ornl

OAK RIDGE NATIONAL LABORATORY

LOCKHEED MARTIN

ORNL/TM-13096

Inverse Kinematics of Redundant Systems Driver IKORv1.0–2.0

Charles J. Hacker
Gregory A. Fries
François G. Pin

MASTER

MANAGED AND OPERATED BY
LOCKHEED MARTIN ENERGY RESEARCH CORPORATION
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

ORNL-27 (3-96)

ORNL/TM-13096

Robotics and Process Systems Division

INVERSE KINEMATICS OF REDUNDANT SYSTEMS DRIVER IKORv1.0–2.0

(FULL SPACE PARAMETERIZATION WITH ORIENTATION CONTROL, PLATFORM MOBILITY, AND PORTABILITY)

Charles J. Hacker, Gregory A. Fries, and François G. Pin

Date Published—January 1997

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
Managed by
LOCKHEED MARTIN ENERGY RESEARCH CORP.
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-96OR22464

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Few optimization methods exist for path planning of kinematically redundant manipulators. Among these, a universal method is lacking that takes advantage of a manipulator's redundancy while satisfying possibly varying constraints and task requirements. Full Space Parameterization (FSP) is a new method that generates the entire solution space of underspecified systems of algebraic equations and then calculates the unique solution satisfying specific constraints and optimization criteria. The FSP method has been previously tested on several configurations of the redundant manipulator HERMIES-III. This report deals with the extension of the FSP driver, Inverse Kinematics On Redundant systems (IKOR), to include three-dimensional manipulation systems, possibly incorporating a mobile platform, with and without orientation control. The driver was also extended by integrating two optimized versions of the FSP solution generator as well as the ability to easily port to any manipulator.

IKOR was first altered to include the ability to handle orientation control and to integrate an optimized solution generator. The resulting system was tested on a 4 degrees-of-redundancy manipulator arm and was found to successfully perform trajectories with least norm criteria while avoiding obstacles and joint limits. Next, the system was adapted and tested on a manipulator arm placed on a mobile platform yielding 7 degrees of redundancy. After successful testing on least norm trajectories while avoiding obstacles and joint limits, IKORv1.0 was developed. The system was successfully verified using comparisons with a current industry standard, the Moore Penrose Pseudo-Inverse. Finally, IKORv2.0 was created, which includes both the *one shot* and *two step* methods, manipulator portability, integration of a second optimized solution generator, and finally a more robust and usable code design.

1. INTRODUCTION

A new method, Full Space Parameterization (FSP), was recently developed[1] for the solution of underspecified systems of algebraic equations. This method was tested for application to the joint velocity calculations of redundant manipulators with kinematic and environmental constraints.[1,2] The overall development, enhancement, and testing program for the FSP initially involved simulations in a UNIX workstation environment using a preliminary version of the FSP code in the ANSI "C" programming language. A two-dimensional (2-D), 4 degrees-of-freedom (DOF) planar manipulator, with 1 degree of redundancy (DOR), was used to test the new method. The results were compared to those of the popular Least Norm optimization criterion (the Moore Penrose Pseudo-Inverse). The comparison study proved that the new method matched the results of the industry-accepted optimization criterion to within five significant digits, that is, to within the expected truncation errors of the calculational code.

A workbench simulator was then developed,[3] and the FSP method was tested using a 7-DOF, three-dimensional (3-D) manipulator using orientation control. The algorithm was again tested against the Pseudo-Inverse and again proved to equivalently match its results. The versatility of the algorithm was also tested during this stage by including joint limits in the manipulator kinematics.

Obstacle avoidance was then implemented on the algorithm and tested on a 2-D manipulator with 2 DOR.[4] The algorithm was shown to avoid two obstacles while maintaining a least norm movement. A revised code was then generated, including both obstacle and joint limit avoidance, and was tested on a 2-D, 2-DOR system as well as on a 3-D, 7-DOF system.[2]

As the DOR increased, so did the complexity of the algorithm. A revised and enhanced code was thus generated that not only solved arbitrary DOF systems with arbitrary DOR but also improved the solution generator both for speed and complications based on ill-conditioned Jacobians.[5] An optimized solution generator was further developed to produce FSP version 2.0.[6]

This report describes the incorporation of FSP version 2.0 in a user-friendly driver that allows development, application, and/or testing of the FSP on a wide variety of systems in various control configurations.

2. SYSTEMS OVERVIEW

As shown in Fig. 1, the Inverse Kinematics On Redundant systems (IKOR) driver is the user's interface to FSP. This section introduces the system using a bottom-up approach. First, the need for inverse kinematics in the field of robotics is examined. Next, the scope of possibilities for the FSP inverse kinematics system is addressed. Finally, the IKOR driver is discussed as the user's interface that updates, initializes, and drives FSP and possibly other inverse kinematics routines.

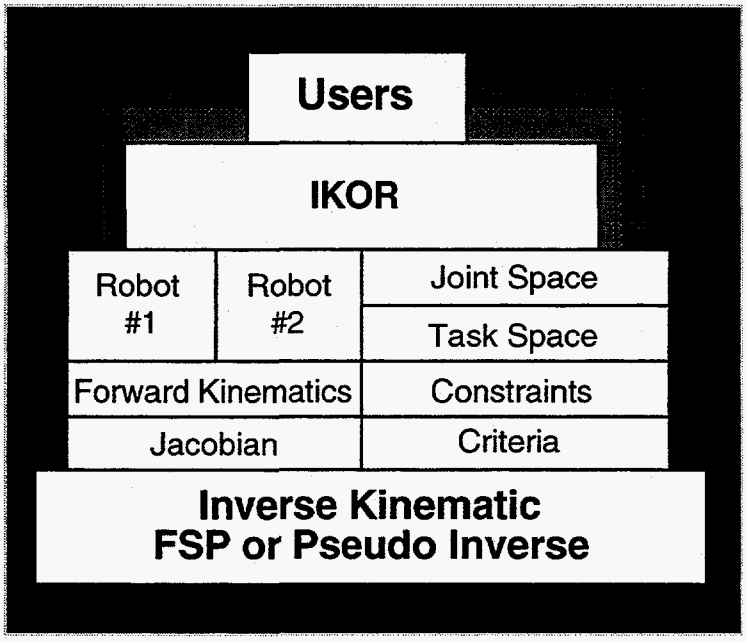


Fig. 1. System layers.

2.1. INVERSE KINEMATICS ON REDUNDANT SYSTEMS

A new interest in the field of robotics is concerned with the motion of redundant manipulators. Whatever the robotic task being considered, the motion generated should be smooth, follow criteria such as strength optimization or motion-specific minimization, and be acceptable and adaptable to environmental hazards such as obstacles. Some of the distinctive problems needing to be solved include inverse kinematics, motion planning, obstacle avoidance, and grasping.[7]

A robot arm that has more joints than required to move an end-effector into any configuration is considered to be a redundant manipulator. Throughout this study, M represents the dimension of the joint space or the number of joints and N represents the dimension of the task space that describes the required position and orientation of the manipulator. For example, a 3-D, 7-DOF manipulator, for which both position and orientation are controlled, has 1 DOR.

$M = 7$: DOF
$N = 6$: Position(x, y, and z) + Orientation (yaw, pitch, roll)
$M - N = 1$: DOR

The analytical description of the forward kinematics problem maps the set of joint angles, $\bar{q}$, of the manipulator to a resulting end-effector position and orientation vector $\bar{X}$.

$$(F) : (\bar{q}) -> (\bar{X}) \quad . \tag{1}$$

The function F is referred to as the forward kinematics function, and its determination is primarily a geometry problem.

To initiate path planning however, the situation is reversed. Joint angles that represent a specific position of the end-effector are desired. This mapping of angles given a specific position is referred to as inverse kinematics which deals primarily with the inverse of the mapping function F described previously:

$$(G) : (\bar{X}) -> (\bar{q}) \quad . \tag{2}$$

The forward kinematics equation

$$\bar{X} = F(\bar{q}) \tag{3}$$

can be differentiated to provide for changes in position:

$$\dot{\bar{X}} = J(\bar{q})\dot{\bar{q}} \quad \text{with} \quad J_{ij} = \frac{\partial F_i}{\partial q_j} \quad , \tag{4}$$

where the matrix J is referred to as the Jacobian matrix. Equation (4) represents a nonlinear relationship that is typically linearized over discrete time steps Δt for controls:

$$\Delta \bar{X} \approx J\Big|_{\Delta t} \Delta \bar{q} \quad . \tag{5}$$

Drift errors with this linearization are generally corrected with the introduction of a feedback loop (see Sect. 2.2). Assuming very small changes in position, the feedback mechanism can provide for accurate motion. Because redundant systems have more joints than requested elements in the task space, the solution involves the inversion of a nonsquare, $M \times N$, Jacobian.

There is no direct method for inverting a nonsquare matrix. However, the increase in complexity is justified and desired since these systems allow for an infinite amount of solutions to a given point in a trajectory path. Several methods, such as the Pseudo-Inverse and Extended Jacobian have been developed to provide for particular

solutions of the inverse problem.[8,9] Unfortunately, these methods generally fail to satisfy those who implement real-time, sensor-based systems because only one solution is readily found by each method. The Pseudo-Inverse method provides the solution that involves only the least norm motion of the joints. This method cannot adhere to different criteria such as strength optimization, or any additional constraints such as obstacles or joint limits unless additional computationally intensive procedures are used to develop homogeneous solutions. In these cases, however, the solution loses its initial criterion. The extended Jacobian, which also finds only a particular solution, extends the Jacobian into a square matrix by adding constraints that are computationally intensive as well as difficult to implement in a dynamic environment. To take full advantage of redundant systems, solutions corresponding to a wide variety of constraints and/or criteria should be found readily and efficiently.

If the entire space of solutions could be captured for the inverse kinematics problem, solutions corresponding to particular criteria and constraints could be generated analytically. The FSP does just that.[1-6]

2.2. FULL SPACE PARAMETERIZATION

Instead of finding only a particular solution to the inverse kinematics problem, the FSP method enables the entire space of solutions to be determined. FSP derives a sufficient number of solution vectors, $\bar{g}_i$, from the Jacobian to fully parameterize the entire solution space. Except for special cases,[5,6] this sufficient number of linearly independent vectors is $M-N+1$.[1] Equation (6) describes the entire solution space of the unconstrained Eq. (5).

$$S,S=\left\{\Delta\bar{q}\in\Re^m,\Delta\bar{q}=\sum_{i=1}^{M-N+1}t_i\bar{g}_i,\quad\sum_{i=1}^{M-N+1}t_i=1\right\}\quad . \tag{6}$$

The vectors $\bar{g}_i$ are found by blocking $M-N$ columns from the Jacobian and inverting the resulting square matrix. Although $M-N+1$ inversions must be made, the entire solution space is found and parameterized. Consequently, analytical methods can be used to determine the optimum solution with respect to possibly changing task criteria and constraints.

An added benefit of FSP is its ability to readily find the entire solution space for motions in the null space, that is, motions that change the arm configuration while keeping the end-effector in the same place:

$$\mathrm{N,N}=\left\{\Delta\bar{q}\in\Re^m,\Delta\bar{q}=\sum_{i=1}^{M-N+1}t_i\bar{g}_i,\quad\sum_{i=1}^{M-N+1}t_i=0\right\}\quad . \tag{7}$$

As shown subsequently, the IKOR driver allows for the modular insertion of optimization criteria such as least norm, minimum power, minimum torque or optimum

acceleration. Additionally, the ability to include obstacle avoidance, joint limits, and other constraint modules is present.

2.3. IKOR DRIVER FOR THE FULL SPACE PARAMETERIZATION

IKOR consists of several stages and tests, as outlined in Fig. 2. The following is an overview of the IKOR system. Specific details, data flows, and implementation notes can be found in Appendix C, "System Data Management."

The user begins by selecting a robot, platform, orientation control, and a trajectory. IKOR then compiles the chosen manipulator into the system, initializes the manipulator, and cycles through the trajectory one time step at a time.

Once IKOR reads a data file containing initial joint angles and a trajectory, the requested change in x, y, z, and if selected, yaw, pitch, and roll of the end-effector, are calculated by taking the forward kinematics of the current joint angles and subtracting the values from the next point in the trajectory. A Jacobian based on the current configuration is calculated and this, along with $\Delta\overline{X}$ in Eq. (5), is sent to the FSP's solution generator subroutine. The solution generator proceeds to determine square submatrices from the Jacobian and produce the set of $M-N+1$ linearly independent solution vectors. The generator then returns an exit status, the solution vectors, and any "beta vectors" due to constraints related to the rank lost optimization process.[5]

The code initiates either a *one shot* or *two step* method, based on user preference. For the *one shot* method (see Sect. 4.2.3), the system determines at the beginning of the time step if the current state of the robot is violating any constraints. If so, "beta vectors" necessary to guide the parameterization are calculated.[1,2] The parametric solutions are then calculated based on the given criterion and the beta vectors. The change in angles necessary to complete the motion are produced based on the avoidance of constraints and the requested criterion. This flow is demonstrated in Fig. 2.

However, the user may also choose a *two step* approach (see Sect. 4.1.2), in which a motion based only on the given criterion is generated. In other words, no obstacle or joint limit constraints are used to generate this motion. Then, still in the current time step, constraints are checked and if any are violated, beta vectors are produced and a new constrained parametric solution is found. However, this constraint avoidance is performed in the null space [see Eq. (7)]. When added to the previously calculated solution, a motion based on the avoidance of constraints and the requested criterion is performed.

If there are remaining points in the trajectory, the system again calculates the change in position by performing the forward kinematics of the new joint angles and subtracting it from the next trajectory point. This is the feedback mechanism that provides for an accurate motion, as mentioned previously.

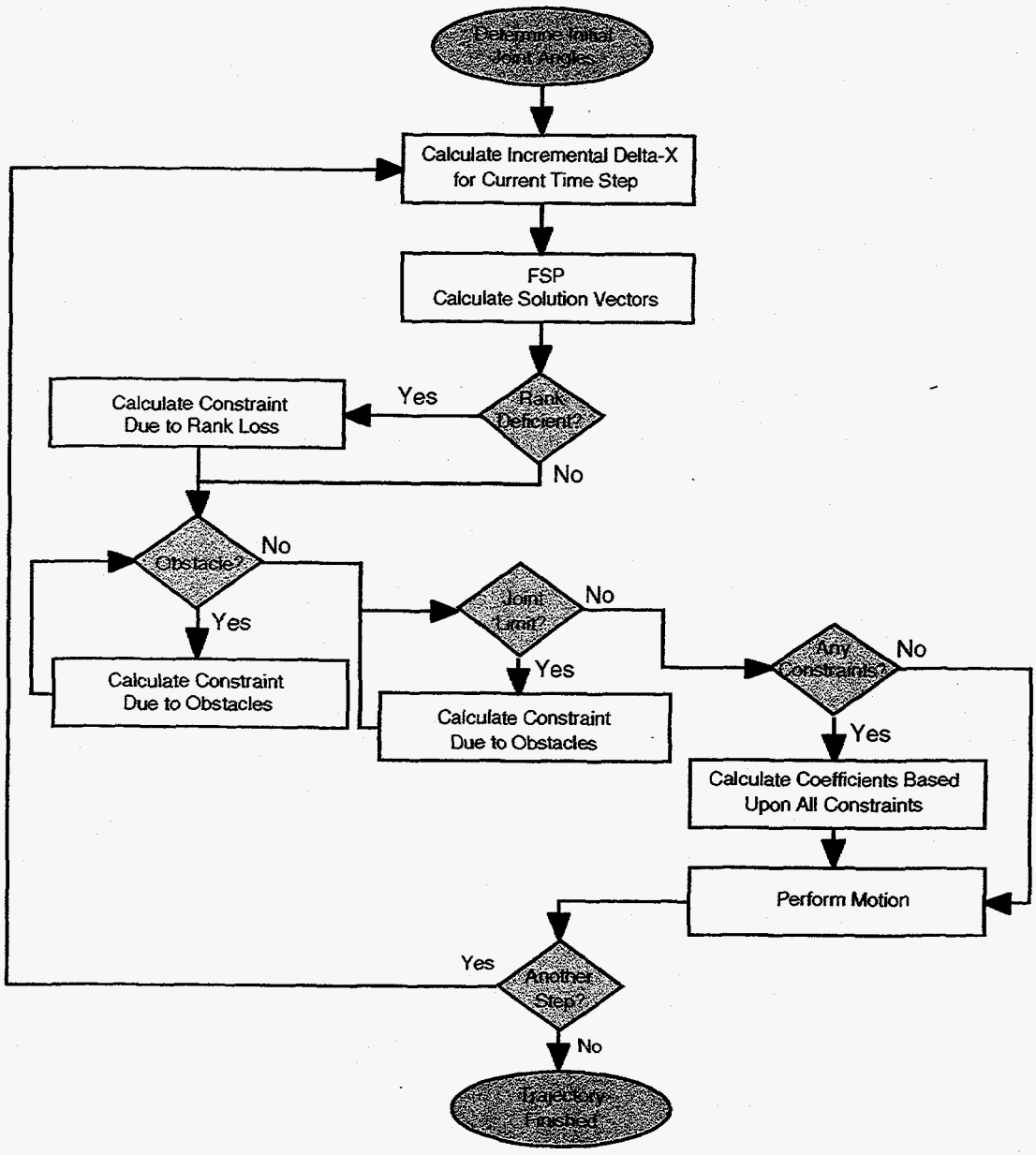


Fig. 2. System flowchart.

3. SPECIAL SYSTEM REQUIREMENTS

Every system has special requirements that drive its development. Two such requirements for IKOR involve performance and continuity. The first requirement, performance, is necessary because FSP/IKOR's future is in the real-time control-system arena. The second special requirement, continuity, is based on the desire to keep the system entropy low through intelligent design.

3.1. PERFORMANCE

IKOR has been demonstrated in a real-time environment on TELEGRIP, a UNIX-based simulation software program (see Appendix B2, "Silicon Graphics Station"). Real-time systems require the minimization of two timing mechanisms. These mechanisms are maximum time and average time which are discussed on a time step basis in the following.

Although neither mechanism should be overlooked, maximum time is the delimiter. This timing mechanism involves the greatest amount of possible time consumed in a single time step. For FSP/IKOR as well as many other systems, maintaining the minimization of this time is a strict requirement. It has been shown that although almost a trillion (see Sect. 4.1.1) possible solution combinations exist for a highly redundant manipulator, the search can be considerably optimized.[5,6] It must be assumed that the maximum number of constraints has also occurred. However, the generation of beta vectors and the calculation of the parametric solutions take a relatively constant amount of time and are inconsequential relative to the solution generator's time.[2] The solution generator process uses most of its time inverting the $M - N + 1$ square submatrices.

For a system such as this, the average time is most easily calculated empirically. By taking many trajectory samples, the average time can be determined by dividing the total time by the number of steps. The average time for FSP/IKOR must necessarily be considerably less than the maximum time since it is rare for the system to go through all solution combinations completely.

Therefore, to be seen as a viable real-time method, a constant mind-set must be directed towards keeping maximum run time at a minimum. For the linear model and average time mechanism, much work has already been done in both FSPv1.0[5] and FSPv2.0[6] as well as IKORv1.0 and IKORv2.0. In-depth analysis and assembly code appear to be the next increase in speed for the linear model. The parallel model must not be neglected as a viable model that may hold instrumental gains for both maximum and average system times.

3.2. CONTINUITY

The IKOR/FSP systems-development teams have been varied. With several people responsible for updating and modifying the system with little if any overlap, the potential

for the design to become less flexible, less adaptable, and more costly is high. Therefore, it is necessary for modifiers of this system to project what the needs of the future will be and design accordingly. Special attention should be paid to modular subsystem design.

Many studies have been made showing entropy of software systems over time.[10] This environment provides no exception, and developers of this system will quickly become aware of this fact. With the number of people who modify the code and their difficult task of learning the entire system in a relatively little amount of time, thoughtful design becomes increasingly more important. Therefore, a design view that focuses on the continuity of the system is a special system requirement.

4. MAJOR OBJECTIVES

Several points were the focus of the enhancements of IKORv1.0 and IKORv2.0. These objectives materialized in modifications to both the FSP system and its driver. The details of the modifications are presented in the following.

4.1. IKORv1.0

IKOR began as a loose collection of procedures that solved for least norm motion on a 2-D planar manipulator as well as the 3-D HERMIES-III CESARm manipulator. As DORs increased however, it was necessary to implement a more integrated and modular system. With the integration of the enhanced solution generator, IKOR matured into IKORv1.0. This version consisted of numerous improvements over its predecessor. IKORv1.0 included in one software package the capabilities for orientation control, platform control, the preliminary ability to port to different manipulators, and increased modularity.

4.1.1. Integration of FSPv1.0 Solution Generator

The FSP method had been previously tested on a 2-D, 4-DOF planar manipulator, with and without orientation control. This manipulator with orientation control had a three element task space x, y, and pitch, yielding $4-3=1$ DOR. Without orientation control, the task space has two elements x and y, yielding $4-2=2$ DOR. A 7-DOF, 3-D robot was also tested with orientation control and a task space of six elements, y, x, z, yaw, pitch, and roll, yielding $7-6=1$ DOR.

All three of these initial manipulator setups contained at most 2 DOR and a relatively small number of possible inversions necessary to produce the solution vectors that span the entire space. For example, the 2-D, 4-DOF case cited previously, without orientation, would involve a maximum of $C_2^4=6$ inversions and therefore six possible solution vectors to choose three independent vectors (recall that the number of solution vectors necessary are $M-N+1$ or $4-2+1=3$). This is a maximum of $C_3^6=20$ possible combinations. The 3-D, 7-DOF case was only slightly more computationally difficult, with $C_6^7=7$ possible inversions and $C_2^7=21$ possible solution combinations. It was foreseen that the CESARm manipulator on a platform and without orientation control would be much more intensive. Such a manipulator has 10 DOF with 7 DOR and would require a possible $C_7^{10}=120$ inversions which, with eight independent solution vectors required, yields $C_8^{120}=840,261,910,994$ possible solution combinations. It was obvious that FSP would require a sophisticated optimization routine to quickly find one in nearly a trillion combinations (it should be noted that many of these combinations provide for an adequate solution space). FSPv1.0 represented the birth of the optimized solution generator.

The upgrade to the solution generator included modifications to the code that implemented the calculations of the analytical solutions. This upgrade was performed on a previous system that worked with two solution vectors, the 3-D, 7-DOF with orientation control, and 1-DOR HERMIES system, see Appendix B3, "Modifications of IKOR."

The optimized FSPv1.0 was integrated into IKOR through a structural reorganization. The system being upgraded computed the solution vectors before the least norm motion (see Fig. 2). The conversion process consisted of replacing several procedures with the single FSP subroutine function call as well as the conversion and standardization of the function parameters.

Test trajectories were run on both the original and enhanced versions of the 3-D system without orientation control code. The results from several tests showed complete compatibility. Indeed, this was expected since the enhanced FSP code was mathematically proven to be just as competent as the original code. The enhanced solution generator code was implemented only to increase efficiency. The obstacle and joint limit avoidance were still operating under the original FSP code.

The system was tested against the Pseudo-Inverse algorithm to compare the method with an industry standard. Appendix A1 (a–d) provides the results from four test trajectories that compare FSP Least Norm vs Pseudo-Inverse. For each trajectory, four configurations (3×10, 3×7, 6×10, 6×7) were tested. For the enhanced code integration experiments, a 7-DOF, 3-D system with orientation control was used. These data are labeled as 6×7 in the graphs. In each case, FSP and Pseudo-Inverse algorithms agreed to within 1E-5 accuracy. Because the Pseudo-Inverse does not deal with constraints, the test trajectories did not include any joint limit or obstacle avoidance, the inclusion of which would inevitably alter the motion. However, timing tests were conducted that did include obstacles to give a run time comparison of FSP's strengths (see Sect. 4.2.3).

4.1.2. Adaptation of FSPv1.0's Analytical System

The experiments conducted to show FSP's abilities included testing of the ability to do both joint limit and obstacle avoidance while remaining true to a least norm criterion. This was first accomplished using the *two step* method. During the first step, a least norm motion was performed. During the second step, constraints were avoided by moving the arm in the null space, thereby losing the least norm criteria. References 1 and 2 provide the analytical solutions for null space motion, including obstacle and joint limit constraints. Essentially, a constrained optimization Lagrangian is constructed and differentiated to obtain the optimal condition and to solve for the parameter vector $\bar{t}$ with component t_i, $i = 1, M - N + 1$. The motion solution is then obtained as

$$\Delta\bar{q} = \sum_{i=1}^{M-N+1} t_i \bar{g}_i \quad . \tag{8}$$

For the first step, least norm motion without any constraints,[1] $\bar{t}$ reduces to

$$\bar{t} = \frac{G^{-1}\bar{e}}{\bar{e}^T G^{-1}\bar{e}} \quad , \tag{9}$$

where $e_i = 1,\ i = 1,\ M - N + 1$, and each element of the Grammian matrix G is a dot product of the $\bar{g}$ vectors:

$$G_{ij} = \langle \bar{g}_i\, \bar{g}_j \rangle \quad . \tag{10}$$

To satisfy constraints in the null space (see Ref. 2), the following equation is used:

$$\bar{t} = -\mu G^{-1}\bar{e} - G^{-1}v^i\bar{\beta}^i \quad , \tag{11}$$

where μ and v^i and Lagrange multipliers, and the constraints due to either joint limits, obstacles, or loss in rank (see Refs. 6 and 7 for loss in rank) are represented by the $\bar{\beta}^i$ vectors.

For obstacle avoidance, the $\bar{\beta}$ vectors are calculated using the unit vector normal to the obstacle, $\bar{n}$, and the distance, d, needing to be traveled by the point on the manipulator (with Jacobian J^*) closest to the obstacle,[2] according to

$$\bar{\beta}, \beta_k = \sum_{i=1}^{3} \sum_{j=1}^{m} (J^*_{ij}\ g_{k_j} n_i)/d \quad . \tag{12}$$

For joint limit avoidance, the $\bar{\beta}$ vector is given by

$$\bar{\beta}, \beta_k = g_k/d \quad , \tag{13}$$

where d is the angle by which the specific joint exceeds its limit.

The obstacle avoidance, joint limit avoidance, and analytical solutions were rewritten to facilitate the same DOR as the enhanced FSP code (i.e., up to computer limitations). One problem encountered was the use of the matrix manipulation routines created by Henry Dorum (see Appendix B3, "Modification of IKOR, Using MATRIX Code.") The subroutines are the backbone for most matrix manipulations throughout the system. The incorrect use of these subroutines caused considerable memory leaks, which in turn, resulted in inconsistent behaviors. Appendix B3 should be consulted for a full description for the appropriate use of this code.

4.1.3. Orientation Control

The conversion from orientation control to no orientation control allowed for 3 more DOR on the HERMIES system for a total of 4 DOR (4 DOF are theoretical since the roll element of the Euler angles for the HERMIES arm has no bearing on the end-effector's

x, y, z position. Therefore, there were only 3 effective DOR.) With orientation control, the end-effector's x, y, and z as well as final yaw, pitch, and roll are necessary to determine a target position for the end-effector. In many instances however, it is not necessary to have orientation control until the arm is considerably closer to the target. Eliminating the yaw, pitch, and roll from the system's operational space allows the manipulator to be more agile and able to avoid obstacles or other constraints. It is also possible to alter the software to include any one, two, or three of the Euler angles as needed. For instance, if the manipulator was used to carry spillables, pitch and roll could be constrained while the yaw could be free to change to avoid constraints.

Upon observation of the code, only two sections needed to be altered to convert the position control from six elements (x, y, z, yaw, pitch, and roll) down to three elements (x, y, and z). In the first section, the size of $\Delta\bar{x}$ was reduced. Next, the Jacobian was altered to use only the necessary position elements.

