2:2).EQ.'I'.OR.JCON(I,2)(1:1).EQ.'S')THEN
            SHFT2=JPOS(I,2)
         ELSE
            SHFT2=JPOS(I,2)+RL
         END IF
         RJ(1)=0.DO
         RJ(2)=0.DO
         RJ(3)=SHFT2
         PHI=OFTZ(I)
         CALL ROTATE(UZ,PHI,RJ)
         PHI=OFTY(I)
         CALL ROTATE(UY,PHI,RJ)
         PHI=OFTX(I)
         CALL ROTATE(UX,PHI,RJ)
         VECT(I,2,1)=VECT(I,3,1)+OFX(I)-VECT(I,1,1)+RJ(1)
```

```
         VECT(I,2,2)=VECT(I,3,2)+OFY(I)-VECT(I,1,2)+RJ(2)
         VECT(I,2,3)=VECT(I,3,3)+OFZ(I)-VECT(I,1,3)+RJ(3)-SHFT1
         DO 160 J=1,3
            TVECT(I,J,1)=VECT(I,J,1)
            TVECT(I,J,2)=VECT(I,J,2)
            TVECT(I,J,3)=VECT(I,J,3)
160      CONTINUE
150   CONTINUE
      IF(IJMP.NE.1) THEN
      CALL       MOVLNK(JSV,JLEN,JPOS,GLEN,GPOS,JNT,POSX,POSY,
     >                  POSZ,TZ,TY,TX,JOINT,JTRK,JCOUNT,OFX,OFY,OFZ,
     >                  OFTZ,OFTY,OFTX,IP,RO,RI,RL,RS,NIN,NOUT,OFMAX,
     >                  VECT,TVECT,LNK,JCON,CON)
      END IF
* THE LINKS HAVE ALL BEEN RESHAPED (IF THAT WAS POSSIBLE)
* TIME TO CREATE THE FILES FOR D. MONTGOMERY'S ANIMATION

*  WRITE OUT THE POSITION FILE
      THEMC2=THEMEC(1:ITHEMC)//'.POS'
      OPEN(20,FILE=THEMC2)
      WRITE(20,200)MECHNM,IDFILE
200   FORMAT(A80,A16)
      WRITE(20,201)IP
201   FORMAT(I3,' ANIMATION STEPS')
      DO 205 J=1,IP
         DO 210 I=1,LNK
            LCOUNT=100+I
            RZ=TZ(I,J)*180.D0/DACOS(-1.D0)
            RY=TY(I,J)*180.D0/DACOS(-1.D0)
            RX=TX(I,J)*180.D0/DACOS(-1.D0)
            WRITE(20,215)J,LCOUNT,RZ,RY,RX,
     >                              POSX(I,J),POSY(I,J),POSZ(I,J)
215         FORMAT(2I5,6F12.4)
210      CONTINUE
205   CONTINUE
      CLOSE (20)
*  WRITE OUT THE MODEL FILE
      THEMC2=THEMEC(1:ITHEMC)//'.MOD'
      OPEN (30,FILE=THEMC2)
      WRITE(30,200)MECHNM,IDFILE
      MODCNT=0
      LNKCNT=0
      DO 300 I=1,LNK
         DO 310 J=1,2
            MODCNT=MODCNT+1
            IF(J.EQ.1)THEN
               M1=MODCNT
            ELSE
               M2=MODCNT
            END IF
            IF      (JCON(I,J).EQ.'RE')THEN
               MODTY=1
               MODNM='EXTERNAL REVOLUTE JOINT'
            ELSE IF(JCON(I,J).EQ.'RI')THEN
               MODTY=2
               MODNM='INTERNAL REVOLUTE JOINT'
            ELSE IF(JCON(I,J).EQ.'PE')THEN
               MODTY=5
               MODNM='EXTERNAL PRISMATIC JOINT'
```

```
            ELSE IF(JCON(I,J).EQ.'PI')THEN
               MODTY=6
               MODNM='INTERNAL PRISMATIC JOINT'
            ELSE IF(JCON(I,J).EQ.'CE')THEN
               MODTY=7
               MODNM='EXTERNAL CYLINDRIC JOINT'
            ELSE IF(JCON(I,J).EQ.'CI')THEN
               MODTY=8
               MODNM='INTERNAL CYLINDRIC JOINT'
            ELSE IF(JCON(I,J).EQ.'SE')THEN
               MODTY=9
               MODNM='EXTERNAL SPHERIC JOINT'
            ELSE IF(JCON(I,J).EQ.'SI')THEN
               MODTY=10
               MODNM='INTERNAL SPHERIC JOINT'
            END IF
            IF(J.EQ.1)THEN
*            THIS IS NOT TOTALLY TRUE.  RZ MUST BE NOTED FOR PRISMATICS
               RZ=0.D0
               RY=0.D0
               RX=0.D0
               X=0.D0
               Y=0.D0
               Z=JPOS(I,1)
            ELSE
               RZ=OFTZ(I)*180.D0/DACOS(-1.D0)
               RY=OFTY(I)*180.D0/DACOS(-1.D0)
               RX=OFTX(I)*180.D0/DACOS(-1.D0)
               X=OFX(I)
               Y=OFY(I)
               Z=OFZ(I)
            END IF
            IF     (MODTY.EQ.1.OR.MODTY.EQ.5.OR.MODTY.EQ.7)THEN
               Z=Z+RL
               WRITE(30,315)MODTY,MODCNT,MODNM,RO,RI,JLEN(I,J),
     >                     RZ,RY,RX,X,Y,Z
            ELSE IF(MODTY.EQ.2.OR.MODTY.EQ.6.OR.MODTY.EQ.8)THEN
               E1=JPOS(I,J)
               E2=E1+JLEN(I,J)
               WRITE(30,316)MODTY,MODCNT,MODNM,RI,E1,E2,RZ,RY,RX,X,Y,Z
            ELSE IF(MODTY.EQ.9)THEN
*------------THIS IS SUBTLE, THE NUB DETERMINES ROTATION -- TOTALLY!
              IF(J.EQ.1)THEN
                  U(1)=1.D0
                  U(2)=0.D0
                  U(3)=0.D0
                  V(1)=-TVECT(I,1,1)
                  V(2)=-TVECT(I,1,2)
                  V(3)=-TVECT(I,1,3)
                  IF(V(2).EQ.0.D0.AND.V(3).EQ.0.D0)THEN
                     RX=0.D0
                  ELSE
                     RX=DATAN2(V(3),V(2))-DACOS(-1.D0)/2.D0
                     TRX=-RX
                     RX=RX*180.D0/PI
                  END IF
                  CALL ROTATE(U,TRX,V)
                  IF(V(1).EQ.0.D0.AND.V(2).EQ.0.D0)THEN
                     STOP 'DATAN2 ERROR 2'
```

```
                  END IF
                  RY=DATAN2(V(1),V(3))*180.D0/PI
               ELSE
                  U(1)=1.D0
                  U(2)=0.D0
                  U(3)=0.D0
                  V(1)=-TVECT(I,3,1)
                  V(2)=-TVECT(I,3,2)
                  V(3)=-TVECT(I,3,3)
                  IF(V(2).EQ.0.D0.AND.V(3).EQ.0.D0)THEN
                     RX=0.D0
                  ELSE
                     RX=DATAN2(V(3),V(2))-DACOS(-1.D0)/2.D0
                     TRX=-RX
                     RX=RX*180.D0/PI
                  END IF
                  CALL ROTATE(U,TRX,V)
                  IF(V(1).EQ.0.D0.AND.V(2).EQ.0.D0)THEN
                     STOP 'DATAN2 ERROR 4'
                  END IF
                  RY=DATAN2(V(1),V(3))*180.D0/PI
               END IF
               R=0.9D0*RS
               WRITE(30,317)MODTY,MODCNT,MODNM,RS,R,RZ,RY,RX,X,Y,Z
            ELSE IF(MODTY.EQ.10)THEN
               R=0.9D0*RS
               WRITE(30,318)MODTY,MODCNT,MODNM,R,X,Y,Z
            ELSE
               WRITE(6,*)'MAJOR SCREW UP AT THE END'
            END IF
315      FORMAT(2I4,2X,A80/9F12.4)
316      FORMAT(2I4,2X,A80/9F12.4)
317      FORMAT(2I4,2X,A80/8F12.4)
318      FORMAT(2I4,2X,A80/4F12.4)
310      CONTINUE
         MODCNT=MODCNT+1
         MODNM='CONNECTING ELEMENT'
         MODTY=3
         V(1)=0.D0
         V(2)=0.D0
         IF(JCON(I,1)(1:1).EQ.'S')THEN
            V(3)=0.D0
         ELSE
            V(3)=JPOS(I,1)+RL
         END IF
         P(1,1)=V(1)+TVECT(I,1,1)
         P(1,2)=V(2)+TVECT(I,1,2)
         P(1,3)=V(3)+TVECT(I,1,3)
         P(2,1)=P(1,1)+TVECT(I,2,1)
         P(2,2)=P(1,2)+TVECT(I,2,2)
         P(2,3)=P(1,3)+TVECT(I,2,3)
         P(3,1)=P(2,1)-TVECT(I,3,1)
         P(3,2)=P(2,2)-TVECT(I,3,2)
         P(3,3)=P(2,3)-TVECT(I,3,3)
         N=4
         WRITE(30,319)MODTY,MODCNT,MODNM,RI,N,V(1),V(2),V(3),
     >   P(1,1),P(1,2),P(1,3),P(2,1),P(2,2),P(2,3),P(3,1),P(3,2),P(3,3)
319      FORMAT(2I4,2X,A80/F12.4,I4/3F12.4/3F12.4/3F12.4/3F12.4)
         M3=MODCNT
```

```
         MODCNT=MODCNT+1
         LNKCNT=LNKCNT+1
         LCOUNT=I+100
         MODTY=4
         MODNM=JCON(I,1)//JCON(I,2)//' LINK '
         N=3
         WRITE(30,320)MODTY,LCOUNT,MODNM,LNKCNT,N,M1,M2,M3
320      FORMAT(2I4,2X,A12,I4/4I4)
300   CONTINUE
      CLOSE (30)
      STOP
      END

      SUBROUTINE     AVCYL(V,OPT,SV,EV,B,TVECT,LNK1,JPOSI,JNT,GOOD)
*  THIS ROUTINE IS FOR AVOIDANCE OF CYLINDRIC INTRUDERS
      REAL *8 TVECT(20,3,3),JNT(20,2,3),RJ(3)
      REAL *8 SV(2,3),EV(2,3),DOT,UX(3),UY(3),UZ(3),B,PHI,U(3)
      REAL *8 A(3),AP(3),C(3),D(3),MAGA,MAGD
      REAL *8 AXD(3),AXDMAG,AXDXA(3),BETA,CDOTA,CHECK(3),DAXIS(3)
      REAL *8 DCHECK,DOT1,DOT2,DOTD,DOTDU,DOTE,DOTMAX,DOTMIN,DOTP,DOTS
      REAL *8 DOTVCE,DOTVCI,DOTX,DTHET,DTOT,DTST(3),MAG2,MAGC,MAGVC
      REAL *8 NEWB,NEWTA,NEWVC,PERP(3),RATIO,ROTN,ROTP,SA(3),SD(3),T1(3)
      REAL *8 T1MAG,TA(3),TAP(3),TCROSS(3),TE(3),TEMPD(3),TEST1,TEST2
      REAL *8 TESTD,TEV(2,3),THET0,THET1,THET2,THETC,VS(3),VE(3)
      REAL *8 TS(3),TSV(2,3),UMAG,VC(3),MAGE,MAGS,CMAGE,CMAGS
      REAL *8 DVS,ANGS,DVE,ANGE,MAGVS,MAGVE,TMAGA
      real *8 uxdt,tmag7,pi
      INTEGER LNK1,JPOSI,OPT(20,3),V,TESPOS
      LOGICAL RSTRT,GOOD,ITSE,ITSS
      ITSE=.FALSE.
      ITSS=.FALSE.
      B=B*1.001D0
      UX(1)=1.D0
      UX(2)=0.D0
      UX(3)=0.D0
      UY(1)=0.D0
      UY(2)=1.D0
      UY(3)=0.D0
      UZ(1)=0.D0
      UZ(2)=0.D0
      UZ(3)=1.D0
      A(1)=EV(1,1)-SV(1,1)
      A(2)=EV(1,2)-SV(1,2)
      A(3)=EV(1,3)-SV(1,3)
      AP(1)=A(1)
      AP(2)=A(2)
      AP(3)=A(3)
      D(1)=EV(2,1)-SV(2,1)
      D(2)=EV(2,2)-SV(2,2)
      D(3)=EV(2,3)-SV(2,3)
      MAGA=DSQRT(A(1)*A(1)+A(2)*A(2)+A(3)*A(3))
      MAGD=DSQRT(D(1)*D(1)+D(2)*D(2)+D(3)*D(3))
* C IS POSITION VECTOR FROM TAIL OF V1 TO SPHERE CENTER
      C(1)=SV(2,1)-SV(1,1)
      C(2)=SV(2,2)-SV(1,2)
      C(3)=SV(2,3)-SV(1,3)
      IF(V.EQ.2)THEN
         TESPOS=2*(JPOSI/2)
         IF(OPT(LNK1,2).EQ.1.OR.(OPT(LNK1,2).EQ.3.AND.TESPOS.NE.
```

```
     >       JPOSI))THEN
*          MINUS SIGN USED TO GET POSITIVE INDEXING ALONG V1 FOR
*          POSITIVE ROTATION
           RJ(1)=-(TVECT(LNK1,2,2)*TVECT(LNK1,1,3)-TVECT(LNK1,2,3)*
     >          TVECT(LNK1,1,2))
           RJ(2)=-(TVECT(LNK1,2,3)*TVECT(LNK1,1,1)-TVECT(LNK1,2,1)*
     >          TVECT(LNK1,1,3))
           RJ(3)=-(TVECT(LNK1,2,1)*TVECT(LNK1,1,2)-TVECT(LNK1,2,2)*
     >          TVECT(LNK1,1,1))
           RSTRT=.FALSE.
         ELSE
           RJ(1)=TVECT(LNK1,2,2)*TVECT(LNK1,3,3)-TVECT(LNK1,2,3)*
     >             TVECT(LNK1,3,2)
           RJ(2)=TVECT(LNK1,2,3)*TVECT(LNK1,3,1)-TVECT(LNK1,2,1)*
     >             TVECT(LNK1,3,3)
           RJ(3)=TVECT(LNK1,2,1)*TVECT(LNK1,3,2)-TVECT(LNK1,2,2)*
     >             TVECT(LNK1,3,1)
           RSTRT=.TRUE.
         END IF
         UMAG=DSQRT(RJ(1)*RJ(1)+RJ(2)*RJ(2)+RJ(3)*RJ(3))
         IF(UMAG.EQ.0.D0) THEN
*          V2 IS COLINEAR WITH V1 OR V3 (DEPENDING ON RSTRT)
*         NEED TO TEST 3 CASES
           AXD(1)=A(2)*D(3)-A(3)*D(2)
           AXD(2)=A(3)*D(1)-A(1)*D(3)
           AXD(3)=A(1)*D(2)-A(2)*D(1)
           AXDMAG=DSQRT(AXD(1)*AXD(1)+AXD(2)*AXD(2)+AXD(3)*AXD(3))
           IF(AXDMAG.EQ.0.D0)THEN
*          FIRST THE PARALLEL CASE (FIND PROJECTIONS OF BOTH ENDS OF
*          D ON A)
             DOT1=(C(1)*A(1)+C(2)*A(2)+C(3)*A(3))/MAGA
             DOT2=((C(1)+D(1))*A(1)+(C(2)+D(2))*A(2)+(C(3)+D(3))*A(3))
     >            /MAGA
             DOTMAX=DMAX1(DOT1,DOT2)
             DOTMIN=DMIN1(DOT1,DOT2)
             IF(RSTRT) THEN
               DOT=DMAX1(DOT1,DOT2)
               TVECT(LNK1,3,1)=TVECT(LNK1,3,1)-(1.D0-DOT)*
     >                         TVECT(LNK1,2,1)
               TVECT(LNK1,3,2)=TVECT(LNK1,3,2)-(1.D0-DOT)*
     >                         TVECT(LNK1,2,2)
               TVECT(LNK1,3,3)=TVECT(LNK1,3,3)-(1.D0-DOT)*
     >                         TVECT(LNK1,2,3)
               TVECT(LNK1,2,1)=TVECT(LNK1,2,1)*DOT
               TVECT(LNK1,2,2)=TVECT(LNK1,2,2)*DOT
               TVECT(LNK1,2,3)=TVECT(LNK1,2,3)*DOT
             ELSE
               DOT=DMIN1(DOT1,DOT2)
               TVECT(LNK1,1,1)=TVECT(LNK1,1,1)+DOT*TVECT(LNK1,2,1)
               TVECT(LNK1,1,2)=TVECT(LNK1,1,2)+DOT*TVECT(LNK1,2,2)
               TVECT(LNK1,1,3)=TVECT(LNK1,1,3)+DOT*TVECT(LNK1,2,3)
               TVECT(LNK1,2,1)=TVECT(LNK1,2,1)*(1.D0-DOT)
               TVECT(LNK1,2,2)=TVECT(LNK1,2,2)*(1.D0-DOT)
               TVECT(LNK1,2,3)=TVECT(LNK1,2,3)*(1.D0-DOT)
             END IF
           ELSE
*          CHECK THE OTHER CASE
*            CROSS THE AXD WITH A TO GET A NORMAL TO A ON WHICH D CAN
*            BE PROJECTED
```

```
                AXDXA(1)=AXD(2)*A(3)-AXD(3)*A(2)
                AXDXA(2)=AXD(3)*A(1)-AXD(1)*A(3)
                AXDXA(3)=AXD(1)*A(2)-AXD(2)*A(1)
                DOT1=C(1)*AXDXA(1)+C(2)*AXDXA(2)+C(3)*AXDXA(3)
                DOT2=(C(1)+D(1))*AXDXA(1)+(C(2)+D(2))*AXDXA(2)+(C(3)+
     >               D(3))*AXDXA(3)
                IF(DOT1*DOT2.LT.0.D0)THEN
*                 FIND THE CROSS POINT BY PROPORTIONS
                  DTOT=DABS(DOT1)+DABS(DOT2)
                  RATIO=DABS(DOT1)/DTOT
                  DOT=((C(1)+RATIO*D(1))*A(1)+(C(2)+RATIO*D(2))*
     >                 A(2)+(C(3)+RATIO*D(3))*A(3))/MAGA
                ELSE
*                 DOT THE END CLOSEST TO THE PLANE OF ROTATION
                  IF(DABS(DOT1).GT.DABS(DOT2))THEN
                   DOT=((C(1)+D(1))*A(1)+(C(2)+D(2))*A(2)+(C(3)+D(3))*
     >                  A(1))/MAGA
                  ELSE
                   DOT=(C(1)*A(1)+C(2)*A(2)+C(3)*A(3))/MAGA
                  END IF
                END IF
                IF(RSTRT) THEN
                  TVECT(LNK1,3,1)=TVECT(LNK1,3,1)-(1.D0-DOT)*
     >                            TVECT(LNK1,2,1)
                  TVECT(LNK1,3,2)=TVECT(LNK1,3,2)-(1.D0-DOT)*
     >                            TVECT(LNK1,2,2)
                  TVECT(LNK1,3,3)=TVECT(LNK1,3,3)-(1.D0-DOT)*
     >                            TVECT(LNK1,2,3)
                  TVECT(LNK1,2,1)=TVECT(LNK1,2,1)*DOT
                  TVECT(LNK1,2,2)=TVECT(LNK1,2,2)*DOT
                  TVECT(LNK1,2,3)=TVECT(LNK1,2,3)*DOT
                ELSE
                  TVECT(LNK1,1,1)=TVECT(LNK1,1,1)+DOT*TVECT(LNK1,2,1)
                  TVECT(LNK1,1,2)=TVECT(LNK1,1,2)+DOT*TVECT(LNK1,2,2)
                  TVECT(LNK1,1,3)=TVECT(LNK1,1,3)+DOT*TVECT(LNK1,2,3)
                  TVECT(LNK1,2,1)=TVECT(LNK1,2,1)*(1.D0-DOT)
                  TVECT(LNK1,2,2)=TVECT(LNK1,2,2)*(1.D0-DOT)
                  TVECT(LNK1,2,3)=TVECT(LNK1,2,3)*(1.D0-DOT)
                END IF
             END IF
          ELSE
*         IT IS NOT COLINEAR WITH V2
*         NEED TO TEST THE 4 CASES
*         REPOSITION VECTOR 2 TO AVOID THE INTRUDER
             RJ(1)=RJ(1)/UMAG
             RJ(2)=RJ(2)/UMAG
             RJ(3)=RJ(3)/UMAG
             IF(RSTRT)THEN
               TA(1)=A(1)
               TA(2)=A(2)
               TA(3)=A(3)
               SA(1)=SV(1,1)
               SA(2)=SV(1,2)
               SA(3)=SV(1,3)
             ELSE
               TA(1)=-A(1)
               TA(2)=-A(2)
               TA(3)=-A(3)
               SA(1)=EV(1,1)
```

```
              SA(2)=EV(1,2)
              SA(3)=EV(1,3)
              C(1)=C(1)-A(1)
              C(2)=C(2)-A(2)
              C(3)=C(3)-A(3)
            END IF
            TAP(1)=TA(1)
            TAP(2)=TA(2)
            TAP(3)=TA(3)
*           CHECK CASE WHERE TAIL OF A IS CLOSER THAN B TO D AT
*           PROJECTION POINT ONTO D (REPRESENTED AS AN INFINITE LINE).
* CDOTD, DTST, AND TESTD NEW 1-1-90.  OLD TEST FOUND TO BE IN ERROR.
            CDOTD=(C(1)*D(1)+C(2)*D(2)+C(3)*D(3))/MAGD
            DTST(1)=C(1)-CDOTD*D(1)/MAGD
            DTST(2)=C(2)-CDOTD*D(2)/MAGD
            DTST(3)=C(3)-CDOTD*D(3)/MAGD
            TESTD=DSQRT(DTST(1)**2+DTST(2)**2+DTST(3)**2)
            DOT1=C(1)*RJ(1)+C(2)*RJ(2)+C(3)*RJ(3)
            DOT2=(C(1)+D(1))*RJ(1)+(C(2)+D(2))*RJ(2)+(C(3)+D(3))*
     >           RJ(3)
            MAGC=DSQRT(C(1)**2+C(2)**2+C(3)**2)
            MAG2=DSQRT((C(1)+D(1))**2+(C(2)+D(2))**2+(C(3)+D(3))**2)
            DOTDU=(D(1)*RJ(1)+D(2)*RJ(2)+D(3)*RJ(3))/MAGD
            IF(TESTD.LT.B)THEN
*            THIS IS CASE ONE AND A SPHERIC SOLUTION WILL WORK.  FIND
*            THE END OF D CLOSEST TO THE ROTATION AXIS.
              IF(MAGC.LT.MAG2)THEN
                 TSV(2,1)=SV(2,1)
                 TSV(2,2)=SV(2,2)
                 TSV(2,3)=SV(2,3)
                 TEV(2,1)=SV(2,1)
                 TEV(2,2)=SV(2,2)
                 TEV(2,3)=SV(2,3)
              ELSE
                 TSV(2,1)=EV(2,1)
                 TSV(2,2)=EV(2,2)
                 TSV(2,3)=EV(2,3)
                 TEV(2,1)=EV(2,1)
                 TEV(2,2)=EV(2,2)
                 TEV(2,3)=EV(2,3)
              END IF
              TSV(1,1)=SV(1,1)
              TSV(1,2)=SV(1,2)
              TSV(1,3)=SV(1,3)
              TEV(1,1)=EV(1,1)
              TEV(1,2)=EV(1,2)
              TEV(1,3)=EV(1,3)
              B=B/1.001D0
              CALL AVSPH(V,OPT,TSV,TEV,B,TVECT,LNK1,JPOSI,JNT,GOOD)
              IF(.NOT.GOOD)THEN
                 WRITE(6,*)'FAILED IN AVCYL CASE 1 V2'
              END IF
            ELSE IF(DOTDU.EQ.0.D0)THEN
*           IT IS CASE TWO (PERPENDICULAR TO ROTATION AXIS)
              IF(RSTRT)THEN
                 VC(1)=TVECT(LNK1,3,1)
                 VC(2)=TVECT(LNK1,3,2)
                 VC(3)=TVECT(LNK1,3,3)
              ELSE
```

```
            VC(1)=TVECT(LNK1,1,1)
            VC(2)=TVECT(LNK1,1,2)
            VC(3)=TVECT(LNK1,1,3)
         END IF
         MAGVC=DSQRT(VC(1)*VC(1)+VC(2)*VC(2)+VC(3)*VC(3))
         PERP(1)=(VC(2)*RJ(3)-VC(3)*RJ(2))/MAGVC
         PERP(2)=(VC(3)*RJ(1)-VC(1)*RJ(3))/MAGVC
         PERP(3)=(VC(1)*RJ(2)-VC(2)*RJ(1))/MAGVC
         DOTP=TA(1)*PERP(1)+TA(2)*PERP(2)+TA(3)*PERP(3)
         PERP(1)=PERP(1)*DOTP
         PERP(2)=PERP(2)*DOTP
         PERP(3)=PERP(3)*DOTP
         IF(RSTRT)THEN
            TS(1)=SV(2,1)-SV(1,1)
            TS(2)=SV(2,2)-SV(1,2)
            TS(3)=SV(2,3)-SV(1,3)
            TE(1)=EV(2,1)-SV(1,1)
            TE(2)=EV(2,2)-SV(1,2)
            TE(3)=EV(2,3)-SV(1,3)
         ELSE
            TS(1)=SV(2,1)-EV(1,1)
            TS(2)=SV(2,2)-EV(1,2)
            TS(3)=SV(2,3)-EV(1,3)
            TE(1)=EV(2,1)-EV(1,1)
            TE(2)=EV(2,2)-EV(1,2)
            TE(3)=EV(2,3)-EV(1,3)
         END IF
         TCROSS(1)=TS(2)*TE(3)-TS(3)*TE(2)
         TCROSS(2)=TS(3)*TE(1)-TS(1)*TE(3)
         TCROSS(3)=TS(1)*TE(2)-TS(2)*TE(1)
         DOTX=RJ(1)*TCROSS(1)+RJ(2)*TCROSS(2)+RJ(3)*TCROSS(3)
*        WHEN DOTX IS > 0 THERE IS A POSITIVE ANGULAR TRANSITION
*        FROM START TO END ABOUT RJ
         DOTE=PERP(1)*TE(1)+PERP(2)*TE(2)+PERP(3)*TE(3)
         DOTS=PERP(1)*TS(1)+PERP(2)*TS(2)+PERP(3)*TS(3)
         IF(DOTX.LT.0.D0)THEN
            IF(DOTS.GT.0.D0)THEN
               TSV(2,1)=SV(2,1)
               TSV(2,2)=SV(2,2)
               TSV(2,3)=SV(2,3)
               TEV(2,1)=SV(2,1)
               TEV(2,2)=SV(2,2)
               TEV(2,3)=SV(2,3)
               TSV(1,1)=SV(1,1)
               TSV(1,2)=SV(1,2)
               TSV(1,3)=SV(1,3)
               TEV(1,1)=EV(1,1)
               TEV(1,2)=EV(1,2)
               TEV(1,3)=EV(1,3)
               B=B/1.001D0
             CALL AVSPH(V,OPT,TSV,TEV,B,TVECT,LNK1,JPOSI,JNT,GOOD)
            ELSE
               WRITE(6,*)'FAILED IN AVCYL1'
               GOOD=.FALSE.
            END IF
         ELSE
            IF(DOTE.GT.0.D0)THEN
               IF(DOTE.LE.DOTP.AND.DOTS.LE.DOTP)THEN
                  TSV(2,1)=EV(2,1)
```