In both cases, a conditional statement based upon the value of the task space, N, was placed around the section of code in question. Calculation of $\Delta\bar{x}$ for the Euler angles yaw, pitch, and roll were blocked. For the second case, to remove the bottom three rows, a conditional statement was placed around the code in the subroutine that computed the Jacobian. Global variables for M and N, corresponding to joint and operational space respectively, are available throughout the code, permitting dynamic alteration. In IKORv2.0, a new data structure was implemented to remove the necessity of the two global variables.

Once complete, the new system was tested against the Pseudo-Inverse. Appendix A1 (a–d) includes results from four test trajectories that show nearly identical behaviors. Because previous experiments consistently demonstrated equivalent compatibility with the Pseudo-Inverse, this was expected. For each trajectory, four configurations (3×10, 3×7, 6×7, and 6×10) were tested. For the data discussed in this section, 7-DOF, 3-D systems without orientation control were used (these data are labeled 3×7 in the graphs).

4.1.4. Platform Introduction and Obstacle Avoidance Update

In the search to further exploit the benefits of the FSP method, the manipulator was placed upon a moving platform, producing a 10-DOF system. Using the appropriate Jacobian and forward kinematics, and no orientation control, a 7-DOR mobile manipulator was enabled.[5] Also, a 4-DOR system was created by including orientation control and platform mobility.

At this stage, conversion from orientation to no orientation control involved only the manipulation of the global variable N that acknowledged either six or three elements in the position vector. This system was tested against the Pseudo-Inverse. Appendix A1 (a–d) includes the results from these tests (see the data labeled 3×10 and 7, 6×10). Comparisons of the results showed that FSP and the Pseudo-Inverse produced identical results.

To detect intersection with obstacles, the forward kinematics subroutine must provide beginning and ending points for each link. This was achieved by implementing two drive routines that used the forward kinematics expressions. One driver allowed the altering of the link lengths, while the other used the full link lengths. Also, the calculation of obstacle constraints involved a reduced Jacobian based on the point of the "intersection-terminated" manipulator arm. In a like manner, two Jacobian drivers were also required. One allowed for dynamically alterable link lengths, while the other included fixed, full link lengths. These revisions allowed the forward kinematics and Jacobian subroutines to operate with differing link lengths. When porting the code to another system, it is only necessary to swap out these subroutines, not the drivers (see Sects. 4.1.5 and 4.2.4).

4.1.5. Preliminary Portability

Since all test systems to date involved the HERMIES-III manipulator arm as a prototype, it was desirable to determine the software's ability to port to other manipulators. Therefore, IKOR was modified to allow for easier robot porting. To test this portion of the code, the forward kinematics and Jacobian were derived for the AIRARM, a manipulator under study in a current project at Oak Ridge National Laboratory.[11]

The conversion process involved liberating the dependencies of the routines for the forward kinematics and the Jacobian calculations from the simulation system. The HERMIES' routines were separated, and the AIRARM's routines were developed to conform to the system. This involved the alteration of macros and other data variables for the number of links, link lengths, and platform length and width values.

Because of the AIRARM's prismatic joint, not available on the HERMIES manipulator arm, additional consideration of optimization criteria had to be addressed. The problem resulted from the fact that changes in angles and linear displacements are not equivalent. To correct the problem, a new mechanism was introduced through Eq. (10). Although not shown in Eq. (10), the formalism developed in Ref. 12 introduces a matrix B, such that

$$G_{ij} = \bar{g}_i B^T B \bar{g}_j \quad . \tag{14}$$

The solution involves using this B matrix as a normalizing diagonal matrix based on the type of joint.[12] For the kinematics developed for the AIRARM, rotational joint displacements were in radians, while the prismatic displacements were in inches. Therefore, the component in the matrix B used for the translational joint was .042, the conversion factor between inches and meters. Components for the other rotational joints were set to 1. Note that use of the B matrix can be further generalized to create an interesting mechanism for real-time control of joint activity, allowing the relative control of the contribution of any of the joints dynamically.

4.1.6. IKORv1.0 Complete

With the addition of the preliminary ability to port to other manipulators, IKORv1.0 was complete. The integration of all four of the configurations for 3-D manipulators was the only remaining issue. The goal was to dynamically allow or limit the maneuverability of the mobile platform (alter M to provide either seven or ten elements in the joint space) aside from the use of weights. Using the platform code as a base, it was desirable to have the ability to disable the platform to achieve the original 7 DOF. The change involved the implementation of conditional statements into the subroutine that calculated the Jacobian matrix. In essence, when the joint space was desired to be seven-dimensional, the last three columns of the Jacobian matrix were dropped, thereby disallowing the joint changes associated with the platform control. This dynamic ability to change from arm plus the platform to arm only is currently dependent on the platform being the last three elements of the Jacobian. This is scheduled to be changed to the first three elements for the next release of IKOR.

Because all the configurations discussed were now resident in one software package, it was now feasible to make reliable time comparisons between different configurations as well as optimization methods. Timing data can be found in Appendix A1. The graph in Fig. 3 demonstrates the relative speeds of the FSP algorithm vs an optimized version of the Pseudo-Inverse from Ref. 13. All joint and obstacle avoidance subroutines for FSP were disabled.

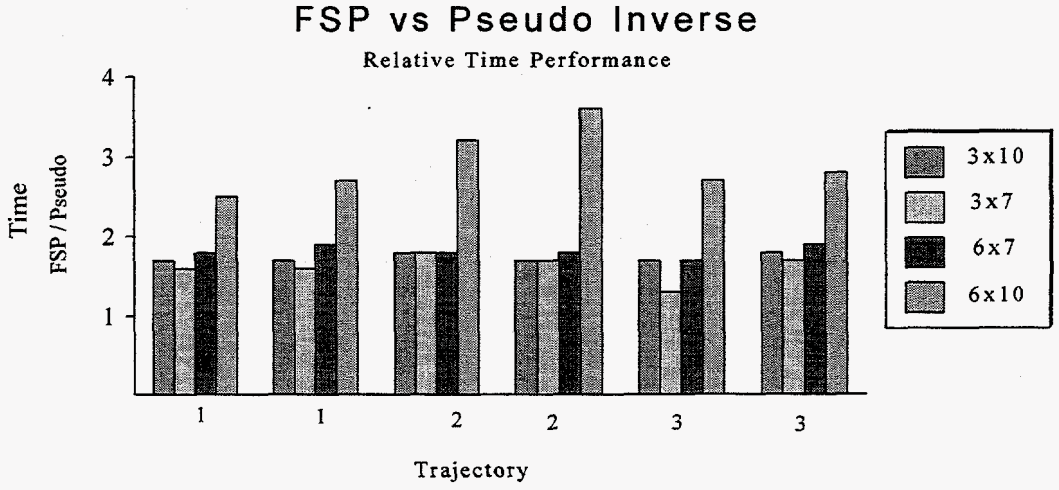


Fig. 3. Time considerations.

Each of the three trajectories was tested using a different number of points. For instance, the identical path of trajectory one was divided into 200 and 800 points, respectively. Of particular interest is the surge of the 6×10 Jacobian during all the trajectories, especially trajectory two. The cause of this trajectory's high score has not

yet been determined, although it is expected to be caused by a near singularity. (A singularity occurs when a small motion of the end-effector calls for large changes in joint angles.)

The problem with singularities is that once they are found it is usually too late. An accepted solution that may be quite impossible to implement in real time is to avoid trajectories where this might occur.[9] By using a loop-back subsystem, FSP may be able to dynamically deal with such a problem (i.e., when a singularity occurs, the arm can be run backwards then moved forward again with a new constraint for the joint that necessitated the greatest change).

4.2. IKORv2.0

As a further enhancement, the FSPv1.0's solution generator was extended to handle more special cases. These changes, along with the desire for more system features, inspired further changes to the IKOR/FSP interface. The analytical system was updated with the implementation of the *one shot* method. IKOR was still implementing the *two step* method of making the least norm motion and then using the null space to avoid constraints. However, this method required two calculations of the entire solution space vectors as well as two calculations of the parametric solutions whenever there were any constraints to be avoided. It was therefore desired to increase efficiency by having a method that performed the criteria and incorporated all constraints with the *one shot* method. Additionally it was desired to have a robust method of porting manipulators.

4.2.1. Integration of FSPv2.0 Solution Generator

Throughout the testing phase of IKORv1.0, several trajectories were found in which there were not enough solution vectors found to completely span the solution space. After the completion of IKORv1.0, an investigation was performed. Along with an overhaul of the blocking pattern search, two special cases were uncovered.[6] The changes brought about during the integration of FSPv2.0 were numerous. Compatibility with FSPv1.0 was maintained throughout IKORv2.03. However, updating time and the presence of FSPv2.0 prohibited further compatibility.

A new data structure was created to increase the ease of data flow and to better understand the numerous amount of data elements involved in the system. This data structure, detailed in Appendix C, is FSP_data. It is an instance of the structure Solutions and maintains a conglomerate of information: joint and task space sizes, solution vectors, current joint values, number of constraints, and beta vectors. Since the number of solution vectors was no longer equal to $M - N + 1$ because of the special cases, the macro SPAN was redefined to be equivalent to the number of solution vectors returned by the solution generator.

4.2.2. Adaptation of FSPv2.0's Analytical System

The introduction of FSP_data as well as the introduction of the structure Robot described in Sect. 4.2.3 had a direct bearing on the analytical system. The adaptation also entailed a restructuring of the solution generator, constraint, and analytical system interfaces. This resolved the problems encountered in FSPv1.0 with the solution vectors being sent to the analytical procedures in a broken down state that included flags, vectors, and other variables relating to the optimization.

The data structures FSP_data and Robot were relatively straightforward adaptations of the existing code. Aside from function prototypes and declarations as well as some internal modifications, these structures mainly allowed for increased flexibility, adaptability, and maintainability.

The solution generator's interface was also adapted by introducing a routine to rebuild the solution vectors. All reduced columns introduced a zero element in each solution vector, while reduced rows generated a beta vector. Therefore, only the solution vectors were returned as well as applicable betas (see Ref. 6). This provided an excellent medium, and it corrected IKORv1.0's inability to perform rank deficient cases.

4.2.3. FSPv2.0's One Shot Analytical Solution Approach

For the *two step* method, full space motion followed by null space constraint avoidance constituted a seemingly wasteful process. One of FSP's strengths is that it takes nearly the same amount of time to calculate motion with or without constraints. However, in the *two step* process when constraints were present, it took nearly twice as long because solution vectors had to be generated again after the full space motion since the arm was in a new position.

It was desirable to produce the *one shot* process. The key difference between the implementation of the *one shot* and *two step* method is the difference between a full and null space motion. In the analytical solution, this translates to only a slight difference in calculating the Lagrange parameters (see Refs. 2 or 12). These were the FSP modifications necessary to change the motion from null space to full space. IKOR, on the other hand, needed a mechanism that allowed the selection of either method. The compile macro, *TWOSTEP*, was introduced such that when the macro is defined, the old *two step* method is used. Otherwise, and by default, the *one shot* method is used. Figure 4 shows the time relationship of these two methods using the AIRARM manipulator. In trajectory no. 1, IKOR is set to ignore all constraints. In trajectories no. 2 and no. 3, IKOR is set to avoid obstacles and joint limits. While every step in trajectory no. 2 contains a constraint adjustment, trajectory no. 3 has constraints initiated for only 80% of the steps. The *one shot* method dramatically defeats the *two step* method when obstacles are encountered and otherwise produces equivalent timing results. This method is clearly the most efficient method for the least norm criterion. Furthermore, it is obvious from Fig. 4 that no additional time is required for the dynamic avoidance of constraints using

the *one shot* code, as the comparison between trajectories no. 2 and no. 3 with obstacles, and trajectory no. 1 without an obstacle, illustrates. Data collected for this test can be found in Appendix A2.

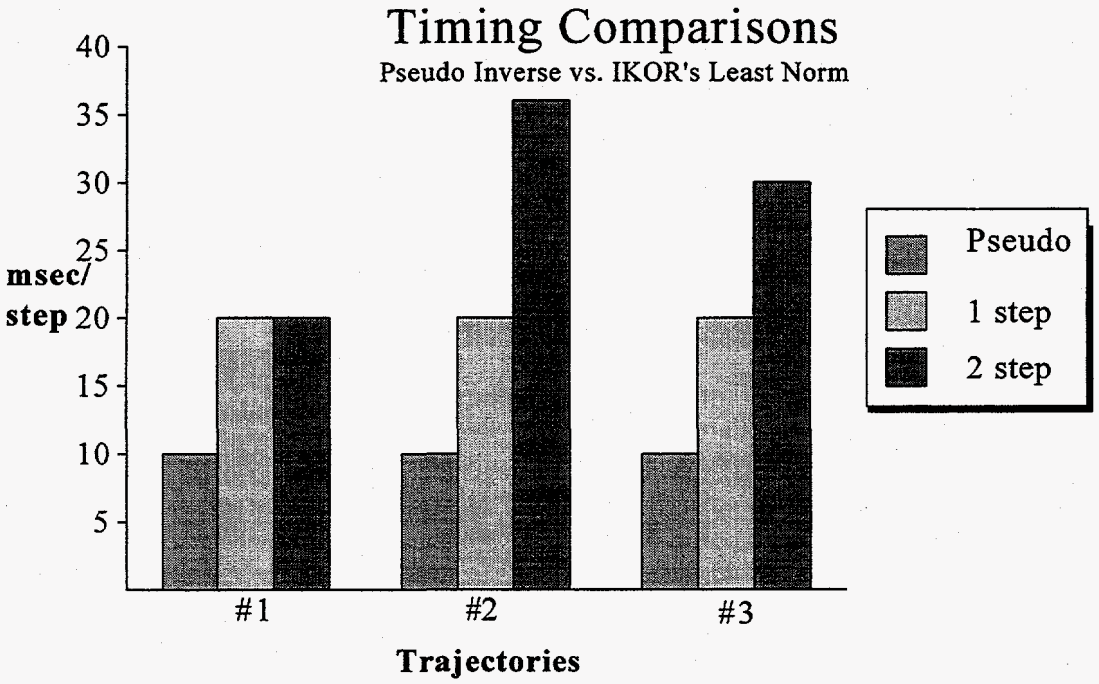


Fig. 4. Constraint avoidance timing runs.

4.2.4. IKORv2.0's Advanced Manipulator Portability

An inverse kinematics driver is incomplete without the ability to be used on many different manipulators. It was therefore a high priority for IKORv2.0 to be ported easily and effectively to other redundant manipulators. The first experiments conducted during IKORv1.0 proved that more thought was needed on the subject. For the completed IKORv2.0 system, two subroutines and a data file were the selected method that differentiated manipulators. The two subroutines were GET_JACOB() and GET_ACTUAL_X(), which calculated the forward and inverse kinematics. The data file was ROBOT.dat, which contains parameters such as link lengths.

A sample data file for the AIRARM (shown in Appendix D, "Code Listing, Manipulator Porting Programs") shows the fields, data, and a brief description of their meaning.

The subroutines GET_JACOB() and GET_ACTUAL_X() for the AIRARM or the HERMIES can also be found in Appendix D. These two procedures use different variables for the link lengths. For GET_JACOB(), the link lengths should be referenced as LL[0]...LL[NL−1], where NL represents the number of links present in the manipulator as found in the data file. However, GET_ACTUAL_X() must use

LINKS[0]...LINKS[NL−1] for the link lengths. LINKS and NL can be found inside the variable Robot. LL and Robot can currently be accessed globally.

GET_JACOB() performs the Jacobian calculation described in Eqs. (3)–(5) of Sect. 2.1. The subprocedure must receive two matrices: a declared Jacobian with N rows and M columns (which is empty although not necessarily all zeros) and Qarray, an $M \times 1$ matrix (which holds the current joint angles). The Jacobian must then be calculated.

The variables declared in the beginning of the example files are merely to aid the readability of the code. This Jacobian routine, as well as the forward kinematics routine assumes the building point to be the origin. Next, based on the platform's rotation, the temporary Jacobian is then incorporated into the real Jacobian. Note that the change in the x dimension is the first row of the Jacobian, change in the y dimension is the second row, and change in the z dimension is the third row. In the HERMIES arm, the remaining three rows represent the yaw, pitch, and roll of the orientation. The AIRARM's orientation elements have not yet been developed.

As a standard, the platform's x, y, and rotation are declared as the last three elements of the joint space of the forward kinematics (the last three columns). Therefore, based on the value of the joint space, the platform can be removed by supplying an M of the appropriate size. This mechanism is far from sophisticated, but does perform the necessary job.

The GET_ACTUAL_X() routine is based on transformation matrices. There are several matrix declarations for the transformation matrices at the beginning of the procedures in the example files. The initializations of A–E as well as XW–EW are a spin-off of CheezyIGRIP[3] and are required. CheezyIGRIP is a simulator that has been developed concurrently with IKOR/FSP. A–E specify the length of the respective link lengths as well as where the link-referencing letter is located on the arm as displayed by the CheezyIGRIP simulator. Following this, the platform is built, and then each link is added one after the other. Since XW–EW contain the x, y, and z coordinates of each link, they are placed into x_of_link that the system uses to determine the end-effector's position and the position of each link for the obstacle avoidance routines.

4.2.5. Orientation Update

Certain trajectories using the platform for the HERMIES arm were not completing properly when orientation control was being used. The Jacobian was the cause of this error and the following demonstrates its proper development. The last three rows of the Jacobian control the orientation of the manipulator. The method uses coordinate frame transformations for each of the DOF. A series of Euler transformations are used (small case "c" and "s" stand for cosine and sine, respectively). The matrices for the rotation around x, y, and z are as follows:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & c\theta & -s\theta \\ 0 & s\theta & c\theta \end{bmatrix}, \begin{bmatrix} c\theta & 0 & s\theta \\ 0 & 1 & 0 \\ -s\theta & 0 & c\theta \end{bmatrix}, \text{ and } \begin{bmatrix} c\theta & -s\theta & 0 \\ s\theta & c\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (15)$$

For the HERMIES configuration without a platform, the first DOF is a rotation around the positive z axis or $[0,0,1]^T$. To get the z axis of our current coordinate axis to point in the direction of the axis of rotation of joint 2, the coordinate frame must be rotated 90° about the current x axis and around the z axis by $q1$, the value of the first joint. The calculation with transformation matrices follows.

$$\begin{bmatrix} c1 & -s1 & 0 \\ s1 & c1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} c1 & 0 & s1 \\ s1 & 0 & -c1 \\ 0 & 1 & 0 \end{bmatrix}. \quad (16)$$

Since the axis of rotation is pointed in the z direction of our new transformed coordinate axis, the vector obtained is: $[s1,-c1,0]^T$. This matrix must then be rotated by −90° around the x of the new coordinate frame and around the z axis by $q2$, the value of the second joint. After making this computation, the result is a transformation to the last coordinate frame. Therefore, to relate it to the original frame, it must be multiplied by the previous transformation matrix.

$$\begin{bmatrix} c2 & -s2 & 0 \\ s2 & c2 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} c2 & 0 & -s2 \\ s2 & 0 & c2 \\ 0 & -1 & 0 \end{bmatrix}, \quad (17)$$

and

$$\begin{bmatrix} c1 & 0 & s1 \\ s1 & 0 & -c1 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} c2 & 0 & -s2 \\ s2 & 0 & c2 \\ 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} c1c2 & s1 & -c1s2 \\ s1c2 & -c1 & -s1s2 \\ s2 & 0 & c2 \end{bmatrix}. \quad (18)$$

Only the third column is of interest because the axis of rotation is pointing in the direction of the transformed z axis [that is, $(-c1s2, -s1s2, c2)$]. The next two DOF are special in the sense that despite any change in either one's value, there is no corresponding change in the axis around which the other rotates. Thus, the values in the transformation matrices for these two angles are equal. To get this similar axis we must rotate the current axis around the $-x$ axis by 90° and the $-z$ axis by $q3$, or the value of the third joint. This computation follows. Notice that the result is multiplied by the previous transformation to relate it back to the original coordinate axis.

$$\begin{bmatrix} c3 & s3 & 0 \\ -s3 & c3 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}=\begin{bmatrix} c3 & 0 & s3 \\ -s3 & 0 & c3 \\ 0 & -1 & 0 \end{bmatrix}, \tag{19}$$

and

$$\begin{bmatrix} c1c2 & s1 & -c1s2 \\ s1c2 & -c & -s1s2 \\ s2 & 0 & c2 \end{bmatrix}\begin{bmatrix} c3 & 0 & s3 \\ -s3 & 0 & c3 \\ 0 & -1 & 0 \end{bmatrix}\begin{bmatrix} c1c2c3-s1s3 & c1s2 & c1c2s3+s1c3 \\ s1c2c3+c1s3 & s1s2 & s1c2s3-c1c3 \\ s2c3 & -c2 & s2s3 \end{bmatrix}. \tag{20}$$

The desired vector is again in the third column of the transformation matrix. This last example demonstrates the growing nature of these vectors. To obtain the remaining two vectors, the remaining transformations must be derived so that the z axis is in the direction of rotation. This must then be multiplied by the transformations made up to this point. Doing this for the remaining three DOF, the final solution is as follows:

$$\begin{bmatrix} (-xx\cdot s4-c1s2c4)c5-(xx\cdot c4-c1s2s4)s5 \\ (-yy\cdot s4-s1s2c4)c5-(yy\cdot c4-s1s2s4)s5 \\ (c2c4-zz\cdot s4)c5-(c2s4+zz\cdot c4)s5 \end{bmatrix}, \tag{21}$$

$$\begin{bmatrix} ((-xx\cdot s4-c1s2c4)s5+(xx\cdot c4-c1s2s4)c5)s6+(c1cc2s3+s1c3)c6 \\ ((-yy\cdot s4-s1s2c4)s5+(yy\cdot c4-s1s2s4)s5)s6-(c1c3-s1c2s3)c6 \\ ((c2c4-zz\cdot s4)s5+(c2s4+zz\cdot c4)c5)s6+s2s3c6 \end{bmatrix}, \tag{22}$$

with

$$\left.\begin{aligned} xx &= c1c2c3-s1s3 \\ yy &= s1c2c3+c1s3 \\ zz &= s2c3 \end{aligned}\right\}. \tag{23}$$

When the HERMIES manipulator arm is used in conjunction with a platform, the previous development must be modified. This modification is due to the three added DOF. The first two DOF only change the position of the base of the arm and therefore do not change the orientation of the end-effector. However, the rotation of the platform effects orientation. Before rederiving the previous development, it is important to note that this rotation is in the same axis as the first DOF. It may therefore be coupled with the transformation for the first DOF $q1$ (i.e., around the z axis). A substitution can then be affected into the instances of $c1$ and $s1$. Additionally, the final three vectors must be added on: [0,0,0] for the translation and [0,0,1] for the rotation.

4.2.6. IKORv2.0 Complete

These modifications completed the IKORv2.0. There were many intermediate versions created throughout IKORv2.0's development. The version chart in Appendix B3, "Modification of IKOR, Version List," shows some of the contents and compatibilities during these intermediate stages.

5. CONCLUSIONS

During the trials and tests presented throughout this paper, the new method for IKOR is shown to be a successful and capable algorithm for autonomous redundant manipulators. Not only does it allow for obstacle and joint limit avoidance, but it does so at the relatively low time ratio of 2:1 compared to an optimized, nonconstrained Pseudo-Inverse code. As discussed in Sect. 3.1, this average timing scale is shadowed by the maximum time phenomenon that has yet to be fully determined. Both FSPv1.0 and FSPv2.0 have begun to pave the way for reducing the maximum time.[5,6]

The 3-D IKORv1.0 system was adapted to dynamically include both orientation and nonorientation control, as well as platform and nonplatform mobility. This system allowed for a solid testbed to compare the motions of different optimization criteria, as well as for testing other popular industry methods. The software package was also modified to allow for easier portability to other manipulators. Since the HERMIES manipulator is a research manipulator only, it will become increasingly necessary to port this algorithm to other manipulators. During this preliminary development, a joint weighting scheme was introduced that provides a mechanism to dynamically alter the contribution of any joint in the desired motion.

Performance and portability are the earmarks of IKORv2.0. With the introduction of the *one shot* method, both obstacle and joint limit avoidance can be accomplished in the same amount of time as the least norm motion alone. It was then possible to show that although it ran about twice longer than the Pseudo-Inverse, IKOR was able to treat constraints such as obstacles and joint limits avoidance during that same amount of time. Finally, with the introduction of the manipulator data file and streamlined subroutines, a feasible system of porting was introduced.

6. REFERENCES

1. F. G. Pin et al., "A New Method for the Inverse Kinematic Joint Velocity Calculations of Redundant Manipulators," pp. 96–102 in *Proceedings of the IEEE International Conference on Robotics and Automation*, San Diego, California, May 8–13, 1994.
2. F. G. Pin and F. A. Tulloch, "Resolving Kinematic Redundancy with Constraints Using the FSP (Full Space Parameterization) Approach," pp. 1897–1902 in *Proceedings of the IEEE International Conference on Robotics and Automation*, Minneapolis, Minnesota, April 22–28, 1996, Vol. 2.
3. D. Carlson, *The Implementation of CESAR's New Redundancy Resolution Algorithm on a 7 D.O.F. Redundant Manipulator*, ORNL/SERS report, May 1993.
4. D. Boozer and C. Lupo, *Applying a New Path Planning Algorithm to a Manipulator with Two Degrees of Redundancy*, ORNL/SERS report, December 1993.
5. K. A. Morgansen and F. G. Pin, *Enhanced Code for the Full Space Parameterization Approach to Solving Underspecified Systems of Algebraic Equations, Version 1.0*, ORNL/TM-12816, Martin Marietta Energy Systems, Inc., Oak Ridge Nat. Lab., 1995.
6. G. A. Fries, C. J. Hacker, and F. G. Pin, *FSP (Full Space Parameterization) Version 2.0*, ORNL/TM-13021, Martin Marietta Energy Systems, Inc., Oak Ridge Nat. Lab., 1995.
7. R. Brockett, "On the Computer Control of Movement," pp. 534–40 in *Proceedings of the IEEE Conference on Robotics and Automation*, Philadelphia, Pennsylvania, April 24–29, 1988.
8. B. Siciliano, "Kinematic Control of Redundant Robot Manipulators: A Tutorial," *Journal of Intelligent and Robotic Systems* **3**, 201–212 (1990).
9. J. Baillieul, "Resolution of Kinematic Redundancy," *Notices of the American Mathematical Society* **36**(8) (October 1989).
10. K. E. Kendall and J. E. Kendall, *Systems Analysis and Design*, 2d ed., Prentice-Hall, New Jersey, 1988.
11. T. E. Deeter et al., "The Next Generation Munitions Handler Advanced Telerobotics Technology Demonstrator Program," in *Proceedings of ISRAM '96, Sixth International Symposium on Robotics and Manufacturing*, Montpellier, France, May 27–30, 1996.
12. F. G. Pin et al., "Motion Planning for Mobile Manipulators with a Non-Holonomic Constraint Using the FSP (Full Space Parameterization) Method," *Journal of Robotic Systems* **13**(11), 723–36, 1996.
13. W. H. Press et al., *Numerical Recipes in C*, 2d ed., Cambridge University Press, New York, 1992.

APPENDIXES

Appendix A contains some of the data collected during the overall testing program. Appendix A1 (a–d) contains four trajectories on which FSP was tested against the Pseudo-Inverse for each of the four HERMIES configurations (3×10, 3×7, 6×7, and 6×10). Appendix A2 contains information on how these timing studies were performed, as well as the data taken from these studies.

Appendix B contains the User's Guide. Beginning with push button directions on the use of IKOR, followed by suggestions on maintaining documentation and by comments on using some of the "sticky" aspects of the code effectively, this appendix was designed to provide the user with key information about the design and use of the system. Also included is a version chart and a compilation guide.

Appendix C contains the data flow diagrams and the data element dictionary for the system. All major data flows are presented in the diagrams between four entities involved with the system. These entities are the USER, DESIGN ENGINEERS, IKOR, and FSP.

Appendix D includes all of the code for the system, arranged in subcategories and briefly discussed.

Appendix A: Least Norm Compatibility and Efficiency Studies

This appendix contains the data and the data gathering techniques for the testing program. Appendix A1 discusses IKOR/FSP's comparisons with the Pseudo-Inverse code that can be found in Ref. 13. Test after test proved the compatibility and identical results of the two methods, and three trajectories are presented and discussed in detail here to illustrate this fact. Appendix A is organized as follows:

For sake of brevity, least norm is used here to denote the sum of the squares of the requested change in joint values. First, a graphical representation of the least norm and IKOR/FSP compatibility is presented. The first graph for each trajectory section is the performance of IKOR/FSP, while the second graph is the performance of the Moore Penrose Pseudo-Inverse. An interesting trend is shown by the use of orientation control. In general, orientation control increases the least norm values. Also, without platform capabilities, the joints are required to make larger motions. When coupled with orientation control, these plots would not fit on the same scale and are therefore removed from the graphical representation (e.g., for trajectory #2, 6×7 case, the maximum least norm value is 1,472). It should be noted that these plots show identical behavior between the Pseudo-Inverse and the FSP, and that the HERMIES arm behaves within its parameters, without singularities for all trajectories and in all configurations.

Next, the data gathered is presented for the trajectories. For each trajectory, four configurations of the HERMIES arm were tested. Each configuration's data set is presented. For each time step, the step's number, the least norm values for the joint angles produced by FSP, the least norm values for the joint angles produced by the Pseudo-Inverse, and their differences are presented respectively. At the end of each configuration's data, a summary is presented. The summary includes the largest and the smallest least norm value found in the data set, the largest error between FSP and Pseudo-Inverse encountered in the data set, the sum of the differences and the percentage of the error. Additionally, the time step number where each of these conditions were found is shown.

Appendix A2 discusses IKOR/FSP's timing runs comparisons with the Pseudo-Inverse. Two studies were performed on each of three trajectories. Also included in this section is the technique used to time the systems.

Appendix A1: Least Norm Compatibility
Graphical Data Display
Trajectory #1

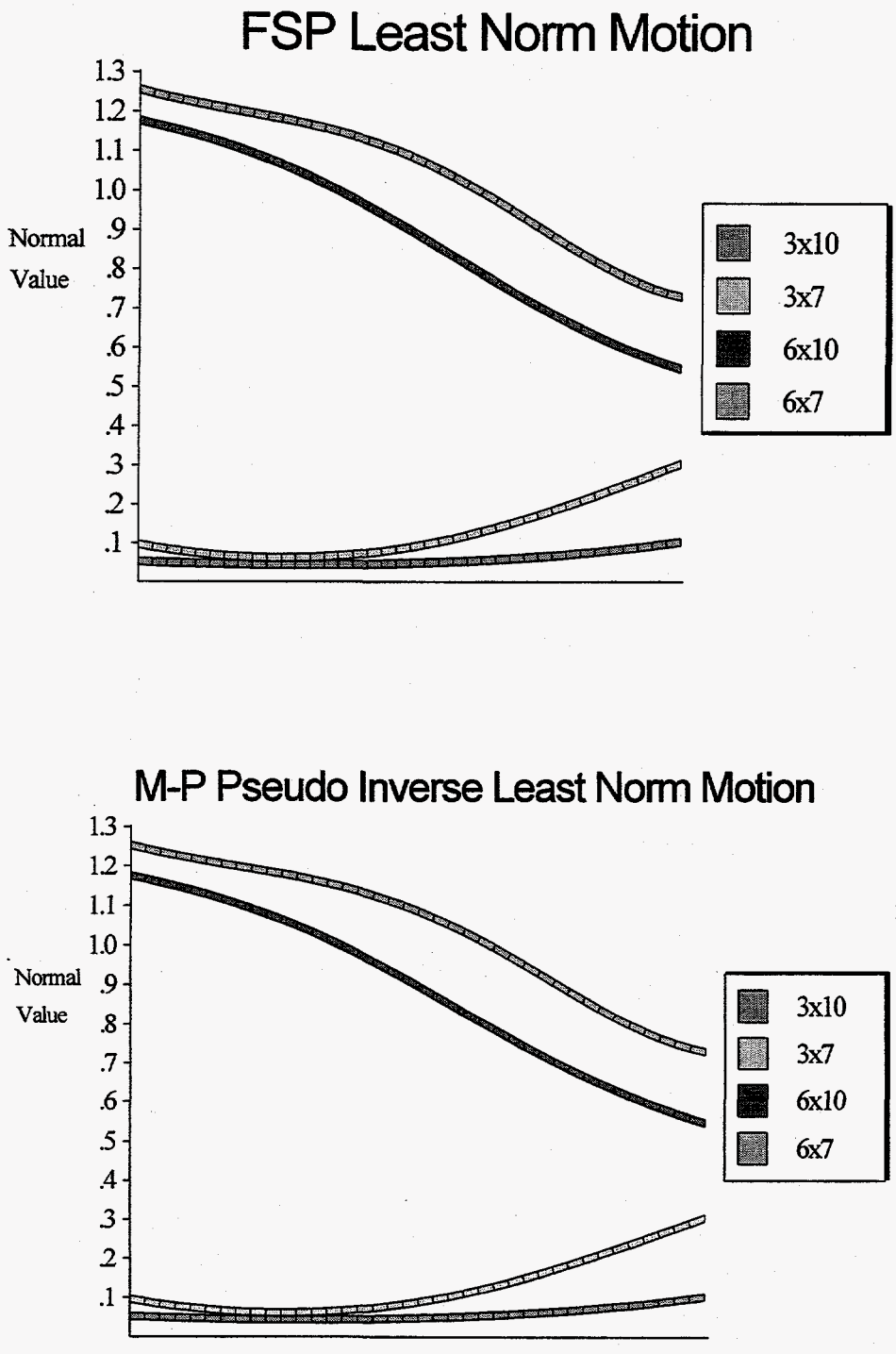
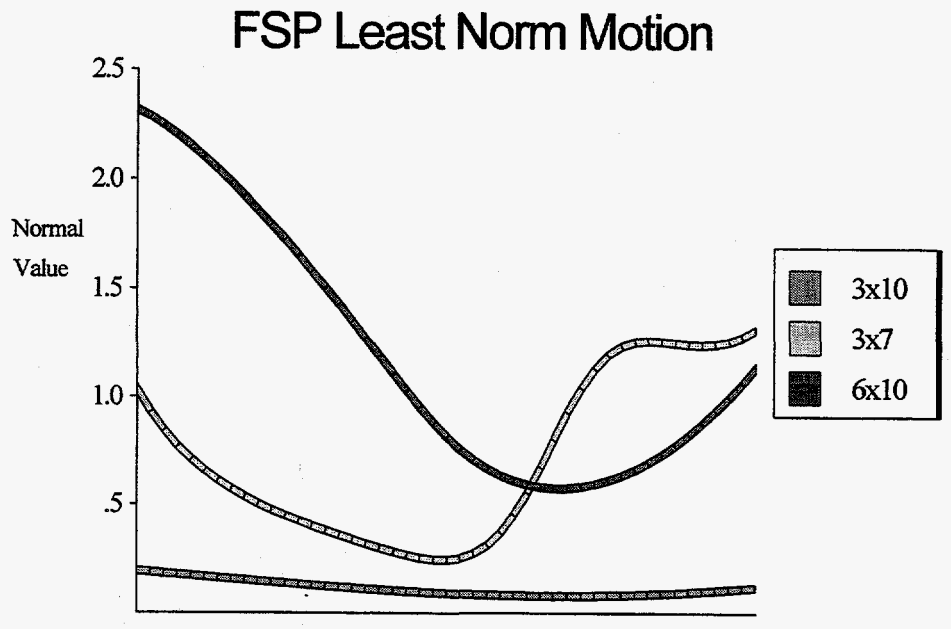


Fig. A1. Least Norm FSP vs. Pseudo Inverse Trajectory #1.

Appendix A1: Least Norm Compatibility
Trajectory #2



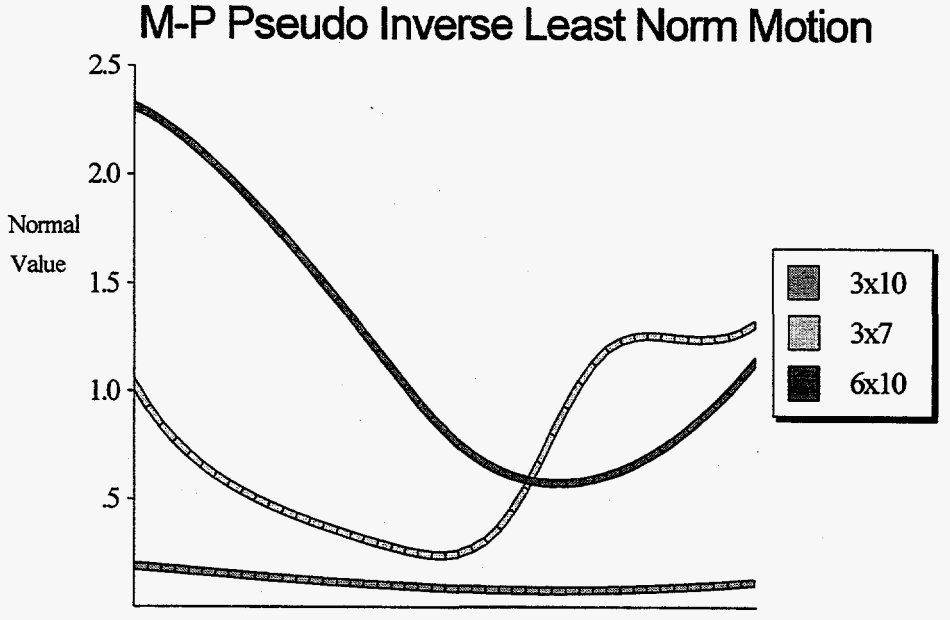
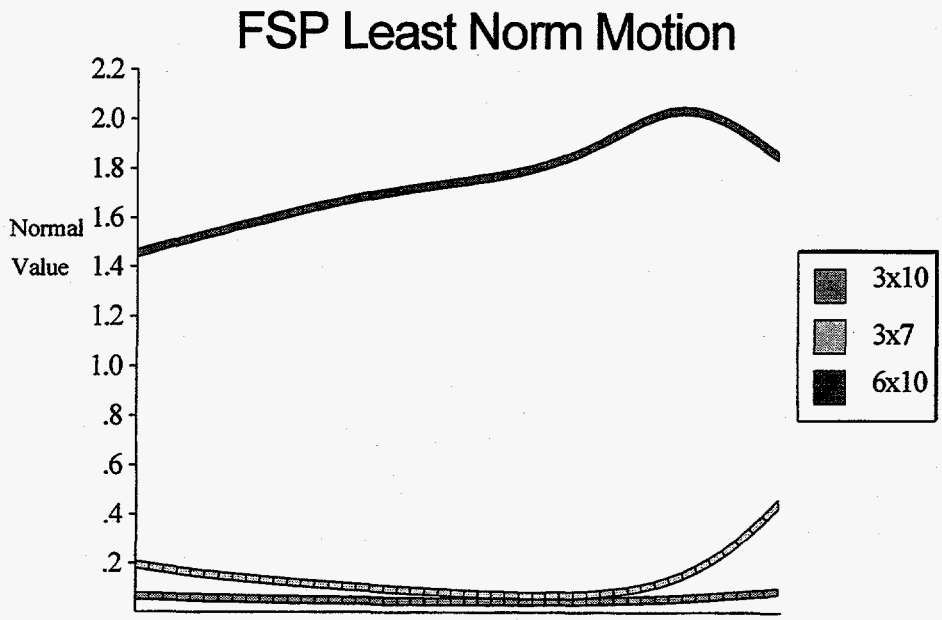


Fig. A2. Least Norm FSP vs. Pseudo Inverse Trajectory #2.

Appendix A1: Least Norm Compatibility
Trajectory #3



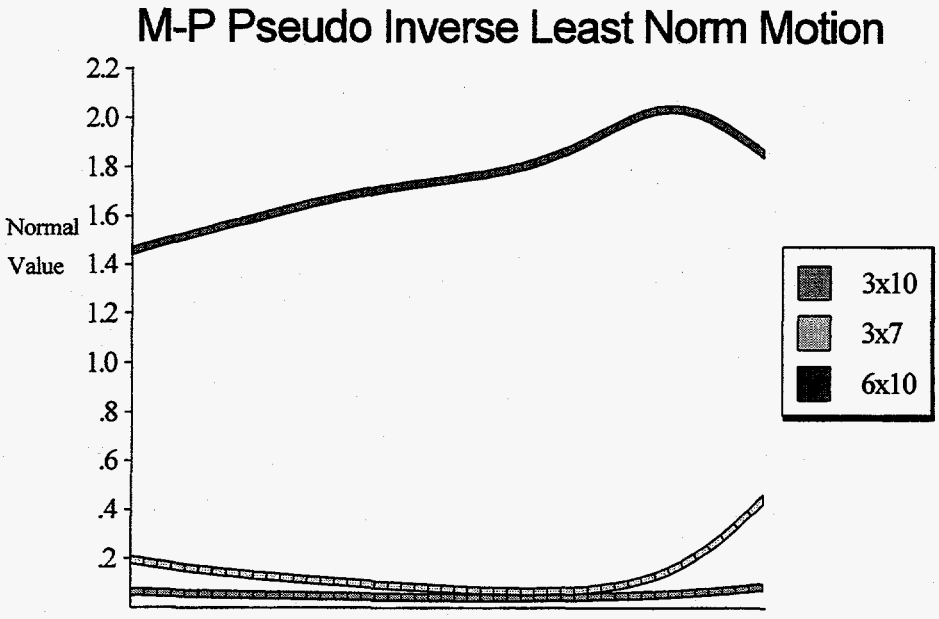


Fig. A3. Least Norm FSP vs. Pseudo Inverse Trajectory #3.

Trajectory #1 3x10

:::::::: Norms for 3x10 Traj :#1 ::::::::

----------differences: ----------

STEP	FSP		Pseudo		diff
# 1#	0.055882	-	0.055882	=	0.000000
# 2#	0.051797	-	0.051797	=	0.000000
# 3#	0.051686	-	0.051686	=	0.000000
# 4#	0.051375	-	0.051375	=	0.000000
# 5#	0.051095	-	0.051095	=	0.000000
# 6#	0.050813	-	0.050813	=	0.000000
# 7#	0.050556	-	0.050556	=	0.000000
# 8#	0.050272	-	0.050272	=	0.000000
# 9#	0.050004	-	0.050004	=	0.000000
# 10#	0.049755	-	0.049755	=	0.000000
# 11#	0.049508	-	0.049508	=	0.000000
# 12#	0.049279	-	0.049279	=	0.000000
# 13#	0.049021	-	0.049020	=	0.000001
# 14#	0.048790	-	0.048790	=	0.000000
# 15#	0.048556	-	0.048556	=	0.000000
# 16#	0.048342	-	0.048342	=	0.000000
# 17#	0.048146	-	0.048145	=	0.000001
# 18#	0.047914	-	0.047914	=	0.000000
# 19#	0.047713	-	0.047713	=	0.000000
# 20#	0.047508	-	0.047508	=	0.000000
# 21#	0.047306	-	0.047306	=	0.000000
# 22#	0.047150	-	0.047150	=	0.000000
# 23#	0.046945	-	0.046945	=	0.000000
# 24#	0.046767	-	0.046767	=	0.000000
# 25#	0.046595	-	0.046597	=	-0.000002
# 26#	0.046443	-	0.046444	=	-0.000001
# 27#	0.046260	-	0.046260	=	0.000000
# 28#	0.046113	-	0.046114	=	-0.000001
# 29#	0.045958	-	0.045960	=	-0.000002
# 30#	0.045810	-	0.045811	=	-0.000001
# 31#	0.045693	-	0.045692	=	0.000001
# 32#	0.045537	-	0.045538	=	-0.000001
# 33#	0.045402	-	0.045401	=	0.000001
# 34#	0.045285	-	0.045286	=	-0.000001
# 35#	0.045159	-	0.045159	=	0.000000
# 36#	0.045069	-	0.045069	=	0.000000
# 37#	0.044941	-	0.044940	=	0.000001
# 38#	0.044830	-	0.044834	=	-0.000004
# 39#	0.044736	-	0.044736	=	0.000000
# 40#	0.044659	-	0.044658	=	0.000001
# 41#	0.044556	-	0.044556	=	0.000000
# 42#	0.044470	-	0.044473	=	-0.000003
# 43#	0.044393	-	0.044394	=	-0.000001
# 44#	0.044313	-	0.044315	=	-0.000002
# 45#	0.044270	-	0.044269	=	0.000001
# 46#	0.044186	-	0.044190	=	-0.000004
# 47#	0.044135	-	0.044131	=	0.000004
# 48#	0.044078	-	0.044078	=	0.000000
# 49#	0.044025	-	0.044026	=	-0.000001
# 50#	0.044001	-	0.044003	=	-0.000002
# 51#	0.043951	-	0.043950	=	0.000001
# 52#	0.043919	-	0.043919	=	0.000000
# 53#	0.043887	-	0.043892	=	-0.000005

# 54#	0.043865 -	0.043863 =	0.000002
# 55#	0.043867 -	0.043867 =	0.000000
# 56#	0.043836 -	0.043835 =	0.000001
# 57#	0.043831 -	0.043830 =	0.000001
# 58#	0.043828 -	0.043827 =	0.000001
# 59#	0.043848 -	0.043848 =	0.000000
# 60#	0.043840 -	0.043843 =	-0.000003
# 61#	0.043855 -	0.043852 =	0.000003
# 62#	0.043866 -	0.043871 =	-0.000005
# 63#	0.043892 -	0.043894 =	-0.000002
# 64#	0.043943 -	0.043941 =	0.000002
# 65#	0.043961 -	0.043957 =	0.000004
# 66#	0.043996 -	0.043999 =	-0.000003
# 67#	0.044043 -	0.044042 =	0.000001
# 68#	0.044084 -	0.044081 =	0.000003
# 69#	0.044163 -	0.044163 =	0.000000
# 70#	0.044204 -	0.044202 =	0.000002
# 71#	0.044270 -	0.044270 =	0.000000
# 72#	0.044333 -	0.044333 =	0.000000
# 73#	0.044411 -	0.044411 =	0.000000
# 74#	0.044515 -	0.044514 =	0.000001
# 75#	0.044577 -	0.044577 =	0.000000
# 76#	0.044670 -	0.044670 =	0.000000
# 77#	0.044759 -	0.044760 =	-0.000001
# 78#	0.044887 -	0.044888 =	-0.000001
# 79#	0.044976 -	0.044974 =	0.000002
# 80#	0.045085 -	0.045087 =	-0.000002
# 81#	0.045203 -	0.045206 =	-0.000003
# 82#	0.045322 -	0.045324 =	-0.000002
# 83#	0.045474 -	0.045472 =	0.000002
# 84#	0.045589 -	0.045585 =	0.000004
# 85#	0.045726 -	0.045727 =	-0.000001
# 86#	0.045864 -	0.045867 =	-0.000003
# 87#	0.046023 -	0.046024 =	-0.000001
# 88#	0.046194 -	0.046191 =	0.000003
# 89#	0.046334 -	0.046340 =	-0.000006
# 90#	0.046503 -	0.046505 =	-0.000002
# 91#	0.046671 -	0.046669 =	0.000002
# 92#	0.046872 -	0.046872 =	0.000000
# 93#	0.047038 -	0.047034 =	0.000004
# 94#	0.047223 -	0.047225 =	-0.000002
# 95#	0.047414 -	0.047413 =	0.000001
# 96#	0.047622 -	0.047620 =	0.000002
# 97#	0.047849 -	0.047845 =	0.000004
# 98#	0.048038 -	0.048038 =	0.000000
# 99#	0.048258 -	0.048260 =	-0.000002
#100#	0.048473 -	0.048472 =	0.000001
#101#	0.048712 -	0.048709 =	0.000003
#102#	0.048962 -	0.048964 =	-0.000002
#103#	0.049188 -	0.049188 =	0.000000
#104#	0.049431 -	0.049437 =	-0.000006
#105#	0.049676 -	0.049675 =	0.000001
#106#	0.049947 -	0.049944 =	0.000003
#107#	0.050230 -	0.050229 =	0.000001
#108#	0.050485 -	0.050486 =	-0.000001
#109#	0.050753 -	0.050753 =	0.000000
#110#	0.051042 -	0.051042 =	0.000000
#111#	0.051355 -	0.051354 =	0.000001
#112#	0.051630 -	0.051630 =	0.000000
#113#	0.051932 -	0.051932 =	0.000000

#114#	0.052240	-	0.052240	= 0.000000
#115#	0.052558	-	0.052556	= 0.000002
#116#	0.052902	-	0.052902	= 0.000000
#117#	0.053210	-	0.053211	= -0.000001
#118#	0.053545	-	0.053548	= -0.000003
#119#	0.053882	-	0.053885	= -0.000003
#120#	0.054244	-	0.054241	= 0.000003
#121#	0.054610	-	0.054612	= -0.000002
#122#	0.054953	-	0.054958	= -0.000005
#123#	0.055321	-	0.055317	= 0.000004
#124#	0.055701	-	0.055703	= -0.000002
#125#	0.056088	-	0.056085	= 0.000003
#126#	0.056506	-	0.056504	= 0.000002
#127#	0.056874	-	0.056874	= 0.000000
#128#	0.057274	-	0.057275	= -0.000001
#129#	0.057697	-	0.057695	= 0.000002
#130#	0.058132	-	0.058132	= 0.000000
#131#	0.058539	-	0.058541	= -0.000002
#132#	0.058984	-	0.058980	= 0.000004
#133#	0.059407	-	0.059409	= -0.000002
#134#	0.059864	-	0.059865	= -0.000001
#135#	0.060344	-	0.060346	= -0.000002
#136#	0.060790	-	0.060789	= 0.000001
#137#	0.061258	-	0.061255	= 0.000003
#138#	0.061745	-	0.061748	= -0.000003
#139#	0.062234	-	0.062233	= 0.000001
#140#	0.062754	-	0.062750	= 0.000004
#141#	0.063226	-	0.063236	= -0.000010
#142#	0.063735	-	0.063737	= -0.000002
#143#	0.064283	-	0.064269	= 0.000014
#144#	0.064788	-	0.064822	= -0.000034
#145#	0.065373	-	0.065340	= 0.000033
#146#	0.065881	-	0.065883	= -0.000002
#147#	0.066401	-	0.066433	= -0.000032
#148#	0.067024	-	0.067005	= 0.000019
#149#	0.067617	-	0.067609	= 0.000008
#150#	0.068156	-	0.068161	= -0.000005
#151#	0.068737	-	0.068735	= 0.000002
#152#	0.069357	-	0.069351	= 0.000006
#153#	0.069958	-	0.069953	= 0.000005
#154#	0.070604	-	0.070603	= 0.000001
#155#	0.071215	-	0.071218	= -0.000003
#156#	0.071825	-	0.071835	= -0.000010
#157#	0.072490	-	0.072482	= 0.000008
#158#	0.073149	-	0.073146	= 0.000003
#159#	0.073838	-	0.073839	= -0.000001
#160#	0.074484	-	0.074482	= 0.000002
#161#	0.075173	-	0.075168	= 0.000005
#162#	0.075871	-	0.075868	= 0.000003
#163#	0.076600	-	0.076606	= -0.000006
#164#	0.077291	-	0.077296	= -0.000005
#165#	0.078005	-	0.078003	= 0.000002
#166#	0.078755	-	0.078755	= 0.000000
#167#	0.079507	-	0.079512	= -0.000005
#168#	0.080289	-	0.080293	= -0.000004
#169#	0.081033	-	0.081027	= 0.000006
#170#	0.081799	-	0.081798	= 0.000001
#171#	0.082599	-	0.082602	= -0.000003
#172#	0.083398	-	0.083401	= -0.000003
#173#	0.084258	-	0.084259	= -0.000001

```
#174#          0.085026 -          0.085021 =          0.000005
#175#          0.085873 -          0.085878 =         -0.000005
#176#          0.086721 -          0.086719 =          0.000002
#177#          0.087580 -          0.087581 =         -0.000001
#178#          0.088487 -          0.088484 =          0.000003
#179#          0.089334 -          0.089331 =          0.000003
#180#          0.090222 -          0.090234 =         -0.000012
#181#          0.091137 -          0.091138 =         -0.000001
#182#          0.092089 -          0.092086 =          0.000003
#183#          0.092983 -          0.092988 =         -0.000005
#184#          0.093922 -          0.093927 =         -0.000005
#185#          0.094881 -          0.094881 =          0.000000
#186#          0.095848 -          0.095849 =         -0.000001
#187#          0.096872 -          0.096877 =         -0.000005
#188#          0.097815 -          0.097815 =          0.000000
#189#          0.098826 -          0.098828 =         -0.000002
#190#          0.099868 -          0.099858 =          0.000010
#191#          0.100883 -          0.100883 =          0.000000
#192#          0.101967 -          0.101968 =         -0.000001
#193#          0.102975 -          0.102973 =          0.000002
#194#          0.104054 -          0.104058 =         -0.000004
#195#          0.105140 -          0.105137 =          0.000003
Largest : 0.105140 @ line #195#
Smallest: 0.043827 @ line # 58#
Error   : 0.000034 @ line #144#
Error Ratio: 0.05%
```

Trajectory #1 3x7

```
::::::::: Norms for 3x7 Traj :#1 :::::::::
---------------differences: -----------------
STEP              FSP                  Pseudo                 diff

#  1#          0.103820 -           0.103820 =           0.000000
#  2#          0.095774 -           0.095774 =           0.000000
#  3#          0.094632 -           0.094632 =           0.000000
#  4#          0.093265 -           0.093265 =           0.000000
#  5#          0.091931 -           0.091931 =           0.000000
#  6#          0.090619 -           0.090619 =           0.000000
#  7#          0.089369 -           0.089369 =           0.000000
#  8#          0.088083 -           0.088083 =           0.000000
#  9#          0.086824 -           0.086824 =           0.000000
# 10#          0.085673 -           0.085673 =           0.000000
# 11#          0.084508 -           0.084508 =           0.000000
# 12#          0.083390 -           0.083390 =           0.000000
# 13#          0.082258 -           0.082258 =           0.000000
# 14#          0.081177 -           0.081177 =           0.000000
# 15#          0.080094 -           0.080094 =           0.000000
# 16#          0.079104 -           0.079104 =           0.000000
# 17#          0.078120 -           0.078116 =           0.000004
# 18#          0.077141 -           0.077145 =          -0.000004
# 19#          0.076191 -           0.076189 =           0.000002
# 20#          0.075287 -           0.075286 =           0.000001
# 21#          0.074358 -           0.074361 =          -0.000003
# 22#          0.073568 -           0.073565 =           0.000003
# 23#          0.072705 -           0.072708 =          -0.000003
# 24#          0.071909 -           0.071909 =           0.000000
# 25#          0.071129 -           0.071131 =          -0.000002
# 26#          0.070402 -           0.070402 =           0.000000
# 27#          0.069642 -           0.069640 =           0.000002
# 28#          0.068975 -           0.068972 =           0.000003
# 29#          0.068304 -           0.068303 =           0.000001
# 30#          0.067664 -           0.067669 =          -0.000005
# 31#          0.067082 -           0.067083 =          -0.000001
# 32#          0.066483 -           0.066481 =           0.000002
# 33#          0.065903 -           0.065902 =           0.000001
# 34#          0.065401 -           0.065399 =           0.000002
# 35#          0.064890 -           0.064891 =          -0.000001
# 36#          0.064448 -           0.064450 =          -0.000002
# 37#          0.063986 -           0.063981 =           0.000005
# 38#          0.063545 -           0.063553 =          -0.000008
# 39#          0.063177 -           0.063174 =           0.000003
# 40#          0.062827 -           0.062829 =          -0.000002
# 41#          0.062480 -           0.062485 =          -0.000005
# 42#          0.062176 -           0.062173 =           0.000003
# 43#          0.061898 -           0.061896 =           0.000002
# 44#          0.061625 -           0.061623 =           0.000002
# 45#          0.061441 -           0.061442 =          -0.000001
# 46#          0.061230 -           0.061231 =          -0.000001
# 47#          0.061061 -           0.061061 =           0.000000
# 48#          0.060927 -           0.060922 =           0.000005
# 49#          0.060798 -           0.060799 =          -0.000001
# 50#          0.060740 -           0.060742 =          -0.000002
# 51#          0.060674 -           0.060676 =          -0.000002
# 52#          0.060645 -           0.060645 =           0.000000
```