```
                        TSV(2,2)=EV(2,2)
                        TSV(2,3)=EV(2,3)
                        TEV(2,1)=EV(2,1)
                        TEV(2,2)=EV(2,2)
                        TEV(2,3)=EV(2,3)
                        TSV(1,1)=SV(1,1)
                        TSV(1,2)=SV(1,2)
                        TSV(1,3)=SV(1,3)
                        TEV(1,1)=EV(1,1)
                        TEV(1,2)=EV(1,2)
                        TEV(1,3)=EV(1,3)
                        B=B/1.001D0
                        CALL AVSPH(V,OPT,TSV,TEV,B,TVECT,LNK1,JPOSI,
     >                              JNT,GOOD)
                     ELSE
                        TSV(2,1)=SV(2,1)+D(1)*(DOTP-DOTS)/(DOTE-DOTS)
                        TSV(2,2)=SV(2,2)+D(2)*(DOTP-DOTS)/(DOTE-DOTS)
                        TSV(2,3)=SV(2,3)+D(3)*(DOTP-DOTS)/(DOTE-DOTS)
                        TEV(2,1)=SV(2,1)+D(1)*(DOTP-DOTS)/(DOTE-DOTS)
                        TEV(2,2)=SV(2,2)+D(2)*(DOTP-DOTS)/(DOTE-DOTS)
                        TEV(2,3)=SV(2,3)+D(3)*(DOTP-DOTS)/(DOTE-DOTS)
                        TSV(1,1)=SV(1,1)
                        TSV(1,2)=SV(1,2)
                        TSV(1,3)=SV(1,3)
                        TEV(1,1)=EV(1,1)
                        TEV(1,2)=EV(1,2)
                        TEV(1,3)=EV(1,3)
                        B=B/1.001D0
                        CALL AVSPH(V,OPT,TSV,TEV,B,TVECT,LNK1,JPOSI,
     >                              JNT,GOOD)
                     END IF
                  ELSE
                     WRITE(6,*)'FAILED IN AVCYL2'
                     GOOD=.FALSE.
                  END IF
               END IF
               IF(.NOT.GOOD)THEN
                  WRITE(6,*)'FAILED IN AVCYL CASE 2 V2'
               END IF
            ELSE IF(DABS(DOTDU).EQ.1.D0)THEN
*           IT IS CASE THREE (PARALLEL TO ROTATION AXIS)
               TS(1)=SV(2,1)-SV(1,1)
               TS(2)=SV(2,2)-SV(1,2)
               TS(3)=SV(2,3)-SV(1,3)
               TE(1)=EV(2,1)-SV(1,1)
               TE(2)=EV(2,2)-SV(1,2)
               TE(3)=EV(2,3)-SV(1,3)
               DOTE=RJ(1)*TE(1)+RJ(2)*TE(2)+RJ(3)*TE(3)
               DOTS=RJ(1)*TS(1)+RJ(2)*TS(2)+RJ(3)*TS(3)
               IF(DOTE*DOTS.GT.0.D0)THEN
                  IF(DABS(DOTE).GT.DABS(DOTS))THEN
                        TSV(2,1)=SV(2,1)
                        TSV(2,2)=SV(2,2)
                        TSV(2,3)=SV(2,3)
                        TEV(2,1)=SV(2,1)
                        TEV(2,2)=SV(2,2)
                        TEV(2,3)=SV(2,3)
                  ELSE
                        TSV(2,1)=EV(2,1)
```

```
                        TSV(2,2)=EV(2,2)
                        TSV(2,3)=EV(2,3)
                        TEV(2,1)=EV(2,1)
                        TEV(2,2)=EV(2,2)
                        TEV(2,3)=EV(2,3)
                     END IF
                  ELSE
                     DOTD=D(1)*RJ(1)+D(2)*RJ(2)+D(3)*RJ(3)
                     TSV(2,1)=SV(2,1)+D(1)*DABS(DOTS)/DABS(DOTD)
                     TSV(2,2)=SV(2,2)+D(2)*DABS(DOTS)/DABS(DOTD)
                     TSV(2,3)=SV(2,3)+D(3)*DABS(DOTS)/DABS(DOTD)
                     TEV(2,1)=SV(2,1)+D(1)*DABS(DOTS)/DABS(DOTD)
                     TEV(2,2)=SV(2,2)+D(2)*DABS(DOTS)/DABS(DOTD)
                     TEV(2,3)=SV(2,3)+D(3)*DABS(DOTS)/DABS(DOTD)
                  END IF
                  TSV(1,1)=SV(1,1)
                  TSV(1,2)=SV(1,2)
                  TSV(1,3)=SV(1,3)
                  TEV(1,1)=EV(1,1)
                  TEV(1,2)=EV(1,2)
                  TEV(1,3)=EV(1,3)
                  B=B/1.001D0
                  CALL AVSPH(V,OPT,TSV,TEV,B,TVECT,LNK1,JPOSI,JNT,GOOD)
                  IF(.NOT.GOOD)THEN
                     WRITE(6,*)'FAILED IN AVCYL CASE 3 V2'
                  END IF
               ELSE
*              IT IS CASE 4 (GENERAL CASE)
                  SD(1)=SV(2,1)
                  SD(2)=SV(2,2)
                  SD(3)=SV(2,3)
                  CALL RODIR(B,SA,TA,SD,D,RJ,ROTP,ROTN)
                  CALL ROTATE(RJ,ROTP,TA)
*  FIND OUT IF THE ENDS OF D ARE GREATER THAN B FROM THE PLANE OF
*  ROTATION, AND ON THE ROTATION CENTER SIDE OF VC.
                  DOTS=C(1)*RJ(1)+C(2)*RJ(2)+C(3)*RJ(3)
                  DOTE=(C(1)+D(1))*RJ(1)+(C(2)+D(2))*RJ(2)+
     >                 (C(3)+D(3))*RJ(3)
*     FIND OUT HOW FAR THE START AND END OF D PROJECT ONTO A VECTOR
*     WHICH IS PERPENDICULAR TO VC FROM SA.
                  IF(RSTRT)THEN
                     VC(1)=TVECT(LNK1,3,1)
                     VC(2)=TVECT(LNK1,3,2)
                     VC(3)=TVECT(LNK1,3,3)
                  ELSE
                     VC(1)=TVECT(LNK1,1,1)
                     VC(2)=TVECT(LNK1,1,2)
                     VC(3)=TVECT(LNK1,1,3)
                  END IF
                  MAGVC=DSQRT(VC(1)*VC(1)+VC(2)*VC(2)+VC(3)*VC(3))
                  PERP(1)=(VC(2)*RJ(3)-VC(3)*RJ(2))/MAGVC
                  PERP(2)=(VC(3)*RJ(1)-VC(1)*RJ(3))/MAGVC
                  PERP(3)=(VC(1)*RJ(2)-VC(2)*RJ(1))/MAGVC
                  TMAGA=TAP(1)*PERP(1)+TAP(2)*PERP(2)+TAP(3)*PERP(3)

*     WITH THAT DONE, FIND WHERE C AND C+D PROJECT ONTO PERP
                  CMAGS=C(1)*PERP(1)+C(2)*PERP(2)+C(2)*PERP(2)
                  CMAGE=(C(1)+D(1))*PERP(1)+(C(2)+D(2))*PERP(2)+
     >                 (C(3)+D(3))*PERP(3)
```

```
                VS(1)=C(1)-DOTS*RJ(1)
                VS(2)=C(2)-DOTS*RJ(2)
                VS(3)=C(3)-DOTS*RJ(3)
                VE(1)=C(1)+D(1)-DOTE*RJ(1)
                VE(2)=C(2)+D(2)-DOTE*RJ(2)
                VE(3)=C(3)+D(3)-DOTE*RJ(3)
                DVS=RJ(1)*(TAP(2)*VS(3)-TAP(3)*VS(2))+
     >              RJ(2)*(TAP(3)*VS(1)-TAP(1)*VS(3))+
     >              RJ(3)*(TAP(1)*VS(2)-TAP(2)*VS(1))
                DVE=RJ(1)*(TAP(2)*VE(3)-TAP(3)*VE(2))+
     >              RJ(2)*(TAP(3)*VE(1)-TAP(1)*VE(3))+
     >              RJ(3)*(TAP(1)*VE(2)-TAP(2)*VE(1))
                IF(DVE.GT.0.D0.AND.DVS.GT.0.D0)THEN
*             FIND THE MOST POSITIVE END IN TERMS OF ANGLE FROM TAP
                    MAGVE=DSQRT(VE(1)**2+VE(2)**2+VE(3)**2)
                    MAGVS=DSQRT(VS(1)**2+VS(2)**2+VS(3)**2)
                    ANGE=DACOS(
     >                (VE(1)*TAP(1)+VE(2)*TAP(2)+VE(3)*TAP(3))/MAGA/MAGVE)
                    ANGS=DACOS(
     >                (VS(1)*TAP(1)+VS(2)*TAP(2)+VS(3)*TAP(3))/MAGA/MAGVS)
                    IF(ANGE.GT.ANGS)THEN
                       ITSE=.TRUE.
                    ELSE
                       ITSS=.TRUE.
                    END IF
                    ELSE IF(DVE.GT.0.D0)THEN
                       ITSE=.TRUE.
                    ELSE IF(DVS.GT.0.D0)THEN
                       ITSS=.TRUE.
                END IF

*>>>>>>>>>>>>>
                IF(ITSS.AND.DABS(DOTS).LT.B)THEN
                   IF(CMAGS.LT.TMAGA)THEN
                      TSV(2,1)=SV(2,1)
                      TSV(2,2)=SV(2,2)
                      TSV(2,3)=SV(2,3)
                      TEV(2,1)=SV(2,1)
                      TEV(2,2)=SV(2,2)
                      TEV(2,3)=SV(2,3)
                      TSV(1,1)=SV(1,1)
                      TSV(1,2)=SV(1,2)
                      TSV(1,3)=SV(1,3)
                      TEV(1,1)=EV(1,1)
                      TEV(1,2)=EV(1,2)
                      TEV(1,3)=EV(1,3)
                      B=B/1.001D0
                      CALL AVSPH(V,OPT,TSV,TEV,B,TVECT,LNK1,JPOSI,JNT,
     >                           GOOD)
                   ELSE
*     SINCE THE END IS WITHIN B OF THE PLANE OF ROTATION, SHORTEN THE
*     VECTOR SO ITS IMAGE ON PERP IS A MAGNITUDE OF TMAGA AND USE THIS
*     AS THE LOACTION OF A SPHERE FOR AVOIDANCE
                      VS(1)=VS(1)*TMAGA/CMAGS
                      VS(2)=VS(2)*TMAGA/CMAGS
                      VS(3)=VS(3)*TMAGA/CMAGS
                      TSV(2,1)=SV(1,1)+VS(1)+RJ(1)*DOTS
                      TSV(2,2)=SV(1,2)+VS(2)+RJ(2)*DOTS
                      TSV(2,3)=SV(1,3)+VS(3)+RJ(3)*DOTS
```

```
                  TEV(2,1)=TSV(2,1)
                  TEV(2,2)=TSV(2,2)
                  TEV(2,3)=TSV(2,3)
                  TSV(1,1)=SV(1,1)
                  TSV(1,2)=SV(1,2)
                  TSV(1,3)=SV(1,3)
                  TEV(1,1)=EV(1,1)
                  TEV(1,2)=EV(1,2)
                  TEV(1,3)=EV(1,3)
                  B=B/1.001D0
                  CALL AVSPH(V,OPT,TSV,TEV,B,TVECT,LNK1,JPOSI,JNT,
     >                       GOOD)
               END IF
            ELSE IF(ITSE.AND.DABS(DOTE).LT.B)THEN
               IF(CMAGE.LT.TMAGA)THEN
                  TSV(2,1)=EV(2,1)
                  TSV(2,2)=EV(2,2)
                  TSV(2,3)=EV(2,3)
                  TEV(2,1)=EV(2,1)
                  TEV(2,2)=EV(2,2)
                  TEV(2,3)=EV(2,3)
                  TSV(1,1)=SV(1,1)
                  TSV(1,2)=SV(1,2)
                  TSV(1,3)=SV(1,3)
                  TEV(1,1)=EV(1,1)
                  TEV(1,2)=EV(1,2)
                  TEV(1,3)=EV(1,3)
                  B=B/1.001D0
                  CALL AVSPH(V,OPT,TSV,TEV,B,TVECT,LNK1,JPOSI,JNT,
     >                       GOOD)
               ELSE
*     SINCE THE END IS WITHIN B OF THE PLANE OF ROTATION, SHORTEN ITS
*     IMAGE IN THE PLANE TO A MAGNITUDE OF MAGA AND USE THIS AS THE
*     LOACTION OF A SPHERE FOR AVOIDANCE
                  VE(1)=VE(1)*TMAGA/CMAGE
                  VE(2)=VE(2)*TMAGA/CMAGE
                  VE(3)=VE(3)*TMAGA/CMAGE
                  TSV(2,1)=SV(1,1)+VE(1)+RJ(1)*DOTE
                  TSV(2,2)=SV(1,2)+VE(2)+RJ(2)*DOTE
                  TSV(2,3)=SV(1,3)+VE(3)+RJ(3)*DOTE
                  TEV(2,1)=TSV(2,1)
                  TEV(2,2)=TSV(2,2)
                  TEV(2,3)=TSV(2,3)
                  TSV(1,1)=SV(1,1)
                  TSV(1,2)=SV(1,2)
                  TSV(1,3)=SV(1,3)
                  TEV(1,1)=EV(1,1)
                  TEV(1,2)=EV(1,2)
                  TEV(1,3)=EV(1,3)
                  B=B/1.001D0
                  CALL AVSPH(V,OPT,TSV,TEV,B,TVECT,LNK1,JPOSI,JNT,
     >                       GOOD)
               END IF
            ELSE
*>>>>>>>>>>>>
*              CHECK THE ANGLE BETWEEN TA AND D. IF IT IS TOO GREAT
*              THEN GOOD IS FALSE. OTHERWISE RESIZE V3 OR V1 AND A
```

```
                  DOT=(TA(2)*VC(3)-TA(3)*VC(2))*RJ(1)+
     >                 (TA(3)*VC(1)-TA(1)*VC(3))*RJ(2)+
     >                 (TA(1)*VC(2)-TA(2)*VC(1))*RJ(3)

                  IF(DOT.LE.0.DO)THEN
                     GOOD=.FALSE.
                  ELSE
                     GOOD=.TRUE.
                     DOTVCI=(TAP(1)*VC(1)+TAP(2)*VC(2)+TAP(3)*
     >                        VC(3))/MAGVC
                     DOTVCE=(TA(1)*VC(1)+TA(2)*VC(2)+TA(3)*VC(3))/MAGVC
                     THET1=DACOS(DOTVCI/MAGA)
                     THET2=DACOS(DOTVCE/MAGA)
                     NEWTA=DSIN(THET1)/DSIN(THET2)
                     TA(1)=TA(1)*NEWTA
                     TA(2)=TA(2)*NEWTA
                     TA(3)=TA(3)*NEWTA
                     DOTVCE=(TA(1)*VC(1)+TA(2)*VC(2)+TA(3)*VC(3))/MAGVC
                     NEWVC=MAGVC+DOTVCE-DOTVCI
                     VC(1)=VC(1)*NEWVC/MAGVC
                     VC(2)=VC(2)*NEWVC/MAGVC
                     VC(3)=VC(3)*NEWVC/MAGVC
                     IF(RSTRT)THEN
                        TVECT(LNK1,2,1)=TA(1)
                        TVECT(LNK1,2,2)=TA(2)
                        TVECT(LNK1,2,3)=TA(3)
                        TVECT(LNK1,3,1)=VC(1)
                        TVECT(LNK1,3,2)=VC(2)
                        TVECT(LNK1,3,3)=VC(3)
                     ELSE
                        TVECT(LNK1,2,1)=-TA(1)
                        TVECT(LNK1,2,2)=-TA(2)
                        TVECT(LNK1,2,3)=-TA(3)
                        TVECT(LNK1,1,1)=VC(1)
                        TVECT(LNK1,1,2)=VC(2)
                        TVECT(LNK1,1,3)=VC(3)
                     END IF
                  END IF
               END IF
            END IF
         END IF
      ELSE
*  437    SET UP THE ROTATION VECTOR FOR A
         IF(V.EQ.1)THEN
            RJ(1)=JNT(LNK1,1,1)
            RJ(2)=JNT(LNK1,1,2)
            RJ(3)=JNT(LNK1,1,3)
            IF(OPT(LNK1,1).EQ.2)THEN
               RJ(1)=-RJ(1)
               RJ(2)=-RJ(2)
               RJ(3)=-RJ(3)
            END IF
         ELSE IF(V.EQ.3)THEN
*           THE SECOND INDEX OF JNT ONLY GOES UP TO 2 (SECOND JOINT AXIS)
            RJ(1)=JNT(LNK1,2,1)
            RJ(2)=JNT(LNK1,2,2)
            RJ(3)=JNT(LNK1,2,3)
            IF(OPT(LNK1,3).EQ.2)THEN
               RJ(1)=-RJ(1)
```

```
               RJ(2)=-RJ(2)
               RJ(3)=-RJ(3)
            END IF
         END IF
*       SOLVE THE 4 CASES
*        CHECK CASE WHERE TAIL OF A IS CLOSER THAN B TO D AT
*        PROJECTION POINT ONTO D.
* CDOTD, DTST, AND TESTD NEW 1-1-90.  OLD TEST FOUND TO BE IN ERROR.
            CDOTD=(C(1)*D(1)+C(2)*D(2)+C(3)*D(3))/MAGD
            DTST(1)=C(1)-CDOTD*D(1)/MAGD
            DTST(2)=C(2)-CDOTD*D(2)/MAGD
            DTST(3)=C(3)-CDOTD*D(3)/MAGD
            TESTD=DSQRT(DTST(1)**2+DTST(2)**2+DTST(3)**2)
            MAGC=DSQRT(C(1)**2+C(2)**2+C(3)**2)
         MAG2=DSQRT((C(1)+D(1))**2+(C(2)+D(2))**2+(C(3)+D(3))**2)
         DOTDU=(D(1)*RJ(1)+D(2)*RJ(2)+D(3)*RJ(3))/MAGD
         IF(TESTD.LT.B)THEN
*         THIS IS CASE ONE AND A SPHERIC SOLUTION WILL WORK.  FIND
*         THE END OF D CLOSEST TO THE ROTATION AXIS.
            IF(MAGC.LT.MAG2)THEN
               TSV(2,1)=SV(2,1)
               TSV(2,2)=SV(2,2)
               TSV(2,3)=SV(2,3)
               TEV(2,1)=SV(2,1)
               TEV(2,2)=SV(2,2)
               TEV(2,3)=SV(2,3)
            ELSE
               TSV(2,1)=EV(2,1)
               TSV(2,2)=EV(2,2)
               TSV(2,3)=EV(2,3)
               TEV(2,1)=EV(2,1)
               TEV(2,2)=EV(2,2)
               TEV(2,3)=EV(2,3)
            END IF
            TSV(1,1)=SV(1,1)
            TSV(1,2)=SV(1,2)
            TSV(1,3)=SV(1,3)
            TEV(1,1)=EV(1,1)
            TEV(1,2)=EV(1,2)
            TEV(1,3)=EV(1,3)
            B=B/1.001D0
            CALL AVSPH(V,OPT,TSV,TEV,B,TVECT,LNK1,JPOSI,JNT,GOOD)
            IF(.NOT.GOOD)THEN
               WRITE(6,*)'FAILED IN AVCYL CASE 1 V1 OR V3'
            END IF
         ELSE IF(DOTDU.EQ.0.D0)THEN
*        IT IS CASE TWO (PERPENDICULAR TO ROTATION AXIS)
            PERP(1)=(D(2)*RJ(3)-D(3)*RJ(2))/MAGD
            PERP(2)=(D(3)*RJ(1)-D(1)*RJ(3))/MAGD
            PERP(3)=(D(1)*RJ(2)-D(2)*RJ(1))/MAGD
            DOTP=A(1)*PERP(1)+A(2)*PERP(2)+A(3)*PERP(3)
            PERP(1)=PERP(1)*DOTP
            PERP(2)=PERP(2)*DOTP
            PERP(3)=PERP(3)*DOTP
            TS(1)=SV(2,1)-SV(1,1)
            TS(2)=SV(2,2)-SV(1,2)
            TS(3)=SV(2,3)-SV(1,3)
            TE(1)=EV(2,1)-SV(1,1)
            TE(2)=EV(2,2)-SV(1,2)
```

```
            TE(3)=EV(2,3)-SV(1,3)
            DOTP=TE(1)*PERP(1)+TE(2)*PERP(2)+TE(3)*PERP(3)
            PERP(1)=PERP(1)*DOTP
            PERP(2)=PERP(2)*DOTP
            PERP(3)=PERP(3)*DOTP
            TCROSS(1)=TS(2)*TE(3)-TS(3)*TE(2)
            TCROSS(2)=TS(3)*TE(1)-TS(1)*TE(3)
            TCROSS(3)=TS(1)*TE(2)-TS(2)*TE(1)
            DOTX=RJ(1)*TCROSS(1)+RJ(2)*TCROSS(2)+RJ(3)*TCROSS(3)
*           WHEN DOTX IS > 0 THERE IS A POSITIVE ANGULAR TRANSITION
*           FROM START TO END ABOUT RJ
            DOTE=RJ(1)*TE(1)+RJ(2)*TE(2)+RJ(3)*TE(3)
            DAXIS(1)=RJ(1)*DOTE
            DAXIS(2)=RJ(2)*DOTE
            DAXIS(3)=RJ(3)*DOTE
            IF(DOTX.LT.0.D0)THEN
*            WORRY ABOUT THE TAIL
               T1(1)=TS(1)-DAXIS(1)
               T1(2)=TS(2)-DAXIS(2)
               T1(3)=TS(3)-DAXIS(3)
               T1MAG=DSQRT(T1(1)*T1(1)+T1(2)*T1(2)+T1(3)*T1(3))
               IF(T1MAG.LE.MAGA)THEN
*                THE TAIL IS INSIDE THE CYLINDER DESCRIBED BY A
                  TSV(2,1)=SV(2,1)
                  TSV(2,2)=SV(2,2)
                  TSV(2,3)=SV(2,3)
                  TEV(2,1)=SV(2,1)
                  TEV(2,2)=SV(2,2)
                  TEV(2,3)=SV(2,3)
                  TSV(1,1)=SV(1,1)
                  TSV(1,2)=SV(1,2)
                  TSV(1,3)=SV(1,3)
                  TEV(1,1)=EV(1,1)
                  TEV(1,2)=EV(1,2)
                  TEV(1,3)=EV(1,3)
                  B=B/1.001D0
                  CALL AVSPH(V,OPT,TSV,TEV,B,TVECT,LNK1,JPOSI,JNT,GOOD)
               ELSE
                  BETA=DACOS(DABS(DOTE)/B)
                  NEWB=B*DSIN(BETA)
                  TEMPD(1)=-D(1)
                  TEMPD(2)=-D(2)
                  TEMPD(3)=-D(3)
                  THET0=(A(1)*TEMPD(1)+A(2)*TEMPD(2)+A(3)*TEMPD(3))
     >                 /MAGA/MAGD
                  THET2=DACOS((DABS(DOTP)-NEWB)/MAGA)
                  DTHET=THET0-THET2
                  CALL ROTATE(RJ,DTHET,A)
                  CHECK(1)=A(1)-TE(1)
                  CHECK(2)=A(2)-TE(2)
                  CHECK(3)=A(3)-TE(3)
                  DCHECK=(TEMPD(1)*CHECK(1)+TEMPD(2)*CHECK(2)+
     >                 TEMPD(3)*CHECK(3))/MAGD
                  IF(DCHECK.GT.MAGD)THEN
                     TSV(2,1)=SV(2,1)
                     TSV(2,2)=SV(2,2)
                     TSV(2,3)=SV(2,3)
                     TEV(2,1)=SV(2,1)
                     TEV(2,2)=SV(2,2)
```

```
                  TEV(2,3)=SV(2,3)
                  TSV(1,1)=SV(1,1)
                  TSV(1,2)=SV(1,2)
                  TSV(1,3)=SV(1,3)
                  TEV(1,1)=EV(1,1)
                  TEV(1,2)=EV(1,2)
                  TEV(1,3)=EV(1,3)
                  B=B/1.001D0
                 CALL AVSPH(V,OPT,TSV,TEV,B,TVECT,LNK1,JPOSI,JNT,GOOD)
               ELSE
                  GOOD=.TRUE.
                  IF(V.EQ.1)THEN
                     TVECT(LNK1,1,1)=A(1)
                     TVECT(LNK1,1,2)=A(2)
                     TVECT(LNK1,1,3)=A(3)
                     TVECT(LNK1,2,1)=TVECT(LNK1,2,1)+AP(1)-A(1)
                     TVECT(LNK1,2,2)=TVECT(LNK1,2,2)+AP(2)-A(2)
                     TVECT(LNK1,2,3)=TVECT(LNK1,2,3)+AP(3)-A(3)
                  ELSE IF(V.EQ.3)THEN
                     TVECT(LNK1,3,1)=A(1)
                     TVECT(LNK1,3,2)=A(2)
                     TVECT(LNK1,3,3)=A(3)
                     TVECT(LNK1,2,1)=TVECT(LNK1,2,1)+A(1)-AP(1)
                     TVECT(LNK1,2,2)=TVECT(LNK1,2,2)+A(2)-AP(2)
                     TVECT(LNK1,2,3)=TVECT(LNK1,2,3)+A(3)-AP(3)
                  ELSE
                     WRITE(6,*)'MAJOR MESS-UP IN AVCYL 1'
                  END IF
               END IF
            END IF
           ELSE
*           WORRY ABOUT THE HEAD
              T1(1)=TE(1)-DAXIS(1)
              T1(2)=TE(2)-DAXIS(2)
              T1(3)=TE(3)-DAXIS(3)
              T1MAG=DSQRT(T1(1)*T1(1)+T1(2)*T1(2)+T1(3)*T1(3))
              IF(T1MAG.LE.MAGA)THEN
                 TSV(2,1)=EV(2,1)
                 TSV(2,2)=EV(2,2)
                 TSV(2,3)=EV(2,3)
                 TEV(2,1)=EV(2,1)
                 TEV(2,2)=EV(2,2)
                 TEV(2,3)=EV(2,3)
                 TSV(1,1)=SV(1,1)
                 TSV(1,2)=SV(1,2)
                 TSV(1,3)=SV(1,3)
                 TEV(1,1)=EV(1,1)
                 TEV(1,2)=EV(1,2)
                 TEV(1,3)=EV(1,3)
                 B=B/1.001D0
                 CALL AVSPH(V,OPT,TSV,TEV,B,TVECT,LNK1,JPOSI,JNT,GOOD)
              ELSE
                 BETA=DACOS(DABS(DOTE)/B)
                 NEWB=B*DSIN(BETA)
                 TEMPD(1)=D(1)
                 TEMPD(2)=D(2)
                 TEMPD(3)=D(3)
                 THETO=(A(1)*TEMPD(1)+A(2)*TEMPD(2)+A(3)*TEMPD(3))
     >                  /MAGA/MAGD
```

```
                  THET2=DACOS((DABS(DOTP)-NEWB)/MAGA)
                  DTHET=THETO-THET2
                  CALL ROTATE(RJ,DTHET,A)
                  CHECK(1)=A(1)-TS(1)
                  CHECK(2)=A(2)-TS(2)
                  CHECK(3)=A(3)-TS(3)
                  DCHECK=(TEMPD(1)*CHECK(1)+TEMPD(2)*CHECK(2)+
     >                    TEMPD(3)*CHECK(3))/MAGD
                  IF(DCHECK.GT.MAGD)THEN
                     TSV(2,1)=EV(2,1)
                     TSV(2,2)=EV(2,2)
                     TSV(2,3)=EV(2,3)
                     TEV(2,1)=EV(2,1)
                     TEV(2,2)=EV(2,2)
                     TEV(2,3)=EV(2,3)
                     TSV(1,1)=SV(1,1)
                     TSV(1,2)=SV(1,2)
                     TSV(1,3)=SV(1,3)
                     TEV(1,1)=EV(1,1)
                     TEV(1,2)=EV(1,2)
                     TEV(1,3)=EV(1,3)
                     B=B/1.001D0
                   CALL AVSPH(V,OPT,TSV,TEV,B,TVECT,LNK1,JPOSI,JNT,GOOD)
                  ELSE
                     GOOD=.TRUE.
                     IF(V.EQ.1)THEN
                        TVECT(LNK1,1,1)=A(1)
                        TVECT(LNK1,1,2)=A(2)
                        TVECT(LNK1,1,3)=A(3)
                        TVECT(LNK1,2,1)=TVECT(LNK1,2,1)+AP(1)-A(1)
                        TVECT(LNK1,2,2)=TVECT(LNK1,2,2)+AP(2)-A(2)
                        TVECT(LNK1,2,3)=TVECT(LNK1,2,3)+AP(3)-A(3)
                     ELSE IF(V.EQ.3)THEN
                        TVECT(LNK1,3,1)=A(1)
                        TVECT(LNK1,3,2)=A(2)
                        TVECT(LNK1,3,3)=A(3)
                        TVECT(LNK1,2,1)=TVECT(LNK1,2,1)+A(1)-AP(1)
                        TVECT(LNK1,2,2)=TVECT(LNK1,2,2)+A(2)-AP(2)
                        TVECT(LNK1,2,3)=TVECT(LNK1,2,3)+A(3)-AP(3)
                     ELSE
                        WRITE(6,*)'MAJOR MESS-UP IN AVCYL 2'
                     END IF
                  END IF
               END IF
            END IF
            IF(.NOT.GOOD)THEN
*              IT SHOULD BE IMPOSSIBLE TO FAIL HERE, BUT I WANT TO KNOW
               WRITE(6,*)'FAILED IN AVCYL CASE 2 V1 OR V3'
            END IF
         ELSE IF(DABS(DOTDU).EQ.1.D0)THEN
*        IT IS CASE THREE (PARALLEL TO ROTATION AXIS)
            TS(1)=SV(2,1)-SV(1,1)
            TS(2)=SV(2,2)-SV(1,2)
            TS(3)=SV(2,3)-SV(1,3)
            TE(1)=EV(2,1)-SV(1,1)
            TE(2)=EV(2,2)-SV(1,2)
            TE(3)=EV(2,3)-SV(1,3)
            DOTE=RJ(1)*TE(1)+RJ(2)*TE(2)+RJ(3)*TE(3)
            DOTS=RJ(1)*TS(1)+RJ(2)*TS(2)+RJ(3)*TS(3)
```

```
            IF(DOTE*DOTS.GT.0.D0)THEN
               IF(DABS(DOTE).GT.DABS(DOTS))THEN
                     TSV(2,1)=SV(2,1)
                     TSV(2,2)=SV(2,2)
                     TSV(2,3)=SV(2,3)
                     TEV(2,1)=SV(2,1)
                     TEV(2,2)=SV(2,2)
                     TEV(2,3)=SV(2,3)
               ELSE
                     TSV(2,1)=EV(2,1)
                     TSV(2,2)=EV(2,2)
                     TSV(2,3)=EV(2,3)
                     TEV(2,1)=EV(2,1)
                     TEV(2,2)=EV(2,2)
                     TEV(2,3)=EV(2,3)
               END IF
            ELSE
               DOTD=D(1)*RJ(1)+D(2)*RJ(2)+D(3)*RJ(3)
               TSV(2,1)=SV(2,1)+D(1)*DABS(DOTS)/DABS(DOTD)
               TSV(2,2)=SV(2,2)+D(2)*DABS(DOTS)/DABS(DOTD)
               TSV(2,3)=SV(2,3)+D(3)*DABS(DOTS)/DABS(DOTD)
               TEV(2,1)=SV(2,1)+D(1)*DABS(DOTS)/DABS(DOTD)
               TEV(2,2)=SV(2,2)+D(2)*DABS(DOTS)/DABS(DOTD)
               TEV(2,3)=SV(2,3)+D(3)*DABS(DOTS)/DABS(DOTD)
            END IF
            TSV(1,1)=SV(1,1)
            TSV(1,2)=SV(1,2)
            TSV(1,3)=SV(1,3)
            TEV(1,1)=EV(1,1)
            TEV(1,2)=EV(1,2)
            TEV(1,3)=EV(1,3)
            B=B/1.001D0
            CALL AVSPH(V,OPT,TSV,TEV,B,TVECT,LNK1,JPOSI,JNT,GOOD)
            IF(.NOT.GOOD)THEN
               WRITE(6,*)'FAILED IN AVCYL CASE 3 V1 OR V3'
            END IF
         ELSE
*        IT IS CASE 4 (GENERAL CASE)
            SA(1)=SV(1,1)
            SA(2)=SV(1,2)
            SA(3)=SV(1,3)
            SD(1)=SV(2,1)
            SD(2)=SV(2,2)
            SD(3)=SV(2,3)
            CALL RODIR(B,SA,A,SD,D,RJ,ROTP,ROTN)
            CALL ROTATE(RJ,ROTP,A)
*  FIND OUT IF THE ENDS OF D ARE GREATER THAN B FROM THE PLANE OF
*  ROTATION, AND INSIDE THE CYLINDER OF ROTATION.
            DOTS=C(1)*RJ(1)+C(2)*RJ(2)+C(3)*RJ(3)
            DOTE=(C(1)+D(1))*RJ(1)+(C(2)+D(2))*RJ(2)+
     >           (C(3)+D(3))*RJ(3)
            MAGS=C(1)**2+C(2)**2+C(3)**2
            MAGE=(C(1)+D(1))**2+(C(2)+D(2))**2+(C(3)+D(3))**2
            CMAGS=DSQRT(MAGS-DOTS**2)
            CMAGE=DSQRT(MAGE-DOTE**2)
            VS(1)=C(1)-DOTS*RJ(1)
            VS(2)=C(2)-DOTS*RJ(2)
            VS(3)=C(3)-DOTS*RJ(3)
            VE(1)=C(1)+D(1)-DOTE*RJ(1)
```

```
            VE(2)=C(2)+D(2)-DOTE*RJ(2)
            VE(3)=C(3)+D(3)-DOTE*RJ(3)
            DVS=RJ(1)*(AP(2)*VS(3)-AP(3)*VS(2))+
     >           RJ(2)*(AP(3)*VS(1)-AP(1)*VS(3))+
     >           RJ(3)*(AP(1)*VS(2)-AP(2)*VS(1))
            DVE=RJ(1)*(AP(2)*VE(3)-AP(3)*VE(2))+
     >           RJ(2)*(AP(3)*VE(1)-AP(1)*VE(3))+
     >           RJ(3)*(AP(1)*VE(2)-AP(2)*VE(1))
            IF(DVE.GT.0.DO.AND.DVS.GT.0.DO)THEN
*            FIND THE MOST POSITIVE END IN TERMS OF ANGLE FROM AP
              MAGVE=DSQRT(VE(1)**2+VE(2)**2+VE(3)**2)
              MAGVS=DSQRT(VS(1)**2+VS(2)**2+VS(3)**2)
              ANGE=DACOS(
     >              (VE(1)*AP(1)+VE(2)*AP(2)+VE(3)*AP(3))/MAGA/MAGVE)
              ANGS=DACOS(
     >              (VS(1)*AP(1)+VS(2)*AP(2)+VS(3)*AP(3))/MAGA/MAGVS)
              IF(ANGE.GT.ANGS)THEN
                 ITSE=.TRUE.
              ELSE
                 ITSS=.TRUE.
              END IF
            ELSE IF(DVE.GT.0.DO)THEN
              ITSE=.TRUE.
            ELSE IF(DVS.GT.0.DO)THEN
                 ITSS=.TRUE.
            END IF
            IF(ITSS.AND.DABS(DOTS).LT.B)THEN
              IF(CMAGS.LT.MAGA)THEN
                 TSV(2,1)=SV(2,1)
                 TSV(2,2)=SV(2,2)
                 TSV(2,3)=SV(2,3)
                 TEV(2,1)=SV(2,1)
                 TEV(2,2)=SV(2,2)
                 TEV(2,3)=SV(2,3)
                 TSV(1,1)=SV(1,1)
                 TSV(1,2)=SV(1,2)
                 TSV(1,3)=SV(1,3)
                 TEV(1,1)=EV(1,1)
                 TEV(1,2)=EV(1,2)
                 TEV(1,3)=EV(1,3)
                 B=B/1.001D0
                 CALL AVSPH(V,OPT,TSV,TEV,B,TVECT,LNK1,JPOSI,JNT,GOOD)
              ELSE
*    SINCE THE END IS WITHIN B OF THE PLANE OF ROTATION, SHORTEN ITS
*    IMAGE IN THE PLANE TO A MAGNITUDE OF MAGA AND USE THIS AS THE
*    LOACTION OF A SPHERE FOR AVOIDANCE
                 MAGVS=DSQRT(VS(1)**2+VS(2)**2+VS(3)**2)
                 VS(1)=VS(1)*MAGA/MAGVS
                 VS(2)=VS(2)*MAGA/MAGVS
                 VS(3)=VS(3)*MAGA/MAGVS
                 TSV(2,1)=SV(1,1)+VS(1)+RJ(1)*DOTS
                 TSV(2,2)=SV(1,2)+VS(2)+RJ(2)*DOTS
                 TSV(2,3)=SV(1,3)+VS(3)+RJ(3)*DOTS
                 TEV(2,1)=TSV(2,1)
                 TEV(2,2)=TSV(2,2)
                 TEV(2,3)=TSV(2,3)
                 TSV(1,1)=SV(1,1)
                 TSV(1,2)=SV(1,2)
                 TSV(1,3)=SV(1,3)
```

```
                  TEV(1,1)=EV(1,1)
                  TEV(1,2)=EV(1,2)
                  TEV(1,3)=EV(1,3)
                  B=B/1.001D0
                  CALL AVSPH(V,OPT,TSV,TEV,B,TVECT,LNK1,JPOSI,JNT,GOOD)
               END IF
            ELSE IF(ITSE.AND.DABS(DOTE).LT.B)THEN
               IF(CMAGE.LT.MAGA)THEN
                  TSV(2,1)=EV(2,1)
                  TSV(2,2)=EV(2,2)
                  TSV(2,3)=EV(2,3)
                  TEV(2,1)=EV(2,1)
                  TEV(2,2)=EV(2,2)
                  TEV(2,3)=EV(2,3)
                  TSV(1,1)=SV(1,1)
                  TSV(1,2)=SV(1,2)
                  TSV(1,3)=SV(1,3)
                  TEV(1,1)=EV(1,1)
                  TEV(1,2)=EV(1,2)
                  TEV(1,3)=EV(1,3)
                  B=B/1.001D0
                  CALL AVSPH(V,OPT,TSV,TEV,B,TVECT,LNK1,JPOSI,JNT,GOOD)
               ELSE
*     SINCE THE END IS WITHIN B OF THE PLANE OF ROTATION, SHORTEN ITS
*     IMAGE IN THE PLANE TO A MAGNITUDE OF MAGA AND USE THIS AS THE
*     LOACTION OF A SPHERE FOR AVOIDANCE
                  MAGVE=DSQRT(VE(1)**2+VE(2)**2+VE(3)**2)
                  VE(1)=VE(1)*MAGA/MAGVE
                  VE(2)=VE(2)*MAGA/MAGVE
                  VE(3)=VE(3)*MAGA/MAGVE
                  TSV(2,1)=SV(1,1)+VE(1)+RJ(1)*DOTE
                  TSV(2,2)=SV(1,2)+VE(2)+RJ(2)*DOTE
                  TSV(2,3)=SV(1,3)+VE(3)+RJ(3)*DOTE
                  TEV(2,1)=TSV(2,1)
                  TEV(2,2)=TSV(2,2)
                  TEV(2,3)=TSV(2,3)
                  TSV(1,1)=SV(1,1)
                  TSV(1,2)=SV(1,2)
                  TSV(1,3)=SV(1,3)
                  TEV(1,1)=EV(1,1)
                  TEV(1,2)=EV(1,2)
                  TEV(1,3)=EV(1,3)
                  B=B/1.001D0
                  CALL AVSPH(V,OPT,TSV,TEV,B,TVECT,LNK1,JPOSI,JNT,GOOD)
               END IF

            ELSE
*               IF
*                 THE CASE WHERE D IS FOUND TO BE OFF THE END OF A
*                 WILL NOT BE IMPLEMENTED AT THIS TIME 8-10-89.
*                 (THE ONLY CASE WHICH CAN ONLY BE FOUND NUMERICALLY)
*               ELSE
                  GOOD=.TRUE.
                  IF(V.EQ.1)THEN
                     TVECT(LNK1,1,1)=A(1)
                     TVECT(LNK1,1,2)=A(2)
                     TVECT(LNK1,1,3)=A(3)
                     TVECT(LNK1,2,1)=TVECT(LNK1,2,1)+AP(1)-A(1)
                     TVECT(LNK1,2,2)=TVECT(LNK1,2,2)+AP(2)-A(2)
```

```
                  TVECT(LNK1,2,3)=TVECT(LNK1,2,3)+AP(3)-A(3)
               ELSE IF(V.EQ.3)THEN
                  TVECT(LNK1,3,1)=A(1)
                  TVECT(LNK1,3,2)=A(2)
                  TVECT(LNK1,3,3)=A(3)
                  TVECT(LNK1,2,1)=TVECT(LNK1,2,1)+A(1)-AP(1)
                  TVECT(LNK1,2,2)=TVECT(LNK1,2,2)+A(2)-AP(2)
                  TVECT(LNK1,2,3)=TVECT(LNK1,2,3)+A(3)-AP(3)
               ELSE
                  WRITE(6,*)'MAJOR MESS-UP IN AVCYL 3'
               END IF
*           END IF
         END IF
      END IF
   END IF
   RETURN
   END

   SUBROUTINE     AVSPH(V,OPT,SV,EV,B,TVECT,LNK1,JPOSI,JNT,GOOD)
* NOTES MAY 13 THRU JUNE
*  THIS ROUTINE IS FOR SPHERICAL INTRUDER AVIDANCE
   REAL *8 TVECT(20,3,3),JNT(20,2,3),RJ(3)
   REAL *8 SV(2,3),EV(2,3),DOT,UX(3),UY(3),UZ(3),B,PHI,U(3)
   REAL *8 A(3),AP(3),C(3),AMAG,D(3),MAGD,ROTP,GAMA,THETA,DOTA,DOTA2
   REAL *8 UXD(3),MAGF,MAGV1,MAGA2,MAGV3
   REAL *8 DP(3),MAGDP,PI,V3(3),V1(3)
   REAL *8 THET3,THET1,THET2,MAGA,UMAG,TEST
   INTEGER LNK1,JPOSI,OPT(20,3),V,TESPOS
   LOGICAL RSTRT,GOOD
   GOOD=.TRUE.
   B=B*1.001D0
   UX(1)=1.D0
   UX(2)=0.D0
   UX(3)=0.D0
   UY(1)=0.D0
   UY(2)=1.D0
   UY(3)=0.D0
   UZ(1)=0.D0
   UZ(2)=0.D0
   UZ(3)=1.D0
   A(1)=EV(1,1)-SV(1,1)
   A(2)=EV(1,2)-SV(1,2)
   A(3)=EV(1,3)-SV(1,3)
   AP(1)=A(1)
   AP(2)=A(2)
   AP(3)=A(3)
   MAGA=DSQRT(A(1)*A(1)+A(2)*A(2)+A(3)*A(3))
* C IS POSITION VECTOR FROM TAIL OF V1 TO SPHERE CENTER
   C(1)=SV(2,1)-SV(1,1)
   C(2)=SV(2,2)-SV(1,2)
   C(3)=SV(2,3)-SV(1,3)
   IF(V.EQ.2)THEN
      TESPOS=2*(JPOSI/2)
      IF(OPT(LNK1,2).EQ.1.OR.(OPT(LNK1,2).EQ.3.AND.TESPOS.NE.
  >       JPOSI))THEN
*        MINUS SIGN USED TO GET POSITIVE INDEXING ALONG V1 FOR
*        POSITIVE ROTATION
         RJ(1)=-(TVECT(LNK1,2,2)*TVECT(LNK1,1,3)-TVECT(LNK1,2,3)*
  >          TVECT(LNK1,1,2))
```

```
            RJ(2)=-(TVECT(LNK1,2,3)*TVECT(LNK1,1,1)-TVECT(LNK1,2,1)*
     >           TVECT(LNK1,1,3))
            RJ(3)=-(TVECT(LNK1,2,1)*TVECT(LNK1,1,2)-TVECT(LNK1,2,2)*
     >           TVECT(LNK1,1,1))
            RSTRT=.FALSE.
         ELSE
            RJ(1)=TVECT(LNK1,2,2)*TVECT(LNK1,3,3)-TVECT(LNK1,2,3)*
     >              TVECT(LNK1,3,2)
            RJ(2)=TVECT(LNK1,2,3)*TVECT(LNK1,3,1)-TVECT(LNK1,2,1)*
     >              TVECT(LNK1,3,3)
            RJ(3)=TVECT(LNK1,2,1)*TVECT(LNK1,3,2)-TVECT(LNK1,2,2)*
     >              TVECT(LNK1,3,1)
            RSTRT=.TRUE.
         END IF
         UMAG=DSQRT(RJ(1)*RJ(1)+RJ(2)*RJ(2)+RJ(3)*RJ(3))
         IF(UMAG.EQ.0.D0)THEN
*          PROJECT THE SPHERE ONTO THE VECTOR AND SHORTEN THE VECTOR.
            DOT=(A(1)*C(1)+A(2)*C(2)+A(3)*C(3))/MAGA
            IF(RSTRT)THEN
               TVECT(LNK1,3,1)=TVECT(LNK1,3,1)-(1.D0-DOT)*
     >                         TVECT(LNK1,2,1)
               TVECT(LNK1,3,2)=TVECT(LNK1,3,2)-(1.D0-DOT)*
     >                         TVECT(LNK1,2,2)
               TVECT(LNK1,3,3)=TVECT(LNK1,3,3)-(1.D0-DOT)*
     >                         TVECT(LNK1,2,3)
               TVECT(LNK1,2,1)=TVECT(LNK1,2,1)*DOT
               TVECT(LNK1,2,2)=TVECT(LNK1,2,2)*DOT
               TVECT(LNK1,2,3)=TVECT(LNK1,2,3)*DOT
            ELSE
               TVECT(LNK1,1,1)=TVECT(LNK1,1,1)+DOT*TVECT(LNK1,2,1)
               TVECT(LNK1,1,2)=TVECT(LNK1,1,2)+DOT*TVECT(LNK1,2,2)
               TVECT(LNK1,1,3)=TVECT(LNK1,1,3)+DOT*TVECT(LNK1,2,3)
               TVECT(LNK1,2,1)=TVECT(LNK1,2,1)*(1.D0-DOT)
               TVECT(LNK1,2,2)=TVECT(LNK1,2,2)*(1.D0-DOT)
               TVECT(LNK1,2,3)=TVECT(LNK1,2,3)*(1.D0-DOT)
            END IF
         ELSE
            RJ(1)=RJ(1)/UMAG
            RJ(2)=RJ(2)/UMAG
            RJ(3)=RJ(3)/UMAG
            DOT=(RJ(1)*C(1)+RJ(2)*C(2)+RJ(3)*C(3))
            D(1)=C(1)-DOT*RJ(1)
            D(2)=C(2)-DOT*RJ(2)
            D(3)=C(3)-DOT*RJ(3)
            MAGF=DABS(B*DSIN(DACOS(DABS(DOT/B))))
            IF(RSTRT)THEN
               MAGD=DSQRT(D(1)*D(1)+D(2)*D(2)+D(3)*D(3))
               GAMA=DASIN(MAGF/MAGD)
               THETA=DACOS((A(1)*D(1)+A(2)*D(2)+A(3)*D(3))/MAGA/MAGD)
               UXD(1)=RJ(2)*D(3)-RJ(3)*D(2)
               UXD(2)=RJ(3)*D(1)-RJ(1)*D(3)
               UXD(3)=RJ(1)*D(2)-RJ(2)*D(1)
               DOTA=A(1)*UXD(1)+A(2)*UXD(2)+A(3)*UXD(3)
               IF(DOTA.GE.0.D0)THEN
                  ROTP=GAMA-THETA
               ELSE
                  ROTP=GAMA+THETA
               END IF
            ELSE
```

```
            DP(1)=D(1)-A(1)
            DP(2)=D(2)-A(2)
            DP(3)=D(3)-A(3)
            MAGDP=DSQRT(DP(1)*DP(1)+DP(2)*DP(2)+DP(3)*DP(3))
            GAMA=DASIN(MAGF/MAGDP)
            THETA=DACOS((-A(1)*DP(1)-A(2)*DP(2)-A(3)*DP(3))/MAGA/
     >                  MAGDP)
            UXD(1)=RJ(2)*DP(3)-RJ(3)*DP(2)
            UXD(2)=RJ(3)*DP(1)-RJ(1)*DP(3)
            UXD(3)=RJ(1)*DP(2)-RJ(2)*DP(1)
            DOTA=A(1)*UXD(1)+A(2)*UXD(2)+A(3)*UXD(3)
            IF(DOTA.GE.0.D0)THEN
               ROTP=GAMA+THETA
            ELSE
               ROTP=GAMA-THETA
            END IF
          END IF
          CALL ROTATE(RJ,ROTP,A)
          PI=DACOS(-1.D0)
          IF(RSTRT)THEN
*          ROTATE ABOUT POINT ON V1 TO LENGTHEN V3.
           V3(1)=TVECT(LNK1,3,1)
           V3(2)=TVECT(LNK1,3,2)
           V3(3)=TVECT(LNK1,3,3)
           TEST=(A(2)*V3(3)-A(3)*V3(2))*RJ(1)+(
     >     A(3)*V3(1)-A(1)*V3(3))*RJ(2)+(A(1)*V3(2)-A(2)*V3(1))*RJ(3)
           IF(TEST.LT.0.D0)THEN
              GOOD=.FALSE.
              WRITE(6,*)'FAILED IN AVSPH'
              GO TO 999
           END IF
           MAGV3=DSQRT(V3(1)*V3(1)+V3(2)*V3(2)+V3(3)*V3(3))
           THET2=DACOS((AP(1)*V3(1)+AP(2)*V3(2)+AP(3)*V3(3))/
     >                 MAGA/MAGV3)
           THET3=THET2-ROTP
           MAGA2=MAGA*DSIN(THET2)/DSIN(THET3)
           TVECT(LNK1,2,1)=A(1)*MAGA2/MAGA
           TVECT(LNK1,2,2)=A(2)*MAGA2/MAGA
           TVECT(LNK1,2,3)=A(3)*MAGA2/MAGA
           TVECT(LNK1,3,1)=V3(1)-AP(1)+TVECT(LNK1,2,1)
           TVECT(LNK1,3,2)=V3(2)-AP(2)+TVECT(LNK1,2,2)
           TVECT(LNK1,3,3)=V3(3)-AP(3)+TVECT(LNK1,2,3)
          ELSE
*          ROTATE ABOUT POINT ON V3 TO LENGTHEN V1.
           V1(1)=TVECT(LNK1,1,1)
           V1(2)=TVECT(LNK1,1,2)
           V1(3)=TVECT(LNK1,1,3)
           TEST=(A(2)*V1(3)-A(3)*V1(2))*RJ(1)+(
     >     A(3)*V1(1)-A(1)*V1(3))*RJ(2)+(A(1)*V1(2)-A(2)*V1(1))*RJ(3)
           IF(TEST.GT.0.D0)THEN
              GOOD=.FALSE.
              WRITE(6,*)'FAILED IN AVSPH2'
              GO TO 999
           END IF
           MAGV1=DSQRT(V1(1)*V1(1)+V1(2)*V1(2)+V1(3)*V1(3))
           THET2=DACOS((AP(1)*V1(1)+AP(2)*V1(2)+AP(3)*V1(3))/
     >                 MAGA/MAGV1)
           THET1=ROTP+THET2
           MAGA2=MAGA*DSIN(THET2)/DSIN(THET1)
```