# 53#	0.060643	-	0.060643	= 0.000000
# 54#	0.060659	-	0.060660	= -0.000001
# 55#	0.060748	-	0.060747	= 0.000001
# 56#	0.060812	-	0.060810	= 0.000002
# 57#	0.060927	-	0.060928	= -0.000001
# 58#	0.061051	-	0.061054	= -0.000003
# 59#	0.061258	-	0.061257	= 0.000001
# 60#	0.061431	-	0.061431	= 0.000000
# 61#	0.061658	-	0.061659	= -0.000001
# 62#	0.061914	-	0.061914	= 0.000000
# 63#	0.062181	-	0.062179	= 0.000002
# 64#	0.062518	-	0.062518	= 0.000000
# 65#	0.062843	-	0.062842	= 0.000001
# 66#	0.063205	-	0.063209	= -0.000004
# 67#	0.063602	-	0.063601	= 0.000001
# 68#	0.064017	-	0.064016	= 0.000001
# 69#	0.064490	-	0.064491	= -0.000001
# 70#	0.064955	-	0.064956	= -0.000001
# 71#	0.065462	-	0.065464	= -0.000002
# 72#	0.065976	-	0.065980	= -0.000004
# 73#	0.066569	-	0.066574	= -0.000005
# 74#	0.067177	-	0.067171	= 0.000006
# 75#	0.067772	-	0.067776	= -0.000004
# 76#	0.068429	-	0.068424	= 0.000005
# 77#	0.069081	-	0.069079	= 0.000002
# 78#	0.069822	-	0.069830	= -0.000008
# 79#	0.070555	-	0.070554	= 0.000001
# 80#	0.071300	-	0.071302	= -0.000002
# 81#	0.072097	-	0.072098	= -0.000001
# 82#	0.072893	-	0.072886	= 0.000007
# 83#	0.073779	-	0.073777	= 0.000002
# 84#	0.074637	-	0.074643	= -0.000006
# 85#	0.075555	-	0.075554	= 0.000001
# 86#	0.076444	-	0.076440	= 0.000004
# 87#	0.077424	-	0.077430	= -0.000006
# 88#	0.078440	-	0.078436	= 0.000004
# 89#	0.079426	-	0.079427	= -0.000001
# 90#	0.080477	-	0.080475	= 0.000002
# 91#	0.081533	-	0.081536	= -0.000003
# 92#	0.082665	-	0.082663	= 0.000002
# 93#	0.083772	-	0.083769	= 0.000003
# 94#	0.084918	-	0.084923	= -0.000005
# 95#	0.086076	-	0.086076	= 0.000000
# 96#	0.087310	-	0.087307	= 0.000003
# 97#	0.088598	-	0.088598	= 0.000000
# 98#	0.089816	-	0.089815	= 0.000001
# 99#	0.091110	-	0.091105	= 0.000005
#100#	0.092388	-	0.092385	= 0.000003
#101#	0.093763	-	0.093764	= -0.000001
#102#	0.095171	-	0.095171	= 0.000000
#103#	0.096568	-	0.096565	= 0.000003
#104#	0.097976	-	0.097970	= 0.000006
#105#	0.099377	-	0.099383	= -0.000006
#106#	0.100904	-	0.100909	= -0.000005
#107#	0.102438	-	0.102429	= 0.000009
#108#	0.103979	-	0.103979	= 0.000000
#109#	0.105462	-	0.105454	= 0.000008
#110#	0.107089	-	0.107089	= 0.000000
#111#	0.108721	-	0.108724	= -0.000003
#112#	0.110339	-	0.110340	= -0.000001

#113#	0.111995	-	0.111993	= 0.000002
#114#	0.113669	-	0.113675	= -0.000006
#115#	0.115415	-	0.115404	= 0.000011
#116#	0.117168	-	0.117168	= 0.000000
#117#	0.118909	-	0.118916	= -0.000007
#118#	0.120700	-	0.120685	= 0.000015
#119#	0.122464	-	0.122468	= -0.000004
#120#	0.124382	-	0.124386	= -0.000004
#121#	0.126238	-	0.126245	= -0.000007
#122#	0.128110	-	0.128099	= 0.000011
#123#	0.129953	-	0.129953	= 0.000000
#124#	0.131938	-	0.131945	= -0.000007
#125#	0.133895	-	0.133906	= -0.000011
#126#	0.135974	-	0.135960	= 0.000014
#127#	0.137888	-	0.137886	= 0.000002
#128#	0.139858	-	0.139873	= -0.000015
#129#	0.141968	-	0.141960	= 0.000008
#130#	0.144068	-	0.144070	= -0.000002
#131#	0.146139	-	0.146139	= 0.000000
#132#	0.148288	-	0.148291	= -0.000003
#133#	0.150329	-	0.150333	= -0.000004
#134#	0.152539	-	0.152528	= 0.000011
#135#	0.154769	-	0.154775	= -0.000006
#136#	0.156919	-	0.156918	= 0.000001
#137#	0.159083	-	0.159087	= -0.000004
#138#	0.161419	-	0.161431	= -0.000012
#139#	0.163643	-	0.163633	= 0.000010
#140#	0.165966	-	0.165967	= -0.000001
#141#	0.168218	-	0.168218	= 0.000000
#142#	0.170482	-	0.170482	= 0.000000
#143#	0.172917	-	0.172911	= 0.000006
#144#	0.175262	-	0.175252	= 0.000010
#145#	0.177590	-	0.177599	= -0.000009
#146#	0.179984	-	0.179980	= 0.000004
#147#	0.182316	-	0.182323	= -0.000007
#148#	0.184803	-	0.184813	= -0.000010
#149#	0.187344	-	0.187343	= 0.000001
#150#	0.189698	-	0.189687	= 0.000011
#151#	0.192095	-	0.192099	= -0.000004
#152#	0.194642	-	0.194627	= 0.000015
#153#	0.197130	-	0.197126	= 0.000004
#154#	0.199692	-	0.199707	= -0.000015
#155#	0.202243	-	0.202241	= 0.000002
#156#	0.204624	-	0.204630	= -0.000006
#157#	0.207274	-	0.207262	= 0.000012
#158#	0.209824	-	0.209818	= 0.000006
#159#	0.212445	-	0.212449	= -0.000004
#160#	0.215022	-	0.215003	= 0.000019
#161#	0.217558	-	0.217563	= -0.000005
#162#	0.220191	-	0.220205	= -0.000014
#163#	0.222876	-	0.222878	= -0.000002
#164#	0.225456	-	0.225459	= -0.000003
#165#	0.228008	-	0.227993	= 0.000015
#166#	0.230742	-	0.230752	= -0.000010
#167#	0.233482	-	0.233467	= 0.000015
#168#	0.236123	-	0.236139	= -0.000016
#169#	0.238749	-	0.238753	= -0.000004
#170#	0.241339	-	0.241361	= -0.000022
#171#	0.244172	-	0.244132	= 0.000040
#172#	0.246826	-	0.246844	= -0.000018

```
#173#          0.249688 -          0.249671 =          0.000017
#174#          0.252190 -          0.252201 =         -0.000011
#175#          0.254985 -          0.254987 =         -0.000002
#176#          0.257748 -          0.257737 =          0.000011
#177#          0.260501 -          0.260495 =          0.000006
#178#          0.263286 -          0.263308 =         -0.000022
#179#          0.265960 -          0.265958 =          0.000002
#180#          0.268757 -          0.268770 =         -0.000013
#181#          0.271549 -          0.271546 =          0.000003
#182#          0.274382 -          0.274391 =         -0.000009
#183#          0.277117 -          0.277102 =          0.000015
#184#          0.279901 -          0.279901 =          0.000000
#185#          0.282698 -          0.282710 =         -0.000012
#186#          0.285544 -          0.285540 =          0.000004
#187#          0.288470 -          0.288449 =          0.000021
#188#          0.291097 -          0.291091 =          0.000006
#189#          0.294056 -          0.294068 =         -0.000012
#190#          0.296980 -          0.296988 =         -0.000008
#191#          0.299781 -          0.299776 =          0.000005
#192#          0.302739 -          0.302734 =          0.000005
#193#          0.305482 -          0.305471 =          0.000011
#194#          0.308482 -          0.308458 =          0.000024
#195#          0.311383 -          0.311400 =         -0.000017
Largest : 0.311400 @ line #195#
Smallest: 0.060643 @ line # 53#
Error   : 0.000040 @ line #171#
Error Ratio: 0.02%
```

Trajectory #1 6x7

:::::::: Norms for 6x7 Traj :#1 ::::::::

----------differences: ----------

STEP	FSP		Pseudo		diff
# 1#	1.261517	-	1.261517	=	0.000000
# 2#	1.250471	-	1.250471	=	0.000000
# 3#	1.248612	-	1.248612	=	0.000000
# 4#	1.246483	-	1.246485	=	-0.000002
# 5#	1.244665	-	1.244665	=	0.000000
# 6#	1.242662	-	1.242661	=	0.000001
# 7#	1.241042	-	1.241042	=	0.000000
# 8#	1.239136	-	1.239137	=	-0.000001
# 9#	1.237312	-	1.237318	=	-0.000006
# 10#	1.235632	-	1.235629	=	0.000003
# 11#	1.233860	-	1.233854	=	0.000006
# 12#	1.232387	-	1.232386	=	0.000001
# 13#	1.230658	-	1.230656	=	0.000002
# 14#	1.228998	-	1.228999	=	-0.000001
# 15#	1.227426	-	1.227428	=	-0.000002
# 16#	1.225846	-	1.225846	=	0.000000
# 17#	1.224560	-	1.224555	=	0.000005
# 18#	1.222945	-	1.222943	=	0.000002
# 19#	1.221471	-	1.221483	=	-0.000012
# 20#	1.220035	-	1.220036	=	-0.000001
# 21#	1.218404	-	1.218411	=	-0.000007
# 22#	1.217328	-	1.217316	=	0.000012
# 23#	1.215744	-	1.215746	=	-0.000002
# 24#	1.214417	-	1.214418	=	-0.000001
# 25#	1.213068	-	1.213067	=	0.000001
# 26#	1.211734	-	1.211748	=	-0.000014
# 27#	1.210342	-	1.210328	=	0.000014
# 28#	1.209080	-	1.209084	=	-0.000004
# 29#	1.207796	-	1.207782	=	0.000014
# 30#	1.206442	-	1.206450	=	-0.000008
# 31#	1.205179	-	1.205163	=	0.000016
# 32#	1.203914	-	1.203935	=	-0.000021
# 33#	1.202485	-	1.202494	=	-0.000009
# 34#	1.201339	-	1.201344	=	-0.000005
# 35#	1.200079	-	1.200054	=	0.000025
# 36#	1.198790	-	1.198803	=	-0.000013
# 37#	1.197527	-	1.197526	=	0.000001
# 38#	1.196249	-	1.196243	=	0.000006
# 39#	1.194991	-	1.194992	=	-0.000001
# 40#	1.193803	-	1.193811	=	-0.000008
# 41#	1.192344	-	1.192347	=	-0.000003
# 42#	1.191102	-	1.191109	=	-0.000007
# 43#	1.189880	-	1.189869	=	0.000011
# 44#	1.188550	-	1.188543	=	0.000007
# 45#	1.187281	-	1.187284	=	-0.000003
# 46#	1.185983	-	1.185966	=	0.000017
# 47#	1.184695	-	1.184691	=	0.000004
# 48#	1.183324	-	1.183334	=	-0.000010
# 49#	1.181976	-	1.181980	=	-0.000004
# 50#	1.180582	-	1.180578	=	0.000004
# 51#	1.179208	-	1.179190	=	0.000018
# 52#	1.177896	-	1.177892	=	0.000004
# 53#	1.176436	-	1.176476	=	-0.000040

# 54#	1.175012	-	1.174998	= 0.000014
# 55#	1.173606	-	1.173605	= 0.000001
# 56#	1.172107	-	1.172115	= -0.000008
# 57#	1.170670	-	1.170658	= 0.000012
# 58#	1.169134	-	1.169139	= -0.000005
# 59#	1.167719	-	1.167720	= -0.000001
# 60#	1.165965	-	1.165946	= 0.000019
# 61#	1.164456	-	1.164473	= -0.000017
# 62#	1.162850	-	1.162853	= -0.000003
# 63#	1.161212	-	1.161208	= 0.000004
# 64#	1.159726	-	1.159721	= 0.000005
# 65#	1.157798	-	1.157794	= 0.000004
# 66#	1.156178	-	1.156174	= 0.000004
# 67#	1.154472	-	1.154458	= 0.000014
# 68#	1.152565	-	1.152567	= -0.000002
# 69#	1.150928	-	1.150932	= -0.000004
# 70#	1.148849	-	1.148836	= 0.000013
# 71#	1.147104	-	1.147105	= -0.000001
# 72#	1.145128	-	1.145118	= 0.000010
# 73#	1.143202	-	1.143224	= -0.000022
# 74#	1.141320	-	1.141316	= 0.000004
# 75#	1.139076	-	1.139083	= -0.000007
# 76#	1.137154	-	1.137147	= 0.000007
# 77#	1.134957	-	1.134974	= -0.000017
# 78#	1.132928	-	1.132936	= -0.000008
# 79#	1.130652	-	1.130669	= -0.000017
# 80#	1.128238	-	1.128227	= 0.000011
# 81#	1.126111	-	1.126128	= -0.000017
# 82#	1.123798	-	1.123805	= -0.000007
# 83#	1.121523	-	1.121506	= 0.000017
# 84#	1.119014	-	1.119026	= -0.000012
# 85#	1.116458	-	1.116455	= 0.000003
# 86#	1.114007	-	1.114019	= -0.000012
# 87#	1.111498	-	1.111501	= -0.000003
# 88#	1.109036	-	1.109045	= -0.000009
# 89#	1.106217	-	1.106213	= 0.000004
# 90#	1.103457	-	1.103461	= -0.000004
# 91#	1.100789	-	1.100787	= 0.000002
# 92#	1.098130	-	1.098130	= 0.000000
# 93#	1.095260	-	1.095262	= -0.000002
# 94#	1.092204	-	1.092195	= 0.000009
# 95#	1.089338	-	1.089337	= 0.000001
# 96#	1.086436	-	1.086459	= -0.000023
# 97#	1.083425	-	1.083417	= 0.000008
# 98#	1.080233	-	1.080227	= 0.000006
# 99#	1.077103	-	1.077105	= -0.000002
#100#	1.073919	-	1.073926	= -0.000007
#101#	1.070758	-	1.070737	= 0.000021
#102#	1.067608	-	1.067608	= 0.000000
#103#	1.064107	-	1.064127	= -0.000020
#104#	1.060671	-	1.060662	= 0.000009
#105#	1.057369	-	1.057364	= 0.000005
#106#	1.053874	-	1.053888	= -0.000014
#107#	1.050451	-	1.050431	= 0.000020
#108#	1.046771	-	1.046777	= -0.000006
#109#	1.043056	-	1.043039	= 0.000017
#110#	1.039532	-	1.039536	= -0.000004
#111#	1.035981	-	1.035965	= 0.000016
#112#	1.032108	-	1.032099	= 0.000009
#113#	1.028281	-	1.028297	= -0.000016

#114#	1.024368	-	1.024356	= 0.000012
#115#	1.020600	-	1.020595	= 0.000005
#116#	1.016766	-	1.016776	= -0.000010
#117#	1.012741	-	1.012750	= -0.000009
#118#	1.008756	-	1.008744	= 0.000012
#119#	1.004555	-	1.004537	= 0.000018
#120#	1.000674	-	1.000662	= 0.000012
#121#	0.996615	-	0.996588	= 0.000027
#122#	0.992386	-	0.992392	= -0.000006
#123#	0.988220	-	0.988240	= -0.000020
#124#	0.983917	-	0.983909	= 0.000008
#125#	0.979772	-	0.979781	= -0.000009
#126#	0.975595	-	0.975605	= -0.000010
#127#	0.971171	-	0.971182	= -0.000011
#128#	0.966837	-	0.966823	= 0.000014
#129#	0.962492	-	0.962492	= 0.000000
#130#	0.958241	-	0.958237	= 0.000004
#131#	0.953776	-	0.953783	= -0.000007
#132#	0.949382	-	0.949367	= 0.000015
#133#	0.944869	-	0.944886	= -0.000017
#134#	0.940379	-	0.940359	= 0.000020
#135#	0.936148	-	0.936154	= -0.000006
#136#	0.931509	-	0.931512	= -0.000003
#137#	0.927033	-	0.927052	= -0.000019
#138#	0.922514	-	0.922513	= 0.000001
#139#	0.917924	-	0.917929	= -0.000005
#140#	0.913665	-	0.913666	= -0.000001
#141#	0.909012	-	0.909006	= 0.000006
#142#	0.904425	-	0.904409	= 0.000016
#143#	0.899905	-	0.899912	= -0.000007
#144#	0.895509	-	0.895529	= -0.000020
#145#	0.890930	-	0.890886	= 0.000044
#146#	0.886497	-	0.886509	= -0.000012
#147#	0.881892	-	0.881886	= 0.000006
#148#	0.877346	-	0.877393	= -0.000047
#149#	0.873101	-	0.873084	= 0.000017
#150#	0.868543	-	0.868548	= -0.000005
#151#	0.864050	-	0.864040	= 0.000010
#152#	0.859756	-	0.859760	= -0.000004
#153#	0.855211	-	0.855215	= -0.000004
#154#	0.850997	-	0.851010	= -0.000013
#155#	0.846619	-	0.846613	= 0.000006
#156#	0.842233	-	0.842236	= -0.000003
#157#	0.838052	-	0.838013	= 0.000039
#158#	0.833743	-	0.833766	= -0.000023
#159#	0.829638	-	0.829640	= -0.000002
#160#	0.825406	-	0.825406	= 0.000000
#161#	0.821280	-	0.821305	= -0.000025
#162#	0.817235	-	0.817217	= 0.000018
#163#	0.813155	-	0.813185	= -0.000030
#164#	0.809253	-	0.809266	= -0.000013
#165#	0.805275	-	0.805252	= 0.000023
#166#	0.801420	-	0.801424	= -0.000004
#167#	0.797675	-	0.797669	= 0.000006
#168#	0.793868	-	0.793861	= 0.000007
#169#	0.790178	-	0.790195	= -0.000017
#170#	0.786552	-	0.786543	= 0.000009
#171#	0.783021	-	0.783016	= 0.000005
#172#	0.779499	-	0.779529	= -0.000030
#173#	0.776147	-	0.776134	= 0.000013

```
#174#          0.772704 -          0.772706 =         -0.000002
#175#          0.769510 -          0.769538 =         -0.000028
#176#          0.766393 -          0.766393 =          0.000000
#177#          0.763240 -          0.763240 =          0.000000
#178#          0.760216 -          0.760230 =         -0.000014
#179#          0.757309 -          0.757303 =          0.000006
#180#          0.754458 -          0.754468 =         -0.000010
#181#          0.751740 -          0.751750 =         -0.000010
#182#          0.749203 -          0.749184 =          0.000019
#183#          0.746463 -          0.746477 =         -0.000014
#184#          0.744079 -          0.744070 =          0.000009
#185#          0.741746 -          0.741752 =         -0.000006
#186#          0.739481 -          0.739494 =         -0.000013
#187#          0.737445 -          0.737449 =         -0.000004
#188#          0.735190 -          0.735176 =          0.000014
#189#          0.733419 -          0.733416 =          0.000003
#190#          0.731639 -          0.731632 =          0.000007
#191#          0.729882 -          0.729889 =         -0.000007
#192#          0.728314 -          0.728324 =         -0.000010
#193#          0.726819 -          0.726833 =         -0.000014
#194#          0.725505 -          0.725507 =         -0.000002
#195#          0.724303 -          0.724310 =         -0.000007
Largest : 1.261517 @ line #  1#
Smallest: 0.724303 @ line #195#
Error   : 0.000047 @ line #148#
Error Ratio: 0.01%
```

Trajectory #1 6x10

::::::::: Norms for 6x10 Traj :#1 :::::::::
----------differences: ----------

STEP	FSP		Pseudo		diff
# 1#	1.173674	-	1.173673	=	0.000001
# 2#	1.173262	-	1.173262	=	0.000000
# 3#	1.171644	-	1.171646	=	-0.000002
# 4#	1.169785	-	1.169785	=	0.000000
# 5#	1.168145	-	1.168149	=	-0.000004
# 6#	1.166319	-	1.166315	=	0.000004
# 7#	1.164801	-	1.164797	=	0.000004
# 8#	1.162957	-	1.162964	=	-0.000007
# 9#	1.161161	-	1.161166	=	-0.000005
# 10#	1.159399	-	1.159405	=	-0.000006
# 11#	1.157501	-	1.157487	=	0.000014
# 12#	1.155879	-	1.155876	=	0.000003
# 13#	1.153952	-	1.153953	=	-0.000001
# 14#	1.152016	-	1.152014	=	0.000002
# 15#	1.150194	-	1.150199	=	-0.000005
# 16#	1.148208	-	1.148205	=	0.000003
# 17#	1.146492	-	1.146485	=	0.000007
# 18#	1.144433	-	1.144445	=	-0.000012
# 19#	1.142482	-	1.142488	=	-0.000006
# 20#	1.140515	-	1.140509	=	0.000006
# 21#	1.138354	-	1.138357	=	-0.000003
# 22#	1.136557	-	1.136542	=	0.000015
# 23#	1.134327	-	1.134340	=	-0.000013
# 24#	1.132299	-	1.132308	=	-0.000009
# 25#	1.130205	-	1.130199	=	0.000006
# 26#	1.128045	-	1.128024	=	0.000021
# 27#	1.125880	-	1.125902	=	-0.000022
# 28#	1.123718	-	1.123718	=	0.000000
# 29#	1.121493	-	1.121489	=	0.000004
# 30#	1.119238	-	1.119246	=	-0.000008
# 31#	1.116940	-	1.116934	=	0.000006
# 32#	1.114676	-	1.114675	=	0.000001
# 33#	1.112260	-	1.112267	=	-0.000007
# 34#	1.109951	-	1.109943	=	0.000008
# 35#	1.107576	-	1.107589	=	-0.000013
# 36#	1.105112	-	1.105112	=	0.000000
# 37#	1.102691	-	1.102686	=	0.000005
# 38#	1.100189	-	1.100200	=	-0.000011
# 39#	1.097680	-	1.097677	=	0.000003
# 40#	1.095206	-	1.095203	=	0.000003
# 41#	1.092444	-	1.092439	=	0.000005
# 42#	1.089842	-	1.089851	=	-0.000009
# 43#	1.087258	-	1.087250	=	0.000008
# 44#	1.084571	-	1.084582	=	-0.000011
# 45#	1.081824	-	1.081824	=	0.000000
# 46#	1.079087	-	1.079100	=	-0.000013
# 47#	1.076322	-	1.076319	=	0.000003
# 48#	1.073476	-	1.073481	=	-0.000005
# 49#	1.070645	-	1.070624	=	0.000021
# 50#	1.067688	-	1.067700	=	-0.000012
# 51#	1.064746	-	1.064744	=	0.000002
# 52#	1.061843	-	1.061841	=	0.000002
# 53#	1.058825	-	1.058831	=	-0.000006

# 54#	1.055799	-	1.055801	= -0.000002
# 55#	1.052696	-	1.052694	= 0.000002
# 56#	1.049619	-	1.049616	= 0.000003
# 57#	1.046472	-	1.046475	= -0.000003
# 58#	1.043301	-	1.043300	= 0.000001
# 59#	1.040155	-	1.040167	= -0.000012
# 60#	1.036737	-	1.036729	= 0.000008
# 61#	1.033500	-	1.033508	= -0.000008
# 62#	1.030203	-	1.030206	= -0.000003
# 63#	1.026872	-	1.026864	= 0.000008
# 64#	1.023548	-	1.023561	= -0.000013
# 65#	1.019964	-	1.019955	= 0.000009
# 66#	1.016589	-	1.016589	= 0.000000
# 67#	1.013119	-	1.013106	= 0.000013
# 68#	1.009554	-	1.009547	= 0.000007
# 69#	1.006071	-	1.006092	= -0.000021
# 70#	1.002276	-	1.002267	= 0.000009
# 71#	0.998770	-	0.998777	= -0.000007
# 72#	0.995121	-	0.995109	= 0.000012
# 73#	0.991423	-	0.991438	= -0.000015
# 74#	0.987781	-	0.987771	= 0.000010
# 75#	0.983841	-	0.983839	= 0.000002
# 76#	0.980177	-	0.980176	= 0.000001
# 77#	0.976337	-	0.976335	= 0.000002
# 78#	0.972569	-	0.972565	= 0.000004
# 79#	0.968628	-	0.968637	= -0.000009
# 80#	0.964580	-	0.964565	= 0.000015
# 81#	0.960792	-	0.960787	= 0.000005
# 82#	0.956842	-	0.956852	= -0.000010
# 83#	0.952908	-	0.952918	= -0.000010
# 84#	0.948848	-	0.948835	= 0.000013
# 85#	0.944727	-	0.944722	= 0.000005
# 86#	0.940735	-	0.940728	= 0.000007
# 87#	0.936678	-	0.936679	= -0.000001
# 88#	0.932635	-	0.932632	= 0.000003
# 89#	0.928396	-	0.928390	= 0.000006
# 90#	0.924201	-	0.924183	= 0.000018
# 91#	0.920124	-	0.920130	= -0.000006
# 92#	0.915998	-	0.915989	= 0.000009
# 93#	0.911760	-	0.911754	= 0.000006
# 94#	0.907408	-	0.907423	= -0.000015
# 95#	0.903264	-	0.903270	= -0.000006
# 96#	0.899043	-	0.899041	= 0.000002
# 97#	0.894796	-	0.894804	= -0.000008
# 98#	0.890426	-	0.890424	= 0.000002
# 99#	0.886103	-	0.886108	= -0.000005
#100#	0.881862	-	0.881869	= -0.000007
#101#	0.877563	-	0.877556	= 0.000007
#102#	0.873296	-	0.873294	= 0.000002
#103#	0.868874	-	0.868878	= -0.000004
#104#	0.864447	-	0.864456	= -0.000009
#105#	0.860207	-	0.860215	= -0.000008
#106#	0.855846	-	0.855842	= 0.000004
#107#	0.851522	-	0.851529	= -0.000007
#108#	0.847075	-	0.847091	= -0.000016
#109#	0.842638	-	0.842631	= 0.000007
#110#	0.838373	-	0.838367	= 0.000006
#111#	0.834080	-	0.834081	= -0.000001
#112#	0.829623	-	0.829614	= 0.000009
#113#	0.825256	-	0.825264	= -0.000008