```
        TVECT(LNK1,2,1)=A(1)*MAGA2/MAGA
        TVECT(LNK1,2,2)=A(2)*MAGA2/MAGA
        TVECT(LNK1,2,3)=A(3)*MAGA2/MAGA
        TVECT(LNK1,1,1)=V1(1)+AP(1)-TVECT(LNK1,2,1)
        TVECT(LNK1,1,2)=V1(2)+AP(2)-TVECT(LNK1,2,2)
        TVECT(LNK1,1,3)=V1(3)+AP(3)-TVECT(LNK1,2,3)
      END IF
   END IF
ELSE IF(V.EQ.1)THEN
   RJ(1)=JNT(LNK1,1,1)
   RJ(2)=JNT(LNK1,1,2)
   RJ(3)=JNT(LNK1,1,3)
   IF(OPT(LNK1,1).EQ.2)THEN
      RJ(1)=-RJ(1)
      RJ(2)=-RJ(2)
      RJ(3)=-RJ(3)
   END IF
   DOT=(RJ(1)*C(1)+RJ(2)*C(2)+RJ(3)*C(3))
   D(1)=C(1)-DOT*RJ(1)
   D(2)=C(2)-DOT*RJ(2)
   D(3)=C(3)-DOT*RJ(3)
   MAGF=DABS(B*DSIN(DACOS(DABS(DOT/B))))
   MAGD=DSQRT(D(1)*D(1)+D(2)*D(2)+D(3)*D(3))
   GAMA=DASIN(MAGF/MAGD)
   IF((A(1)*D(1)+A(2)*D(2)+A(3)*D(3))/MAGA/MAGD.GT.1.D0)THEN
      THETA=0.D0
   ELSE IF((A(1)*D(1)+A(2)*D(2)+A(3)*D(3))/MAGA/MAGD.LT.-1.D0)THEN
      THETA=DACOS(-1.D0)
   ELSE
      THETA=DACOS((A(1)*D(1)+A(2)*D(2)+A(3)*D(3))/MAGA/MAGD)
   END IF
   UXD(1)=RJ(2)*D(3)-RJ(3)*D(2)
   UXD(2)=RJ(3)*D(1)-RJ(1)*D(3)
   UXD(3)=RJ(1)*D(2)-RJ(2)*D(1)
   DOTA=A(1)*UXD(1)+A(2)*UXD(2)+A(3)*UXD(3)
   IF(DOTA.GE.0.D0)THEN
      ROTP=GAMA-THETA
   ELSE
      ROTP=GAMA+THETA
   END IF
   CALL ROTATE(RJ,ROTP,A)
   DOTA2=(C(1)*A(1)+C(2)*A(2)+C(3)*A(3))/MAGA
   IF(DOTA2.GT.MAGA)THEN
      GAMA=DACOS((MAGA*MAGA+MAGD*MAGD-B*B)/2.D0/MAGD/MAGA)
      IF(DOTA.GE.0.D0)THEN
         ROTP=GAMA-THETA
      ELSE
         ROTP=GAMA+THETA
      END IF
      A(1)=AP(1)
      A(2)=AP(2)
      A(3)=AP(3)
      CALL ROTATE(RJ,ROTP,A)
   END IF
   TVECT(LNK1,2,1)=-A(1)+AP(1)+TVECT(LNK1,2,1)
   TVECT(LNK1,2,2)=-A(2)+AP(2)+TVECT(LNK1,2,2)
   TVECT(LNK1,2,3)=-A(3)+AP(3)+TVECT(LNK1,2,3)
   TVECT(LNK1,1,1)=A(1)
   TVECT(LNK1,1,2)=A(2)
```

```
         TVECT(LNK1,1,3)=A(3)
      ELSE IF(V.EQ.3)THEN
         RJ(1)=JNT(LNK1,2,1)
         RJ(2)=JNT(LNK1,2,2)
         RJ(3)=JNT(LNK1,2,3)
         IF(OPT(LNK1,3).EQ.2)THEN
            RJ(1)=-RJ(1)
            RJ(2)=-RJ(2)
            RJ(3)=-RJ(3)
         END IF
         DOT=(RJ(1)*C(1)+RJ(2)*C(2)+RJ(3)*C(3))
         D(1)=C(1)-DOT*RJ(1)
         D(2)=C(2)-DOT*RJ(2)
         D(3)=C(3)-DOT*RJ(3)
         MAGF=DABS(B*DSIN(DACOS(DABS(DOT/B))))
         MAGD=DSQRT(D(1)*D(1)+D(2)*D(2)+D(3)*D(3))
         GAMA=DASIN(MAGF/MAGD)
         THETA=DACOS((A(1)*D(1)+A(2)*D(2)+A(3)*D(3))/MAGA/MAGD)
         UXD(1)=RJ(2)*D(3)-RJ(3)*D(2)
         UXD(2)=RJ(3)*D(1)-RJ(1)*D(3)
         UXD(3)=RJ(1)*D(2)-RJ(2)*D(1)
         DOTA=A(1)*UXD(1)+A(2)*UXD(2)+A(3)*UXD(3)
         IF(DOTA.GE.0.D0)THEN
            ROTP=GAMA-THETA
         ELSE
            ROTP=GAMA+THETA
         END IF
         CALL ROTATE(RJ,ROTP,A)
         DOTA2=(C(1)*A(1)+C(2)*A(2)+C(3)*A(3))/MAGA
         IF(DOTA2.GT.MAGA)THEN
            GAMA=DACOS((MAGA*MAGA+MAGD*MAGD-B*B)/2.D0/MAGD/MAGA)
            IF(DOTA.GE.0.D0)THEN
               ROTP=GAMA-THETA
            ELSE
               ROTP=GAMA+THETA
            END IF
            A(1)=AP(1)
            A(2)=AP(2)
            A(3)=AP(3)
            CALL ROTATE(RJ,ROTP,A)
         END IF
         TVECT(LNK1,2,1)=A(1)-AP(1)+TVECT(LNK1,2,1)
         TVECT(LNK1,2,2)=A(2)-AP(2)+TVECT(LNK1,2,2)
         TVECT(LNK1,2,3)=A(3)-AP(3)+TVECT(LNK1,2,3)
         TVECT(LNK1,3,1)=A(1)
         TVECT(LNK1,3,2)=A(2)
         TVECT(LNK1,3,3)=A(3)
      END IF
999   CONTINUE
      RETURN
      END

      SUBROUTINE CHKONE(SV,EV,B,HIT)
* THIS ROUTINE USE USED TO CHECK INTERFERENCE BETWEEN TWO CYLINDRICAL
* ELEMENTS
      CHARACTER *7 VERF
      REAL*8 SV(2,3),EV(2,3),LAM1,LAM2,LAM3,NX,NY,NZ
      REAL*8 MAGA,MAGB,MAGN,MAG,B,MAXX1,MINX1,MAXY1,MINY1,MAXZ1,MINZ1
      REAL*8 MAXX2,MINX2,MAXY2,MINY2,MAXZ2,MINZ2,S2,BR,CR,UPB,PN
```

```
      REAL*8 D1,D2,D3,D4,MIND,P,Q,R,S,T,U,DOT
      REAL*8 AX,AY,AZ,BX,BY,BZ,CX,CY,CZ,DX,DY,DZ
      LOGICAL HIT,PARL
      VERF='CHKONE '
      HIT=.TRUE.
      MAXX1=DMAX1(SV(1,1),EV(1,1))+B/2.D0
      MINX1=DMIN1(SV(1,1),EV(1,1))-B/2.D0
      MAXY1=DMAX1(SV(1,2),EV(1,2))+B/2.D0
      MINY1=DMIN1(SV(1,2),EV(1,2))-B/2.D0
      MAXZ1=DMAX1(SV(1,3),EV(1,3))+B/2.D0
      MINZ1=DMIN1(SV(1,3),EV(1,3))-B/2.D0
      MAXX2=DMAX1(SV(2,1),EV(2,1))+B/2.D0
      MINX2=DMIN1(SV(2,1),EV(2,1))-B/2.D0
      MAXY2=DMAX1(SV(2,2),EV(2,2))+B/2.D0
      MINY2=DMIN1(SV(2,2),EV(2,2))-B/2.D0
      MAXZ2=DMAX1(SV(2,3),EV(2,3))+B/2.D0
      MINZ2=DMIN1(SV(2,3),EV(2,3))-B/2.D0
      IF(MINX1.GT.MAXX2.OR.MINX2.GT.MAXX1.OR.MINY1.GT.MAXY2.OR.
     >   MINY2.GT.MAXY1.OR.MINZ1.GT.MAXZ2.OR.MINZ2.GT.MAXZ1)THEN
         HIT=.FALSE.
      ELSE

      AX=EV(1,1)-SV(1,1)
      AY=EV(1,2)-SV(1,2)
      AZ=EV(1,3)-SV(1,3)
      BX=EV(2,1)-SV(2,1)
      BY=EV(2,2)-SV(2,2)
      BZ=EV(2,3)-SV(2,3)
      CX=SV(2,1)-SV(1,1)
      CY=SV(2,2)-SV(1,2)
      CZ=SV(2,3)-SV(1,3)
      NX=AY*BZ-AZ*BY
      NY=AZ*BX-BZ*AX
      NZ=AX*BY-AY*BX
      IF(NX.EQ.0.0.AND.NY.EQ.0.0.AND.NZ.EQ.0.0)THEN
* *   SOLVE THE PARALLEL PROBLEM
         S2=(BX*BX+BY*BY+BZ*BZ)
         BR=(-CX*BX-CY*BY-CZ*BZ)/S2
         CR=BR+(AX*BX+AY*BY+AZ*BZ)/S2
         MAGB=DSQRT(S2)
         UPB=MAGB+B
         IF((BR*MAGB.GE.UPB.AND.CR.GE.UPB).OR.
     >       (BR.LT.-B.AND.CR.LT.-B))THEN
            HIT=.FALSE.
         ELSE
              PN=DSQRT((CX+BR*BX)*(CX+BR*BX)+(CY+BR*BY)*(CY+BR*BY)+
     >         (CZ+BR*BZ)*(CZ+BR*BZ))
              IF(PN.GE.B)THEN
                  HIT=.FALSE.
              ELSE
                  D1=DSQRT(CX*CX+CY*CY+CZ*CZ)
                  D2=DSQRT((EV(2,1)-SV(1,1))**2+(EV(2,2)-SV(1,2))**2+
     >                     (EV(2,3)-SV(1,3))**2)
                  D3=DSQRT((EV(2,1)-EV(1,1))**2+(EV(2,2)-EV(1,2))**2+
     >                     (EV(2,3)-EV(1,3))**2)
                  D4=DSQRT((SV(2,1)-EV(1,1))**2+(SV(2,2)-EV(1,2))**2+
     >                     (SV(2,3)-EV(1,3))**2)
                  MIND=DMIN1(D1,D2,D3,D4)
                  IF(((BR.GE.1.D0.AND.CR.GE.1.D0).OR.
```

```
>                (BR.LT.0.DO.AND.CR.LT.0.DO)).AND.MIND.GE.B)THEN
                  HIT=.FALSE.
               ELSE
                  PARL=.TRUE.
               END IF
            END IF
      END IF
 ELSE
      MAGN=DSQRT(NX*NX+NY*NY+NZ*NZ)
      NX=NX/MAGN
      NY=NY/MAGN
      NZ=NZ/MAGN
      IF(ABS(NZ).GE.ABS(NX).AND.ABS(NZ).GE.ABS(NY))THEN
            IF(ABS(AX).GT.ABS(AY))THEN
               P=AY*BX-AX*BY
               Q=AY*NX-AX*NY
               R=AX*CY-AY*CX
               S=AZ*BX-AX*BZ
               T=AZ*NX-AX*NZ
               U=AX*CZ-AZ*CX
               LAM3=(P*U-S*R)/(P*T-S*Q)
               IF(DABS(LAM3).GE.B)THEN
                  HIT=.FALSE.
               ELSE
                  LAM2=(R-LAM3*Q)/P
                  LAM1=(CX+BX*LAM2+NX*LAM3)/AX
               END IF
            ELSE
               P=AX*BY-AY*BX
               Q=AX*NY-AY*NX
               R=AY*CX-AX*CY
               S=AZ*BY-AY*BZ
               T=AZ*NY-AY*NZ
               U=AY*CZ-AZ*CY
               LAM3=(P*U-S*R)/(P*T-S*Q)
               IF(DABS(LAM3).GE.B)THEN
                  HIT=.FALSE.
               ELSE
                  LAM2=(R-LAM3*Q)/P
                  LAM1=(CY+BY*LAM2+NY*LAM3)/AY
               END IF
            END IF
      ELSE IF(ABS(NY).GE.ABS(NX))THEN
            IF(ABS(AX).GT.ABS(AZ))THEN
               P=AZ*BX-AX*BZ
               Q=AZ*NX-AX*NZ
               R=AX*CZ-AZ*CX
               S=AY*BX-AX*BY
               T=AY*NX-AX*NY
               U=AX*CY-AY*CX
               LAM3=(P*U-S*R)/(P*T-S*Q)
               IF(DABS(LAM3).GE.B)THEN
                  HIT=.FALSE.
               ELSE
                  LAM2=(R-LAM3*Q)/P
                  LAM1=(CX+BX*LAM2+NX*LAM3)/AX
               END IF
            ELSE
               P=AX*BZ-AZ*BX
```

```
            Q=AX*NZ-AZ*NX
            R=AZ*CX-AX*CZ
            S=AY*BZ-AZ*BY
            T=AY*NZ-AZ*NY
            U=AZ*CY-AY*CZ
            LAM3=(P*U-S*R)/(P*T-S*Q)
            IF(DABS(LAM3).GE.B)THEN
               HIT=.FALSE.
            ELSE
               LAM2=(R-LAM3*Q)/P
               LAM1=(CZ+BZ*LAM2+NZ*LAM3)/AZ
            END IF
         END IF
      ELSE
         IF(ABS(AZ).GT.ABS(AY))THEN
            P=AY*BZ-AZ*BY
            Q=AY*NZ-AZ*NY
            R=AZ*CY-AY*CZ
            S=AX*BZ-AZ*BX
            T=AX*NZ-AZ*NX
            U=AZ*CX-AX*CZ
            LAM3=(P*U-S*R)/(P*T-S*Q)
            IF(DABS(LAM3).GE.B)THEN
               HIT=.FALSE.
            ELSE
               LAM2=(R-LAM3*Q)/P
               LAM1=(CZ+BZ*LAM2+NZ*LAM3)/AZ
            END IF
         ELSE
            P=AZ*BY-AY*BZ
            Q=AZ*NY-AY*NZ
            R=AY*CZ-AZ*CY
            S=AX*BY-AY*BX
            T=AX*NY-AY*NX
            U=AY*CX-AX*CY
            LAM3=(P*U-S*R)/(P*T-S*Q)
            IF(DABS(LAM3).GE.B)THEN
               HIT=.FALSE.
            ELSE
               LAM2=(R-LAM3*Q)/P
               LAM1=(CY+BY*LAM2+NY*LAM3)/AY
            END IF
         END IF
      END IF
      IF(HIT)THEN
         IF(LAM1.GE.0.D0.AND.LAM1.LE.1.D0.AND.
     >      LAM2.GE.0.D0.AND.LAM2.LE.1.D0)THEN
            PARL=.FALSE.
         ELSE IF(LAM1.GE.0.D0.AND.LAM1.LE.1.D0)THEN
            MAGA=DSQRT(AX*AX+AY*AY+AZ*AZ)
            IF(LAM2.GT.1.D0)THEN
               DOT=((CX+BX)*AX+(CY+BY)*AY+(CZ+BZ)*AZ)/MAGA
               DX=DOT*AX/MAGA
               DY=DOT*AY/MAGA
               DZ=DOT*AZ/MAGA
               MAG=DSQRT((CX+BX-DX)**2+(CY+BY-DY)**2+(CZ+BZ-DZ)**2)
               IF(MAG.GE.B)THEN
                  HIT=.FALSE.
               ELSE
```

```
            PARL=.FALSE.
         END IF
      ELSE
         DOT=(CX*AX+CY*AY+CZ*AZ)/MAGA
         DX=DOT*AX/MAGA
         DY=DOT*AY/MAGA
         DZ=DOT*AZ/MAGA
         MAG=DSQRT((CX-DX)**2+(CY-DY)**2+(CZ-DZ)**2)
         IF(MAG.GE.B)THEN
            HIT=.FALSE.
         ELSE
            PARL=.FALSE.
         END IF
      END IF
   ELSE IF(LAM2.GE.0.D0.AND.LAM2.LE.1.D0)THEN
      MAGB=DSQRT(BX*BX+BY*BY+BZ*BZ)
      IF(LAM1.GT.1.D0)THEN
         DOT=((-CX+AX)*BX+(-CY+AY)*BY+(-CZ+AZ)*BZ)/MAGB
         DX=DOT*BX/MAGB
         DY=DOT*BY/MAGB
         DZ=DOT*BZ/MAGB
         MAG=DSQRT((-CX+AX-DX)**2+(-CY+AY-DY)**2+
     >                (-CZ+AZ-DZ)**2)
         IF(MAG.GE.B)THEN
            HIT=.FALSE.
         ELSE
            PARL=.FALSE.
         END IF
      ELSE
         DOT=(-CX*BX-CY*BY-CZ*BZ)/MBGB
         DX=DOT*BX/MAGB
         DY=DOT*BY/MAGB
         DZ=DOT*BZ/MAGB
         MAG=DSQRT((-CX-DX)**2+(-CY-DY)**2+(-CZ-DZ)**2)
         IF(MAG.GE.B)THEN
            HIT=.FALSE.
         ELSE
            PARL=.FALSE.
         END IF
      END IF
   ELSE IF(LAM1.GT.1.D0.AND.LAM2.GT.1.D0)THEN
      MAGB=DSQRT(BX*BX+BY*BY+BZ*BZ)
      MAGA=DSQRT(AX*AX+AY*AY+AZ*AZ)
      DOT=((CX+BX)*AX+(CY+BY)*AY+(CZ+BZ)*AZ)/MAGA
      IF(DOT.LT.MAGA.AND.DOT.GT.0.D0)THEN
         DX=DOT*AX/MAGA
         DY=DOT*AY/MAGA
         DZ=DOT*AZ/MAGA
         MAG=DSQRT((CX+BX-DX)**2+(CY+BY-DY)**2+(CZ+BZ-DZ)**2)
         IF(MAG.GE.B)THEN
            HIT=.FALSE.
         ELSE
            PARL=.FALSE.
         END IF
      ELSE
         DOT=((-CX+AX)*BX+(-CY+AY)*BY+(-CZ+AZ)*BZ)/MAGB
         IF(DOT.LT.MAGB.AND.DOT.GT.0.D0)THEN
            DX=DOT*BX/MAGB
            DY=DOT*BY/MAGB
```

```
                DZ=DOT*BZ/MAGB
                MAG=DSQRT((-CX+AX-DX)**2+(-CY+AY-DY)**2+
     >                    (-CZ+AZ-DZ)**2)
                IF(MAG.GE.B)THEN
                   HIT=.FALSE.
                ELSE
                   PARL=.FALSE.
                END IF
             ELSE
                MAG=DSQRT((EV(2,1)-EV(1,1))**2+
     >            (EV(2,2)-EV(1,2))**2+(EV(2,3)-EV(1,3))**2)
                IF(MAG.GE.B)THEN
                   HIT=.FALSE.
                ELSE
                   PARL=.FALSE.
                END IF
             END IF
          END IF
       ELSE IF(LAM1.GT.1.D0.AND.LAM2.LT.0.D0)THEN
          MAGB=DSQRT(BX*BX+BY*BY+BZ*BZ)
          MAGA=DSQRT(AX*AX+AY*AY+AZ*AZ)
          DOT=(CX*AX+CY*AY+CZ*AZ)/MAGA
          IF(DOT.LT.MAGA.AND.DOT.GT.0.D0)THEN
             DX=DOT*AX/MAGA
             DY=DOT*AY/MAGA
             DZ=DOT*AZ/MAGA
             MAG=DSQRT((CX-DX)**2+(CY-DY)**2+(CZ-DZ)**2)
             IF(MAG.GE.B)THEN
                HIT=.FALSE.
             ELSE
                PARL=.FALSE.
             END IF
          ELSE
             DOT=((-CX+AX)*BX+(-CY+AY)*BY+(-CZ+AZ)*BZ)/MAGB
             IF(DOT.GT.0.D0.AND.DOT.LT.MAGB)THEN
                DX=DOT*BX/MAGB
                DY=DOT*BY/MAGB
                DZ=DOT*BZ/MAGB
                MAG=DSQRT((-CX+AX-DX)**2+(-CY+AY-DY)**2+
     >                    (-CZ+AZ-DZ)**2)
                IF(MAG.GE.B)THEN
                   HIT=.FALSE.
                ELSE
                   PARL=.FALSE.
                END IF
             ELSE
                MAG=DSQRT((SV(2,1)-EV(1,1))**2+
     >               (SV(2,2)-EV(1,2))**2+(SV(2,3)-EV(1,3))**2)
                IF(MAG.GE.B)THEN
                   HIT=.FALSE.
                ELSE
                   PARL=.FALSE.
                END IF
             END IF
          END IF
       ELSE IF(LAM1.LT.0.D0.AND.LAM2.GT.1.D0)THEN
          MAGB=DSQRT(BX*BX+BY*BY+BZ*BZ)
          MAGA=DSQRT(AX*AX+AY*AY+AZ*AZ)
          DOT=((CX+BX)*AX+(CY+BY)*AY+(CZ+BZ)*AZ)/MAGA
```

```
            IF(DOT.GT.0.D0.AND.DOT.LT.MAGA)THEN
               DX=DOT*AX/MAGA
               DY=DOT*AY/MAGA
               DZ=DOT*AZ/MAGA
               MAG=DSQRT((CX+BX-DX)**2+(CY+BY-DY)**2+(CZ+BZ-DZ)**2)
               IF(MAG.GE.B)THEN
                  HIT=.FALSE.
               ELSE
                  PARL=.FALSE.
               END IF
            ELSE
               DOT=((-CX)*BX+(-CY)*BY+(-CZ)*BZ)/MAGB
               IF(DOT.LT.MAGB.AND.DOT.GT.0.D0)THEN
                  DX=DOT*BX/MAGB
                  DY=DOT*BY/MAGB
                  DZ=DOT*BZ/MAGB
                  MAG=DSQRT((-CX-DX)**2+(-CY-DY)**2+
     >                      (-CZ-DZ)**2)
                  IF(MAG.GE.B)THEN
                     HIT=.FALSE.
                  ELSE
                     PARL=.FALSE.
                  END IF
               ELSE
                  MAG=DSQRT((EV(2,1)-SV(1,1))**2+
     >                (EV(2,2)-SV(1,2))**2+(EV(2,3)-SV(1,3))**2)
                  IF(MAG.GE.B)THEN
                     HIT=.FALSE.
                  ELSE
                     PARL=.FALSE.
                  END IF
               END IF
            END IF
         ELSE
            MAGB=DSQRT(BX*BX+BY*BY+BZ*BZ)
            MAGA=DSQRT(AX*AX+AY*AY+AZ*AZ)
            DOT=(CX*AX+CY*AY+CZ*AZ)/MAGA
            IF(DOT.GT.0.D0.AND.DOT.LT.MAGA)THEN
               DX=DOT*AX/MAGA
               DY=DOT*AY/MAGA
               DZ=DOT*AZ/MAGA
               MAG=DSQRT((CX-DX)**2+(CY-DY)**2+(CZ-DZ)**2)
               IF(MAG.GE.B)THEN
                  HIT=.FALSE.
               ELSE
                  PARL=.FALSE.
               END IF
            ELSE
               DOT=((-CX)*BX+(-CY)*BY+(-CZ)*BZ)/MAGB
               IF(DOT.GT.0.D0.AND.DOT.LT.MAGB)THEN
                  DX=DOT*BX/MAGB
                  DY=DOT*BY/MAGB
                  DZ=DOT*BZ/MAGB
                  MAG=DSQRT((-CX-DX)**2+(-CY-DY)**2+
     >                      (-CZ-DZ)**2)
                  IF(MAG.GE.B)THEN
                     HIT=.FALSE.
                  ELSE
                     PARL=.FALSE.
```

```
                  END IF
               ELSE
                  MAG=DSQRT(CX*CX+CY*CY+CZ*CZ)
                  IF(MAG.GE.B)THEN
                     HIT=.FALSE.
                  ELSE
                     PARL=.FALSE.
                  END IF
               END IF
            END IF
         END IF
      END IF
      END IF
      RETURN
      END

      SUBROUTINE CHKTWO(SV,EV,B,HIT)
* CHECK FOR INTERFERENCE WITH SPHERICAL ELEMENTS
      REAL*8 SV(2,3),EV(2,3)
      REAL*8 MAGA,B,MAXX1,MINX1,MAXY1,MINY1,MAXZ1,MINZ1
      REAL*8 MAXX2,MINX2,MAXY2,MINY2,MAXZ2,MINZ2,S2,BR,UPB,PN
      REAL*8 D1,D4,MIND
      REAL*8 AX,AY,AZ,CX,CY,CZ
      LOGICAL HIT
      HIT=.TRUE.

      MAXX1=DMAX1(SV(1,1),EV(1,1))+B/2.DO
      MINX1=DMIN1(SV(1,1),EV(1,1))-B/2.DO
      MAXY1=DMAX1(SV(1,2),EV(1,2))+B/2.DO
      MINY1=DMIN1(SV(1,2),EV(1,2))-B/2.DO
      MAXZ1=DMAX1(SV(1,3),EV(1,3))+B/2.DO
      MINZ1=DMIN1(SV(1,3),EV(1,3))-B/2.DO
      MAXX2=SV(2,1)+B/2.DO
      MINX2=SV(2,1)-B/2.DO
      MAXY2=SV(2,2)+B/2.DO
      MINY2=SV(2,2)-B/2.DO
      MAXZ2=SV(2,3)+B/2.DO
      MINZ2=SV(2,3)-B/2.DO
      IF(MINX1.GT.MAXX2.OR.MINX2.GT.MAXX1.OR.MINY1.GT.MAXY2.OR.
     >   MINY2.GT.MAXY1.OR.MINZ1.GT.MAXZ2.OR.MINZ2.GT.MAXZ1)THEN
         HIT=.FALSE.
      ELSE
         AX=EV(1,1)-SV(1,1)
         AY=EV(1,2)-SV(1,2)
         AZ=EV(1,3)-SV(1,3)
         CX=SV(2,1)-SV(1,1)
         CY=SV(2,2)-SV(1,2)
         CZ=SV(2,3)-SV(1,3)
         S2=(AX*AX+AY*AY+AZ*AZ)
         BR=(CX*AX+CY*AY+CZ*AZ)/S2
         MAGA=DSQRT(S2)
         UPB=MAGA+B
         IF(BR*MAGA.GE.UPB.OR.BR.LT.-B)THEN
            HIT=.FALSE.
         ELSE
            PN=DSQRT((CX-BR*AX)*(CX-BR*AX)+(CY-BR*AY)*(CY-BR*AY)+
     >         (CZ-BR*AZ)*(CZ-BR*AZ))
            IF(PN.GE.B)THEN
```

```
                HIT=.FALSE.
              ELSE
                D1=DSQRT(CX*CX+CY*CY+CZ*CZ)
                D4=DSQRT((SV(2,1)-EV(1,1))**2+(SV(2,2)-EV(1,2))**2+
     >                   (SV(2,3)-EV(1,3))**2)
                MIND=DMIN1(D1,D4)
                IF((BR.GE.1.DO.OR.
     >              BR.LT.0.DO).AND.MIND.GE.B)THEN
                   HIT=.FALSE.
                END IF
              END IF
           END IF
        END IF
        RETURN
        END

        SUBROUTINE MOVE1(PDIR,SV,EV,SHIFT,B)
* REPOSITION A JOINT PAST A CYLINDRICAL ELEMENT
        LOGICAL PDIR
        REAL *8 SV(2,3),EV(2,3),SHIFT,B,POINT(3),SLIDE(3),MAGS,POINT2(3)
        REAL *8 AX,AY,AZ,BX,BY,BZ,CX,CY,CZ,LAM1,LAM2,LAM3
        REAL *8 NX,NY,NZ,DOT,NORM,P,Q,R,S,T,U,MAGB,PHI,MAGN
        IF(PDIR)THEN
           POINT(1)=SV(1,1)
           POINT(2)=SV(1,2)
           POINT(3)=SV(1,3)
           SLIDE(1)=EV(1,1)-SV(1,1)
           SLIDE(2)=EV(1,2)-SV(1,2)
           SLIDE(3)=EV(1,3)-SV(1,3)
        ELSE
           POINT(1)=EV(1,1)
           POINT(2)=EV(1,2)
           POINT(3)=EV(1,3)
           SLIDE(1)=SV(1,1)-EV(1,1)
           SLIDE(2)=SV(1,2)-EV(1,2)
           SLIDE(3)=SV(1,3)-EV(1,3)
        END IF
        MAGS=DSQRT(SLIDE(1)**2+SLIDE(2)**2+SLIDE(3)**2)
        AX=SLIDE(1)/MAGS
        AY=SLIDE(2)/MAGS
        AZ=SLIDE(3)/MAGS
        BX=EV(2,1)-SV(2,1)
        BY=EV(2,2)-SV(2,2)
        BZ=EV(2,3)-SV(2,3)
        CX=SV(2,1)-SV(1,1)
        CY=SV(2,2)-SV(1,2)
        CZ=SV(2,3)-SV(1,3)
        NX=AY*BZ-AZ*BY
        NY=AZ*BX-BZ*AX
        NZ=AX*BY-AY*BX
        MAGN=DSQRT(NX*NX+NY*NY+NZ*NZ)
        NX=NX/MAGN
        NY=NY/MAGN
        NZ=NZ/MAGN
        IF(ABS(NZ).GE.ABS(NX).AND.ABS(NZ).GE.ABS(NY))THEN
             IF(ABS(AX).GT.ABS(AY))THEN
                P=AY*BX-AX*BY
                Q=AY*NX-AX*NY
                R=AX*CY-AY*CX
```

```
          S=AZ*BX-AX*BZ
          T=AZ*NX-AX*NZ
          U=AX*CZ-AZ*CX
          LAM3=(P*U-S*R)/(P*T-S*Q)
          LAM2=(R-LAM3*Q)/P
          LAM1=(CX+BX*LAM2+NX*LAM3)/AX
     ELSE
          P=AX*BY-AY*BX
          Q=AX*NY-AY*NX
          R=AY*CX-AX*CY
          S=AZ*BY-AY*BZ
          T=AZ*NY-AY*NZ
          U=AY*CZ-AZ*CY
          LAM3=(P*U-S*R)/(P*T-S*Q)
          LAM2=(R-LAM3*Q)/P
          LAM1=(CY+BY*LAM2+NY*LAM3)/AY
     END IF
ELSE IF(ABS(NY).GE.ABS(NX))THEN
     IF(ABS(AX).GT.ABS(AZ))THEN
          P=AZ*BX-AX*BZ
          Q=AZ*NX-AX*NZ
          R=AX*CZ-AZ*CX
          S=AY*BX-AX*BY
          T=AY*NX-AX*NY
          U=AX*CY-AY*CX
          LAM3=(P*U-S*R)/(P*T-S*Q)
          LAM2=(R-LAM3*Q)/P
          LAM1=(CX+BX*LAM2+NX*LAM3)/AX
     ELSE
          P=AX*BZ-AZ*BX
          Q=AX*NZ-AZ*NX
          R=AZ*CX-AX*CZ
          S=AY*BZ-AZ*BY
          T=AY*NZ-AZ*NY
          U=AZ*CY-AY*CZ
          LAM3=(P*U-S*R)/(P*T-S*Q)
          LAM2=(R-LAM3*Q)/P
          LAM1=(CZ+BZ*LAM2+NZ*LAM3)/AZ
     END IF
ELSE
     IF(ABS(AZ).GT.ABS(AY))THEN
          P=AY*BZ-AZ*BY
          Q=AY*NZ-AZ*NY
          R=AZ*CY-AY*CZ
          S=AX*BZ-AZ*BX
          T=AX*NZ-AZ*NX
          U=AZ*CX-AX*CZ
          LAM3=(P*U-S*R)/(P*T-S*Q)
          LAM2=(R-LAM3*Q)/P
          LAM1=(CZ+BZ*LAM2+NZ*LAM3)/AZ
     ELSE
          P=AZ*BY-AY*BZ
          Q=AZ*NY-AY*NZ
          R=AY*CZ-AZ*CY
          S=AX*BY-AY*BX
          T=AX*NY-AY*NX
          U=AY*CX-AX*CY
          LAM3=(P*U-S*R)/(P*T-S*Q)
          LAM2=(R-LAM3*Q)/P
```

```
            LAM1=(CY+BY*LAM2+NY*LAM3)/AY
         END IF
      END IF
      MAGB=DSQRT(BX*BX+BY*BY+BZ*BZ)
      PHI=MAGN/MAGB
      NORM=DABS(LAM3)
      SHIFT=LAM1+B*DCOS(DASIN(NORM/B))/DSIN(PHI)
*  NOW PROJECT THE SHIFTED END ONTO VECTOR B  (MAR 25 NOTES)
      POINT2(1)=POINT(1)+AX*SHIFT
      POINT2(2)=POINT(2)+AY*SHIFT
      POINT2(3)=POINT(3)+AZ*SHIFT
      CX=POINT2(1)-SV(2,1)
      CY=POINT2(2)-SV(2,2)
      CZ=POINT2(3)-SV(2,3)
      DOT=(BX*CX+BY*CY+BZ*CZ)/MAGB
      IF(DOT.GT.MAGB)THEN
         CX=EV(2,1)-POINT(1)
         CY=EV(2,2)-POINT(2)
         CZ=EV(2,3)-POINT(3)
         DOT=CX*AX+CY*AY+CZ*AZ
         NX=CX-DOT*AX
         NY=CY-DOT*AY
         NZ=CZ-DOT*AZ
         NORM=DSQRT(NX*NX+NY*NY+NZ*NZ)
         SHIFT=DOT+B*DCOS(DASIN(NORM/B))
      ELSE IF(DOT.LT.0.D0)THEN
         CX=SV(2,1)-POINT(1)
         CY=SV(2,2)-POINT(2)
         CX=SV(2,3)-POINT(3)
         DOT=CX*AX+CY*AY+CZ*AZ
         NX=CX-DOT*AX
         NY=CY-DOT*AY
         NZ=CZ-DOT*AZ
         NORM=DSQRT(NX*NX+NY*NY+NZ*NZ)
         SHIFT=DOT+B*DCOS(DASIN(NORM/B))
      END IF
      RETURN
      END

      SUBROUTINE MOVE2(PDIR,SV,EV,SHIFT,B)
* REPOSITION A JOINT PAST A SPHERICAL ELEMENT
      LOGICAL PDIR
      REAL *8 SV(2,3),EV(2,3),SHIFT,B,POINT(3),SLIDE(3),MAGS
      REAL *8 AX,AY,AZ,CX,CY,CZ
      REAL *8 NX,NY,NZ,DOT,NORM
      IF(PDIR)THEN
         POINT(1)=SV(1,1)
         POINT(2)=SV(1,2)
         POINT(3)=SV(1,3)
         SLIDE(1)=EV(1,1)-SV(1,1)
         SLIDE(2)=EV(1,2)-SV(1,2)
         SLIDE(3)=EV(1,3)-SV(1,3)
      ELSE
         POINT(1)=EV(1,1)
         POINT(2)=EV(1,2)
         POINT(3)=EV(1,3)
         SLIDE(1)=SV(1,1)-EV(1,1)
         SLIDE(2)=SV(1,2)-EV(1,2)
         SLIDE(3)=SV(1,3)-EV(1,3)
```

```
      END IF
      MAGS=DSQRT(SLIDE(1)**2+SLIDE(2)**2+SLIDE(3)**2)
      CX=SV(2,1)-POINT(1)
      CY=SV(2,2)-POINT(2)
      CZ=SV(2,3)-POINT(3)
      DOT=CX*SLIDE(1)+CY*SLIDE(2)+CZ*SLIDE(3)
      AX=DOT*SLIDE(1)
      AY=DOT*SLIDE(2)
      AZ=DOT*SLIDE(3)
      NX=CX-AX
      NY=CY-AY
      NZ=CZ-AZ
      NORM=DSQRT(NX*NX+NY*NY+NZ*NZ)
      SHIFT=B*DCOS(DASIN(NORM/B))+DOT
      RETURN
      END

      SUBROUTINE MOVJNT(JSV,JLEN,JPOS,GLEN,GPOS,JNT,POSX,POSY,
     >          POSZ,TZ,TY,TX,JOINT,JTRK,JCOUNT,OFX,OFY,OFZ,
     >               OFTZ,OFTY,OFTX,IP,RO,RI,RL,RS,NIN,NOUT,OFMAX,CON)
* THE COMMAND ROUTINE FOR ORDERING JOINTS INTO A PRIORITY FOR MOVEMENT
* WHEN AN INTERFERENCE EXISTS.  ROUTINES FOR REPOSITIONING JOINTS ARE
* CALLED FROM HERE.

      REAL *8 JLEN(20,2),JPOS(20,2),GLEN(20),GPOS(20)
      REAL *8 JSV(20,2,3),JNT(20,2,3),TJSV(20,2,3),TJNT(20,2,3)
      REAL *8 POSX(20,360),POSY(20,360),POSZ(20,360),
     >TZ(20,360),TY(20,360),TX(20,360),RO,RI,RL,RS,PHI,U(3)
      REAL *8 OFX(20),OFY(20),OFZ(20),OFTZ(20),OFTY(20),OFTX(20),OFMAX
      INTEGER JOINT(20,2),JCOUNT,IP,NIN,NOUT,TLINK2,CON(20,2)
      CHARACTER *1 JTRK(20),RESP

* VARIABLES UNIQUE TO THIS ROUTINE
** B - THE MAGNITUDE OF THE BOUND AROUND JOINTS FOR INTERFERENCE
**     CHECKING
** K - THE NUMBER OF RELOCATABLE JOINTS (NON-SPHERICS)
** KU - THE NUMBER OF NON-RELOCATABLE JOINTS (SPHERICS)
** SORT(20) - PERMUTATION MATRIX (1 THRU K CAN BE SHUFFLED)
** FIRST - A LOGICAL INDICATING WHETHER THIS IS THE FIRST ORDERING OF
**         THE SORT MATRIX
** PFLIP - LOGICAL USED TO DETERMINE IF THE MIRROR IMAGE OF THE
**         CURRENT PERMUTATION HAS BEEN EVALUATED
** PARL - LOGICAL FOR TESTING PARALLELISM
** GRND - LOGICAL FOR TESTING GROUND ATTACHMENT
** CASE1 - LOGICAL FOR TESTING LINE-LINE COLLISION (LINE-SPHERE OTHER)
** PDIR - LOGICAL FOR TESTING IF THE CURRENT MODE OF JOINT MOVEMENT IS
**        POSITIVE RELATIVE TO THE JOINTS Z COORDINATE
** OVER - LOGICAL WHICH TESTS IF A LINK IS TOO LONG
** TOF(3) - TEMPORARY VARIABLE FOR STORING THE POSITION OFFSET OF A
**          JOINT IN THE LINK COORDINATE SYSTEM
** TTOF(3) - TEMPORARY VARIABLE FOR STORING THE ROTATION OFFSET OF A
**           JOINT IN THE LINK COORDINATE SYSTEM
** TPOS(3) - TEMPORARY VARIABLE FOR STORING THE POSITION OF A LINK
** TTH(3) - TEMPORARY VARIABLE FOR STORING THE ROTATIONS OF A LINK
** TJPOS - TEMPORARY VARIABLE FOR STORING THE JOINT POSITION ALONG IT'S
**         Z AXIS
** TJLEN - TEMPORARY VARIABLE FOR STORING THE JOINT LENGTH
** TSV(3) - TEMPORARY VARIABLE FOR STORING THE START OF A JOINT
** TEV(3) - TEMPORARY VARIABLE FOR STORING THE END OF A JOINT
```

```
** SV(2,3)
** EV(2,3)
** TEMPOS(20) - USED TO STORE THE ACCUMULATIONS OF JOINT SHIFTS
** Q - VARIABLE USED IN PERMUTATIONS
** L - VARIABLE USED IN PERMUTATIONS
** TLNK1 - THE FIRST LINK
** TLNK2 - THE SECOND LINK
** J - THE POSITION
** J1 - PERMUTATION MATRIX INDEX
** J2 - THE FIRST JOINT
** J3 - THE SECOND JOINT
** JR - USED TO INDEX TEMPOS FOR RESETTING
** STORE - USED IN MIRRORING A PERMUTATION
** SHIFT - USED TO RECORD THE CURRENT SHIFT TO AVOID COLLISION
** SDIST - USED TO STORE THE DISTANCE BETWEEN SPHERIC JOINTS
** TOTLEN - USED TO SUM THE TOTAL LENGTH OF ELEMENTS BETWEEN JOINT
** JLENMX - THE MAXIMUM VALUE ALLOWED FOR TOTLEN
** DOT
      REAL *8 SV(2,3),EV(2,3),DOT,B
      REAL *8 TOF(3),TTOF(3),TPOS(3),TTH(3),TJPOS,TJLEN,TSV(3),TEV(3)
      REAL *8 TEMPOS(20),SHIFT,SDIST,TOTLEN,JLENMX
      INTEGER K,KU,SORT(20),Q(20),L,TLNK1,TLNK2,TLNKN,J2,J3,JR,STORE
      INTEGER FLNUM
      LOGICAL FIRST,PFLIP,GRND,CASE1,PARL,PDIR,HIT,OVER
      JLENMX=30.DO*OFMAX
      FIRST=.FALSE.
      PFLIP=.FALSE.
      OVER=.FALSE.
      K=0
      KU=0
      DO 10 I=1,JCOUNT
         TEMPOS(I)=0.DO
         IF(JTRK(I).NE.'S')THEN
            K=K+1
            SORT(K)=I
         ELSE
            SORT(JCOUNT-KU)=I
            KU=KU+1
         END IF
10    CONTINUE
      IF(K.GT.12)THEN
         K=12
      END IF
      NFACT=1
      IF(K.GT.1)THEN
         DO 20 I=2,K
            NFACT=NFACT*I
20       CONTINUE
      END IF
*     FOR ALL ALLOWABLE PERMUTATIONS (UNTIL SUCCESS OR DEFINITE FAILURE)
      DO 30 I=1,NFACT/2
         IF(I.GT.1)THEN
            CALL PERMS(K,SORT,FIRST,L,Q)
         END IF
33       CONTINUE
*         INVESTIGATE ALL POSITIONS

         DO 40 J1=1,JCOUNT-1
            J2=SORT(J1)
```

```
            PDIR=.FALSE.
36          CONTINUE
            PDIR=.NOT.PDIR
37          CONTINUE
*         TEST ALL JOINTS
            DO 50 J=1,IP
*            CHECK ALL JOINTS FOR INTERFERENCE
* FIND THE CONNECTIVITY OF JOINT 1
               GRND=.FALSE.
               TLNK1=JOINT(J2,1)
               IF     (CON(JOINT(J2,1),1).EQ.J2)THEN
                  TOF(1)=0
                  TOF(2)=0
                  TOF(3)=0
                  TTOF(1)=0
                  TTOF(2)=0
                  TTOF(3)=0
                  TPOS(1)=POSX(TLNK1,J)
                  TPOS(2)=POSY(TLNK1,J)
                  TPOS(3)=POSZ(TLNK1,J)
                  TTH(1)=TX(TLNK1,J)
                  TTH(2)=TY(TLNK1,J)
                  TTH(3)=TZ(TLNK1,J)
                  TJPOS=JPOS(TLNK1,1)+TEMPOS(J2)
                  TJLEN=JLEN(TLNK1,1)
               ELSE IF(CON(JOINT(J2,1),2).EQ.J2)THEN
                  TOF(1)=OFX(TLNK1)
                  TOF(2)=OFY(TLNK1)
                  TOF(3)=OFZ(TLNK1)
                  TTOF(1)=OFTX(TLNK1)
                  TTOF(2)=OFTY(TLNK1)
                  TTOF(3)=OFTZ(TLNK1)
                  TPOS(1)=POSX(TLNK1,J)
                  TPOS(2)=POSY(TLNK1,J)
                  TPOS(3)=POSZ(TLNK1,J)
                  TTH(1)=TX(TLNK1,J)
                  TTH(2)=TY(TLNK1,J)
                  TTH(3)=TZ(TLNK1,J)
*           THE SUBSCRIPTS IN CON, JPOS, AND JLEN MUST MATCH (1-7-90)
                  TJPOS=JPOS(TLNK1,2)+TEMPOS(J2)
                  TJLEN=JLEN(TLNK1,2)
               ELSE IF(CON(JOINT(J2,2),1).EQ.J2)THEN
* IF DOWN HERE, THE INTERNAL IS GROUND
                  GRND=.TRUE.
                  TLNK1=JOINT(J2,2)
                  TOF(1)=0
                  TOF(2)=0
                  TOF(3)=0
                  TTOF(1)=0
                  TTOF(2)=0
                  TTOF(3)=0
                  TPOS(1)=POSX(TLNK1,1)
                  TPOS(2)=POSY(TLNK1,1)
                  TPOS(3)=POSZ(TLNK1,1)
                  TTH(1)=TX(TLNK1,1)
                  TTH(2)=TY(TLNK1,1)
                  TTH(3)=TZ(TLNK1,1)
                  TJPOS=GPOS(J2)+TEMPOS(J2)
                  TJLEN=GLEN(J2)
```

```
ELSE IF(CON(JOINT(J2,2),2).EQ.J2)THEN
   GRND=.TRUE.
   TLNK1=JOINT(J2,2)
   TOF(1)=OFX(TLNK1)
   TOF(2)=OFY(TLNK1)
   TOF(3)=OFZ(TLNK1)
   TTOF(1)=OFTX(TLNK1)
   TTOF(2)=OFTY(TLNK1)
   TTOF(3)=OFTZ(TLNK1)
   TPOS(1)=POSX(TLNK1,1)
   TPOS(2)=POSY(TLNK1,1)
   TPOS(3)=POSZ(TLNK1,1)
   TTH(1)=TX(TLNK1,1)
   TTH(2)=TY(TLNK1,1)
   TTH(3)=TZ(TLNK1,1)
   TJPOS=GPOS(J2)+TEMPOS(J2)
   TJLEN=GLEN(J2)
ELSE
   PRINT *,'****ERROR**** IN MOVJNT 1'
   STOP
END IF
CALL     GETJVS(TOF,TTOF,TPOS,TTH,TJPOS,TJLEN,TSV,TEV)
SV(1,1)=TSV(1)
SV(1,2)=TSV(2)
SV(1,3)=TSV(3)
EV(1,1)=TEV(1)
EV(1,2)=TEV(2)
EV(1,3)=TEV(3)

DO 60 J3=1,JCOUNT
   IF(J2.EQ.J3)GO TO 60
   TLNK2=JOINT(J3,1)
   IF      (CON(JOINT(J3,1),1).EQ.J3)THEN
      TOF(1)=0
      TOF(2)=0
      TOF(3)=0
      TTOF(1)=0
      TTOF(2)=0
      TTOF(3)=0
      TPOS(1)=POSX(TLNK2,J)
      TPOS(2)=POSY(TLNK2,J)
      TPOS(3)=POSZ(TLNK2,J)
      TTH(1)=TX(TLNK2,J)
      TTH(2)=TY(TLNK2,J)
      TTH(3)=TZ(TLNK2,J)
      TJPOS=JPOS(TLNK2,1)
      TJLEN=JLEN(TLNK2,1)
   ELSE IF(CON(JOINT(J3,1),2).EQ.J3)THEN
      TOF(1)=OFX(TLNK2)
      TOF(2)=OFY(TLNK2)
      TOF(3)=OFZ(TLNK2)
      TTOF(1)=OFTX(TLNK2)
      TTOF(2)=OFTY(TLNK2)
      TTOF(3)=OFTZ(TLNK2)
      TPOS(1)=POSX(TLNK2,J)
      TPOS(2)=POSY(TLNK2,J)
      TPOS(3)=POSZ(TLNK2,J)
      TTH(1)=TX(TLNK2,J)
      TTH(2)=TY(TLNK2,J)
```

```
                    TTH(3)=TZ(TLNK2,J)
*           THE SUBSCRIPTS IN CON, JPOS, AND JLEN MUST MATCH
                    TJPOS=JPOS(TLNK2,2)
                    TJLEN=JLEN(TLNK2,2)
                  ELSE IF(CON(JOINT(J3,2),1).EQ.J3)THEN
* IF DOWN HERE, THE SECOND JOINT IS GROUND
                    GRND=.TRUE.
                    TLNK2=JOINT(J3,2)
                    TOF(1)=0
                    TOF(2)=0
                    TOF(3)=0
                    TTOF(1)=0
                    TTOF(2)=0
                    TTOF(3)=0
                    TPOS(1)=POSX(TLNK2,1)
                    TPOS(2)=POSY(TLNK2,1)
                    TPOS(3)=POSZ(TLNK2,1)
                    TTH(1)=TX(TLNK2,1)
                    TTH(2)=TY(TLNK2,1)
                    TTH(3)=TZ(TLNK2,1)
                    TJPOS=GPOS(J3)
                    TJLEN=GLEN(J3)
                  ELSE IF(CON(JOINT(J3,2),2).EQ.J3)THEN
                    GRND=.TRUE.
                    TLNK2=JOINT(J3,2)
                    TOF(1)=OFX(TLNK2)
                    TOF(2)=OFY(TLNK2)
                    TOF(3)=OFZ(TLNK2)
                    TTOF(1)=OFTX(TLNK2)
                    TTOF(2)=OFTY(TLNK2)
                    TTOF(3)=OFTZ(TLNK2)
                    TPOS(1)=POSX(TLNK2,1)
                    TPOS(2)=POSY(TLNK2,1)
                    TPOS(3)=POSZ(TLNK2,1)
                    TTH(1)=TX(TLNK2,1)
                    TTH(2)=TY(TLNK2,1)
                    TTH(3)=TZ(TLNK2,1)
                    TJPOS=GPOS(J3)
                    TJLEN=GLEN(J3)
                  ELSE
                    PRINT *,'****ERROR**** IN MOVJNT 2'
                    STOP
                  END IF
                  CALL     GETJVS(TOF,TTOF,TPOS,TTH,TJPOS,TJLEN,TSV,TEV)
                  SV(2,1)=TSV(1)
                  SV(2,2)=TSV(2)
                  SV(2,3)=TSV(3)
                  EV(2,1)=TEV(1)
                  EV(2,2)=TEV(2)
                  EV(2,3)=TEV(3)
                  IF(JTRK(J2).EQ.'S')THEN
                    SDIST=DSQRT((SV(2,1)-SV(1,1))**2+
     >                     (SV(2,2)-SV(1,2))**2+(SV(2,3)-SV(1,3))**2)
                    IF(SDIST.LT.RO)THEN
                      PRINT *,'SPHERICS ARE HITTING. CONTINUE?(Y/N)'
                      READ(NIN,70)RESP
70                    FORMAT(A1)
                      IF(RESP.EQ.'Y')THEN
                        GO TO 999
```