#114#	0.820812	-	0.820792	= 0.000020
#115#	0.816512	-	0.816519	= -0.000007
#116#	0.812205	-	0.812200	= 0.000005
#117#	0.807787	-	0.807790	= -0.000003
#118#	0.803452	-	0.803439	= 0.000013
#119#	0.798988	-	0.799000	= -0.000012
#120#	0.794778	-	0.794776	= 0.000002
#121#	0.790468	-	0.790476	= -0.000008
#122#	0.786106	-	0.786103	= 0.000003
#123#	0.781816	-	0.781805	= 0.000011
#124#	0.777402	-	0.777410	= -0.000008
#125#	0.773203	-	0.773211	= -0.000008
#126#	0.768987	-	0.768981	= 0.000006
#127#	0.764633	-	0.764641	= -0.000008
#128#	0.760416	-	0.760398	= 0.000018
#129#	0.756140	-	0.756123	= 0.000017
#130#	0.752014	-	0.752000	= 0.000014
#131#	0.747768	-	0.747766	= 0.000002
#132#	0.743601	-	0.743609	= -0.000008
#133#	0.739420	-	0.739404	= 0.000016
#134#	0.735205	-	0.735191	= 0.000014
#135#	0.731223	-	0.731217	= 0.000006
#136#	0.727021	-	0.727016	= 0.000005
#137#	0.722985	-	0.722975	= 0.000010
#138#	0.718887	-	0.718889	= -0.000002
#139#	0.714789	-	0.714796	= -0.000007
#140#	0.710894	-	0.710899	= -0.000005
#141#	0.706851	-	0.706846	= 0.000005
#142#	0.702863	-	0.702861	= 0.000002
#143#	0.698898	-	0.698881	= 0.000017
#144#	0.695054	-	0.695048	= 0.000006
#145#	0.691093	-	0.691100	= -0.000007
#146#	0.687297	-	0.687284	= 0.000013
#147#	0.683412	-	0.683416	= -0.000004
#148#	0.679541	-	0.679537	= 0.000004
#149#	0.675891	-	0.675869	= 0.000022
#150#	0.672058	-	0.672077	= -0.000019
#151#	0.668332	-	0.668333	= -0.000001
#152#	0.664688	-	0.664699	= -0.000011
#153#	0.660923	-	0.660911	= 0.000012
#154#	0.657390	-	0.657373	= 0.000017
#155#	0.653744	-	0.653755	= -0.000011
#156#	0.650169	-	0.650152	= 0.000017
#157#	0.646630	-	0.646648	= -0.000018
#158#	0.643100	-	0.643095	= 0.000005
#159#	0.639673	-	0.639683	= -0.000010
#160#	0.636213	-	0.636196	= 0.000017
#161#	0.632823	-	0.632824	= -0.000001
#162#	0.629415	-	0.629431	= -0.000016
#163#	0.626022	-	0.626030	= -0.000008
#164#	0.622792	-	0.622787	= 0.000005
#165#	0.619486	-	0.619485	= 0.000001
#166#	0.616272	-	0.616264	= 0.000008
#167#	0.613082	-	0.613092	= -0.000010
#168#	0.609845	-	0.609823	= 0.000022
#169#	0.606760	-	0.606759	= 0.000001
#170#	0.603670	-	0.603680	= -0.000010
#171#	0.600606	-	0.600588	= 0.000018
#172#	0.597566	-	0.597576	= -0.000010
#173#	0.594564	-	0.594554	= 0.000010

```
#174#          0.591592 -          0.591588 =          0.000004
#175#          0.588713 -          0.588706 =          0.000007
#176#          0.585863 -          0.585850 =          0.000013
#177#          0.582971 -          0.583000 =         -0.000029
#178#          0.580124 -          0.580107 =          0.000017
#179#          0.577412 -          0.577419 =         -0.000007
#180#          0.574653 -          0.574680 =         -0.000027
#181#          0.571973 -          0.571971 =          0.000002
#182#          0.569369 -          0.569363 =          0.000006
#183#          0.566601 -          0.566607 =         -0.000006
#184#          0.564130 -          0.564134 =         -0.000004
#185#          0.561552 -          0.561549 =          0.000003
#186#          0.559036 -          0.559047 =         -0.000011
#187#          0.556619 -          0.556616 =          0.000003
#188#          0.554056 -          0.554072 =         -0.000016
#189#          0.551744 -          0.551736 =          0.000008
#190#          0.549419 -          0.549426 =         -0.000007
#191#          0.547046 -          0.547062 =         -0.000016
#192#          0.544705 -          0.544693 =          0.000012
#193#          0.542544 -          0.542527 =          0.000017
#194#          0.540300 -          0.540320 =         -0.000020
#195#          0.538145 -          0.538135 =          0.000010
Largest : 1.173674 @ line #  1#
Smallest: 0.538135 @ line #195#
Error   : 0.000029 @ line #177#
Error Ratio: 0.00%
```

Trajectory #2 3x10

:::::::: Norms for 3x10 Traj :#2 ::::::::

----------differences: ----------

STEP	FSP		Pseudo		diff
# 1#	0.199948	-	0.199948	=	0.000000
# 2#	0.190548	-	0.190548	=	0.000000
# 3#	0.189842	-	0.189841	=	0.000001
# 4#	0.188750	-	0.188752	=	-0.000002
# 5#	0.187732	-	0.187732	=	0.000000
# 6#	0.186698	-	0.186697	=	0.000001
# 7#	0.185623	-	0.185622	=	0.000001
# 8#	0.184573	-	0.184575	=	-0.000002
# 9#	0.183538	-	0.183535	=	0.000003
# 10#	0.182538	-	0.182536	=	0.000002
# 11#	0.181473	-	0.181475	=	-0.000002
# 12#	0.180397	-	0.180395	=	0.000002
# 13#	0.179400	-	0.179402	=	-0.000002
# 14#	0.178288	-	0.178287	=	0.000001
# 15#	0.177306	-	0.177302	=	0.000004
# 16#	0.176264	-	0.176266	=	-0.000002
# 17#	0.175184	-	0.175182	=	0.000002
# 18#	0.174175	-	0.174175	=	0.000000
# 19#	0.173157	-	0.173160	=	-0.000003
# 20#	0.172060	-	0.172061	=	-0.000001
# 21#	0.171019	-	0.171019	=	0.000000
# 22#	0.169987	-	0.169986	=	0.000001
# 23#	0.168968	-	0.168965	=	0.000003
# 24#	0.167922	-	0.167924	=	-0.000002
# 25#	0.166856	-	0.166856	=	0.000000
# 26#	0.165852	-	0.165849	=	0.000003
# 27#	0.164766	-	0.164762	=	0.000004
# 28#	0.163794	-	0.163805	=	-0.000011
# 29#	0.162739	-	0.162740	=	-0.000001
# 30#	0.161677	-	0.161681	=	-0.000004
# 31#	0.160678	-	0.160676	=	0.000002
# 32#	0.159638	-	0.159636	=	0.000002
# 33#	0.158579	-	0.158578	=	0.000001
# 34#	0.157557	-	0.157566	=	-0.000009
# 35#	0.156531	-	0.156527	=	0.000004
# 36#	0.155524	-	0.155527	=	-0.000003
# 37#	0.154528	-	0.154527	=	0.000001
# 38#	0.153449	-	0.153456	=	-0.000007
# 39#	0.152461	-	0.152454	=	0.000007
# 40#	0.151391	-	0.151394	=	-0.000003
# 41#	0.150423	-	0.150422	=	0.000001
# 42#	0.149409	-	0.149409	=	0.000000
# 43#	0.148370	-	0.148367	=	0.000003
# 44#	0.147385	-	0.147388	=	-0.000003
# 45#	0.146376	-	0.146381	=	-0.000005
# 46#	0.145346	-	0.145338	=	0.000008
# 47#	0.144373	-	0.144375	=	-0.000002
# 48#	0.143342	-	0.143338	=	0.000004
# 49#	0.142369	-	0.142368	=	0.000001
# 50#	0.141372	-	0.141376	=	-0.000004
# 51#	0.140354	-	0.140358	=	-0.000004
# 52#	0.139393	-	0.139395	=	-0.000002
# 53#	0.138391	-	0.138387	=	0.000004

# 54#	0.137395	-	0.137399	= -0.000004
# 55#	0.136448	-	0.136443	= 0.000005
# 56#	0.135466	-	0.135460	= 0.000006
# 57#	0.134494	-	0.134500	= -0.000006
# 58#	0.133527	-	0.133527	= 0.000000
# 59#	0.132521	-	0.132522	= -0.000001
# 60#	0.131604	-	0.131602	= 0.000002
# 61#	0.130615	-	0.130618	= -0.000003
# 62#	0.129696	-	0.129693	= 0.000003
# 63#	0.128739	-	0.128739	= 0.000000
# 64#	0.127774	-	0.127777	= -0.000003
# 65#	0.126862	-	0.126861	= 0.000001
# 66#	0.125922	-	0.125923	= -0.000001
# 67#	0.124967	-	0.124969	= -0.000002
# 68#	0.124063	-	0.124063	= 0.000000
# 69#	0.123123	-	0.123120	= 0.000003
# 70#	0.122232	-	0.122236	= -0.000004
# 71#	0.121314	-	0.121319	= -0.000005
# 72#	0.120380	-	0.120373	= 0.000007
# 73#	0.119516	-	0.119516	= 0.000000
# 74#	0.118616	-	0.118617	= -0.000001
# 75#	0.117727	-	0.117730	= -0.000003
# 76#	0.116843	-	0.116843	= 0.000000
# 77#	0.115948	-	0.115948	= 0.000000
# 78#	0.115085	-	0.115085	= 0.000000
# 79#	0.114233	-	0.114232	= 0.000001
# 80#	0.113350	-	0.113356	= -0.000006
# 81#	0.112520	-	0.112516	= 0.000004
# 82#	0.111657	-	0.111660	= -0.000003
# 83#	0.110852	-	0.110851	= 0.000001
# 84#	0.110002	-	0.110004	= -0.000002
# 85#	0.109144	-	0.109143	= 0.000001
# 86#	0.108358	-	0.108356	= 0.000002
# 87#	0.107531	-	0.107533	= -0.000002
# 88#	0.106742	-	0.106742	= 0.000000
# 89#	0.105943	-	0.105942	= 0.000001
# 90#	0.105136	-	0.105137	= -0.000001
# 91#	0.104368	-	0.104367	= 0.000001
# 92#	0.103615	-	0.103615	= 0.000000
# 93#	0.102816	-	0.102813	= 0.000003
# 94#	0.102075	-	0.102079	= -0.000004
# 95#	0.101309	-	0.101307	= 0.000002
# 96#	0.100583	-	0.100587	= -0.000004
# 97#	0.099842	-	0.099841	= 0.000001
# 98#	0.099109	-	0.099111	= -0.000002
# 99#	0.098413	-	0.098411	= 0.000002
#100#	0.097687	-	0.097686	= 0.000001
#101#	0.097021	-	0.097024	= -0.000003
#102#	0.096314	-	0.096315	= -0.000001
#103#	0.095622	-	0.095620	= 0.000002
#104#	0.094957	-	0.094958	= -0.000001
#105#	0.094306	-	0.094307	= -0.000001
#106#	0.093641	-	0.093637	= 0.000004
#107#	0.093013	-	0.093015	= -0.000002
#108#	0.092366	-	0.092364	= 0.000002
#109#	0.091762	-	0.091764	= -0.000002
#110#	0.091149	-	0.091147	= 0.000002
#111#	0.090533	-	0.090529	= 0.000004
#112#	0.089952	-	0.089956	= -0.000004
#113#	0.089359	-	0.089358	= 0.000001

#114#	0.088798	-	0.088804	= -0.000006
#115#	0.088248	-	0.088246	= 0.000002
#116#	0.087662	-	0.087664	= -0.000002
#117#	0.087159	-	0.087158	= 0.000001
#118#	0.086623	-	0.086627	= -0.000004
#119#	0.086100	-	0.086098	= 0.000002
#120#	0.085617	-	0.085616	= 0.000001
#121#	0.085096	-	0.085096	= 0.000000
#122#	0.084632	-	0.084632	= 0.000000
#123#	0.084152	-	0.084152	= 0.000000
#124#	0.083687	-	0.083688	= -0.000001
#125#	0.083254	-	0.083253	= 0.000001
#126#	0.082802	-	0.082805	= -0.000003
#127#	0.082394	-	0.082394	= 0.000000
#128#	0.081986	-	0.081985	= 0.000001
#129#	0.081570	-	0.081569	= 0.000001
#130#	0.081198	-	0.081201	= -0.000003
#131#	0.080823	-	0.080822	= 0.000001
#132#	0.080446	-	0.080443	= 0.000003
#133#	0.080114	-	0.080113	= 0.000001
#134#	0.079759	-	0.079761	= -0.000002
#135#	0.079442	-	0.079438	= 0.000004
#136#	0.079140	-	0.079144	= -0.000004
#137#	0.078828	-	0.078826	= 0.000002
#138#	0.078563	-	0.078565	= -0.000002
#139#	0.078269	-	0.078269	= 0.000000
#140#	0.078027	-	0.078025	= 0.000002
#141#	0.077776	-	0.077778	= -0.000002
#142#	0.077518	-	0.077519	= -0.000001
#143#	0.077335	-	0.077326	= 0.000009
#144#	0.077122	-	0.077123	= -0.000001
#145#	0.076895	-	0.076913	= -0.000018
#146#	0.076664	-	0.076752	= -0.000088
#147#	0.076704	-	0.076574	= 0.000130
#148#	0.076390	-	0.076421	= -0.000031
#149#	0.076306	-	0.076291	= 0.000015
#150#	0.076157	-	0.076159	= -0.000002
#151#	0.076062	-	0.076063	= -0.000001
#152#	0.075951	-	0.075953	= -0.000002
#153#	0.075888	-	0.075886	= 0.000002
#154#	0.075818	-	0.075819	= -0.000001
#155#	0.075740	-	0.075740	= 0.000000
#156#	0.075731	-	0.075731	= 0.000000
#157#	0.075702	-	0.075698	= 0.000004
#158#	0.075669	-	0.075676	= -0.000007
#159#	0.075695	-	0.075691	= 0.000004
#160#	0.075697	-	0.075699	= -0.000002
#161#	0.075731	-	0.075735	= -0.000004
#162#	0.075789	-	0.075787	= 0.000002
#163#	0.075838	-	0.075835	= 0.000003
#164#	0.075925	-	0.075929	= -0.000004
#165#	0.076018	-	0.076017	= 0.000001
#166#	0.076109	-	0.076106	= 0.000003
#167#	0.076232	-	0.076231	= 0.000001
#168#	0.076361	-	0.076359	= 0.000002
#169#	0.076522	-	0.076525	= -0.000003
#170#	0.076691	-	0.076687	= 0.000004
#171#	0.076848	-	0.076847	= 0.000001
#172#	0.077058	-	0.077060	= -0.000002
#173#	0.077248	-	0.077248	= 0.000000

#174#	0.077477	-	0.077475	= 0.000002
#175#	0.077720	-	0.077723	= -0.000003
#176#	0.077961	-	0.077958	= 0.000003
#177#	0.078240	-	0.078241	= -0.000001
#178#	0.078523	-	0.078520	= 0.000003
#179#	0.078802	-	0.078804	= -0.000002
#180#	0.079116	-	0.079112	= 0.000004
#181#	0.079438	-	0.079434	= 0.000004
#182#	0.079790	-	0.079796	= -0.000006
#183#	0.080147	-	0.080145	= 0.000002
#184#	0.080499	-	0.080496	= 0.000003
#185#	0.080902	-	0.080899	= 0.000003
#186#	0.081268	-	0.081270	= -0.000002
#187#	0.081708	-	0.081708	= 0.000000
#188#	0.082139	-	0.082141	= -0.000002
#189#	0.082565	-	0.082562	= 0.000003
#190#	0.083036	-	0.083036	= 0.000000
#191#	0.083512	-	0.083512	= 0.000000
#192#	0.083977	-	0.083977	= 0.000000
#193#	0.084481	-	0.084477	= 0.000004
#194#	0.084992	-	0.084990	= 0.000002
#195#	0.085543	-	0.085541	= 0.000002
#196#	0.086083	-	0.086084	= -0.000001
#197#	0.086623	-	0.086627	= -0.000004
#198#	0.087217	-	0.087211	= 0.000006
#199#	0.087768	-	0.087770	= -0.000002
#200#	0.088404	-	0.088400	= 0.000004
#201#	0.089018	-	0.089025	= -0.000007
#202#	0.089625	-	0.089620	= 0.000005
#203#	0.090289	-	0.090291	= -0.000002
#204#	0.090950	-	0.090954	= -0.000004
#205#	0.091591	-	0.091590	= 0.000001
#206#	0.092305	-	0.092305	= 0.000000
#207#	0.092992	-	0.092993	= -0.000001
#208#	0.093724	-	0.093721	= 0.000003
#209#	0.094453	-	0.094451	= 0.000002
#210#	0.095174	-	0.095175	= -0.000001
#211#	0.095942	-	0.095942	= 0.000000
#212#	0.096683	-	0.096683	= 0.000000
#213#	0.097501	-	0.097502	= -0.000001
#214#	0.098299	-	0.098296	= 0.000003
#215#	0.099087	-	0.099096	= -0.000009
#216#	0.099937	-	0.099931	= 0.000006
#217#	0.100770	-	0.100771	= -0.000001
#218#	0.101587	-	0.101582	= 0.000005
#219#	0.102488	-	0.102493	= -0.000005
#220#	0.103337	-	0.103337	= 0.000000
#221#	0.104259	-	0.104254	= 0.000005
#222#	0.105161	-	0.105164	= -0.000003
#223#	0.106056	-	0.106055	= 0.000001
#224#	0.107007	-	0.107010	= -0.000003
#225#	0.107915	-	0.107914	= 0.000001
#226#	0.108912	-	0.108908	= 0.000004
#227#	0.109881	-	0.109882	= -0.000001
#228#	0.110838	-	0.110834	= 0.000004
#229#	0.111848	-	0.111856	= -0.000008
#230#	0.112866	-	0.112863	= 0.000003
#231#	0.113842	-	0.113843	= -0.000001
#232#	0.114926	-	0.114919	= 0.000007
#233#	0.115937	-	0.115938	= -0.000001

```
#234#          0.117029 -          0.117022 =          0.000007
Largest : 0.199948 @ line #  1#
Smallest: 0.075669 @ line #158#
Error   : 0.000130 @ line #147#
Error Ratio: 0.17%
```

Trajectory #2 3x7

:::::::: Norms for 3x7 Traj :#2 ::::::::

----------differences: ----------

STEP	FSP		Pseudo		diff
# 1#	1.116755	-	1.116756	=	-0.000001
# 2#	1.031932	-	1.031932	=	0.000000
# 3#	1.010608	-	1.010601	=	0.000007
# 4#	0.988565	-	0.988609	=	-0.000044
# 5#	0.967882	-	0.967870	=	0.000012
# 6#	0.947787	-	0.947798	=	-0.000011
# 7#	0.928313	-	0.928315	=	-0.000002
# 8#	0.909901	-	0.909898	=	0.000003
# 9#	0.891997	-	0.891976	=	0.000021
# 10#	0.875113	-	0.875119	=	-0.000006
# 11#	0.858444	-	0.858436	=	0.000008
# 12#	0.842311	-	0.842334	=	-0.000023
# 13#	0.827100	-	0.827081	=	0.000019
# 14#	0.812012	-	0.812009	=	0.000003
# 15#	0.797875	-	0.797864	=	0.000011
# 16#	0.783992	-	0.783989	=	0.000003
# 17#	0.770461	-	0.770449	=	0.000012
# 18#	0.757612	-	0.757631	=	-0.000019
# 19#	0.745124	-	0.745134	=	-0.000010
# 20#	0.732756	-	0.732721	=	0.000035
# 21#	0.720957	-	0.720976	=	-0.000019
# 22#	0.709497	-	0.709490	=	0.000007
# 23#	0.698481	-	0.698484	=	-0.000003
# 24#	0.687711	-	0.687718	=	-0.000007
# 25#	0.677123	-	0.677111	=	0.000012
# 26#	0.667030	-	0.667052	=	-0.000022
# 27#	0.656987	-	0.656961	=	0.000026
# 28#	0.647663	-	0.647650	=	0.000013
# 29#	0.638152	-	0.638187	=	-0.000035
# 30#	0.628923	-	0.628924	=	-0.000001
# 31#	0.620190	-	0.620153	=	0.000037
# 32#	0.611510	-	0.611537	=	-0.000027
# 33#	0.602962	-	0.602964	=	-0.000002
# 34#	0.594786	-	0.594785	=	0.000001
# 35#	0.586734	-	0.586725	=	0.000009
# 36#	0.578972	-	0.578979	=	-0.000007
# 37#	0.571454	-	0.571454	=	0.000000
# 38#	0.563761	-	0.563780	=	-0.000019
# 39#	0.556587	-	0.556578	=	0.000009
# 40#	0.549247	-	0.549244	=	0.000003
# 41#	0.542442	-	0.542450	=	-0.000008
# 42#	0.535602	-	0.535600	=	0.000002
# 43#	0.528819	-	0.528812	=	0.000007
# 44#	0.522346	-	0.522341	=	0.000005
# 45#	0.515940	-	0.515950	=	-0.000010
# 46#	0.509608	-	0.509614	=	-0.000006
# 47#	0.503465	-	0.503466	=	-0.000001
# 48#	0.497325	-	0.497322	=	0.000003
# 49#	0.491475	-	0.491476	=	-0.000001
# 50#	0.485632	-	0.485639	=	-0.000007
# 51#	0.479850	-	0.479839	=	0.000011
# 52#	0.474275	-	0.474281	=	-0.000006
# 53#	0.468685	-	0.468682	=	0.000003

# 54#	0.463232	-	0.463224	= 0.000008
# 55#	0.457943	-	0.457950	= -0.000007
# 56#	0.452708	-	0.452714	= -0.000006
# 57#	0.447496	-	0.447494	= 0.000002
# 58#	0.442369	-	0.442362	= 0.000007
# 59#	0.437189	-	0.437206	= -0.000017
# 60#	0.432366	-	0.432370	= -0.000004
# 61#	0.427405	-	0.427372	= 0.000033
# 62#	0.422608	-	0.422645	= -0.000037
# 63#	0.417849	-	0.417813	= 0.000036
# 64#	0.413070	-	0.413072	= -0.000002
# 65#	0.408480	-	0.408484	= -0.000004
# 66#	0.403848	-	0.403840	= 0.000008
# 67#	0.399207	-	0.399221	= -0.000014
# 68#	0.394778	-	0.394764	= 0.000014
# 69#	0.390245	-	0.390233	= 0.000012
# 70#	0.385872	-	0.385883	= -0.000011
# 71#	0.381487	-	0.381489	= -0.000002
# 72#	0.377029	-	0.377039	= -0.000010
# 73#	0.372847	-	0.372847	= 0.000000
# 74#	0.368600	-	0.368605	= -0.000005
# 75#	0.364348	-	0.364343	= 0.000005
# 76#	0.360134	-	0.360140	= -0.000006
# 77#	0.355918	-	0.355912	= 0.000006
# 78#	0.351782	-	0.351787	= -0.000005
# 79#	0.347722	-	0.347728	= -0.000006
# 80#	0.343593	-	0.343595	= -0.000002
# 81#	0.339601	-	0.339608	= -0.000007
# 82#	0.335516	-	0.335499	= 0.000017
# 83#	0.331665	-	0.331655	= 0.000010
# 84#	0.327608	-	0.327622	= -0.000014
# 85#	0.323588	-	0.323575	= 0.000013
# 86#	0.319775	-	0.319783	= -0.000008
# 87#	0.315852	-	0.315847	= 0.000005
# 88#	0.312072	-	0.312076	= -0.000004
# 89#	0.308260	-	0.308257	= 0.000003
# 90#	0.304432	-	0.304445	= -0.000013
# 91#	0.300733	-	0.300728	= 0.000005
# 92#	0.297139	-	0.297133	= 0.000006
# 93#	0.293392	-	0.293394	= -0.000002
# 94#	0.289853	-	0.289856	= -0.000003
# 95#	0.286276	-	0.286268	= 0.000008
# 96#	0.282845	-	0.282848	= -0.000003
# 97#	0.279399	-	0.279394	= 0.000005
# 98#	0.276045	-	0.276057	= -0.000012
# 99#	0.272858	-	0.272855	= 0.000003
#100#	0.269630	-	0.269648	= -0.000018
#101#	0.266664	-	0.266650	= 0.000014
#102#	0.263634	-	0.263648	= -0.000014
#103#	0.260755	-	0.260747	= 0.000008
#104#	0.258027	-	0.258034	= -0.000007
#105#	0.255484	-	0.255485	= -0.000001
#106#	0.253011	-	0.253003	= 0.000008
#107#	0.250765	-	0.250772	= -0.000007
#108#	0.248652	-	0.248643	= 0.000009
#109#	0.246789	-	0.246792	= -0.000003
#110#	0.245112	-	0.245110	= 0.000002
#111#	0.243607	-	0.243609	= -0.000002
#112#	0.242458	-	0.242456	= 0.000002
#113#	0.241468	-	0.241467	= 0.000001

#114#	0.240874	-	0.240874	= 0.000000
#115#	0.240527	-	0.240537	= -0.000010
#116#	0.240473	-	0.240466	= 0.000007
#117#	0.240901	-	0.240899	= 0.000002
#118#	0.241652	-	0.241651	= 0.000001
#119#	0.242754	-	0.242749	= 0.000005
#120#	0.244400	-	0.244407	= -0.000007
#121#	0.246413	-	0.246412	= 0.000001
#122#	0.249033	-	0.249034	= -0.000001
#123#	0.252103	-	0.252107	= -0.000004
#124#	0.255703	-	0.255699	= 0.000004
#125#	0.259992	-	0.259987	= 0.000005
#126#	0.264769	-	0.264763	= 0.000006
#127#	0.270327	-	0.270339	= -0.000012
#128#	0.276499	-	0.276497	= 0.000002
#129#	0.283272	-	0.283266	= 0.000006
#130#	0.291002	-	0.290999	= 0.000003
#131#	0.299388	-	0.299393	= -0.000005
#132#	0.308481	-	0.308477	= 0.000004
#133#	0.318560	-	0.318561	= -0.000001
#134#	0.329315	-	0.329319	= -0.000004
#135#	0.341111	-	0.341114	= -0.000003
#136#	0.353704	-	0.353685	= 0.000019
#137#	0.367047	-	0.367062	= -0.000015
#138#	0.381474	-	0.381473	= 0.000001
#139#	0.396736	-	0.396747	= -0.000011
#140#	0.413054	-	0.413052	= 0.000002
#141#	0.430219	-	0.430202	= 0.000017
#142#	0.448082	-	0.448095	= -0.000013
#143#	0.467101	-	0.467088	= 0.000013
#144#	0.486884	-	0.486888	= -0.000004
#145#	0.507357	-	0.507356	= 0.000001
#146#	0.528881	-	0.528905	= -0.000024
#147#	0.550799	-	0.550761	= 0.000038
#148#	0.573951	-	0.573954	= -0.000003
#149#	0.597483	-	0.597471	= 0.000012
#150#	0.621371	-	0.621383	= -0.000012
#151#	0.646179	-	0.646224	= -0.000045
#152#	0.671100	-	0.671057	= 0.000043
#153#	0.696719	-	0.696703	= 0.000016
#154#	0.722430	-	0.722430	= 0.000000
#155#	0.748105	-	0.748090	= 0.000015
#156#	0.774069	-	0.774123	= -0.000054
#157#	0.800243	-	0.800242	= 0.000001
#158#	0.825876	-	0.825833	= 0.000043
#159#	0.851714	-	0.851723	= -0.000009
#160#	0.876832	-	0.876835	= -0.000003
#161#	0.902050	-	0.902007	= 0.000043
#162#	0.926464	-	0.926518	= -0.000054
#163#	0.950079	-	0.950100	= -0.000021
#164#	0.973592	-	0.973581	= 0.000011
#165#	0.995923	-	0.995961	= -0.000038
#166#	1.017542	-	1.017496	= 0.000046
#167#	1.038651	-	1.038682	= -0.000031
#168#	1.058266	-	1.058272	= -0.000006
#169#	1.077513	-	1.077510	= 0.000003
#170#	1.095571	-	1.095499	= 0.000072
#171#	1.112135	-	1.112194	= -0.000059
#172#	1.128405	-	1.128418	= -0.000013
#173#	1.143032	-	1.142986	= 0.000046

#174#	1.157128	-	1.157140	= -0.000012
#175#	1.169811	-	1.169808	= 0.000003
#176#	1.181376	-	1.181381	= -0.000005
#177#	1.192515	-	1.192510	= 0.000005
#178#	1.202232	-	1.202224	= 0.000008
#179#	1.210762	-	1.210767	= -0.000005
#180#	1.218929	-	1.218928	= 0.000001
#181#	1.225617	-	1.225613	= 0.000004
#182#	1.232179	-	1.232142	= 0.000037
#183#	1.237380	-	1.237456	= -0.000076
#184#	1.241735	-	1.241656	= 0.000079
#185#	1.245972	-	1.246011	= -0.000039
#186#	1.248906	-	1.248817	= 0.000089
#187#	1.251807	-	1.251854	= -0.000047
#188#	1.253852	-	1.253842	= 0.000010
#189#	1.255070	-	1.255061	= 0.000009
#190#	1.256399	-	1.256440	= -0.000041
#191#	1.257032	-	1.257034	= -0.000002
#192#	1.257027	-	1.256982	= 0.000045
#193#	1.257057	-	1.257101	= -0.000044
#194#	1.256454	-	1.256496	= -0.000042
#195#	1.256188	-	1.256153	= 0.000035
#196#	1.255226	-	1.255264	= -0.000038
#197#	1.253920	-	1.253883	= 0.000037
#198#	1.253082	-	1.253075	= 0.000007
#199#	1.251392	-	1.251431	= -0.000039
#200#	1.250401	-	1.250366	= 0.000035
#201#	1.249016	-	1.248981	= 0.000035
#202#	1.247197	-	1.247262	= -0.000065
#203#	1.246241	-	1.246240	= 0.000001
#204#	1.244808	-	1.244808	= 0.000000
#205#	1.243203	-	1.243167	= 0.000036
#206#	1.242435	-	1.242437	= -0.000002
#207#	1.241055	-	1.241053	= 0.000002
#208#	1.240415	-	1.240412	= 0.000003
#209#	1.239707	-	1.239707	= 0.000000
#210#	1.238845	-	1.238876	= -0.000031
#211#	1.238709	-	1.238686	= 0.000023
#212#	1.238310	-	1.238271	= 0.000039
#213#	1.238775	-	1.238781	= -0.000006
#214#	1.239169	-	1.239133	= 0.000036
#215#	1.239435	-	1.239464	= -0.000029
#216#	1.240778	-	1.240777	= 0.000001
#217#	1.242009	-	1.242031	= -0.000022
#218#	1.243197	-	1.243210	= -0.000013
#219#	1.245578	-	1.245565	= 0.000013
#220#	1.247481	-	1.247481	= 0.000000
#221#	1.250629	-	1.250567	= 0.000062
#222#	1.253574	-	1.253571	= 0.000003
#223#	1.256830	-	1.256863	= -0.000033
#224#	1.261035	-	1.261023	= 0.000012
#225#	1.265024	-	1.265010	= 0.000014
#226#	1.270266	-	1.270275	= -0.000009
#227#	1.275566	-	1.275578	= -0.000012
#228#	1.281189	-	1.281147	= 0.000042
#229#	1.287739	-	1.287723	= 0.000016
#230#	1.294711	-	1.294773	= -0.000062
#231#	1.301798	-	1.301771	= 0.000027
#232#	1.310259	-	1.310266	= -0.000007
#233#	1.318674	-	1.318694	= -0.000020

```
#234#          1.328463 -          1.328442 =          0.000021
Largest : 1.328463 @ line #234#
Smallest: 0.240466 @ line #116#
Error   : 0.000089 @ line #186#
Error Ratio: 0.01%
```

Trajectory #2 6x7

:::::::: Norms for 6x7 Traj :#2 ::::::::
----------differences: ----------

STEP	FSP		Pseudo		diff
# 1#	6.968414	-	6.968416	=	-0.000002
# 2#	7.301880	-	7.301851	=	0.000029
# 3#	7.267199	-	7.267307	=	-0.000108
# 4#	7.222111	-	7.222023	=	0.000088
# 5#	7.180116	-	7.180136	=	-0.000020
# 6#	7.139550	-	7.139505	=	0.000045
# 7#	7.099812	-	7.099829	=	-0.000017
# 8#	7.062784	-	7.062744	=	0.000040
# 9#	7.024738	-	7.024809	=	-0.000071
# 10#	6.989432	-	6.989447	=	-0.000015
# 11#	6.953091	-	6.953090	=	0.000001
# 12#	6.915750	-	6.915707	=	0.000043
# 13#	6.878975	-	6.879017	=	-0.000042
# 14#	6.840050	-	6.840017	=	0.000033
# 15#	6.800510	-	6.800434	=	0.000076
# 16#	6.758464	-	6.758503	=	-0.000039
# 17#	6.713617	-	6.713698	=	-0.000081
# 18#	6.666522	-	6.666549	=	-0.000027
# 19#	6.616502	-	6.616469	=	0.000033
# 20#	6.562092	-	6.561973	=	0.000119
# 21#	6.504717	-	6.504858	=	-0.000141
# 22#	6.442350	-	6.442323	=	0.000027
# 23#	6.376733	-	6.376764	=	-0.000031
# 24#	6.306626	-	6.306606	=	0.000020
# 25#	6.231925	-	6.231905	=	0.000020
# 26#	6.154296	-	6.154402	=	-0.000106
# 27#	6.073364	-	6.073282	=	0.000082
# 28#	5.990965	-	5.991036	=	-0.000071
# 29#	5.907078	-	5.906994	=	0.000084
# 30#	5.822075	-	5.822098	=	-0.000023
# 31#	5.739517	-	5.739577	=	-0.000060
# 32#	5.658823	-	5.658786	=	0.000037
# 33#	5.581147	-	5.581141	=	0.000006
# 34#	5.508755	-	5.508766	=	-0.000011
# 35#	5.441461	-	5.441432	=	0.000029
# 36#	5.381123	-	5.381198	=	-0.000075
# 37#	5.327163	-	5.327082	=	0.000081
# 38#	5.278909	-	5.278952	=	-0.000043
# 39#	5.238621	-	5.238625	=	-0.000004
# 40#	5.203704	-	5.203669	=	0.000035
# 41#	5.175705	-	5.175694	=	0.000011
# 42#	5.151696	-	5.151688	=	0.000008
# 43#	5.132035	-	5.132029	=	0.000006
# 44#	5.116533	-	5.116547	=	-0.000014
# 45#	5.103150	-	5.103080	=	0.000070
# 46#	5.089862	-	5.089874	=	-0.000012
# 47#	5.077944	-	5.077943	=	0.000001
# 48#	5.064287	-	5.064260	=	0.000027
# 49#	5.050127	-	5.050100	=	0.000027
# 50#	5.032274	-	5.032263	=	0.000011
# 51#	5.010634	-	5.010641	=	-0.000007
# 52#	4.986647	-	4.986612	=	0.000035
# 53#	4.958033	-	4.958008	=	0.000025

# 54#	4.923682	-	4.923722	= -0.000040
# 55#	4.886320	-	4.886289	= 0.000031
# 56#	4.843570	-	4.843547	= 0.000023
# 57#	4.797859	-	4.797834	= 0.000025
# 58#	4.747267	-	4.747261	= 0.000006
# 59#	4.692906	-	4.692935	= -0.000029
# 60#	4.636984	-	4.636971	= 0.000013
# 61#	4.577634	-	4.577653	= -0.000019
# 62#	4.516873	-	4.516809	= 0.000064
# 63#	4.454250	-	4.454309	= -0.000059
# 64#	4.390407	-	4.390377	= 0.000030
# 65#	4.326261	-	4.326252	= 0.000009
# 66#	4.261776	-	4.261758	= 0.000018
# 67#	4.196611	-	4.196588	= 0.000023
# 68#	4.133138	-	4.133156	= -0.000018
# 69#	4.068854	-	4.068863	= -0.000009
# 70#	4.006319	-	4.006312	= 0.000007
# 71#	3.944283	-	3.944305	= -0.000022
# 72#	3.882966	-	3.883006	= -0.000040
# 73#	3.823393	-	3.823366	= 0.000027
# 74#	3.764349	-	3.764355	= -0.000006
# 75#	3.707224	-	3.707219	= 0.000005
# 76#	3.651394	-	3.651382	= 0.000012
# 77#	3.596091	-	3.596104	= -0.000013
# 78#	3.542863	-	3.542911	= -0.000048
# 79#	3.490866	-	3.490805	= 0.000061
# 80#	3.439770	-	3.439804	= -0.000034
# 81#	3.390394	-	3.390388	= 0.000006
# 82#	3.341854	-	3.341862	= -0.000008
# 83#	3.295086	-	3.295065	= 0.000021
# 84#	3.249240	-	3.249247	= -0.000007
# 85#	3.204202	-	3.204187	= 0.000015
# 86#	3.161056	-	3.161060	= -0.000004
# 87#	3.118381	-	3.118368	= 0.000013
# 88#	3.077385	-	3.077433	= -0.000048
# 89#	3.037031	-	3.037010	= 0.000021
# 90#	2.997491	-	2.997495	= -0.000004
# 91#	2.959402	-	2.959394	= 0.000008
# 92#	2.922082	-	2.922088	= -0.000006
# 93#	2.885531	-	2.885518	= 0.000013
# 94#	2.850288	-	2.850299	= -0.000011
# 95#	2.815441	-	2.815414	= 0.000027
# 96#	2.782042	-	2.782031	= 0.000011
# 97#	2.748905	-	2.748934	= -0.000029
# 98#	2.716771	-	2.716724	= 0.000047
# 99#	2.685750	-	2.685753	= -0.000003
#100#	2.655015	-	2.655043	= -0.000028
#101#	2.625667	-	2.625618	= 0.000049
#102#	2.596769	-	2.596816	= -0.000047
#103#	2.568508	-	2.568478	= 0.000030
#104#	2.541322	-	2.541336	= -0.000014
#105#	2.514917	-	2.514887	= 0.000030
#106#	2.488898	-	2.488910	= -0.000012
#107#	2.464185	-	2.464180	= 0.000005
#108#	2.439741	-	2.439728	= 0.000013
#109#	2.416644	-	2.416625	= 0.000019
#110#	2.393949	-	2.393944	= 0.000005
#111#	2.372101	-	2.372123	= -0.000022
#112#	2.351355	-	2.351387	= -0.000032
#113#	2.331125	-	2.331074	= 0.000051

#114#	2.312085	-	2.312100	= -0.000015
#115#	2.293788	-	2.293772	= 0.000016
#116#	2.275994	-	2.276016	= -0.000022
#117#	2.259756	-	2.259735	= 0.000021
#118#	2.244261	-	2.244262	= -0.000001
#119#	2.229525	-	2.229496	= 0.000029
#120#	2.216112	-	2.216105	= 0.000007
#121#	2.203358	-	2.203394	= -0.000036
#122#	2.192027	-	2.192006	= 0.000021
#123#	2.181510	-	2.181510	= 0.000000
#124#	2.172107	-	2.172149	= -0.000042
#125#	2.164079	-	2.164049	= 0.000030
#126#	2.156984	-	2.156991	= -0.000007
#127#	2.151351	-	2.151349	= 0.000002
#128#	2.146831	-	2.146840	= -0.000009
#129#	2.143369	-	2.143394	= -0.000025
#130#	2.141605	-	2.141605	= 0.000000
#131#	2.140959	-	2.140964	= -0.000005
#132#	2.141577	-	2.141533	= 0.000044
#133#	2.143766	-	2.143801	= -0.000035
#134#	2.147307	-	2.147305	= 0.000002
#135#	2.152398	-	2.152403	= -0.000005
#136#	2.159059	-	2.159045	= 0.000014
#137#	2.166973	-	2.167010	= -0.000037
#138#	2.176993	-	2.176967	= 0.000026
#139#	2.187919	-	2.187889	= 0.000030
#140#	2.200994	-	2.200988	= 0.000006
#141#	2.215453	-	2.215446	= 0.000007
#142#	2.231534	-	2.231570	= -0.000036
#143#	2.249741	-	2.249726	= 0.000015
#144#	2.269529	-	2.269512	= 0.000017
#145#	2.290933	-	2.290930	= 0.000003
#146#	2.314465	-	2.314453	= 0.000012
#147#	2.339718	-	2.339802	= -0.000084
#148#	2.366856	-	2.366819	= 0.000037
#149#	2.396048	-	2.396041	= 0.000007
#150#	2.426922	-	2.426954	= -0.000032
#151#	2.460368	-	2.460350	= 0.000018
#152#	2.495596	-	2.495632	= -0.000036
#153#	2.533289	-	2.533273	= 0.000016
#154#	2.573098	-	2.573144	= -0.000046
#155#	2.615041	-	2.615009	= 0.000032
#156#	2.660334	-	2.660303	= 0.000031
#157#	2.707218	-	2.707252	= -0.000034
#158#	2.756886	-	2.756905	= -0.000019
#159#	2.810209	-	2.810211	= -0.000002
#160#	2.866256	-	2.866152	= 0.000104
#161#	2.925799	-	2.925848	= -0.000049
#162#	2.989113	-	2.989043	= 0.000070
#163#	3.056500	-	3.056522	= -0.000022
#164#	3.128820	-	3.128861	= -0.000041
#165#	3.206744	-	3.206781	= -0.000037
#166#	3.289545	-	3.289551	= -0.000006
#167#	3.380877	-	3.380826	= 0.000051
#168#	3.480385	-	3.480381	= 0.000004
#169#	3.590543	-	3.590542	= 0.000001
#170#	3.713624	-	3.713585	= 0.000039
#171#	3.853278	-	3.853352	= -0.000074
#172#	4.014617	-	4.014600	= 0.000017
#173#	4.203823	-	4.203851	= -0.000028

#174#	4.430908	-	4.430919	= -0.000011
#175#	4.713267	-	4.713243	= 0.000024
#176#	5.069776	-	5.069793	= -0.000017
#177#	5.540223	-	5.540285	= -0.000062
#178#	6.183492	-	6.183490	= 0.000002
#179#	7.106391	-	7.106303	= 0.000088
#180#	8.496127	-	8.496183	= -0.000056
#181#	10.717220	-	10.717340	= -0.000120
#182#	14.518382	-	14.518511	= -0.000129
#183#	21.580948	-	21.580397	= 0.000551
#184#	36.015701	-	36.015671	= 0.000030
#185#	68.915428	-	68.915298	= 0.000130
#186#	153.036469	-	153.036804	= -0.000335
#187#	388.466583	-	388.464844	= 0.001739
#188#	1001.560303	-	1001.554199	= 0.006104
#189#	1471.660645	-	1471.660156	= 0.000489
#190#	382.611206	-	382.616699	= -0.005493
#191#	194.489929	-	194.490799	= -0.000870
#192#	113.641403	-	113.641830	= -0.000427
#193#	77.408264	-	77.408279	= -0.000015
#194#	54.886456	-	54.886360	= 0.000096
#195#	41.568371	-	41.568657	= -0.000286
#196#	32.940239	-	32.939949	= 0.000290
#197#	27.057022	-	27.057114	= -0.000092
#198#	22.864288	-	22.864223	= 0.000065
#199#	19.761410	-	19.761532	= -0.000122
#200#	17.401985	-	17.401993	= -0.000008
#201#	15.556313	-	15.556107	= 0.000206
#202#	14.084104	-	14.084389	= -0.000285
#203#	12.884241	-	12.884226	= 0.000015
#204#	11.894434	-	11.894449	= -0.000015
#205#	11.065619	-	11.065678	= -0.000059
#206#	10.363224	-	10.363171	= 0.000053
#207#	9.760885	-	9.760659	= 0.000226
#208#	9.241501	-	9.241610	= -0.000109
#209#	8.788286	-	8.788197	= 0.000089
#210#	8.389639	-	8.389783	= -0.000144
#211#	8.038072	-	8.038127	= -0.000055
#212#	7.723260	-	7.723241	= 0.000019
#213#	7.442240	-	7.442255	= -0.000015
#214#	7.188755	-	7.188712	= 0.000043
#215#	6.958773	-	6.958816	= -0.000043
#216#	6.750341	-	6.750464	= -0.000123
#217#	6.559378	-	6.559294	= 0.000084
#218#	6.383584	-	6.383564	= 0.000020
#219#	6.221402	-	6.221494	= -0.000092
#220#	6.071763	-	6.071697	= 0.000066
#221#	5.931025	-	5.931022	= 0.000003
#222#	5.799909	-	5.799897	= 0.000012
#223#	5.676946	-	5.677041	= -0.000095
#224#	5.560748	-	5.560696	= 0.000052
#225#	5.451454	-	5.451405	= 0.000049
#226#	5.347633	-	5.347651	= -0.000018
#227#	5.248859	-	5.248859	= 0.000000
#228#	5.154617	-	5.154593	= 0.000024
#229#	5.065038	-	5.065052	= -0.000014
#230#	4.978469	-	4.978459	= 0.000010
#231#	4.895885	-	4.895859	= 0.000026
#232#	4.816682	-	4.816736	= -0.000054
#233#	4.741375	-	4.741409	= -0.000034

```
#234#          4.669307 -          4.669287 =          0.000020
Largest : 1471.660645 @ line #189#
Smallest: 2.140959 @ line #131#
Error   : 0.006104 @ line #188#
Error Ratio: 0.00%
```

Trajectory #2 6x10

::::::::: Norms for 6x10 Traj :#2 :::::::::

----------differences: ----------

STEP	FSP		Pseudo		diff
# 1#	2.129827	-	2.129826	=	0.000001
# 2#	2.313666	-	2.313662	=	0.000004
# 3#	2.311746	-	2.311749	=	-0.000003
# 4#	2.304901	-	2.304899	=	0.000002
# 5#	2.297971	-	2.297963	=	0.000008
# 6#	2.290728	-	2.290746	=	-0.000018
# 7#	2.283192	-	2.283182	=	0.000010
# 8#	2.275615	-	2.275620	=	-0.000005
# 9#	2.267645	-	2.267610	=	0.000035
# 10#	2.259753	-	2.259764	=	-0.000011
# 11#	2.251648	-	2.251640	=	0.000008
# 12#	2.243150	-	2.243155	=	-0.000005
# 13#	2.234684	-	2.234684	=	0.000000
# 14#	2.225801	-	2.225796	=	0.000005
# 15#	2.217033	-	2.217055	=	-0.000022
# 16#	2.207985	-	2.207987	=	-0.000002
# 17#	2.198601	-	2.198593	=	0.000008
# 18#	2.189192	-	2.189176	=	0.000016
# 19#	2.179678	-	2.179667	=	0.000011
# 20#	2.169912	-	2.169895	=	0.000017
# 21#	2.160027	-	2.160070	=	-0.000043
# 22#	2.149962	-	2.149948	=	0.000014
# 23#	2.139878	-	2.139867	=	0.000011
# 24#	2.129525	-	2.129537	=	-0.000012
# 25#	2.118953	-	2.118953	=	0.000000
# 26#	2.108312	-	2.108304	=	0.000008
# 27#	2.097510	-	2.097509	=	0.000001
# 28#	2.086703	-	2.086712	=	-0.000009
# 29#	2.075705	-	2.075696	=	0.000009
# 30#	2.064470	-	2.064446	=	0.000024
# 31#	2.053237	-	2.053290	=	-0.000053
# 32#	2.041876	-	2.041828	=	0.000048
# 33#	2.030180	-	2.030170	=	0.000010
# 34#	2.018471	-	2.018481	=	-0.000010
# 35#	2.006777	-	2.006769	=	0.000008
# 36#	1.994960	-	1.994987	=	-0.000027
# 37#	1.982840	-	1.982847	=	-0.000007
# 38#	1.970782	-	1.970722	=	0.000060
# 39#	1.958581	-	1.958604	=	-0.000023
# 40#	1.946143	-	1.946107	=	0.000036
# 41#	1.933737	-	1.933746	=	-0.000009
# 42#	1.921139	-	1.921148	=	-0.000009
# 43#	1.908532	-	1.908532	=	0.000000
# 44#	1.895867	-	1.895849	=	0.000018
# 45#	1.882974	-	1.882931	=	0.000043
# 46#	1.869883	-	1.869874	=	0.000009
# 47#	1.857004	-	1.857036	=	-0.000032
# 48#	1.843812	-	1.843835	=	-0.000023
# 49#	1.830604	-	1.830624	=	-0.000020
# 50#	1.817379	-	1.817340	=	0.000039
# 51#	1.803770	-	1.803770	=	0.000000
# 52#	1.790436	-	1.790468	=	-0.000032
# 53#	1.776760	-	1.776747	=	0.000013

# 54#	1.763159	-	1.763163	= -0.000004
# 55#	1.749440	-	1.749449	= -0.000009
# 56#	1.735507	-	1.735500	= 0.000007
# 57#	1.721647	-	1.721644	= 0.000003
# 58#	1.707724	-	1.707718	= 0.000006
# 59#	1.693389	-	1.693406	= -0.000017
# 60#	1.679464	-	1.679463	= 0.000001
# 61#	1.665097	-	1.665069	= 0.000028
# 62#	1.650868	-	1.650877	= -0.000009
# 63#	1.636475	-	1.636478	= -0.000003
# 64#	1.621879	-	1.621897	= -0.000018
# 65#	1.607283	-	1.607298	= -0.000015
# 66#	1.592773	-	1.592784	= -0.000011
# 67#	1.577908	-	1.577893	= 0.000015
# 68#	1.563184	-	1.563164	= 0.000020
# 69#	1.548288	-	1.548290	= -0.000002
# 70#	1.533389	-	1.533413	= -0.000024
# 71#	1.518337	-	1.518346	= -0.000009
# 72#	1.503060	-	1.503056	= 0.000004
# 73#	1.487996	-	1.488021	= -0.000025
# 74#	1.472673	-	1.472636	= 0.000037
# 75#	1.457290	-	1.457290	= 0.000000
# 76#	1.441872	-	1.441906	= -0.000034
# 77#	1.426414	-	1.426415	= -0.000001
# 78#	1.410845	-	1.410833	= 0.000012
# 79#	1.395229	-	1.395240	= -0.000011
# 80#	1.379420	-	1.379435	= -0.000015
# 81#	1.363740	-	1.363735	= 0.000005
# 82#	1.347849	-	1.347865	= -0.000016
# 83#	1.331915	-	1.331926	= -0.000011
# 84#	1.315889	-	1.315889	= 0.000000
# 85#	1.299920	-	1.299911	= 0.000009
# 86#	1.283921	-	1.283934	= -0.000013
# 87#	1.267803	-	1.267791	= 0.000012
# 88#	1.251608	-	1.251606	= 0.000002
# 89#	1.235485	-	1.235495	= -0.000010
# 90#	1.219267	-	1.219278	= -0.000011
# 91#	1.203027	-	1.203010	= 0.000017
# 92#	1.186643	-	1.186646	= -0.000003
# 93#	1.170569	-	1.170580	= -0.000011
# 94#	1.154372	-	1.154384	= -0.000012
# 95#	1.138170	-	1.138151	= 0.000019
# 96#	1.121924	-	1.121930	= -0.000006
# 97#	1.105819	-	1.105813	= 0.000006
# 98#	1.089742	-	1.089760	= -0.000018
# 99#	1.073771	-	1.073767	= 0.000004
#100#	1.057679	-	1.057672	= 0.000007
#101#	1.041929	-	1.041947	= -0.000018
#102#	1.026247	-	1.026257	= -0.000010
#103#	1.010644	-	1.010608	= 0.000036
#104#	0.995051	-	0.995060	= -0.000009
#105#	0.979832	-	0.979834	= -0.000002
#106#	0.964641	-	0.964626	= 0.000015
#107#	0.949678	-	0.949674	= 0.000004
#108#	0.934750	-	0.934758	= -0.000008
#109#	0.920306	-	0.920287	= 0.000019
#110#	0.905848	-	0.905869	= -0.000021
#111#	0.891749	-	0.891747	= 0.000002
#112#	0.877902	-	0.877909	= -0.000007
#113#	0.864305	-	0.864286	= 0.000019

#114#	0.850915	-	0.850942	= -0.000027
#115#	0.837843	-	0.837819	= 0.000024
#116#	0.825063	-	0.825076	= -0.000013
#117#	0.812574	-	0.812569	= 0.000005
#118#	0.800460	-	0.800455	= 0.000005
#119#	0.788566	-	0.788567	= -0.000001
#120#	0.777027	-	0.777034	= -0.000007
#121#	0.765875	-	0.765878	= -0.000003
#122#	0.755018	-	0.755011	= 0.000007
#123#	0.744442	-	0.744432	= 0.000010
#124#	0.734281	-	0.734289	= -0.000008
#125#	0.724393	-	0.724378	= 0.000015
#126#	0.714920	-	0.714930	= -0.000010
#127#	0.705729	-	0.705730	= -0.000001
#128#	0.696934	-	0.696931	= 0.000003
#129#	0.688423	-	0.688422	= 0.000001
#130#	0.680292	-	0.680283	= 0.000009
#131#	0.672465	-	0.672474	= -0.000009
#132#	0.665000	-	0.665006	= -0.000006
#133#	0.657792	-	0.657798	= -0.000006
#134#	0.651035	-	0.651033	= 0.000002
#135#	0.644514	-	0.644514	= 0.000000
#136#	0.638331	-	0.638340	= -0.000009
#137#	0.632473	-	0.632463	= 0.000010
#138#	0.626877	-	0.626878	= -0.000001
#139#	0.621640	-	0.621636	= 0.000004
#140#	0.616682	-	0.616676	= 0.000006
#141#	0.611952	-	0.611962	= -0.000010
#142#	0.607659	-	0.607656	= 0.000003
#143#	0.603543	-	0.603533	= 0.000010
#144#	0.599716	-	0.599722	= -0.000006
#145#	0.596195	-	0.596200	= -0.000005
#146#	0.592925	-	0.592920	= 0.000005
#147#	0.589913	-	0.589912	= 0.000001
#148#	0.587185	-	0.587199	= -0.000014
#149#	0.584710	-	0.584705	= 0.000005
#150#	0.582451	-	0.582438	= 0.000013
#151#	0.580491	-	0.580505	= -0.000014
#152#	0.578777	-	0.578788	= -0.000011
#153#	0.577319	-	0.577290	= 0.000029
#154#	0.576034	-	0.576045	= -0.000011
#155#	0.575045	-	0.575084	= -0.000039
#156#	0.574326	-	0.574261	= 0.000065
#157#	0.573662	-	0.573732	= -0.000070
#158#	0.573437	-	0.573405	= 0.000032
#159#	0.573346	-	0.573350	= -0.000004
#160#	0.573505	-	0.573512	= -0.000007
#161#	0.573912	-	0.573871	= 0.000041
#162#	0.574084	-	0.574472	= -0.000388
#163#	0.575931	-	0.575296	= 0.000635
#164#	0.576264	-	0.576310	= -0.000046
#165#	0.579286	-	0.577551	= 0.001735
#166#	0.577061	-	0.579020	= -0.001959
#167#	0.580761	-	0.580722	= 0.000039
#168#	0.582576	-	0.582639	= -0.000063
#169#	0.584682	-	0.584738	= -0.000056
#170#	0.587135	-	0.587067	= 0.000068
#171#	0.589581	-	0.589619	= -0.000038
#172#	0.592359	-	0.592352	= 0.000007
#173#	0.595390	-	0.595336	= 0.000054