```
                        ELSE
                           STOP
                        END IF
                     ELSE
                        HIT=.FALSE.
                     END IF
                  ELSE IF(JTRK(J3).EQ.'S')THEN
                     B=RO+RS
                     CASE1=.FALSE.
                     CALL CHKTWO(SV,EV,B,HIT)
                  ELSE
                     B=2.D0*RO
                     CASE1=.TRUE.
                     CALL CHKONE(SV,EV,B,HIT,PARL)
                  END IF
                  IF(HIT)THEN
* THE CONDITION FOR SUCCESS IS TO NOT HAVE INTEFERENCE AT ANY POSITION
* THUS, IF A JOINT IS MOVED ALL POSITIONS MUST BE CHECKED AGAIN
                     IF(CASE1)THEN
                        IF(PARL) THEN
                           DOT=(EV(1,1)-SV(1,1))*(EV(2,1)-SV(2,1))+
     >                          (EV(1,2)-SV(1,2))*(EV(2,2)-SV(2,2))+
     >                          (EV(1,3)-SV(1,3))*(EV(2,3)-SV(2,3))
                           IF(DOT.GT.0)THEN
                              SV(2,1)=EV(2,1)
                           END IF
                           CALL MOVE2(PDIR,SV,EV,SHIFT,B)
                        ELSE
                           CALL MOVE1(PDIR,SV,EV,SHIFT,B)
                        END IF
                     ELSE
                        CALL MOVE2(PDIR,SV,EV,SHIFT,B)
                     END IF
                     IF(PDIR)THEN
                        TEMPOS(J2)=TEMPOS(J2)+SHIFT
                     ELSE
                        TEMPOS(J2)=TEMPOS(J2)-SHIFT
                     END IF
                  END IF
60             CONTINUE
50           CONTINUE
**             PUT THE JOINT SEPARATION TEST HERE
**  ONCE A JOINT HAS MOVED, THE LENGTH OF TWO LINKS ARE AFFECTED
             IF(HIT)THEN
                IF(JOINT(J2,1).NE.0)THEN
                   TLNKN=JOINT(J2,1)
                   TOTLEN=DSQRT(OFX(TLNKN)**2+OFY(TLNKN)**2+
     >                OFZ(TLNKN)**2)+JLEN(TLNKN,1)+JLEN(TLNKN,2)+
     >                DABS(JPOS(TLNKN,1)+JPOS(TLNKN,2)+TEMPOS(J2))
                   IF(TOTLEN.GT.JLENMX)THEN
                      OVER=.TRUE.
                   END IF
                END IF
                IF(JOINT(J2,2).NE.0.AND..NOT.OVER)THEN
                   TLNKN=JOINT(J2,2)
                   TOTLEN=DSQRT(OFX(TLNKN)**2+OFY(TLNKN)**2+
     >                OFZ(TLNKN)**2)+JLEN(TLNKN,1)+JLEN(TLNKN,2)+
     >                DABS(JPOS(TLNKN,1)+JPOS(TLNKN,2)+TEMPOS(J2))
                   IF(TOTLEN.GT.JLENMX)THEN
```

```
                  OVER=.TRUE.
               END IF
            END IF
            IF(OVER)THEN
               OVER=.FALSE.
*           RESET THE OFFSETS TO THE DEFAULTS AND TRY IN THE OTHER
*           DIRECTION (WHATEVER THOSE ARE)
               DO 55 JR=1,JCOUNT
                  TEMPOS(JR)=0
55             CONTINUE
               IF(PDIR)THEN
                  TEMPOS(J2)=0
                  GO TO 36
               ELSE
                  DO 57 JR=1,JCOUNT
                     TEMPOS(JR)=0
57                CONTINUE
                  GO TO 80
               END IF
            ELSE
*            EVEN IF THE SEPARATION CONSTRAINT IS NOT VIOLATED,
*            ALL JOINTS MUST STILL BE CHECKED; BUT PDIR IS NOT RESET
               GO TO 37
            END IF
         END IF
40    CONTINUE
*  THE ONLY CONDITION FOR ANOTHER PERMUTATION IS VIOLATION OF A LINK
*  LENGTH
78    CONTINUE
      GO TO 999
80    CONTINUE
*      MIRROR THE PERMUTATION
      FLNUM=INT(K/2)
      DO 777 IFLIP=1,FLNUM
         STORE=SORT(IFLIP)
         SORT(IFLIP)=SORT(K+1-IFLIP)
         SORT(K+1-IFLIP)=STORE
777   CONTINUE
      PFLIP=.NOT.PFLIP
*      OPERATE ON THE MIRRORED PERMUTATION IF IT HAS NOT BEEN DONE.
      IF(PFLIP)THEN
         GO TO 33
      END IF
30  CONTINUE
999 CONTINUE
    DO 1000 I=1,JCOUNT
**    SET ALL OF THE OFFSETS EFFECTIVE
      TLNK1=JOINT(I,1)
      TLNK2=JOINT(I,2)
      IF(TLNK1.EQ.0)THEN
         GPOS(I)=GPOS(I)+TEMPOS(I)
      ELSE
         IF(CON(TLNK1,1).EQ.I)THEN
            JPOS(TLNK1,1)=JPOS(TLNK1,1)+TEMPOS(I)
         ELSE IF(CON(TLNK1,2).EQ.I)THEN
            JPOS(TLNK1,2)=JPOS(TLNK1,2)+TEMPOS(I)
         ELSE
            PRINT *,'******ERROR IN SETTING JPOS1+TEMPOS*****'
         END IF
```

```
         END IF
         IF(TLNK2.EQ.0)THEN
            GPOS(I)=GPOS(I)+TEMPOS(I)
         ELSE
            IF(CON(TLNK2,1).EQ.I)THEN
               JPOS(TLNK2,1)=JPOS(TLNK2,1)+TEMPOS(I)
            ELSE IF(CON(TLNK2,2).EQ.I)THEN
               JPOS(TLNK2,2)=JPOS(TLNK2,2)+TEMPOS(I)
            ELSE
               PRINT *,'******ERROR IN SETTING JPOS2+TEMPOS*****'
            END IF
         END IF
1000  CONTINUE
      RETURN
      END

      SUBROUTINE MOVLNK(JSV,JLEN,JPOS,GLEN,GPOS,JNT,POSX,POSY,
     >                  POSZ,TZ,TY,TX,JOINT,JTRK,JCOUNT,OFX,OFY,OFZ,
     >                  OFTZ,OFTY,OFTX,IP,RO,RI,RL,RS,NIN,NOUT,OFMAX,
     >                  VECT,TVECT,LNK,JCON,CON)
* THE COMMAND ROUTINE FOR ORDERING AND MOVING INTERFERING LINKS.  ALL
* ROUTINES FOR MOVING LINKS ARE CALLED FROM HERE.
      REAL*8 VECT(20,3,3),TVECT(20,3,3),JSV(20,2,3),JNT(20,2,3),RJ(3)
      REAL*8 RLAM1(20),RLAM2(20),RLAM3(20),SOL(20,3),PHI,PI
      REAL*8 UX(3),UY(3),UZ(3),RJ2(3),B
      REAL*8 JLEN(20,2),TJLEN,JPOS(20,2),TJPOS,GLEN(20),GPOS(20)
      INTEGER IVCNT(20),LNK,ICOUNT,NIN,NOUT,CON(20,2)
      INTEGER INDSPH(20,2),NSPHER,SPHTRK(20),JOINT(20,2),JCOUNT,GCOUNT
      CHARACTER *1 JTRK(20)
      REAL *8 OFX(20),OFY(20),OFZ(20),OFTZ(20),OFTY(20),OFTX(20)
      REAL *8 POSX(20,360),POSY(20,360),POSZ(20,360),
     >TZ(20,360),TY(20,360),TX(20,360),RO,RI,RL,RS
      CHARACTER*2 JCON(20,2)
* FIRST MAKE SURE THAT NONE OF THE LINK ELEMENTS HIT ANY JOINTS
* START THE PERMUTATIONS
      REAL *8 SV(2,3),EV(2,3),DOT,TOF(3),TTOF(3),TPOS(3),TTH(3),TSV(3)
      REAL *8 TEV(3),TEMPV(3,3)
      INTEGER K,KU,SORT(20),Q(20),L,LP,L1,FLNUM,OPT(20,3),MVCNT(20,3)
      INTEGER IV,IV2,V1,V2,TLNKS,STORE
* OPT IS THE POSITIONING OPTION IDENTIFIER
      LOGICAL FIRST,PFLIP,GRND,CASE1,PDIR,SPHERE,GOOD,HIT
      FIRST=.FALSE.
      PFLIP=.FALSE.
      UX(1)=1.D0
      UX(2)=0.D0
      UX(3)=0.D0
      UY(1)=0.D0
      UY(2)=1.D0
      UY(3)=0.D0
      UZ(1)=0.D0
      UZ(2)=0.D0
      UZ(3)=1.D0
      K=0
      KU=0
      DO 10 I=1,LNK
         IF(.NOT.(JCON(I,1)(1:1).EQ.'S'.AND.JCON(I,2)(1:1).EQ.'S'))THEN
            K=K+1
            SORT(K)=I
         ELSE
```

```
            SORT(LNK-KU)=I
            KU=KU+1
         END IF
10    CONTINUE
      IF(K.GT.12)THEN
         K=12
      END IF
      NFACT=1
      IF(K.GT.1)THEN
         DO 20 I=2,K
            NFACT=NFACT*I
20       CONTINUE
      END IF
*     FOR ALL ALLOWABLE PERMUTATIONS (UNTIL SUCCESS OR DEFINITE FAILURE)
      DO 30 I=1,NFACT/2
         IF(I.GT.1)THEN
            CALL PERMS(K,SORT,FIRST,LP,Q)
         END IF
33       CONTINUE
         DO 34 I1=1,LNK
            OPT(I1,1)=1
            OPT(I1,2)=1
            OPT(I1,3)=1
            MVCNT(I1,1)=0
            MVCNT(I1,2)=0
            MVCNT(I1,3)=0
34       CONTINUE
*         INVESTIGATE ALL LINK ELEMENTS
35       CONTINUE
         DO 40 L=1,LNK
          L1=SORT(L)
          TEMPV(1,1)=TVECT(L1,1,1)
          TEMPV(1,2)=TVECT(L1,1,2)
          TEMPV(1,3)=TVECT(L1,1,3)
          TEMPV(2,1)=TVECT(L1,2,1)
          TEMPV(2,2)=TVECT(L1,2,2)
          TEMPV(2,3)=TVECT(L1,2,3)
          TEMPV(3,1)=TVECT(L1,3,1)
          TEMPV(3,2)=TVECT(L1,3,2)
          TEMPV(3,3)=TVECT(L1,3,3)
36        CONTINUE
          DO 50 IV=1,3
            IF(IV.EQ.1)THEN
               V1=1
            ELSE IF(IV.EQ.2)THEN
               TEMPV(1,1)=TVECT(L1,1,1)
               TEMPV(1,2)=TVECT(L1,1,2)
               TEMPV(1,3)=TVECT(L1,1,3)
               TEMPV(2,1)=TVECT(L1,2,1)
               TEMPV(2,2)=TVECT(L1,2,2)
               TEMPV(2,3)=TVECT(L1,2,3)
               V1=3
            ELSE
               TEMPV(2,1)=TVECT(L1,2,1)
               TEMPV(2,2)=TVECT(L1,2,2)
               TEMPV(2,3)=TVECT(L1,2,3)
               TEMPV(3,1)=TVECT(L1,3,1)
               TEMPV(3,2)=TVECT(L1,3,2)
               TEMPV(3,3)=TVECT(L1,3,3)
```

```
               V1=2
             END IF
* FIRST ALL LINKS MUST CLEAR ALL JOINTS TO GUARANTEE THAT NO LINKS
* INTERSECT AT A JOINT CENTER CAUSING A SINGULARITY FOR ROTATION.
            DO 60 J=1,IP
             IF(V1.EQ.1)THEN
                SV(1,1)=0.D0
                SV(1,2)=0.D0
                SV(1,3)=JPOS(L1,1)
             ELSE IF(V1.EQ.2)THEN
                SV(1,1)=TVECT(L1,1,1)
                SV(1,2)=TVECT(L1,1,2)
                SV(1,3)=TVECT(L1,1,3)+JPOS(L1,1)
             ELSE
                SV(1,1)=TVECT(L1,1,1)+TVECT(L1,2,1)-TVECT(L1,3,1)
                SV(1,2)=TVECT(L1,1,2)+TVECT(L1,2,2)-TVECT(L1,3,2)
                SV(1,3)=TVECT(L1,1,3)+JPOS(L1,1)+TVECT(L1,2,3)-
     >                  TVECT(L1,3,3)
             END IF
             IF(JCON(L1,1)(2:2).EQ.'E'.AND.JCON(L1,1)(1:1).NE.'S')THEN
                SV(1,3)=SV(1,3)+RL
             END IF
             EV(1,1)=SV(1,1)+TVECT(L1,V1,1)
             EV(1,2)=SV(1,2)+TVECT(L1,V1,2)
             EV(1,3)=SV(1,3)+TVECT(L1,V1,3)
* THESE ARE THE FIRST LINK VECTORS IN THE LOCAL FRAME.
             DO 70 JS=1,JCOUNT
*             VECTORS EMANATING FROM JOINTS DO NOT INTERFERE WITH THOSE
*             JOINTS, SO CHECK THAT CONDITION.
              IF(V1.EQ.1.AND.CON(L1,1).EQ.JS)GO TO 70
              IF(V1.EQ.3.AND.CON(L1,2).EQ.JS)GO TO 70
              TLNKS=JOINT(JS,1)
              IF(TLNKS.EQ.0)THEN
               TLNKS=JOINT(JS,2)
              END IF
              IF     (JOINT(JS,1).NE.0.AND.CON(TLNKS,1).EQ.JS)THEN
               TOF(1)=0
               TOF(2)=0
               TOF(3)=0
               TTOF(1)=0
               TTOF(2)=0
               TTOF(3)=0
               TPOS(1)=POSX(TLNKS,J)
               TPOS(2)=POSY(TLNKS,J)
               TPOS(3)=POSZ(TLNKS,J)
               TTH(1)=TX(TLNKS,J)
               TTH(2)=TY(TLNKS,J)
               TTH(3)=TZ(TLNKS,J)
               TJPOS=JPOS(TLNKS,1)
               TJLEN=JLEN(TLNKS,1)
              ELSE IF(JOINT(JS,1).NE.0.AND.CON(TLNKS,2).EQ.JS)THEN
               TOF(1)=OFX(TLNKS)
               TOF(2)=OFY(TLNKS)
               TOF(3)=OFZ(TLNKS)
               TTOF(1)=OFTX(TLNKS)
               TTOF(2)=OFTY(TLNKS)
               TTOF(3)=OFTZ(TLNKS)
               TPOS(1)=POSX(TLNKS,J)
               TPOS(2)=POSY(TLNKS,J)
```

```
              TPOS(3)=POSZ(TLNKS,J)
              TTH(1)=TX(TLNKS,J)
              TTH(2)=TY(TLNKS,J)
              TTH(3)=TZ(TLNKS,J)
*           NOTE FOR CON, JPOS, AND JLEN; BOTH SUBSCRIPTS MUST MATCH
              TJPOS=JPOS(TLNKS,2)
              TJLEN=JLEN(TLNKS,2)
             ELSE IF(JOINT(JS,2).NE.0.AND.CON(TLNKS,1).EQ.JS)THEN
* IF DOWN HERE, THE JOINT IS GROUND
              GRND=.TRUE.
              TLNKS=JOINT(JS,2)
              TOF(1)=0
              TOF(2)=0
              TOF(3)=0
              TTOF(1)=0
              TTOF(2)=0
              TTOF(3)=0
              TPOS(1)=POSX(TLNKS,1)
              TPOS(2)=POSY(TLNKS,1)
              TPOS(3)=POSZ(TLNKS,1)
              TTH(1)=TX(TLNKS,1)
              TTH(2)=TY(TLNKS,1)
              TTH(3)=TZ(TLNKS,1)
              TJPOS=GPOS(JS)
              TJLEN=GLEN(JS)
             ELSE IF(JOINT(JS,2).NE.0.AND.CON(TLNKS,2).EQ.JS)THEN
              GRND=.TRUE.
              TLNKS=JOINT(JS,2)
              TOF(1)=OFX(TLNKS)
              TOF(2)=OFY(TLNKS)
              TOF(3)=OFZ(TLNKS)
              TTOF(1)=OFTX(TLNKS)
              TTOF(2)=OFTY(TLNKS)
              TTOF(3)=OFTZ(TLNKS)
              TPOS(1)=POSX(TLNKS,1)
              TPOS(2)=POSY(TLNKS,1)
              TPOS(3)=POSZ(TLNKS,1)
              TTH(1)=TX(TLNKS,1)
              TTH(2)=TY(TLNKS,1)
              TTH(3)=TZ(TLNKS,1)
              TJPOS=GPOS(JS)
              TJLEN=GLEN(JS)
             ELSE
              PRINT *,'****JOINT ERROR**** IN MOVLNK 2'
              STOP
             END IF
             CALL     GETJVS(TOF,TTOF,TPOS,TTH,TJPOS,TJLEN,TSV,TEV)
* NOW TRANSFORM VECTOR 2 BACK INTO THE VECTOR 1 LOCAL FRAME OF
* REFERENCE SO THAT INTERFERENCE CHECKING CAN START.
             TSV(1)=TSV(1)-POSX(L1,J)
             TSV(2)=TSV(2)-POSY(L1,J)
             TSV(3)=TSV(3)-POSZ(L1,J)
             TEV(1)=TEV(1)-POSX(L1,J)
             TEV(2)=TEV(2)-POSY(L1,J)
             TEV(3)=TEV(3)-POSZ(L1,J)
             PHI=-TX(L1,J)
             CALL ROTATE(UX,PHI,TSV)
             CALL ROTATE(UX,PHI,TEV)
             PHI=-TY(L1,J)
```

```
                CALL ROTATE(UY,PHI,TSV)
                CALL ROTATE(UY,PHI,TEV)
                PHI=-TZ(L1,J)
                CALL ROTATE(UZ,PHI,TSV)
                CALL ROTATE(UZ,PHI,TEV)
                SV(2,1)=TSV(1)
                SV(2,2)=TSV(2)
                SV(2,3)=TSV(3)
                EV(2,1)=TEV(1)
                EV(2,2)=TEV(2)
                EV(2,3)=TEV(3)
                IF(JTRK(JS).EQ.'S')THEN
                 SPHERE=.TRUE.
                 B=RS+RI
                 CALL CHKTWO(SV,EV,B,HIT)
                ELSE
                 SPHERE=.FALSE.
                 B=RO+RI
                 CALL CHKONE(SV,EV,B,HIT)
                END IF
                IF(HIT)THEN
                  MVCNT(L1,V1)=MVCNT(L1,V1)+1
*                *IF THE MOVEMENT COUNT IS TOO HIGH OR A NUB IS CONNECTED
*                *TO A SPHERE THEN THIS TRY FAILED.
                 IF(MVCNT(L1,V1).GE.IP.OR.(V1.EQ.1.AND.JCON(L1,1)(1:1)
     >           .EQ.'S').OR.(V1.EQ.3.AND.JCON(L1,2)(1:1).EQ.'S'))THEN
*                 ANOTHER PERMUTATION OR OPTION IS NEEDED SO RESET TVECT
*                 TO VECT AND TRY AGAIN.
                  OPT(L1,V1)=OPT(L1,V1)+1
                  MVCNT(L1,V1)=0
                  IF(V1.EQ.1.OR.V1.EQ.3)THEN
                   IF(OPT(L1,V1).GT.2)THEN
                    GO TO 80
                   END IF
                  ELSE IF(OPT(L1,V1).GT.3)THEN
                   GO TO 80
                  END IF
*                   RESET ONLY THIS TVECT AND TRY ANOTHER OPTION ON THE
*                   TVECT THAT FAILED (DO NOT RESET ANY PREVIOUS LINKS AT
*                   THIS POINT
                  CALL RESET(TVECT,TEMPV,L1)
                  GO TO 36
                 END IF
                 LNK1=L1
                 JPOSI=J
                 IF(SPHERE)THEN
                    CALL AVSPH(V1,OPT,SV,EV,B,TVECT,LNK1,JPOSI,JNT,GOOD)
                    IF(.NOT.GOOD)THEN
                     OPT(L1,V1)=OPT(L1,V1)+1
                     MVCNT(L1,V1)=0
                     IF(OPT(L1,V1).GT.3)THEN
                      GO TO 80
                     END IF
*                    RESET ONLY THIS TVECT AND TRY ANOTHER OPTION ON THE
*                    TVECT THAT FAILED (DO NOT RESET ANY PREVIOUS LINKS AT
*                    THIS POINT
                     CALL RESET(TVECT,TEMPV,L1)
                    END IF
                 ELSE
```

```
*                AVOID THE ROUND ENDED CYLINDER (THE ONLY CASE FOR
*                LINKS)
                 CALL AVCYL(V1,OPT,SV,EV,B,TVECT,LNK1,JPOSI,JNT,GOOD)
                 tvs1(1)=vect(l1,1,1)+vect(l1,2,1)-vect(l1,3,1)
                 tvs1(2)=vect(l1,1,2)+vect(l1,2,2)-vect(l1,3,2)
                 tvs1(3)=vect(l1,1,3)+vect(l1,2,3)-vect(l1,3,3)
                 tvs2(1)=tvect(l1,1,1)+tvect(l1,2,1)-tvect(l1,3,1)
                 tvs2(2)=tvect(l1,1,2)+tvect(l1,2,2)-tvect(l1,3,2)
                 tvs2(3)=tvect(l1,1,3)+tvect(l1,2,3)-tvect(l1,3,3)

                 IF(.NOT.GOOD)THEN
                  OPT(L1,V1)=OPT(L1,V1)+1
                  MVCNT(L1,V1)=0
                  IF(OPT(L1,V1).GT.3)THEN
                   GO TO 80
                  END IF
*                 RESET ONLY THIS TVECT AND TRY ANOTHER OPTION ON THE
*                 TVECT THAT FAILED (DO NOT RESET ANY PREVIOUS LINKS AT
*                 THIS POINT
                  CALL RESET(TVECT,TEMPV,L1)
                 END IF
               END IF
               GO TO 36
             END IF
70           CONTINUE
60          CONTINUE
50         CONTINUE
40        CONTINUE
*          INVESTIGATE ALL LINK ELEMENTS
          DO 42 L=1,LNK
           L1=SORT(L)
           TEMPV(1,1)=TVECT(L1,1,1)
           TEMPV(1,2)=TVECT(L1,1,2)
           TEMPV(1,3)=TVECT(L1,1,3)
           TEMPV(2,1)=TVECT(L1,2,1)
           TEMPV(2,2)=TVECT(L1,2,2)
           TEMPV(2,3)=TVECT(L1,2,3)
           TEMPV(3,1)=TVECT(L1,3,1)
           TEMPV(3,2)=TVECT(L1,3,2)
           TEMPV(3,3)=TVECT(L1,3,3)
37         CONTINUE
           DO 52 IV=1,3
             IF(IV.EQ.1)THEN
                V1=1
             ELSE IF(IV.EQ.2)THEN
                TEMPV(1,1)=TVECT(L1,1,1)
                TEMPV(1,2)=TVECT(L1,1,2)
                TEMPV(1,3)=TVECT(L1,1,3)
                TEMPV(2,1)=TVECT(L1,2,1)
                TEMPV(2,2)=TVECT(L1,2,2)
                TEMPV(2,3)=TVECT(L1,2,3)
                V1=3
             ELSE
                TEMPV(2,1)=TVECT(L1,2,1)
                TEMPV(2,2)=TVECT(L1,2,2)
                TEMPV(2,3)=TVECT(L1,2,3)
                TEMPV(3,1)=TVECT(L1,3,1)
                TEMPV(3,2)=TVECT(L1,3,2)
                TEMPV(3,3)=TVECT(L1,3,3)
```

```
               V1=2
             END IF
* FIRST ALL LINKS MUST CLEAR ALL JOINTS TO GUARANTEE THAT NO LINKS
* INTERSECT AT A JOINT CENTER CAUSING A SINGULARITY FOR ROTATION.
            DO 62 J=1,IP
             IF(V1.EQ.1)THEN
                SV(1,1)=0.D0
                SV(1,2)=0.D0
                SV(1,3)=JPOS(L1,1)
             ELSE IF(V1.EQ.2)THEN
                SV(1,1)=TVECT(L1,1,1)
                SV(1,2)=TVECT(L1,1,2)
                SV(1,3)=TVECT(L1,1,3)+JPOS(L1,1)
             ELSE
                SV(1,1)=TVECT(L1,1,1)+TVECT(L1,2,1)-TVECT(L1,3,1)
                SV(1,2)=TVECT(L1,1,2)+TVECT(L1,2,2)-TVECT(L1,3,2)
                SV(1,3)=TVECT(L1,1,3)+JPOS(L1,1)+TVECT(L1,2,3)-
     >                  TVECT(L1,3,3)
             END IF
             IF(JCON(L1,1)(2:2).EQ.'E'.AND.JCON(L1,1)(1:1).NE.'S')THEN
                SV(1,3)=SV(1,3)+RL
             END IF
             EV(1,1)=SV(1,1)+TVECT(L1,V1,1)
             EV(1,2)=SV(1,2)+TVECT(L1,V1,2)
             EV(1,3)=SV(1,3)+TVECT(L1,V1,3)
* THESE ARE THE FIRST LINK VECTORS IN THE LOCAL FRAME.
             DO 420 L2=1,LNK
*             NOW CHECK THE LINK ELEMENTS
              IF(L2.EQ.L1) GO TO 420
              DO 520 IV2=1,3
                IF(IV2.EQ.1)THEN
                   V2=1
                ELSE IF(IV2.EQ.2)THEN
                   V2=3
                ELSE
                   V2=2
                END IF
* VECTORS EMANATING FROM A COMMON JOINT DO NOT INTERFERE
* SO GO TO 420 IF THAT CONDITION IS FOUND.  9-30-89
                IF(V1.EQ.1.AND.V2.EQ.1)THEN
                  IF(CON(L1,1).NE.0.AND.CON(L1,1).EQ.CON(L2,1))THEN
                     GO TO 520
                  END IF
                ELSE IF(V1.EQ.1.AND.V2.EQ.3)THEN
                  IF(CON(L1,1).NE.0.AND.CON(L1,1).EQ.CON(L2,2))THEN
                     GO TO 520
                  END IF
                ELSE IF(V1.EQ.3.AND.V2.EQ.1)THEN
                  IF(CON(L1,2).NE.0.AND.CON(L1,2).EQ.CON(L2,1))THEN
                     GO TO 520
                  END IF
                ELSE IF(V1.EQ.3.AND.V2.EQ.3)THEN
                  IF(CON(L1,2).NE.0.AND.CON(L1,2).EQ.CON(L2,2))THEN
                     GO TO 520
                  END IF
                END IF
* FIRST ALL LINKS MUST CLEAR ALL JOINTS TO GUARANTEE THAT NO LINKS
* INTERSECT AT A JOINT CENTER CAUSING A SINGULARITY FOR ROTATION.
                IF(V2.EQ.1)THEN
```