#174#	0.598479	-	0.598463	= 0.000016
#175#	0.601891	-	0.601887	= 0.000004
#176#	0.605510	-	0.605498	= 0.000012
#177#	0.609267	-	0.609284	= -0.000017
#178#	0.613301	-	0.613314	= -0.000013
#179#	0.617577	-	0.617557	= 0.000020
#180#	0.621949	-	0.621976	= -0.000027
#181#	0.626670	-	0.626649	= 0.000021
#182#	0.631495	-	0.631497	= -0.000002
#183#	0.636487	-	0.636483	= 0.000004
#184#	0.641846	-	0.641841	= 0.000005
#185#	0.647291	-	0.647311	= -0.000020
#186#	0.653046	-	0.653050	= -0.000004
#187#	0.658911	-	0.658900	= 0.000011
#188#	0.665025	-	0.665030	= -0.000005
#189#	0.671388	-	0.671381	= 0.000007
#190#	0.677896	-	0.677883	= 0.000013
#191#	0.684560	-	0.684579	= -0.000019
#192#	0.691597	-	0.691594	= 0.000003
#193#	0.698758	-	0.698766	= -0.000008
#194#	0.706139	-	0.706143	= -0.000004
#195#	0.713680	-	0.713676	= 0.000004
#196#	0.721465	-	0.721464	= 0.000001
#197#	0.729482	-	0.729474	= 0.000008
#198#	0.737660	-	0.737671	= -0.000011
#199#	0.746016	-	0.746017	= -0.000001
#200#	0.754680	-	0.754665	= 0.000015
#201#	0.763489	-	0.763500	= -0.000011
#202#	0.772587	-	0.772584	= 0.000003
#203#	0.781737	-	0.781749	= -0.000012
#204#	0.791173	-	0.791183	= -0.000010
#205#	0.800840	-	0.800837	= 0.000003
#206#	0.810661	-	0.810678	= -0.000017
#207#	0.820660	-	0.820663	= -0.000003
#208#	0.830971	-	0.830967	= 0.000004
#209#	0.841395	-	0.841416	= -0.000021
#210#	0.852086	-	0.852085	= 0.000001
#211#	0.862985	-	0.862976	= 0.000009
#212#	0.873974	-	0.873988	= -0.000014
#213#	0.885208	-	0.885227	= -0.000019
#214#	0.896690	-	0.896672	= 0.000018
#215#	0.908231	-	0.908251	= -0.000020
#216#	0.920166	-	0.920160	= 0.000006
#217#	0.932187	-	0.932189	= -0.000002
#218#	0.944430	-	0.944452	= -0.000022
#219#	0.956843	-	0.956850	= -0.000007
#220#	0.969549	-	0.969558	= -0.000009
#221#	0.982264	-	0.982260	= 0.000004
#222#	0.995273	-	0.995287	= -0.000014
#223#	1.008457	-	1.008476	= -0.000019
#224#	1.021740	-	1.021751	= -0.000011
#225#	1.035362	-	1.035361	= 0.000001
#226#	1.049106	-	1.049107	= -0.000001
#227#	1.063032	-	1.063062	= -0.000030
#228#	1.077159	-	1.077155	= 0.000004
#229#	1.091489	-	1.091483	= 0.000006
#230#	1.105871	-	1.105870	= 0.000001
#231#	1.120498	-	1.120525	= -0.000027
#232#	1.135252	-	1.135227	= 0.000025
#233#	1.150268	-	1.150313	= -0.000045

```
#234#          1.165468 -          1.165456 =          0.000012
Largest : 2.313666 @ line #  2#
Smallest: 0.573346 @ line #159#
Error   : 0.001959 @ line #166#
Error Ratio: 0.34%
```

Trajectory #3 3x10

:::::::: Norms for 3x10 Traj :#3 ::::::::

----------differences: ----------

STEP	FSP		Pseudo		diff
# 1#	0.065670	-	0.065671	=	-0.000001
# 2#	0.063128	-	0.063128	=	0.000000
# 3#	0.062826	-	0.062825	=	0.000001
# 4#	0.062535	-	0.062535	=	0.000000
# 5#	0.062239	-	0.062239	=	0.000000
# 6#	0.061945	-	0.061946	=	-0.000001
# 7#	0.061634	-	0.061633	=	0.000001
# 8#	0.061308	-	0.061308	=	0.000000
# 9#	0.061045	-	0.061044	=	0.000001
# 10#	0.060747	-	0.060749	=	-0.000002
# 11#	0.060458	-	0.060459	=	-0.000001
# 12#	0.060173	-	0.060172	=	0.000001
# 13#	0.059873	-	0.059872	=	0.000001
# 14#	0.059606	-	0.059606	=	0.000000
# 15#	0.059325	-	0.059327	=	-0.000002
# 16#	0.059071	-	0.059071	=	0.000000
# 17#	0.058777	-	0.058777	=	0.000000
# 18#	0.058481	-	0.058480	=	0.000001
# 19#	0.058232	-	0.058232	=	0.000000
# 20#	0.057968	-	0.057967	=	0.000001
# 21#	0.057701	-	0.057702	=	-0.000001
# 22#	0.057439	-	0.057439	=	0.000000
# 23#	0.057153	-	0.057153	=	0.000000
# 24#	0.056926	-	0.056923	=	0.000003
# 25#	0.056657	-	0.056659	=	-0.000002
# 26#	0.056426	-	0.056426	=	0.000000
# 27#	0.056160	-	0.056160	=	0.000000
# 28#	0.055886	-	0.055885	=	0.000001
# 29#	0.055660	-	0.055664	=	-0.000004
# 30#	0.055412	-	0.055408	=	0.000004
# 31#	0.055169	-	0.055174	=	-0.000005
# 32#	0.054932	-	0.054936	=	-0.000004
# 33#	0.054678	-	0.054671	=	0.000007
# 34#	0.054450	-	0.054451	=	-0.000001
# 35#	0.054224	-	0.054228	=	-0.000004
# 36#	0.054003	-	0.054000	=	0.000003
# 37#	0.053760	-	0.053758	=	0.000002
# 38#	0.053513	-	0.053513	=	0.000000
# 39#	0.053308	-	0.053307	=	0.000001
# 40#	0.053086	-	0.053087	=	-0.000001
# 41#	0.052851	-	0.052854	=	-0.000003
# 42#	0.052646	-	0.052645	=	0.000001
# 43#	0.052404	-	0.052406	=	-0.000002
# 44#	0.052211	-	0.052211	=	0.000000
# 45#	0.051992	-	0.051992	=	0.000000
# 46#	0.051784	-	0.051786	=	-0.000002
# 47#	0.051571	-	0.051573	=	-0.000002
# 48#	0.051370	-	0.051366	=	0.000004
# 49#	0.051162	-	0.051165	=	-0.000003
# 50#	0.050961	-	0.050959	=	0.000002
# 51#	0.050759	-	0.050758	=	0.000001
# 52#	0.050542	-	0.050544	=	-0.000002
# 53#	0.050361	-	0.050362	=	-0.000001

# 54#	0.050169	-	0.050169	= 0.000000
# 55#	0.049976	-	0.049975	= 0.000001
# 56#	0.049787	-	0.049785	= 0.000002
# 57#	0.049594	-	0.049592	= 0.000002
# 58#	0.049407	-	0.049407	= 0.000000
# 59#	0.049226	-	0.049227	= -0.000001
# 60#	0.049039	-	0.049037	= 0.000002
# 61#	0.048865	-	0.048868	= -0.000003
# 62#	0.048663	-	0.048663	= 0.000000
# 63#	0.048510	-	0.048507	= 0.000003
# 64#	0.048327	-	0.048330	= -0.000003
# 65#	0.048160	-	0.048158	= 0.000002
# 66#	0.047986	-	0.047986	= 0.000000
# 67#	0.047820	-	0.047821	= -0.000001
# 68#	0.047654	-	0.047652	= 0.000002
# 69#	0.047488	-	0.047490	= -0.000002
# 70#	0.047326	-	0.047326	= 0.000000
# 71#	0.047165	-	0.047164	= 0.000001
# 72#	0.046995	-	0.046996	= -0.000001
# 73#	0.046850	-	0.046848	= 0.000002
# 74#	0.046698	-	0.046698	= 0.000000
# 75#	0.046542	-	0.046541	= 0.000001
# 76#	0.046396	-	0.046395	= 0.000001
# 77#	0.046242	-	0.046243	= -0.000001
# 78#	0.046102	-	0.046102	= 0.000000
# 79#	0.045957	-	0.045955	= 0.000002
# 80#	0.045812	-	0.045818	= -0.000006
# 81#	0.045679	-	0.045677	= 0.000002
# 82#	0.045522	-	0.045520	= 0.000002
# 83#	0.045404	-	0.045404	= 0.000000
# 84#	0.045267	-	0.045270	= -0.000003
# 85#	0.045140	-	0.045140	= 0.000000
# 86#	0.045007	-	0.045007	= 0.000000
# 87#	0.044870	-	0.044871	= -0.000001
# 88#	0.044772	-	0.044768	= 0.000004
# 89#	0.044636	-	0.044640	= -0.000004
# 90#	0.044514	-	0.044512	= 0.000002
# 91#	0.044400	-	0.044403	= -0.000003
# 92#	0.044264	-	0.044259	= 0.000005
# 93#	0.044173	-	0.044173	= 0.000000
# 94#	0.044055	-	0.044057	= -0.000002
# 95#	0.043950	-	0.043951	= -0.000001
# 96#	0.043843	-	0.043841	= 0.000002
# 97#	0.043723	-	0.043720	= 0.000003
# 98#	0.043648	-	0.043650	= -0.000002
# 99#	0.043535	-	0.043534	= 0.000001
#100#	0.043439	-	0.043441	= -0.000002
#101#	0.043328	-	0.043326	= 0.000002
#102#	0.043254	-	0.043256	= -0.000002
#103#	0.043162	-	0.043159	= 0.000003
#104#	0.043077	-	0.043075	= 0.000002
#105#	0.042984	-	0.042985	= -0.000001
#106#	0.042895	-	0.042897	= -0.000002
#107#	0.042826	-	0.042821	= 0.000005
#108#	0.042764	-	0.042764	= 0.000000
#109#	0.042673	-	0.042674	= -0.000001
#110#	0.042603	-	0.042600	= 0.000003
#111#	0.042518	-	0.042520	= -0.000002
#112#	0.042462	-	0.042463	= -0.000001
#113#	0.042402	-	0.042404	= -0.000002

#114#	0.042334	-	0.042338	= -0.000004
#115#	0.042285	-	0.042283	= 0.000002
#116#	0.042208	-	0.042206	= 0.000002
#117#	0.042170	-	0.042173	= -0.000003
#118#	0.042128	-	0.042128	= 0.000000
#119#	0.042074	-	0.042077	= -0.000003
#120#	0.042025	-	0.042023	= 0.000002
#121#	0.041972	-	0.041971	= 0.000001
#122#	0.041949	-	0.041947	= 0.000002
#123#	0.041910	-	0.041909	= 0.000001
#124#	0.041883	-	0.041878	= 0.000005
#125#	0.041844	-	0.041849	= -0.000005
#126#	0.041808	-	0.041810	= -0.000002
#127#	0.041800	-	0.041797	= 0.000003
#128#	0.041780	-	0.041780	= 0.000000
#129#	0.041770	-	0.041770	= 0.000000
#130#	0.041753	-	0.041751	= 0.000002
#131#	0.041719	-	0.041720	= -0.000001
#132#	0.041734	-	0.041736	= -0.000002
#133#	0.041731	-	0.041727	= 0.000004
#134#	0.041728	-	0.041731	= -0.000003
#135#	0.041733	-	0.041731	= 0.000002
#136#	0.041728	-	0.041728	= 0.000000
#137#	0.041752	-	0.041755	= -0.000003
#138#	0.041770	-	0.041767	= 0.000003
#139#	0.041794	-	0.041796	= -0.000002
#140#	0.041807	-	0.041810	= -0.000003
#141#	0.041822	-	0.041818	= 0.000004
#142#	0.041864	-	0.041866	= -0.000002
#143#	0.041902	-	0.041902	= 0.000000
#144#	0.041938	-	0.041938	= 0.000000
#145#	0.041985	-	0.041983	= 0.000002
#146#	0.042011	-	0.042013	= -0.000002
#147#	0.042083	-	0.042087	= -0.000004
#148#	0.042137	-	0.042133	= 0.000004
#149#	0.042207	-	0.042207	= 0.000000
#150#	0.042263	-	0.042266	= -0.000003
#151#	0.042315	-	0.042312	= 0.000003
#152#	0.042407	-	0.042410	= -0.000003
#153#	0.042486	-	0.042488	= -0.000002
#154#	0.042574	-	0.042573	= 0.000001
#155#	0.042639	-	0.042641	= -0.000002
#156#	0.042760	-	0.042759	= 0.000001
#157#	0.042854	-	0.042855	= -0.000001
#158#	0.042964	-	0.042959	= 0.000005
#159#	0.043078	-	0.043080	= -0.000002
#160#	0.043169	-	0.043171	= -0.000002
#161#	0.043309	-	0.043308	= 0.000001
#162#	0.043437	-	0.043437	= 0.000000
#163#	0.043575	-	0.043570	= 0.000005
#164#	0.043709	-	0.043710	= -0.000001
#165#	0.043835	-	0.043838	= -0.000003
#166#	0.044007	-	0.044010	= -0.000003
#167#	0.044170	-	0.044164	= 0.000006
#168#	0.044328	-	0.044330	= -0.000002
#169#	0.044508	-	0.044510	= -0.000002
#170#	0.044660	-	0.044658	= 0.000002
#171#	0.044867	-	0.044868	= -0.000001
#172#	0.045059	-	0.045060	= -0.000001
#173#	0.045261	-	0.045261	= 0.000000

#174#	0.045465	-	0.045467	= -0.000002
#175#	0.045660	-	0.045659	= 0.000001
#176#	0.045903	-	0.045901	= 0.000002
#177#	0.046136	-	0.046135	= 0.000001
#178#	0.046370	-	0.046374	= -0.000004
#179#	0.046620	-	0.046615	= 0.000005
#180#	0.046858	-	0.046859	= -0.000001
#181#	0.047133	-	0.047136	= -0.000003
#182#	0.047414	-	0.047414	= 0.000000
#183#	0.047694	-	0.047692	= 0.000002
#184#	0.047983	-	0.047986	= -0.000003
#185#	0.048260	-	0.048261	= -0.000001
#186#	0.048599	-	0.048596	= 0.000003
#187#	0.048918	-	0.048918	= 0.000000
#188#	0.049251	-	0.049249	= 0.000002
#189#	0.049586	-	0.049595	= -0.000009
#190#	0.049926	-	0.049920	= 0.000006
#191#	0.050302	-	0.050304	= -0.000002
#192#	0.050682	-	0.050680	= 0.000002
#193#	0.051067	-	0.051070	= -0.000003
#194#	0.051466	-	0.051465	= 0.000001
#195#	0.051851	-	0.051849	= 0.000002
#196#	0.052295	-	0.052298	= -0.000003
#197#	0.052739	-	0.052738	= 0.000001
#198#	0.053180	-	0.053183	= -0.000003
#199#	0.053649	-	0.053648	= 0.000001
#200#	0.054097	-	0.054096	= 0.000001
#201#	0.054617	-	0.054613	= 0.000004
#202#	0.055119	-	0.055117	= 0.000002
#203#	0.055638	-	0.055638	= 0.000000
#204#	0.056174	-	0.056174	= 0.000000
#205#	0.056691	-	0.056692	= -0.000001
#206#	0.057296	-	0.057291	= 0.000005
#207#	0.057871	-	0.057874	= -0.000003
#208#	0.058480	-	0.058477	= 0.000003
#209#	0.059063	-	0.059066	= -0.000003
#210#	0.059735	-	0.059733	= 0.000002
#211#	0.060382	-	0.060381	= 0.000001
#212#	0.061061	-	0.061059	= 0.000002
#213#	0.061752	-	0.061754	= -0.000002
#214#	0.062433	-	0.062435	= -0.000002
#215#	0.063197	-	0.063201	= -0.000004
#216#	0.063953	-	0.063955	= -0.000002
#217#	0.064728	-	0.064728	= 0.000000
#218#	0.065520	-	0.065526	= -0.000006
#219#	0.066315	-	0.066307	= 0.000008
#220#	0.067186	-	0.067190	= -0.000004
#221#	0.068061	-	0.068059	= 0.000002
#222#	0.068947	-	0.068950	= -0.000003
#223#	0.069870	-	0.069866	= 0.000004
#224#	0.070772	-	0.070773	= -0.000001
#225#	0.071779	-	0.071779	= 0.000000
#226#	0.072778	-	0.072779	= -0.000001
#227#	0.073800	-	0.073803	= -0.000003
#228#	0.074856	-	0.074854	= 0.000002
#229#	0.075898	-	0.075902	= -0.000004
#230#	0.077048	-	0.077049	= -0.000001
#231#	0.078193	-	0.078201	= -0.000008
#232#	0.079371	-	0.079373	= -0.000002
#233#	0.080581	-	0.080578	= 0.000003

```
#234#          0.081772 -          0.081778 =         -0.000006
#235#          0.083090 -          0.083095 =         -0.000005
#236#          0.084402 -          0.084406 =         -0.000004
#237#          0.085749 -          0.085748 =          0.000001
#238#          0.087124 -          0.087131 =         -0.000007
#239#          0.088498 -          0.088498 =          0.000000
#240#          0.090002 -          0.089997 =          0.000005
#241#          0.091494 -          0.091494 =          0.000000
Largest : 0.091494 @ line #241#
Smallest: 0.041719 @ line #131#
Error   : 0.000009 @ line #189#
Error Ratio: 0.02%
```

Trajectory #3 3x7

:::::::: Norms for 3x7 Traj :#3 ::::::::

----------differences: ----------

STEP	FSP		Pseudo		diff
# 1#	0.208086	-	0.208087	=	-0.000001
# 2#	0.193074	-	0.193074	=	0.000000
# 3#	0.191782	-	0.191783	=	-0.000001
# 4#	0.189954	-	0.189953	=	0.000001
# 5#	0.188204	-	0.188204	=	0.000000
# 6#	0.186508	-	0.186507	=	0.000001
# 7#	0.184735	-	0.184735	=	0.000000
# 8#	0.182956	-	0.182956	=	0.000000
# 9#	0.181382	-	0.181382	=	0.000000
# 10#	0.179705	-	0.179706	=	-0.000001
# 11#	0.178108	-	0.178103	=	0.000005
# 12#	0.176519	-	0.176528	=	-0.000009
# 13#	0.174930	-	0.174915	=	0.000015
# 14#	0.173420	-	0.173435	=	-0.000015
# 15#	0.171898	-	0.171886	=	0.000012
# 16#	0.170505	-	0.170506	=	-0.000001
# 17#	0.168924	-	0.168934	=	-0.000010
# 18#	0.167454	-	0.167450	=	0.000004
# 19#	0.166063	-	0.166051	=	0.000012
# 20#	0.164644	-	0.164658	=	-0.000014
# 21#	0.163257	-	0.163254	=	0.000003
# 22#	0.161905	-	0.161904	=	0.000001
# 23#	0.160502	-	0.160502	=	0.000000
# 24#	0.159218	-	0.159216	=	0.000002
# 25#	0.157885	-	0.157871	=	0.000014
# 26#	0.156680	-	0.156694	=	-0.000014
# 27#	0.155314	-	0.155317	=	-0.000003
# 28#	0.154010	-	0.154014	=	-0.000004
# 29#	0.152819	-	0.152800	=	0.000019
# 30#	0.151562	-	0.151558	=	0.000004
# 31#	0.150335	-	0.150351	=	-0.000016
# 32#	0.149139	-	0.149132	=	0.000007
# 33#	0.147918	-	0.147921	=	-0.000003
# 34#	0.146782	-	0.146774	=	0.000008
# 35#	0.145607	-	0.145608	=	-0.000001
# 36#	0.144520	-	0.144532	=	-0.000012
# 37#	0.143329	-	0.143327	=	0.000002
# 38#	0.142173	-	0.142164	=	0.000009
# 39#	0.141079	-	0.141090	=	-0.000011
# 40#	0.140010	-	0.140003	=	0.000007
# 41#	0.138892	-	0.138892	=	0.000000
# 42#	0.137829	-	0.137828	=	0.000001
# 43#	0.136728	-	0.136699	=	0.000029
# 44#	0.135711	-	0.135717	=	-0.000006
# 45#	0.134640	-	0.134654	=	-0.000014
# 46#	0.133645	-	0.133633	=	0.000012
# 47#	0.132615	-	0.132629	=	-0.000014
# 48#	0.131582	-	0.131589	=	-0.000007
# 49#	0.130574	-	0.130575	=	-0.000001
# 50#	0.129592	-	0.129591	=	0.000001
# 51#	0.128590	-	0.128591	=	-0.000001
# 52#	0.127602	-	0.127602	=	0.000000
# 53#	0.126648	-	0.126648	=	0.000000

# 54#	0.125699	-	0.125691	= 0.000008
# 55#	0.124724	-	0.124737	= -0.000013
# 56#	0.123792	-	0.123792	= 0.000000
# 57#	0.122903	-	0.122902	= 0.000001
# 58#	0.121925	-	0.121918	= 0.000007
# 59#	0.121013	-	0.121012	= 0.000001
# 60#	0.120099	-	0.120097	= 0.000002
# 61#	0.119185	-	0.119202	= -0.000017
# 62#	0.118262	-	0.118261	= 0.000001
# 63#	0.117400	-	0.117400	= 0.000000
# 64#	0.116525	-	0.116517	= 0.000008
# 65#	0.115635	-	0.115644	= -0.000009
# 66#	0.114764	-	0.114754	= 0.000010
# 67#	0.113945	-	0.113944	= 0.000001
# 68#	0.113050	-	0.113048	= 0.000002
# 69#	0.112183	-	0.112185	= -0.000002
# 70#	0.111347	-	0.111347	= 0.000000
# 71#	0.110489	-	0.110497	= -0.000008
# 72#	0.109659	-	0.109659	= 0.000000
# 73#	0.108842	-	0.108835	= 0.000007
# 74#	0.108017	-	0.108019	= -0.000002
# 75#	0.107194	-	0.107193	= 0.000001
# 76#	0.106388	-	0.106396	= -0.000008
# 77#	0.105633	-	0.105623	= 0.000010
# 78#	0.104777	-	0.104786	= -0.000009
# 79#	0.103982	-	0.103977	= 0.000005
# 80#	0.103213	-	0.103204	= 0.000009
# 81#	0.102416	-	0.102419	= -0.000003
# 82#	0.101612	-	0.101623	= -0.000011
# 83#	0.100866	-	0.100858	= 0.000008
# 84#	0.100078	-	0.100082	= -0.000004
# 85#	0.099330	-	0.099331	= -0.000001
# 86#	0.098567	-	0.098567	= 0.000000
# 87#	0.097795	-	0.097795	= 0.000000
# 88#	0.097119	-	0.097119	= 0.000000
# 89#	0.096330	-	0.096328	= 0.000002
# 90#	0.095569	-	0.095569	= 0.000000
# 91#	0.094855	-	0.094867	= -0.000012
# 92#	0.094099	-	0.094089	= 0.000010
# 93#	0.093408	-	0.093404	= 0.000004
# 94#	0.092658	-	0.092664	= -0.000006
# 95#	0.091975	-	0.091976	= -0.000001
# 96#	0.091260	-	0.091252	= 0.000008
# 97#	0.090529	-	0.090528	= 0.000001
# 98#	0.089911	-	0.089913	= -0.000002
# 99#	0.089152	-	0.089157	= -0.000005
#100#	0.088483	-	0.088484	= -0.000001
#101#	0.087771	-	0.087766	= 0.000005
#102#	0.087126	-	0.087125	= 0.000001
#103#	0.086435	-	0.086440	= -0.000005
#104#	0.085793	-	0.085781	= 0.000012
#105#	0.085113	-	0.085114	= -0.000001
#106#	0.084458	-	0.084464	= -0.000006
#107#	0.083809	-	0.083814	= -0.000005
#108#	0.083237	-	0.083230	= 0.000007
#109#	0.082546	-	0.082558	= -0.000012
#110#	0.081913	-	0.081901	= 0.000012
#111#	0.081292	-	0.081293	= -0.000001
#112#	0.080674	-	0.080679	= -0.000005
#113#	0.080077	-	0.080084	= -0.000007

#114#	0.079475	-	0.079471	= 0.000004
#115#	0.078891	-	0.078890	= 0.000001
#116#	0.078281	-	0.078276	= 0.000005
#117#	0.077727	-	0.077730	= -0.000003
#118#	0.077188	-	0.077188	= 0.000000
#119#	0.076600	-	0.076598	= 0.000002
#120#	0.076033	-	0.076034	= -0.000001
#121#	0.075492	-	0.075498	= -0.000006
#122#	0.074981	-	0.074979	= 0.000002
#123#	0.074447	-	0.074440	= 0.000007
#124#	0.073940	-	0.073941	= -0.000001
#125#	0.073420	-	0.073426	= -0.000006
#126#	0.072936	-	0.072933	= 0.000003
#127#	0.072461	-	0.072462	= -0.000001
#128#	0.072000	-	0.071998	= 0.000002
#129#	0.071567	-	0.071567	= 0.000000
#130#	0.071099	-	0.071098	= 0.000001
#131#	0.070642	-	0.070641	= 0.000001
#132#	0.070259	-	0.070261	= -0.000002
#133#	0.069849	-	0.069845	= 0.000004
#134#	0.069468	-	0.069464	= 0.000004
#135#	0.069081	-	0.069088	= -0.000007
#136#	0.068723	-	0.068725	= -0.000002
#137#	0.068406	-	0.068407	= -0.000001
#138#	0.068071	-	0.068073	= -0.000002
#139#	0.067803	-	0.067801	= 0.000002
#140#	0.067489	-	0.067484	= 0.000005
#141#	0.067209	-	0.067212	= -0.000003
#142#	0.066975	-	0.066978	= -0.000003
#143#	0.066758	-	0.066761	= -0.000003
#144#	0.066551	-	0.066550	= 0.000001
#145#	0.066378	-	0.066376	= 0.000002
#146#	0.066199	-	0.066195	= 0.000004
#147#	0.066097	-	0.066099	= -0.000002
#148#	0.065978	-	0.065978	= 0.000000
#149#	0.065940	-	0.065936	= 0.000004
#150#	0.065864	-	0.065869	= -0.000005
#151#	0.065815	-	0.065813	= 0.000002
#152#	0.065858	-	0.065856	= 0.000002
#153#	0.065887	-	0.065891	= -0.000004
#154#	0.065975	-	0.065971	= 0.000004
#155#	0.066053	-	0.066050	= 0.000003
#156#	0.066225	-	0.066231	= -0.000006
#157#	0.066403	-	0.066398	= 0.000005
#158#	0.066633	-	0.066633	= 0.000000
#159#	0.066900	-	0.066898	= 0.000002
#160#	0.067161	-	0.067161	= 0.000000
#161#	0.067533	-	0.067530	= 0.000003
#162#	0.067926	-	0.067928	= -0.000002
#163#	0.068370	-	0.068370	= 0.000000
#164#	0.068843	-	0.068841	= 0.000002
#165#	0.069361	-	0.069360	= 0.000001
#166#	0.069968	-	0.069969	= -0.000001
#167#	0.070622	-	0.070623	= -0.000001
#168#	0.071313	-	0.071315	= -0.000002
#169#	0.072094	-	0.072094	= 0.000000
#170#	0.072857	-	0.072855	= 0.000002
#171#	0.073792	-	0.073796	= -0.000004
#172#	0.074734	-	0.074733	= 0.000001
#173#	0.075762	-	0.075764	= -0.000002