```
                  TSV(1)=0.DO
                  TSV(2)=0.DO
                  TSV(3)=JPOS(L2,1)
               ELSE IF(V2.EQ.2)THEN
                  TSV(1)=TVECT(L2,1,1)
                  TSV(2)=TVECT(L2,1,2)
                  TSV(3)=TVECT(L2,1,3)+JPOS(L2,1)
               ELSE
                  TSV(1)=TVECT(L2,1,1)+TVECT(L2,2,1)-TVECT(L2,3,1)
                  TSV(2)=TVECT(L2,1,2)+TVECT(L2,2,2)-TVECT(L2,3,2)
                  TSV(3)=TVECT(L2,1,3)+JPOS(L2,1)+TVECT(L2,2,3)-
     >                    TVECT(L2,3,3)
               END IF
               IF(JCON(L2,1)(2:2).EQ.'E'.AND.JCON(L2,1)(1:1).NE.'S')
     >         THEN
                  TSV(3)=TSV(3)+RL
               END IF
               TEV(1)=TSV(1)+TVECT(L2,V2,1)
               TEV(2)=TSV(2)+TVECT(L2,V2,2)
               TEV(3)=TSV(3)+TVECT(L2,V2,3)
* THESE ARE THE FIRST LINK VECTORS IN THE LOCAL FRAME.
               PHI=TZ(L2,J)
               CALL ROTATE(UZ,PHI,TSV)
               CALL ROTATE(UZ,PHI,TEV)
               PHI=TY(L2,J)
               CALL ROTATE(UY,PHI,TSV)
               CALL ROTATE(UY,PHI,TEV)
               PHI=TX(L2,J)
               CALL ROTATE(UX,PHI,TSV)
               CALL ROTATE(UX,PHI,TEV)
               TSV(1)=TSV(1)+POSX(L2,J)-POSX(L1,J)
               TSV(2)=TSV(2)+POSY(L2,J)-POSY(L1,J)
               TSV(3)=TSV(3)+POSZ(L2,J)-POSZ(L1,J)
               TEV(1)=TEV(1)+POSX(L2,J)-POSX(L1,J)
               TEV(2)=TEV(2)+POSY(L2,J)-POSY(L1,J)
               TEV(3)=TEV(3)+POSZ(L2,J)-POSZ(L1,J)
* NOW TRANSFORM VECTOR 2 BACK INTO THE VECTOR 1 LOCAL FRAME OF
* REFERENCE SO THAT INTERFERENCE CHECKING CAN START.
               PHI=-TX(L1,J)
               CALL ROTATE(UX,PHI,TSV)
               CALL ROTATE(UX,PHI,TEV)
               PHI=-TY(L1,J)
               CALL ROTATE(UY,PHI,TSV)
               CALL ROTATE(UY,PHI,TEV)
               PHI=-TZ(L1,J)
               CALL ROTATE(UZ,PHI,TSV)
               CALL ROTATE(UZ,PHI,TEV)
               SV(2,1)=TSV(1)
               SV(2,2)=TSV(2)
               SV(2,3)=TSV(3)
               EV(2,1)=TEV(1)
               EV(2,2)=TEV(2)
               EV(2,3)=TEV(3)
               B=RI+RI
               CALL CHKONE(SV,EV,B,HIT)
               IF(HIT)THEN
                 MVCNT(L1,V1)=MVCNT(L1,V1)+1
*             *IF THE MOVEMENT COUNT IS TOO HIGH OR A NUB IS CONNECTED
*             *TO A SPHERE THEN THIS TRY FAILED.
```

```
                IF(MVCNT(L1,V1).GE.IP.OR.(V1.EQ.1.AND.JCON(L1,1)(1:1)
     >           .EQ.'S').OR.(V1.EQ.3.AND.JCON(L1,2)(1:1).EQ.'S'))THEN
*               ANOTHER PERMUTATION OR OPTION IS NEEDED SO RESET TVECT
*               TO VECT AND TRY AGAIN.
                 OPT(L1,V1)=OPT(L1,V1)+1
                 MVCNT(L1,V1)=0
                 IF(V1.EQ.1.OR.V1.EQ.3)THEN
                  IF(OPT(L1,V1).GT.2)THEN
                   GO TO 80
                  END IF
                 ELSE IF(OPT(L1,V1).GT.3)THEN
                  GO TO 80
                 END IF
*                  RESET ONLY THIS TVECT AND TRY ANOTHER OPTION ON THE
*                  TVECT THAT FAILED (DO NOT RESET ANY PREVIOUS LINKS AT
*                  THIS POINT
                 CALL RESET(TVECT,TEMPV,L1)
                 GO TO 37
                END IF
                LNK1=L1
                JPOSI=J
*               AVOID THE ROUND ENDED CYLINDER (THE ONLY CASE FOR
*               LINKS)
                CALL AVCYL(V1,OPT,SV,EV,B,TVECT,LNK1,JPOSI,JNT,GOOD)
                IF(.NOT.GOOD)THEN
                 OPT(L1,V1)=OPT(L1,V1)+1
                 MVCNT(L1,V1)=0
                 IF(OPT(L1,V1).GT.3)THEN
                  GO TO 80
                 END IF
*                RESET ONLY THIS TVECT AND TRY ANOTHER OPTION ON THE
*                TVECT THAT FAILED (DO NOT RESET ANY PREVIOUS LINKS AT
*                THIS POINT
                 CALL RESET(TVECT,TEMPV,L1)
                 GO TO 37
                END IF
*               IF THE AVOIDANCE ATTEMPT WAS GOOD. THE JOINTS NEED TO
*               BE CHECKED AGAIN. SO GO BACK TO 35.
                 GO TO 35
               END IF
520           CONTINUE
420         CONTINUE
62         CONTINUE
52        CONTINUE
42       CONTINUE
* THEN MAKE SURE THAT NONE OF THE LINKS HIT EACH OTHER (ALL LINKS AND
* JOINTS MUST BE RECHECKED FOR COLLISION EVERY TIME THE ELEMENT IS
* MOVED) SEQUENCE IS VECT1,VECT3,VECT2
        GO TO 100
80      CONTINUE
        DO 85 IR=1,LNK
*        RESET ALL TVECTS
         DO 86 IVR=1,3
          TVECT(IR,IVR,1)=VECT(IR,IVR,1)
          TVECT(IR,IVR,2)=VECT(IR,IVR,2)
          TVECT(IR,IVR,3)=VECT(IR,IVR,3)
86       CONTINUE
85      CONTINUE
*        MIRROR THE PERMUTATION
```

```
         FLNUM=INT(K/2)
         DO 777 IFLIP=1,FLNUM
            STORE=SORT(IFLIP)
            SORT(IFLIP)=SORT(K+1-IFLIP)
            SORT(K+1-IFLIP)=STORE
777      CONTINUE
         PFLIP=.NOT.PFLIP
*         OPERATE ON THE MIRRORED PERMUTATION IF IT HAS NOT BEEN DONE.
         IF(PFLIP)THEN
            GO TO 33
         END IF
         IF(.NOT.PFLIP.AND.NFACT/2.EQ.I)THEN
            WRITE(NOUT,*)'COLLISION AVOIDANCE UNSUCCESSFUL'
         END IF
30     CONTINUE
100    CONTINUE
       RETURN
       END

       SUBROUTINE NUBS(JNT,OFX,OFY,OFZ,JPOS,OFTZ,OFTY,OFTX,VECT,TVECT
      >                ,LNK,RO,RS,JTRK,CON)
* ATTACHMENT ELEMENTS ARE CREATED IN THIS ROUTINE
       REAL *8 JNT(20,2,3),OFX(20),OFY(20),OFZ(20),RO,RS
       REAL *8 OFTZ(20),OFTY(20),OFTX(20),VECT(20,3,3),TVECT(20,3,3)
       REAL *8 PSIGN,MAG,AX,AY,AZ,CX,CY,CZ,DX,DY,DZ,U(3),RJ(3),PHI
       REAL *8 MAGC,DOT,POINT(3)
       INTEGER LNK,J,K,M
       INTEGER CON(20,2)
       CHARACTER *1 JTRK(20)
       DO 10 I=1,LNK
          DO 20 J=1,2
             MAG=DSQRT(OFX(I)**2+OFY(I)**2+OFZ(I)**2)
             IF(J.EQ.2)THEN
                PSIGN=-1.D0
                K=3
             ELSE
                PSIGN=1.D0
                K=1
             END IF
             POINT(1)=PSIGN*OFX(I)/MAG
             POINT(2)=PSIGN*OFY(I)/MAG
             POINT(3)=PSIGN*OFZ(I)/MAG
             IF(JTRK(CON(I,J)).EQ.'S')THEN
                VECT(I,K,1)=2.D0*RS*POINT(1)
                VECT(I,K,2)=2.D0*RS*POINT(2)
                VECT(I,K,3)=2.D0*RS*POINT(3)
             ELSE
                DOT=JNT(I,J,1)*POINT(1)+JNT(I,J,2)*POINT(2)+
      >             JNT(I,J,3)*POINT(1)
                IF(DOT.EQ.1.D0)THEN
                   IF(J.EQ.1)THEN
                      VECT(I,K,1)=2.D0*RO
                      VECT(I,K,2)=0.D0
                      VECT(I,K,3)=0.D0
                   ELSE
                      RJ(1)=1.D0
                      RJ(2)=0.D0
                      RJ(3)=0.D0
                      U(1)=0.D0
```

```
                  U(2)=1.D0
                  U(3)=0.D0
                  PHI=OFTY(I)
                  CALL ROTATE(U,PHI,RJ)
                  U(1)=1.D0
                  U(2)=0.D0
                  U(3)=0.D0
                  PHI=OFTX(I)
                  CALL ROTATE(U,PHI,RJ)
                  VECT(I,K,1)=RJ(1)*2.D0*RO
                  VECT(I,K,2)=RJ(2)*2.D0*RO
                  VECT(I,K,3)=RJ(3)*2.D0*RO
               END IF
            ELSE
               AX=JNT(I,J,1)
               AY=JNT(I,J,2)
               AZ=JNT(I,J,3)
               CX=AY*POINT(3)-AZ*POINT(2)
               CY=AZ*POINT(1)-AX*POINT(3)
               CZ=AX*POINT(2)-AY*POINT(1)
               MAGC=DSQRT(CX*CX+CY*CY+CZ*CZ)
               CX=CX/MAGC
               CY=CY/MAGC
               CZ=CZ/MAGC
               DX=CY*AZ-CZ*AY
               DY=CZ*AX-CX*AZ
               DZ=CX*AY-CY*AX
               VECT(I,K,1)=DX*2.D0*RO
               VECT(I,K,2)=DY*2.D0*RO
               VECT(I,K,3)=DZ*2.D0*RO
            END IF
         END IF
         DO 30 M=1,3
            TVECT(I,K,M)=VECT(I,K,M)
30       CONTINUE
20    CONTINUE
10  CONTINUE
    RETURN
    END

    SUBROUTINE PERML(N,X,Q,FIRST)
* THIS ROUTINE FINDS THE NEXT PERMUTATION OF A SET OF N
* NUMBERS
    INTEGER X(20),Q(20),N1,N,M,L,J,K
    LOGICAL FIRST
    N1=N-1
    IF(.NOT.FIRST)THEN
       FIRST=.TRUE.
       DO 10 M=1,N1
          Q(M)=N
10     CONTINUE
    END IF
    IF(Q(N1).EQ.N)THEN
       Q(N1)=N1
       L=X(N)
       X(N)=X(N1)
       X(N1)=L
    ELSE
       DO 20 J=1,N1
```

```
            K=N-J
            IF(Q(K).NE.K) GO TO 5
            Q(K)=N
20        CONTINUE
          FIRST=.FALSE.
          K=1
          GO TO 6
5         M=Q(K)
          L=X(M)
          X(M)=X(K)
          X(K)=L
          Q(K)=M-1
          K=K+1
6         M=N
7         L=X(M)
          X(M)=X(K)
          X(K)=L
          M=M-1
          K=K+1
          IF(K.LT.M) GO TO 7
       END IF
22     FORMAT(3X,4I2)
       RETURN
       END

       SUBROUTINE PERMS(N,X,FIRST,L,Q)
* THIS ROUTINE CALLS AND MANAGES THE PERMUTATION ROUTINE
       INTEGER X(20),Q(20),L,N
       LOGICAL FIRST
       IF(.NOT.FIRST)THEN
          L=1
          DO 10 I=3,N
             L=L*I
10        CONTINUE
       END IF
       L=L-1
       IF(L.EQ.0)THEN
          FIRST=.FALSE.
       ELSE
          CALL PERML(N,X,Q,FIRST)
       END IF
       RETURN
       END

       SUBROUTINE POSCUP(VECT,IVCNT,OFTX,OFTY,OFTZ,TX,TY,TZ,NSPHER,INDSPH
      >                  ,SPHTRK,IP,CON,RS)
* ONLY SPHERICAL JOINT DATA IS PROCESSED HERE.  THE NUMBER OF SPHERICAL
* JOINTS IS STORED IN NSPHER WHILE THE PAIRINGS ARE STORED IN INDSPH.
* A NEW "NUB" VECTOR IS CREATED FOR THE LINK WITH THE CUP PORTION OF THE
* SPHERIC JOINT. OFT*(N) VARIABLES ARE USED TO DIRECT THIS NUB VECTOR.
*
* LIST OF VARIABLES NEEDED FOR CUP POSITIONING:
*     VECT(20,3,3),IVCNT(20),OFTX(20),OFTY(20),OFTZ(20),TX(20,360),
*     TY(20,360),TZ(20,360),NSPHER,INDSPH(20,2),SPHTRK(20),IP,CON(20,2)

       INTEGER NSPHER,INDSPH(20,2),IVCNT(20),SPHTRK(20),IP,LINK1,LINK2
      >        ,JD,IP1,IP2,I,J,K,K1,CON(20,2)
       REAL *8 VECT(20,3,3),OFTX(20),OFTY(20),OFTZ(20),TX(20,360)
      >     ,TY(20,360),TZ(20,360),PHI,RJ(3),U(3),UNIT(20,360,3),SPAN(3),
```

```
     >       BISECT(3),NORM(3),TDIST,DIST,BMAG,PNORM,RNMAG,IPANG,TPANG,
     >       INANG,TNANG,ORNT(20,3),SPMAG,PROJ(360),RS,MAGV,AVANG,pi
      LOGICAL ALLG1,NEG,ORIENT

* THE BALLS ARE POSITIONED RELATIVE TO THE CUP (REFERENCE) UNLESS THE
* BALL IS ATTACHED TO AN SS LINK THEN THE BALL IS REFERENCE.
* THE PROCESS BEGINS BY CREATING A UNIT POSITION VECTOR ATTACHED TO THE
* REFERENCE'S LOCAL COORDINATE SYSTEM. THIS VECTOR UNDERGOES ROTATIONAL
* TRANSFORMATIONS RELATIVE TO THE THREE AXIES OF THE LOCAL COORDINATE
* SYSTEM OF THE REFERENCE.  FIRST THE CUP'S TRANSFORMATIONS ARE APPLIED
* THEN THE BALL'S INVERSE TRANSFORMATIONS ARE APPLIED
* (MAKING A TOTAL OF SIX).  THE INVERSE IMAGE OF THE CUP'S FEED VECTOR
* IS MOVED AROUND IN THE BALL'S FRAME OF REFERENCE.  A COLLECTION OF UNIT
* VECTORS IS DEVELOPED BY PROJECTING THE THE TIP OF THE INITIAL VECTOR
* ONTO A GAUSSIAN SPHERE. THIS IS DONE FOR EACH SPHERIC JOINT.  THE AXIS
* OF A CONE CONTAINING ALL OF THE POINTS IS DETERMINED AND THE AXIS WITH
* ALL OF THE POINTS ARE ROTATED UNTIL THE AXIS IS COLINEAR WITH THE FEED
* VECTOR OF THE CUP.
      DO 10 I=1,NSPHER
         LINK2=INDSPH(I,1)
         LINK1=INDSPH(I,2)
         IF(CON(LINK1,1).EQ.SPHTRK(I))THEN
            KL=1
         ELSE IF(CON(LINK1,2).EQ.SPHTRK(I))THEN
            KL=3
         ELSE
            PRINT *,'SPHERICS ATTACHED TO GROUND NOT IMPLEMENTED YET 1'
            STOP
         END IF
         MAGV=DSQRT(VECT(LINK1,KL,1)**2+VECT(LINK1,KL,2)**2
     >             +VECT(LINK1,KL,3)**2)
         DO 20 J=1,IP
            RJ(1)=-VECT(LINK1,KL,1)/MAGV
            RJ(2)=-VECT(LINK1,KL,2)/MAGV
            RJ(3)=-VECT(LINK1,KL,3)/MAGV
            PHI=TZ(LINK1,J)
            U(1)=0.
            U(2)=0.
            U(3)=1.
            CALL ROTATE(U,PHI,RJ)
            PHI=TY(LINK1,J)
            U(1)=0.
            U(2)=1.
            U(3)=0.
            CALL ROTATE(U,PHI,RJ)
            PHI=TX(LINK1,J)
            U(1)=1.
            U(2)=0.
            U(3)=0.
            CALL ROTATE(U,PHI,RJ)
            PHI=-TX(LINK2,J)
            U(1)=1.
            U(2)=0.
            U(3)=0.
            CALL ROTATE(U,PHI,RJ)
            PHI=-TY(LINK2,J)
            U(1)=0.
            U(2)=1.
            U(3)=0.
```

```
            CALL ROTATE(U,PHI,RJ)
            PHI=-TZ(LINK2,J)
            U(1)=0.
            U(2)=0.
            U(3)=1.
            CALL ROTATE(U,PHI,RJ)
            UNIT(I,J,1)=RJ(1)
            UNIT(I,J,2)=RJ(2)
            UNIT(I,J,3)=RJ(3)
20        CONTINUE
10     CONTINUE
* THE GAUSSIAN SPHERE POINTS HAVE BEEN CREATED.  NOW FIND THE TWO WITH
* MAXIMUM SEPARATION.

* LOOK AT EACH SPHERIC JOINT
       DO 30 I=1,NSPHER
* FIND THE TWO GAUSSIAN POINTS WITH GREATEST SEPARATION
          DIST=0.0
          DO 40 J=1,IP-1
             DO 50 K1=J,IP
                TDIST=DSQRT((UNIT(I,J,1)-UNIT(I,K1,1))**2+
     >                      (UNIT(I,J,2)-UNIT(I,K1,2))**2+
     >                      (UNIT(I,J,3)-UNIT(I,K1,3))**2)
                IF(TDIST.GT.DIST)THEN
                   DIST=TDIST
                   IP1=J
                   IP2=K1
                END IF
50           CONTINUE
40        CONTINUE

* THE TWO POINTS WITH GREATEST SEPARATION HAVE BEEN FOUND
* NOW CHECK THE SEPARATION DISTANCE.  IF SEPARATION IS 2 THEN THE CUP IS
* NOT ORIENTABLE.
          ORIENT=.TRUE.
          IF(DIST.LT.2.D0)THEN
             SPAN(1)=UNIT(I,IP2,1)-UNIT(I,IP1,1)
             SPAN(2)=UNIT(I,IP2,2)-UNIT(I,IP1,2)
             SPAN(3)=UNIT(I,IP2,3)-UNIT(I,IP1,3)
             BISECT(1)=SPAN(1)/2+UNIT(I,IP1,1)
             BISECT(2)=SPAN(2)/2+UNIT(I,IP1,2)
             BISECT(3)=SPAN(3)/2+UNIT(I,IP1,3)
             BMAG=DSQRT(BISECT(1)**2+BISECT(2)**2+BISECT(3)**2)
             ALLG1=.TRUE.
             NEG=.FALSE.

* CHECK TO SEE IF THE OTHER POINTS HAVE A POSITIVE PROJECTION ON THE
* BISECTING VECTOR. (STILL IN DO 30 LOOP)
             DO 60 JD=1,IP
                PROJ(JD)=(BISECT(1)*UNIT(I,JD,1)+BISECT(2)*UNIT(I,JD,2)
     >                   +BISECT(3)*UNIT(I,JD,3))/BMAG
                IF(PROJ(JD).LT.BMAG)THEN
                   ALLG1=.FALSE.
                ELSE IF(PROJ(JD).LT.0)THEN
                   NEG=.TRUE.
                END IF
60           CONTINUE
```

```
*  TEST FOR THE METHOD OF DETERMINING THE ORIENTATION VECTOR
            IF(ALLG1)THEN
*            USE BISECT AS THE ORIENTATION VECTOR FOR THIS JOINT *****
               ORNT(I,1)=BISECT(1)/BMAG
               ORNT(I,2)=BISECT(2)/BMAG
               ORNT(I,3)=BISECT(3)/BMAG
            ELSE IF(.NOT.NEG)THEN
*            USE THE LOWEST POINTS ON EITHER SIDE OF SPAN TO DETERMINE
*            AVERAGE VECTORS FOR ORIENTATION  ************
               NORM(1)=SPAN(2)*BISECT(3)-SPAN(3)*BISECT(2)
               NORM(2)=SPAN(3)*BISECT(1)-SPAN(1)*BISECT(3)
               NORM(3)=SPAN(1)*BISECT(2)-SPAN(2)*BISECT(1)
*               PROJECT THE POINTS ONTO NORM THEN USE PROJ AND NORM TO
*               TO FIND THE POINTS WITH GREATEST ANGLE TO BISECT IN
*               THIS PLANE (BOTH THE POSITIVE AND NEGATIVE PROJECTIONS
*               ONTO NORM).
               RNMAG=DSQRT(NORM(1)**2+NORM(2)**2+NORM(3)**2)
               IPANG=0.0
               INANG=0.0
               DO 70 JD=1,IP
                  PNORM=(NORM(1)*UNIT(I,JD,1)+NORM(2)*UNIT(I,JD,2)
     >                    +NORM(3)*UNIT(I,JD,3))/RNMAG
                  IF(PNORM.GT.0)THEN
                     IF(PNORM.EQ.0.D0.AND.PROJ(JD).EQ.0.D0)THEN
                        STOP 'DATAN2 ERROR 5'
                     END IF
                     TPANG=DATAN2(PNORM,PROJ(JD))
                     IF(TPANG.GT.IPANG)THEN
                        IPANG=TPANG
                     END IF
                  ELSE IF(PNORM.LT.0)THEN
                     IF(PNORM.EQ.0.D0.AND.PROJ(JD).EQ.0.D0)THEN
                        STOP 'DATAN2 ERROR 6'
                     END IF
                     TNANG=DABS(DATAN2(PNORM,PROJ(JD)))
                     IF(TNANG.GT.INANG)THEN
                        INANG=TNANG
                     END IF
                  END IF
70             CONTINUE
*               NOW THAT THE MAX ANGLES OF DEVIATION, BOTH POS AND NEG,
*               HAVE BEEN FOUND, AVERAGE THEM AND MAKE THIS THE ORIENT
*               VECTOR.
               AVANG=(IPANG-INANG)/2.
               SPMAG=DSQRT(SPAN(1)**2+SPAN(2)**2+SPAN(3)**2)
               SPAN(1)=SPAN(1)/SPMAG
               SPAN(2)=SPAN(2)/SPMAG
               SPAN(3)=SPAN(3)/SPMAG
               CALL ROTATE(SPAN,AVANG,BISECT)
               ORNT(I,1)=BISECT(1)/BMAG
               ORNT(I,2)=BISECT(2)/BMAG
               ORNT(I,3)=BISECT(3)/BMAG
            ELSE
               ORIENT=.FALSE.
            END IF
         ELSE
            ORIENT=.FALSE.
         END IF
            IF(.NOT.ORIENT)THEN
```

```
*             IF THE CUP IS NOT ORIENTABLE BY THE OTHER RULES, THEN
*             ORIENT IT ALONG THE FIRST GAUSS POINT.
                ORNT(I,1)=UNIT(I,1,1)
                ORNT(I,2)=UNIT(I,1,2)
                ORNT(I,3)=UNIT(I,1,3)
              END IF
              LINK2=INDSPH(I,1)
              LINK1=INDSPH(I,2)
              IF(CON(LINK2,1).EQ.SPHTRK(I))THEN
                 KL=1
              ELSE IF(CON(LINK2,2).EQ.SPHTRK(I))THEN
                 KL=3
              ELSE
                 PRINT *,'SPHERICS ATTACHED TO GROUND NOT IMPLEMENTED YET'
                 STOP
              END IF
              MAGV=DSQRT(VECT(LINK2,KL,1)**2+VECT(LINK2,KL,1)**2
     >                   +VECT(LINK2,KL,1)**2)
              VECT(LINK2,KL,1)=ORNT(I,1)*MAGV
              VECT(LINK2,KL,2)=ORNT(I,2)*MAGV
              VECT(LINK2,KL,3)=ORNT(I,3)*MAGV
30     CONTINUE
*       THE RELATIVE ORIENTATION IS KNOWN.
       RETURN
       END

       SUBROUTINE RESET(TVECT,TEMPV,L1)
* RESET ELEMENT VECTORS TO THEIR INITIAL DESCRIPTION
* AFTER A RESHAPING OPTION HAS FAILED
       REAL*8 TVECT(20,3,3),TEMPV(3,3)
       INTEGER L1
       TVECT(L1,1,1)=TEMPV(1,1)
       TVECT(L1,1,2)=TEMPV(1,2)
       TVECT(L1,1,3)=TEMPV(1,3)
       TVECT(L1,2,1)=TEMPV(2,1)
       TVECT(L1,2,2)=TEMPV(2,2)
       TVECT(L1,2,3)=TEMPV(2,3)
       TVECT(L1,3,1)=TEMPV(3,1)
       TVECT(L1,3,2)=TEMPV(3,2)
       TVECT(L1,3,3)=TEMPV(3,3)
       RETURN
       END

       SUBROUTINE RODIR(RB,SA,A,SB,B,U,ROTP,ROTN)
       REAL*8 SA(3),A(3),U(3),SB(3),B(3),D(3),U1(3),U2(3),V(3),RB
       REAL*8 DVBB,DMAG,PHI,THT,ROTP,ROTN,AXDDU,PTHET1,PTHET2,DMN,DMJ
       REAL*8 DTHET1,DTHET2,UMAG,BMAG,GAMMA,TTHT,TTHT1,TTHT2,PI,AMN,AMJ
       REAL*8 AMAG
* FIND THE ROTATION NECESSARY TO REPOSITION AN INFINITE CYLINDER
* INTERFERING WITH ANOTHER INFINITE CYLINDER USING AN ELLIPSOIDAL
* PROJECTION

* FIND THE VECTOR FROM SA TO SB
       DO 10 I=1,3
          V(I)=SB(I)-SA(I)
 10    CONTINUE

* FIND THE SHORTEST DISTANCE FROM THE LINE B TO THE POINT SA WHICH IS
* THE ORIGIN OF U (THE ROTATION VECTOR FOR A).
```

```
      DVBB=(B(1)*V(1)+B(2)*V(2)+B(3)*V(3))/(B(1)*B(1)+B(2)*B(2)+B(3)*
     >B(3))
      DO 20 I=1,3
         D(I)=-DVBB*B(I)+V(I)
 20   CONTINUE

* FIND THE ROTATION FROM D IN ELLIPSOIDAL PROJECTION
      DMAG=DSQRT(D(1)*D(1)+D(2)*D(2)+D(3)*D(3))
      PHI=DASIN(RB/DMAG)

* FIND THE MAJOR AXIS ORIENTATION FOR THE ELLIPSE FORMED BY LOOKING
* DOWN THE VECTOR B AT A CIRCLE DESCRIBED BY U AND SA.
      U1(1)=U(2)*B(3)-U(3)*B(2)
      U1(2)=U(3)*B(1)-U(1)*B(3)
      U1(3)=U(1)*B(2)-U(2)*B(1)

* IF THE MAJOR AXIS MAGNITUDE IS ZERO, THEN THE CIRCLE AND ELLIPSE
* COORDINATES ARE THE SAME; AND A SLIGHTLY DIFFERENT METHOD IS USED.
      IF(U1(1).EQ.0.0.AND.U1(2).EQ.0.0.AND.U1(3).EQ.0.0)THEN
         AMAG=DSQRT(A(1)*A(1)+A(2)*A(2)+A(3)*A(3))
         THT=DACOS((A(1)*D(1)+A(2)*D(2)+A(3)*D(3))/AMAG/DMAG)
         AXDDU=(A(2)*D(3)-A(3)*D(2))*U(1)+(A(3)*D(1)-A(1)*D(3))*U(2)+
     >   (A(1)*D(2)-A(2)*D(1))*U(3)
         IF(AXDDU.GE.0.)THEN
              ROTP=THT+PHI
              ROTN=THT-PHI
         ELSE
              ROTP=-THT+PHI
              ROTN=-THT-PHI
         END IF

      ELSE
* FIND THE MINOR AXIS
         U2(1)=U1(2)*B(3)-U1(3)*B(2)
         U2(2)=U1(3)*B(1)-U1(1)*B(3)
         U2(3)=U1(1)*B(2)-U1(2)*B(1)

* FIND THE ANGLE OF D IN ELLIPSOIDAL COORDINATES BY PROJECTING IT ONTO
* THE MAJOR AND MINOR AXES.
         DMN=(D(1)*U2(1)+D(2)*U2(2)+D(3)*U2(3))/DSQRT(U2(1)*U2(1)+U2(2)*
     >   U2(2)+U2(3)*U2(3))
         DMJ=(D(1)*U1(1)+D(2)*U1(2)+D(3)*U1(3))/DSQRT(U1(1)*U1(1)+U1(2)*
     >   U1(2)+U1(3)*U1(3))
         IF(DMN.EQ.0.DO.AND.DMJ.EQ.0.DO)THEN
            STOP 'DATAN2 ERROR 7'
         END IF
         PTHET1=DATAN2(DMN,DMJ)

* NOW DO THE SAME WITH VECTOR A TO USE AS A REFERENCE.
         AMN=(A(1)*U2(1)+A(2)*U2(2)+A(3)*U2(3))/DSQRT(U2(1)*U2(1)+U2(2)*
     >   U2(2)+U2(3)*U2(3))
         AMJ=(A(1)*U1(1)+A(2)*U1(2)+A(3)*U1(3))/DSQRT(U1(1)*U1(1)+U1(2)*
     >   U1(2)+U1(3)*U1(3))
         IF(AMN.EQ.0.DO.AND.AMJ.EQ.0.DO)THEN
            STOP 'DATAN2 ERROR 8'
         END IF
         PTHET2=DATAN2(AMN,AMJ)
```

```
* FIND THE ROTATIONS (+ AND -) FROM D TO ELIMINATE INTERFERENCE IN
* THE ELLIPSOIDAL REFERENCE FRAME.
        DTHET1=PTHET1+PHI
        DTHET2=PTHET1-PHI

* FIND THE ANGLE BETWEEN U AND B TO GET READY TO PROJECT THESE ROTATIONS
* BACK ONTO THE CIRCLE FRAME OF REFERENCE.
        UMAG=DSQRT(U(1)*U(1)+U(2)*U(2)+U(3)*U(3))
        BMAG=DSQRT(B(1)*B(1)+B(2)*B(2)+B(3)*B(3))
        GAMMA=DACOS((U(1)*B(1)+U(2)*B(2)+U(3)*B(3))/UMAG/BMAG)

* PROJECT EVERYTHING BACK ONTO THE CIRCLE.
        IF(DSIN(DTHET1)/DCOS(GAMMA).EQ.0.D0.AND.DCOS(DTHET1).EQ.0.D0)
     >  THEN
           STOP 'DATAN2 ERROR 9'
        END IF
        TTHT1=DATAN2(DSIN(DTHET1)/DABS(DCOS(GAMMA)),DCOS(DTHET1))
        IF(DSIN(DTHET2)/DCOS(GAMMA).EQ.0.D0.AND.DCOS(DTHET2).EQ.0.D0)
     >  THEN
           STOP 'DATAN2 ERROR 10'
        END IF
        TTHT2=DATAN2(DSIN(DTHET2)/DABS(DCOS(GAMMA)),DCOS(DTHET2))
        IF(DSIN(PTHET2)/DCOS(GAMMA).EQ.0.D0.AND.DCOS(PTHET2).EQ.0.D0)
     >  THEN
           STOP 'DATAN2 ERROR 11'
        END IF
        TTHT=DATAN2(DSIN(PTHET2)/DABS(DCOS(GAMMA)),DCOS(PTHET2))
        IF(U(1)*B(1)+U(2)*B(2)+U(3)*B(3).LT.0.)THEN
           ROTP=TTHT1-TTHT
           ROTN=TTHT2-TTHT
        ELSE
           ROTN=TTHT-TTHT1
           ROTP=TTHT-TTHT2
        END IF
      ENDIF
      PI=DACOS(-1.D0)
      IF(ROTN.GT.0.D0)THEN
         ROTN=ROTN-2.D0*PI
      END IF
      IF(ROTP.LT.0.D0)THEN
         ROTP=ROTP+2.D0*PI
      END IF
      RETURN
      END

C   SUBROUTINE ROTATE
C   COMPUTES ROTATION MATRIX ELEMENTS IN TERMS OF ANGLE PHI (RADIANS)
C   ABOUT U AND ROTATES VECTOR RJ TO RJ=(RM)*RJ
C   RJ CAN BE A VECTOR OF ANY MAGNITUDE, BUT U MUST BE A UNIT VECTOR
      SUBROUTINE ROTATE(U,PHI,RJ)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION RM(3,3),U(3),RJ(3),TJ(3)
      C=DCOS(PHI)
      S=DSIN(PHI)
      V=1.-C
      RM(1,1)=U(1)*U(1)*V+C
      RM(1,2)=U(1)*U(2)*V-U(3)*S
      RM(1,3)=U(1)*U(3)*V+U(2)*S
      RM(2,1)=U(1)*U(2)*V+U(3)*S
```

```
      RM(2,2)=U(2)*U(2)*V+C
      RM(2,3)=U(2)*U(3)*V-U(1)*S
      RM(3,1)=U(1)*U(3)*V-U(2)*S
      RM(3,2)=U(2)*U(3)*V+U(1)*S
      RM(3,3)=U(3)*U(3)*V+C
      DO 5 I=1,3
    5  TJ(I)=RJ(I)
      DO 10 I=1,3
   10 RJ(I)=RM(I,1)*TJ(1)+RM(I,2)*TJ(2)+RM(I,3)*TJ(3)
      RETURN
      END

      SUBROUTINE SIZE(JSV,JLEN,JPOS,GLEN,GPOS,JNT,POSX,POSY,POSZ,TZ,
     >                TY,TX,JOINT,JTRK,JCOUNT,OFX,OFY,OFZ,
     >                OFTZ,OFTY,OFTX,IP,CON)

* CYLINDRIC AND PRISMATIC JOINTS ARE SIZED HERE

      REAL *8 JSV(20,2,3),JNT(20,2,3)
      REAL *8 JLEN(20,2),JPOS(20,2),GLEN(20),GPOS(20)
      REAL *8 OFX(20),OFY(20),OFZ(20),OFTZ(20),OFTY(20),OFTX(20)
      REAL *8 POSX(20,360),POSY(20,360),POSZ(20,360),
     >TZ(20,360),TY(20,360),TX(20,360),PHI,U(3)
      INTEGER JOINT(20,2),JCOUNT,IP,CON(20,2)
      CHARACTER *1 JTRK(20)
* VARIABLES UNIQUE TO THIS ROUTINE FOLLOW
** POUTER(3)- THE X,Y,Z LOCATION OF THE OUTER JOINT AT THIS INSTANT
** PINNER(3)- THE X,Y,Z LOCATION OF THE INNER JOINT AT THIS INSTANT
** TEMPZ(3)- THE ABSOLUTE DIRECTION OF LOCAL Z AXIS AFTER LINK IS
**           POSITIONED
** INOFF(3)- OFFSET OF AN INNER JOINT CONNECTED TO LINK END 2
** OUTOFF(3)- OFFSET OF AN OUTER JOINT CONNECTED TO LINK END 2
** TEMP(3)- UNIT VECTOR FOR OUTOFF OR INNOFF IN INTERIM CALCULATIONS
** MAGOUT- THE MAGNITUDE OF OUTOFF
** MAGIN- THE MAGNITUDE OF INOFF
** DOT- DOT PRODUCT OF OUTER JOINT OFFSET RELATIVE TO INNER ONTO INNER Z
** TLINKI- LINK CONNECTED TO INNER JOINT
** TLINKO- LINK CONNECTED TO OUTER JOINT
** IEND- LINK END FOR OUTER JOINT WHEN DETERMINING GROUND INNER Z AXIS
** ALL LOGICALS USED TO DETERMINE END OF LINK THAT A JOINT IS CONNECTED
** TO I.E. END1J IS END 1 OF A MOVING INNER JOINT, OENDJG IS END 2 OF A
** GROUNDED OUTER
      REAL *8 PINNER(3),POUTER(3),TEMPZ(3),INOFF(3),OUTOFF(3)
      REAL *8 TEMP(3),MAGOUT,MAGIN,DOT,JPMIN,JPMAX
      INTEGER TLINKI,TLINKO,IEND
      LOGICAL END1J,END2J,END1G,END2G,OEND1J,OEND2J,OEND1G,OEND2G
      DO 10 I=1,JCOUNT
         END1J=.FALSE.
         END2J=.FALSE.
         END1G=.FALSE.
         END2G=.FALSE.
         OEND1J=.FALSE.
         OEND2J=.FALSE.
         OEND1G=.FALSE.
         OEND2G=.FALSE.
         JPMAX=0.D0
         JPMIN=JPMAX
         IF(JTRK(I).EQ.'P'.OR.JTRK(I).EQ.'C')THEN
            TLINKI=JOINT(I,1)
```

```
            TLINKO=JOINT(I,2)
**          FIND THE CONNECTIVITY OF THE INTERNAL LINK
            IF      (CON(TLINKI,1).EQ.I)THEN
               END1J=.TRUE.
            ELSE IF(CON(TLINKI,2).EQ.I)THEN
               END2J=.TRUE.
            ELSE IF(CON(TLINKI,1).EQ.0)THEN
               END1G=.TRUE.
            ELSE IF(CON(TLINKI,2).EQ.0)THEN
               END2G=.TRUE.
            ELSE
               WRITE(6,*)'ERROR INTERNAL JOINT HAS NO GROUND OR LINK'
               WRITE(6,*)'CONNECTIVITY. JOINT NUMBER IS'
               WRITE(6,*)I
               STOP
            END IF
**          FIND THE CONNECTIVITY OF THE EXTERNAL LINK
            IF      (CON(TLINKO,1).EQ.I)THEN
               OEND1J=.TRUE.
            ELSE IF(CON(TLINKO,2).EQ.I)THEN
               OEND2J=.TRUE.
            ELSE IF(CON(TLINKO,1).EQ.0)THEN
               OEND1G=.TRUE.
            ELSE IF(CON(TLINKO,2).EQ.0)THEN
               OEND2G=.TRUE.
            ELSE
               WRITE(6,*)'ERROR EXTERNAL JOINT HAS NO GROUND OR LINK'
               WRITE(6,*)'CONNECTIVITY. JOINT NUMBER IS'
               WRITE(6,*)I
               STOP
            END IF
            IF(OEND1G)THEN
               POUTER(1)=POSX(TLINKO,1)
               POUTER(2)=POSY(TLINKO,1)
               POUTER(3)=POSZ(TLINKO,1)
            ELSE IF(OEND2G)THEN
               OUTOFF(1)=OFX(TLINKO)
               OUTOFF(2)=OFY(TLINKO)
               OUTOFF(3)=OFZ(TLINKO)
               MAGOUT=DSQRT(OUTOFF(1)**2+OUTOFF(2)**2+OUTOFF(3)**2)
               TEMP(1)=OUTOFF(1)/MAGOUT
               TEMP(2)=OUTOFF(2)/MAGOUT
               TEMP(3)=OUTOFF(3)/MAGOUT
               U(1)=0.D0
               U(2)=0.D0
               U(3)=1.D0
               PHI=TZ(TLINKO,1)
               CALL ROTATE(U,PHI,TEMP)
               U(1)=0.D0
               U(2)=1.D0
               U(3)=0.D0
               PHI=TY(TLINKO,1)
               CALL ROTATE(U,PHI,TEMP)
               U(1)=1.D0
               U(2)=0.D0
               U(3)=0.D0
               PHI=TX(TLINKO,1)
               CALL ROTATE(U,PHI,TEMP)
               OUTOFF(1)=TEMP(1)*MAGOUT
```

```
    OUTOFF(2)=TEMP(2)*MAGOUT
    OUTOFF(3)=TEMP(3)*MAGOUT
    POUTER(1)=POSX(TLINKO,1)+OUTOFF(1)
    POUTER(2)=POSY(TLINKO,1)+OUTOFF(2)
    POUTER(3)=POSZ(TLINKO,1)+OUTOFF(3)
END IF
IF(END1G)THEN
    PINNER(1)=POSX(TLINKI,1)
    PINNER(2)=POSY(TLINKI,1)
    PINNER(3)=POSZ(TLINKI,1)
ELSE IF(END2G)THEN
    INOFF(1)=OFX(TLINKI)
    INOFF(2)=OFY(TLINKI)
    INOFF(3)=OFZ(TLINKI)
    MAGIN=DSQRT(INOFF(1)**2+INOFF(2)**2+INOFF(3)**2)
    TEMP(1)=INOFF(1)/MAGIN
    TEMP(2)=INOFF(2)/MAGIN
    TEMP(3)=INOFF(3)/MAGIN
    U(1)=0.D0
    U(2)=0.D0
    U(3)=1.D0
    PHI=TZ(TLINKI,1)
    CALL ROTATE(U,PHI,TEMP)
    U(1)=0.D0
    U(2)=1.D0
    U(3)=0.D0
    PHI=TY(TLINKI,1)
    CALL ROTATE(U,PHI,TEMP)
    U(1)=1.D0
    U(2)=0.D0
    U(3)=0.D0
    PHI=TX(TLINKI,1)
    CALL ROTATE(U,PHI,TEMP)
    INOFF(1)=TEMP(1)*MAGIN
    INOFF(2)=TEMP(2)*MAGIN
    INOFF(3)=TEMP(3)*MAGIN
    PINNER(1)=POSX(TLINKI,1)+INOFF(1)
    PINNER(2)=POSY(TLINKI,1)+INOFF(2)
    PINNER(3)=POSZ(TLINKI,1)+INOFF(3)
END IF
DO 20 J=1,IP
    IF(OEND1J)THEN
      POUTER(1)=POSX(TLINKO,J)
      POUTER(2)=POSY(TLINKO,J)
      POUTER(3)=POSZ(TLINKO,J)
    ELSE IF(OEND2J)THEN
      OUTOFF(1)=OFX(TLINKO)
      OUTOFF(2)=OFY(TLINKO)
      OUTOFF(3)=OFZ(TLINKO)
      MAGOUT=DSQRT(OUTOFF(1)**2+OUTOFF(2)**2+OUTOFF(3)**2)
      TEMP(1)=OUTOFF(1)/MAGOUT
      TEMP(2)=OUTOFF(2)/MAGOUT
      TEMP(3)=OUTOFF(3)/MAGOUT
      U(1)=0.D0
      U(2)=0.D0
      U(3)=1.D0
      PHI=TZ(TLINKO,J)
      CALL ROTATE(U,PHI,TEMP)
      U(1)=0.D0
```

```
      U(2)=1.D0
      U(3)=0.D0
      PHI=TY(TLINKO,J)
      CALL ROTATE(U,PHI,TEMP)
      U(1)=1.D0
      U(2)=0.D0
      U(3)=0.D0
      PHI=TX(TLINKO,J)
      CALL ROTATE(U,PHI,TEMP)
      OUTOFF(1)=TEMP(1)*MAGOUT
      OUTOFF(2)=TEMP(2)*MAGOUT
      OUTOFF(3)=TEMP(3)*MAGOUT
      POUTER(1)=POSX(TLINKO,J)+OUTOFF(1)
      POUTER(2)=POSY(TLINKO,J)+OUTOFF(2)
      POUTER(3)=POSZ(TLINKO,J)+OUTOFF(3)
    END IF
    IF(END1J)THEN
       PINNER(1)=POSX(TLINKI,J)
       PINNER(2)=POSY(TLINKI,J)
       PINNER(3)=POSZ(TLINKI,J)
    ELSE IF(END2J)THEN
       INOFF(1)=OFX(TLINKI)
       INOFF(2)=OFY(TLINKI)
       INOFF(3)=OFZ(TLINKI)
       MAGIN=DSQRT(INOFF(1)**2+INOFF(2)**2+INOFF(3)**2)
       TEMP(1)=INOFF(1)/MAGIN
       TEMP(2)=INOFF(2)/MAGIN
       TEMP(3)=INOFF(3)/MAGIN
       U(1)=0.D0
       U(2)=0.D0
       U(3)=1.D0
       PHI=TZ(TLINKI,J)
       CALL ROTATE(U,PHI,TEMP)
       U(1)=0.D0
       U(2)=1.D0
       U(3)=0.D0
       PHI=TY(TLINKI,J)
       CALL ROTATE(U,PHI,TEMP)
       U(1)=1.D0
       U(2)=0.D0
       U(3)=0.D0
       PHI=TX(TLINKI,J)
       CALL ROTATE(U,PHI,TEMP)
       INOFF(1)=TEMP(1)*MAGIN
       INOFF(2)=TEMP(2)*MAGIN
       INOFF(3)=TEMP(3)*MAGIN
       PINNER(1)=POSX(TLINKI,J)+INOFF(1)
       PINNER(2)=POSY(TLINKI,J)+INOFF(2)
       PINNER(3)=POSZ(TLINKI,J)+INOFF(3)
    END IF
    IF(END1J)THEN
       TEMPZ(1)=JNT(TLINKI,1,1)
       TEMPZ(2)=JNT(TLINKI,1,2)
       TEMPZ(3)=JNT(TLINKI,1,3)
       U(1)=0.D0
       U(2)=0.D0
       U(3)=1.D0
       PHI=TZ(TLINKI,J)
       CALL ROTATE(U,PHI,TEMPZ)
```

```
            U(1)=0.D0
            U(2)=1.D0
            U(3)=0.D0
            PHI=TY(TLINKI,J)
            CALL ROTATE(U,PHI,TEMPZ)
            U(1)=1.D0
            U(2)=0.D0
            U(3)=0.D0
            PHI=TX(TLINKI,J)
            CALL ROTATE(U,PHI,TEMPZ)
            DOT=(POUTER(1)-PINNER(1))*TEMPZ(1)+(POUTER(2)
     >          -PINNER(2))*TEMPZ(2)+(POUTER(3)-PINNER(3))
     >          *TEMPZ(3)
            IF(DOT.GT.JPMAX)THEN
               JLEN(TLINKI,1)=JLEN(TLINKI,1)+DOT-JPMAX
               JPMAX=DOT
            ELSE IF(DOT.LT.JPMIN)THEN
               JLEN(TLINKI,1)=JLEN(TLINKI,1)-DOT+JPMIN
               JPOS(TLINKI,1)=JPOS(TLINKI,1)+DOT-JPMIN
               JPMIN=DOT
            END IF
         ELSE IF(END2J)THEN
            TEMPZ(1)=JNT(TLINKI,2,1)
            TEMPZ(2)=JNT(TLINKI,2,2)
            TEMPZ(3)=JNT(TLINKI,2,3)
            U(1)=0.D0
            U(2)=0.D0
            U(3)=1.D0
            PHI=TZ(TLINKI,J)
            CALL ROTATE(U,PHI,TEMPZ)
            U(1)=0.D0
            U(2)=1.D0
            U(3)=0.D0
            PHI=TY(TLINKI,J)
            CALL ROTATE(U,PHI,TEMPZ)
            U(1)=1.D0
            U(2)=0.D0
            U(3)=0.D0
            PHI=TX(TLINKI,J)
            CALL ROTATE(U,PHI,TEMPZ)
            DOT=(POUTER(1)-PINNER(1))*TEMPZ(1)+(POUTER(2)
     >          -PINNER(2))*TEMPZ(2)+(POUTER(3)-PINNER(3))
     >          *TEMPZ(3)
            IF(DOT.GT.JPMAX)THEN
               JLEN(TLINKI,2)=JLEN(TLINKI,2)+DOT-JPMAX
               JPMAX=DOT
            ELSE IF(DOT.LT.JPMIN)THEN
               JLEN(TLINKI,2)=JLEN(TLINKI,2)-DOT+JPMIN
               JPOS(TLINKI,2)=JPOS(TLINKI,2)+DOT-JPMIN
               JPMIN=DOT
            END IF
         ELSE IF(END1G)THEN
            IF(OEND1J)THEN
               IEND=1
            ELSE IF(OEND2J)THEN
               IEND=2
            END IF
            TEMPZ(1)=JNT(TLINKO,IEND,1)
            TEMPZ(2)=JNT(TLINKO,IEND,2)
```

```
            TEMPZ(3)=JNT(TLINKO,IEND,3)
            U(1)=0.D0
            U(2)=0.D0
            U(3)=1.D0
            PHI=TZ(TLINKI,1)
            CALL ROTATE(U,PHI,TEMPZ)
            U(1)=0.D0
            U(2)=1.D0
            U(3)=0.D0
            PHI=TY(TLINKO,1)
            CALL ROTATE(U,PHI,TEMPZ)
            U(1)=1.D0
            U(2)=0.D0
            U(3)=0.D0
            PHI=TX(TLINKI,1)
            CALL ROTATE(U,PHI,TEMPZ)
            DOT=(POUTER(1)-PINNER(1))*TEMPZ(1)+(POUTER(2)
     >          -PINNER(2))*TEMPZ(2)+(POUTER(3)-PINNER(3))
     >          *TEMPZ(3)
            IF(DOT.GT.JPMAX)THEN
               GLEN(I)=GLEN(I)+DOT-JPMAX
               JPMAX=DOT
            ELSE IF(DOT.LT.JPMIN)THEN
               GLEN(I)=GLEN(I)-DOT+JPMIN
               GPOS(I)=GPOS(I)+DOT-JPMIN
               JPMIN=DOT
            END IF
         ELSE IF(END2G)THEN
            IF(OEND1J)THEN
               IEND=1
            ELSE IF(OEND2J)THEN
               IEND=2
            END IF
            TEMPZ(1)=JNT(TLINKO,IEND,1)
            TEMPZ(2)=JNT(TLINKO,IEND,2)
            TEMPZ(3)=JNT(TLINKO,IEND,3)
            U(1)=0.D0
            U(2)=0.D0
            U(3)=1.D0
            PHI=TZ(TLINKO,1)
            CALL ROTATE(U,PHI,TEMPZ)
            U(1)=0.D0
            U(2)=1.D0
            U(3)=0.D0
            PHI=TY(TLINKO,1)
            CALL ROTATE(U,PHI,TEMPZ)
            U(1)=1.D0
            U(2)=0.D0
            U(3)=0.D0
            PHI=TX(TLINKO,1)
            CALL ROTATE(U,PHI,TEMPZ)
            DOT=(POUTER(1)-PINNER(1))*TEMPZ(1)+(POUTER(2)
     >          -PINNER(2))*TEMPZ(2)+(POUTER(3)-PINNER(3))
     >          *TEMPZ(3)
            IF(DOT.GT.JPMAX)THEN
               GLEN(I)=GLEN(I)+DOT-JPMAX
               JPMAX=DOT
            ELSE IF(DOT.LT.JPMIN)THEN
               GLEN(I)=GLEN(I)-DOT+JPMIN
```

```
                    GPOS(I)=GPOS(I)+DOT-JPMIN
                    JPMIN=DOT
                  END IF
                END IF
20            CONTINUE
            END IF
10      CONTINUE
        RETURN
        END
```

Vita

The author was born in Redwood Falls, Minnesota on June 29, 1949. At VPI&SU, he enrolled in the CO-OP program and served in this CO-OP capacity with Bethlehem Steel Corporation. Work at Bethlehem Steel Corporation involved wear and stress studies on channel rolls as well as work in the area of processing "thick" biaxially oriented polymers. In 1978, he graduated from VPI&SU with a bachelors degree in mechanical engineering.

A five year period of employment at Motorola followed. During this time, the author worked on several portable radio products in the areas of package design, structural analysis, heat transfer, and material selection. He also obtained his professional engineers license and completed his masters degree in mechanical engineering at Florida Atlantic University during this period.

He returned to VPI&SU where he began his doctoral studies. During this time he worked as a part time teaching assistant and graduate fellow under a grant from Westinghouse. He finished his studies while working for Poly-Scientific as a software specialist in CAD applications. The author is currently living in Blacksburg, Virginia with his wife Marla.

Mitchel J. Keil