#174#	0.076846	-	0.076841	= 0.000005
#175#	0.077979	-	0.077977	= 0.000002
#176#	0.079252	-	0.079258	= -0.000006
#177#	0.080586	-	0.080585	= 0.000001
#178#	0.081987	-	0.081986	= 0.000001
#179#	0.083473	-	0.083472	= 0.000001
#180#	0.085004	-	0.085008	= -0.000004
#181#	0.086723	-	0.086722	= 0.000001
#182#	0.088495	-	0.088491	= 0.000004
#183#	0.090359	-	0.090359	= 0.000000
#184#	0.092332	-	0.092330	= 0.000002
#185#	0.094342	-	0.094344	= -0.000002
#186#	0.096592	-	0.096593	= -0.000001
#187#	0.098880	-	0.098879	= 0.000001
#188#	0.101306	-	0.101308	= -0.000002
#189#	0.103830	-	0.103837	= -0.000007
#190#	0.106437	-	0.106432	= 0.000005
#191#	0.109291	-	0.109298	= -0.000007
#192#	0.112221	-	0.112226	= -0.000005
#193#	0.115300	-	0.115303	= -0.000003
#194#	0.118517	-	0.118511	= 0.000006
#195#	0.121797	-	0.121793	= 0.000004
#196#	0.125370	-	0.125371	= -0.000001
#197#	0.129062	-	0.129051	= 0.000011
#198#	0.132879	-	0.132880	= -0.000001
#199#	0.136886	-	0.136886	= 0.000000
#200#	0.140938	-	0.140933	= 0.000005
#201#	0.145396	-	0.145397	= -0.000001
#202#	0.149933	-	0.149931	= 0.000002
#203#	0.154659	-	0.154657	= 0.000002
#204#	0.159568	-	0.159563	= 0.000005
#205#	0.164560	-	0.164569	= -0.000009
#206#	0.169969	-	0.169961	= 0.000008
#207#	0.175480	-	0.175480	= 0.000000
#208#	0.181206	-	0.181213	= -0.000007
#209#	0.187026	-	0.187021	= 0.000005
#210#	0.193274	-	0.193277	= -0.000003
#211#	0.199682	-	0.199665	= 0.000017
#212#	0.206310	-	0.206314	= -0.000004
#213#	0.213135	-	0.213126	= 0.000009
#214#	0.220089	-	0.220085	= 0.000004
#215#	0.227512	-	0.227510	= 0.000002
#216#	0.235078	-	0.235082	= -0.000004
#217#	0.242868	-	0.242844	= 0.000024
#218#	0.250905	-	0.250917	= -0.000012
#219#	0.259022	-	0.259021	= 0.000001
#220#	0.267654	-	0.267659	= -0.000005
#221#	0.276409	-	0.276417	= -0.000008
#222#	0.285423	-	0.285423	= 0.000000
#223#	0.294658	-	0.294671	= -0.000013
#224#	0.303955	-	0.303949	= 0.000006
#225#	0.313783	-	0.313794	= -0.000011
#226#	0.323709	-	0.323706	= 0.000003
#227#	0.333840	-	0.333833	= 0.000007
#228#	0.344104	-	0.344132	= -0.000028
#229#	0.354541	-	0.354459	= 0.000082
#230#	0.365241	-	0.365293	= -0.000052
#231#	0.376126	-	0.376155	= -0.000029
#232#	0.387147	-	0.387166	= -0.000019
#233#	0.398272	-	0.398277	= -0.000005

```
#234#          0.409314 -          0.409284 =          0.000030
#235#          0.420789 -          0.420789 =          0.000000
#236#          0.432179 -          0.432196 =         -0.000017
#237#          0.443611 -          0.443596 =          0.000015
#238#          0.455079 -          0.455081 =         -0.000002
#239#          0.466200 -          0.466202 =         -0.000002
#240#          0.477829 -          0.477838 =         -0.000009
#241#          0.489083 -          0.489081 =          0.000002
Largest : 0.489083 @ line #241#
Smallest: 0.065813 @ line #151#
Error   : 0.000082 @ line #229#
Error Ratio: 0.02%
```

Trajectory #3 6x7

:::::::: Norms for 6x7 Traj :#3 ::::::::

----------differences: ----------

STEP	FSP		Pseudo		diff
# 1#	2.291624	-	2.291625	=	-0.000001
# 2#	2.408646	-	2.408645	=	0.000001
# 3#	2.432993	-	2.432992	=	0.000001
# 4#	2.426930	-	2.426929	=	0.000001
# 5#	2.458176	-	2.458197	=	-0.000021
# 6#	2.443710	-	2.443663	=	0.000047
# 7#	2.475068	-	2.475076	=	-0.000008
# 8#	2.459539	-	2.459572	=	-0.000033
# 9#	2.490581	-	2.490572	=	0.000009
# 10#	2.475086	-	2.475075	=	0.000011
# 11#	2.505185	-	2.505196	=	-0.000011
# 12#	2.490247	-	2.490230	=	0.000017
# 13#	2.518939	-	2.518956	=	-0.000017
# 14#	2.504544	-	2.504554	=	-0.000010
# 15#	2.532308	-	2.532251	=	0.000057
# 16#	2.518891	-	2.518898	=	-0.000007
# 17#	2.544936	-	2.544940	=	-0.000004
# 18#	2.532983	-	2.532964	=	0.000019
# 19#	2.557267	-	2.557274	=	-0.000007
# 20#	2.547248	-	2.547253	=	-0.000005
# 21#	2.569540	-	2.569533	=	0.000007
# 22#	2.561770	-	2.561748	=	0.000022
# 23#	2.569058	-	2.569096	=	-0.000038
# 24#	2.594236	-	2.594233	=	0.000003
# 25#	2.584641	-	2.584613	=	0.000028
# 26#	2.592535	-	2.592569	=	-0.000034
# 27#	2.600847	-	2.600736	=	0.000111
# 28#	2.622931	-	2.622991	=	-0.000060
# 29#	2.618586	-	2.618588	=	-0.000002
# 30#	2.627621	-	2.627611	=	0.000010
# 31#	2.637450	-	2.637441	=	0.000009
# 32#	2.647831	-	2.647885	=	-0.000054
# 33#	2.658650	-	2.658702	=	-0.000052
# 34#	2.670434	-	2.670421	=	0.000013
# 35#	2.682659	-	2.682651	=	0.000008
# 36#	2.681394	-	2.681333	=	0.000061
# 37#	2.709841	-	2.709866	=	-0.000025
# 38#	2.710603	-	2.710574	=	0.000029
# 39#	2.741284	-	2.741289	=	-0.000005
# 40#	2.739875	-	2.739843	=	0.000032
# 41#	2.777660	-	2.777697	=	-0.000037
# 42#	2.774772	-	2.774768	=	0.000004
# 43#	2.806205	-	2.806229	=	-0.000024
# 44#	2.829289	-	2.829261	=	0.000028
# 45#	2.854568	-	2.854588	=	-0.000020
# 46#	2.881619	-	2.881643	=	-0.000024
# 47#	2.911571	-	2.911528	=	0.000043
# 48#	2.944034	-	2.944031	=	0.000003
# 49#	2.979962	-	2.979968	=	-0.000006
# 50#	3.019297	-	3.019302	=	-0.000005
# 51#	3.062697	-	3.062716	=	-0.000019
# 52#	3.110402	-	3.110390	=	0.000012
# 53#	3.163171	-	3.163152	=	0.000019

# 54#	3.221042	-	3.221039	=	0.000003
# 55#	3.285645	-	3.285696	=	-0.000051
# 56#	3.357116	-	3.357143	=	-0.000027
# 57#	3.436488	-	3.436524	=	-0.000036
# 58#	3.524397	-	3.524364	=	0.000033
# 59#	3.622574	-	3.622572	=	0.000002
# 60#	3.731936	-	3.731930	=	0.000006
# 61#	3.854088	-	3.854035	=	0.000053
# 62#	3.992369	-	3.992399	=	-0.000030
# 63#	4.145334	-	4.145357	=	-0.000023
# 64#	4.319675	-	4.319641	=	0.000034
# 65#	4.514539	-	4.514562	=	-0.000023
# 66#	4.737188	-	4.737179	=	0.000009
# 67#	4.988215	-	4.988235	=	-0.000020
# 68#	5.274117	-	5.274121	=	-0.000004
# 69#	5.602203	-	5.602185	=	0.000018
# 70#	5.975360	-	5.975307	=	0.000053
# 71#	6.406528	-	6.406566	=	-0.000038
# 72#	6.900389	-	6.900363	=	0.000026
# 73#	7.472895	-	7.472813	=	0.000082
# 74#	8.131830	-	8.131946	=	-0.000116
# 75#	8.897085	-	8.897067	=	0.000018
# 76#	9.781750	-	9.781681	=	0.000069
# 77#	10.803719	-	10.803707	=	0.000012
# 78#	11.976869	-	11.976909	=	-0.000040
# 79#	13.316942	-	13.316828	=	0.000114
# 80#	14.824564	-	14.824715	=	-0.000151
# 81#	16.492691	-	16.492689	=	0.000002
# 82#	18.297403	-	18.297279	=	0.000124
# 83#	20.175972	-	20.176165	=	-0.000193
# 84#	22.043806	-	22.043598	=	0.000208
# 85#	23.783745	-	23.783285	=	0.000460
# 86#	25.265064	-	25.265007	=	0.000057
# 87#	26.355431	-	26.355942	=	-0.000511
# 88#	26.960964	-	26.961008	=	-0.000044
# 89#	27.020885	-	27.020624	=	0.000261
# 90#	26.566666	-	26.566925	=	-0.000259
# 91#	25.643639	-	25.643101	=	0.000538
# 92#	24.381241	-	24.380665	=	0.000576
# 93#	22.871433	-	22.872675	=	-0.001242
# 94#	21.255779	-	21.255302	=	0.000477
# 95#	19.603230	-	19.602938	=	0.000292
# 96#	18.000753	-	18.000641	=	0.000112
# 97#	16.483776	-	16.484272	=	-0.000496
# 98#	15.077948	-	15.077764	=	0.000184
# 99#	13.800875	-	13.800875	=	0.000000
#100#	12.644335	-	12.644363	=	-0.000028
#101#	11.614658	-	11.614706	=	-0.000048
#102#	10.691275	-	10.691195	=	0.000080
#103#	9.873972	-	9.874155	=	-0.000183
#104#	9.144074	-	9.143955	=	0.000119
#105#	8.499043	-	8.499056	=	-0.000013
#106#	7.921186	-	7.921305	=	-0.000119
#107#	7.409916	-	7.409798	=	0.000118
#108#	6.950144	-	6.950323	=	-0.000179
#109#	6.540135	-	6.539948	=	0.000187
#110#	6.173304	-	6.173414	=	-0.000110
#111#	5.841489	-	5.841393	=	0.000096
#112#	5.544358	-	5.544303	=	0.000055
#113#	5.273881	-	5.274012	=	-0.000131

#114#	5.030035	-	5.030019	= 0.000016
#115#	4.806790	-	4.806813	= -0.000023
#116#	4.605691	-	4.605606	= 0.000085
#117#	4.419624	-	4.419644	= -0.000020
#118#	4.251779	-	4.251801	= -0.000022
#119#	4.095506	-	4.095528	= -0.000022
#120#	3.953461	-	3.953540	= -0.000079
#121#	3.821476	-	3.821250	= 0.000226
#122#	3.699841	-	3.699881	= -0.000040
#123#	3.588265	-	3.588261	= 0.000004
#124#	3.483358	-	3.483414	= -0.000056
#125#	3.387301	-	3.387300	= 0.000001
#126#	3.296914	-	3.296830	= 0.000084
#127#	3.213376	-	3.213396	= -0.000020
#128#	3.134793	-	3.134840	= -0.000047
#129#	3.062460	-	3.062437	= 0.000023
#130#	2.993166	-	2.993163	= 0.000003
#131#	2.929263	-	2.929302	= -0.000039
#132#	2.868860	-	2.868703	= 0.000157
#133#	2.812593	-	2.812607	= -0.000014
#134#	2.758812	-	2.758850	= -0.000038
#135#	2.709209	-	2.709152	= 0.000057
#136#	2.661329	-	2.661375	= -0.000046
#137#	2.616737	-	2.616831	= -0.000094
#138#	2.574832	-	2.574805	= 0.000027
#139#	2.534770	-	2.534849	= -0.000079
#140#	2.497109	-	2.497117	= -0.000008
#141#	2.461054	-	2.461061	= -0.000007
#142#	2.427463	-	2.427466	= -0.000003
#143#	2.394880	-	2.394880	= 0.000000
#144#	2.364668	-	2.364714	= -0.000046
#145#	2.335230	-	2.335220	= 0.000010
#146#	2.308069	-	2.308071	= -0.000002
#147#	2.281274	-	2.281304	= -0.000030
#148#	2.256664	-	2.256551	= 0.000113
#149#	2.232646	-	2.232637	= 0.000009
#150#	2.209706	-	2.209751	= -0.000045
#151#	2.188285	-	2.188378	= -0.000093
#152#	2.167405	-	2.167382	= 0.000023
#153#	2.147999	-	2.148048	= -0.000049
#154#	2.128961	-	2.128945	= 0.000016
#155#	2.111172	-	2.111150	= 0.000022
#156#	2.093921	-	2.093919	= 0.000002
#157#	2.077674	-	2.077652	= 0.000022
#158#	2.061839	-	2.061902	= -0.000063
#159#	2.047099	-	2.047094	= 0.000005
#160#	2.032530	-	2.032521	= 0.000009
#161#	2.019048	-	2.019053	= -0.000005
#162#	2.005810	-	2.005793	= 0.000017
#163#	1.993450	-	1.993460	= -0.000010
#164#	1.981632	-	1.981617	= 0.000015
#165#	1.969869	-	1.969847	= 0.000022
#166#	1.959134	-	1.959113	= 0.000021
#167#	1.948490	-	1.948447	= 0.000043
#168#	1.938603	-	1.938612	= -0.000009
#169#	1.928935	-	1.928945	= -0.000010
#170#	1.919808	-	1.919757	= 0.000051
#171#	1.910922	-	1.910928	= -0.000006
#172#	1.902617	-	1.902609	= 0.000008
#173#	1.894440	-	1.894432	= 0.000008

#174#	1.886911	-	1.886891	= 0.000020
#175#	1.879252	-	1.879333	= -0.000081
#176#	1.872558	-	1.872532	= 0.000026
#177#	1.865677	-	1.865658	= 0.000019
#178#	1.859225	-	1.859237	= -0.000012
#179#	1.853177	-	1.853195	= -0.000018
#180#	1.847248	-	1.847228	= 0.000020
#181#	1.841721	-	1.841705	= 0.000016
#182#	1.836234	-	1.836238	= -0.000004
#183#	1.831223	-	1.831195	= 0.000028
#184#	1.826191	-	1.826169	= 0.000022
#185#	1.821479	-	1.821530	= -0.000051
#186#	1.817025	-	1.817019	= 0.000006
#187#	1.812866	-	1.812874	= -0.000008
#188#	1.808691	-	1.808689	= 0.000002
#189#	1.804935	-	1.804935	= 0.000000
#190#	1.801101	-	1.801076	= 0.000025
#191#	1.797613	-	1.797613	= 0.000000
#192#	1.794292	-	1.794297	= -0.000005
#193#	1.790975	-	1.790944	= 0.000031
#194#	1.787979	-	1.788015	= -0.000036
#195#	1.784915	-	1.784899	= 0.000016
#196#	1.782333	-	1.782318	= 0.000015
#197#	1.779560	-	1.779569	= -0.000009
#198#	1.777132	-	1.777121	= 0.000011
#199#	1.774587	-	1.774551	= 0.000036
#200#	1.772323	-	1.772391	= -0.000068
#201#	1.770109	-	1.770105	= 0.000004
#202#	1.768086	-	1.768084	= 0.000002
#203#	1.766045	-	1.766042	= 0.000003
#204#	1.764127	-	1.764141	= -0.000014
#205#	1.762267	-	1.762253	= 0.000014
#206#	1.760549	-	1.760552	= -0.000003
#207#	1.758929	-	1.758916	= 0.000013
#208#	1.757303	-	1.757333	= -0.000030
#209#	1.755680	-	1.755717	= -0.000037
#210#	1.754396	-	1.754390	= 0.000006
#211#	1.752965	-	1.752939	= 0.000026
#212#	1.751534	-	1.751592	= -0.000058
#213#	1.750321	-	1.750337	= -0.000016
#214#	1.748821	-	1.748802	= 0.000019
#215#	1.747813	-	1.747844	= -0.000031
#216#	1.746558	-	1.746521	= 0.000037
#217#	1.745463	-	1.745464	= -0.000001
#218#	1.744232	-	1.744202	= 0.000030
#219#	1.742994	-	1.742984	= 0.000010
#220#	1.742057	-	1.742065	= -0.000008
#221#	1.741004	-	1.740999	= 0.000005
#222#	1.739935	-	1.739911	= 0.000024
#223#	1.738705	-	1.738708	= -0.000003
#224#	1.737643	-	1.737636	= 0.000007
#225#	1.736573	-	1.736577	= -0.000004
#226#	1.735590	-	1.735582	= 0.000008
#227#	1.734359	-	1.734368	= -0.000009
#228#	1.733276	-	1.733292	= -0.000016
#229#	1.731945	-	1.731918	= 0.000027
#230#	1.730978	-	1.730979	= -0.000001
#231#	1.729711	-	1.729704	= 0.000007
#232#	1.728336	-	1.728365	= -0.000029
#233#	1.727171	-	1.727137	= 0.000034

```
#234#          1.725473 -          1.725500 =         -0.000027
#235#          1.724382 -          1.724342 =          0.000040
#236#          1.722760 -          1.722769 =         -0.000009
#237#          1.721257 -          1.721292 =         -0.000035
#238#          1.719649 -          1.719635 =          0.000014
#239#          1.717784 -          1.717805 =         -0.000021
#240#          1.716188 -          1.716151 =          0.000037
#241#          1.714383 -          1.714417 =         -0.000034
Largest : 27.020885 @ line # 89#
Smallest: 1.714383 @ line #241#
Error   : 0.001242 @ line # 93#
Error Ratio: 0.01%
```

Trajectory #3 6x10

:::::::: Norms for 6x10 Traj :#3 ::::::::

----------differences: ----------

STEP	FSP		Pseudo		diff
# 1#	1.441534	-	1.441534	=	0.000000
# 2#	1.451998	-	1.451993	=	0.000005
# 3#	1.454581	-	1.454586	=	-0.000005
# 4#	1.457944	-	1.457947	=	-0.000003
# 5#	1.461165	-	1.461154	=	0.000011
# 6#	1.464417	-	1.464420	=	-0.000003
# 7#	1.467579	-	1.467582	=	-0.000003
# 8#	1.470697	-	1.470697	=	0.000000
# 9#	1.473962	-	1.473947	=	0.000015
# 10#	1.477141	-	1.477148	=	-0.000007
# 11#	1.480242	-	1.480240	=	0.000002
# 12#	1.483488	-	1.483494	=	-0.000006
# 13#	1.486409	-	1.486409	=	0.000000
# 14#	1.489666	-	1.489661	=	0.000005
# 15#	1.492777	-	1.492768	=	0.000009
# 16#	1.495882	-	1.495902	=	-0.000020
# 17#	1.499009	-	1.499001	=	0.000008
# 18#	1.501921	-	1.501913	=	0.000008
# 19#	1.505161	-	1.505172	=	-0.000011
# 20#	1.508178	-	1.508174	=	0.000004
# 21#	1.511289	-	1.511300	=	-0.000011
# 22#	1.514275	-	1.514281	=	-0.000006
# 23#	1.517329	-	1.517316	=	0.000013
# 24#	1.520397	-	1.520412	=	-0.000015
# 25#	1.523460	-	1.523455	=	0.000005
# 26#	1.526491	-	1.526501	=	-0.000010
# 27#	1.529475	-	1.529462	=	0.000013
# 28#	1.532456	-	1.532475	=	-0.000019
# 29#	1.535569	-	1.535557	=	0.000012
# 30#	1.538575	-	1.538571	=	0.000004
# 31#	1.541533	-	1.541535	=	-0.000002
# 32#	1.544549	-	1.544549	=	0.000000
# 33#	1.547433	-	1.547443	=	-0.000010
# 34#	1.550535	-	1.550533	=	0.000002
# 35#	1.553473	-	1.553471	=	0.000002
# 36#	1.556501	-	1.556497	=	0.000004
# 37#	1.561652	-	1.561641	=	0.000011
# 38#	1.559939	-	1.559955	=	-0.000016
# 39#	1.566869	-	1.566871	=	-0.000002
# 40#	1.566799	-	1.566791	=	0.000008
# 41#	1.572090	-	1.572098	=	-0.000008
# 42#	1.574738	-	1.574730	=	0.000008
# 43#	1.577242	-	1.577242	=	0.000000
# 44#	1.579909	-	1.579904	=	0.000005
# 45#	1.582505	-	1.582505	=	0.000000
# 46#	1.585125	-	1.585125	=	0.000000
# 47#	1.587602	-	1.587601	=	0.000001
# 48#	1.590207	-	1.590202	=	0.000005
# 49#	1.596902	-	1.596905	=	-0.000003
# 50#	1.595351	-	1.595355	=	-0.000004
# 51#	1.602183	-	1.602187	=	-0.000004
# 52#	1.602812	-	1.602798	=	0.000014
# 53#	1.605920	-	1.605922	=	-0.000002

# 54#	1.608695	-	1.608687	= 0.000008
# 55#	1.611499	-	1.611519	= -0.000020
# 56#	1.614352	-	1.614351	= 0.000001
# 57#	1.616992	-	1.616996	= -0.000004
# 58#	1.619873	-	1.619885	= -0.000012
# 59#	1.622607	-	1.622595	= 0.000012
# 60#	1.625399	-	1.625403	= -0.000004
# 61#	1.628068	-	1.628086	= -0.000018
# 62#	1.630804	-	1.630790	= 0.000014
# 63#	1.633473	-	1.633478	= -0.000005
# 64#	1.636197	-	1.636196	= 0.000001
# 65#	1.638798	-	1.638792	= 0.000006
# 66#	1.641489	-	1.641483	= 0.000006
# 67#	1.643959	-	1.643965	= -0.000006
# 68#	1.646640	-	1.646654	= -0.000014
# 69#	1.649269	-	1.649256	= 0.000013
# 70#	1.651774	-	1.651786	= -0.000012
# 71#	1.654354	-	1.654339	= 0.000015
# 72#	1.656713	-	1.656719	= -0.000006
# 73#	1.659323	-	1.659325	= -0.000002
# 74#	1.661760	-	1.661752	= 0.000008
# 75#	1.664382	-	1.664367	= 0.000015
# 76#	1.666602	-	1.666603	= -0.000001
# 77#	1.668929	-	1.668941	= -0.000012
# 78#	1.671333	-	1.671315	= 0.000018
# 79#	1.673765	-	1.673779	= -0.000014
# 80#	1.675975	-	1.675969	= 0.000006
# 81#	1.678241	-	1.678242	= -0.000001
# 82#	1.680467	-	1.680442	= 0.000025
# 83#	1.682701	-	1.682699	= 0.000002
# 84#	1.684941	-	1.684962	= -0.000021
# 85#	1.687069	-	1.687041	= 0.000028
# 86#	1.689241	-	1.689243	= -0.000002
# 87#	1.691182	-	1.691186	= -0.000004
# 88#	1.693395	-	1.693390	= 0.000005
# 89#	1.695391	-	1.695410	= -0.000019
# 90#	1.697476	-	1.697476	= 0.000000
# 91#	1.699396	-	1.699389	= 0.000007
# 92#	1.701312	-	1.701292	= 0.000020
# 93#	1.703256	-	1.703278	= -0.000022
# 94#	1.705231	-	1.705223	= 0.000008
# 95#	1.707015	-	1.707019	= -0.000004
# 96#	1.708948	-	1.708946	= 0.000002
# 97#	1.710610	-	1.710613	= -0.000003
# 98#	1.712451	-	1.712456	= -0.000005
# 99#	1.714278	-	1.714287	= -0.000009
#100#	1.715942	-	1.715934	= 0.000008
#101#	1.717607	-	1.717592	= 0.000015
#102#	1.719322	-	1.719333	= -0.000011
#103#	1.721046	-	1.721040	= 0.000006
#104#	1.722633	-	1.722622	= 0.000011
#105#	1.724283	-	1.724275	= 0.000008
#106#	1.725743	-	1.725714	= 0.000029
#107#	1.727426	-	1.727460	= -0.000034
#108#	1.728954	-	1.728949	= 0.000005
#109#	1.730506	-	1.730482	= 0.000024
#110#	1.732059	-	1.732066	= -0.000007
#111#	1.733414	-	1.733441	= -0.000027
#112#	1.735074	-	1.735057	= 0.000017
#113#	1.736594	-	1.736583	= 0.000011

#114#	1.738053	-	1.738044	= 0.000009
#115#	1.739462	-	1.739487	= -0.000025
#116#	1.740853	-	1.740882	= -0.000029
#117#	1.742429	-	1.742406	= 0.000023
#118#	1.742238	-	1.742265	= -0.000027
#119#	1.747218	-	1.747231	= -0.000013
#120#	1.745886	-	1.745901	= -0.000015
#121#	1.747768	-	1.747730	= 0.000038
#122#	1.749723	-	1.749735	= -0.000012
#123#	1.751622	-	1.751616	= 0.000006
#124#	1.753513	-	1.753493	= 0.000020
#125#	1.755426	-	1.755423	= 0.000003
#126#	1.755346	-	1.755360	= -0.000014
#127#	1.759418	-	1.759446	= -0.000028
#128#	1.757296	-	1.757256	= 0.000040
#129#	1.761122	-	1.761122	= 0.000000
#130#	1.762519	-	1.762547	= -0.000028
#131#	1.764157	-	1.764175	= -0.000018
#132#	1.766034	-	1.766031	= 0.000003
#133#	1.767855	-	1.767884	= -0.000029
#134#	1.769675	-	1.769650	= 0.000025
#135#	1.771617	-	1.771609	= 0.000008
#136#	1.773362	-	1.773355	= 0.000007
#137#	1.775542	-	1.775529	= 0.000013
#138#	1.777659	-	1.777632	= 0.000027
#139#	1.779735	-	1.779738	= -0.000003
#140#	1.781979	-	1.781972	= 0.000007
#141#	1.784066	-	1.784086	= -0.000020
#142#	1.786643	-	1.786633	= 0.000010
#143#	1.788995	-	1.788995	= 0.000000
#144#	1.791562	-	1.791582	= -0.000020
#145#	1.794124	-	1.794131	= -0.000007
#146#	1.796747	-	1.796715	= 0.000032
#147#	1.799604	-	1.799579	= 0.000025
#148#	1.802525	-	1.802531	= -0.000006
#149#	1.805442	-	1.805433	= 0.000009
#150#	1.808531	-	1.808526	= 0.000005
#151#	1.811570	-	1.811573	= -0.000003
#152#	1.814982	-	1.815003	= -0.000021
#153#	1.818483	-	1.818451	= 0.000032
#154#	1.821909	-	1.821909	= 0.000000
#155#	1.825428	-	1.825441	= -0.000013
#156#	1.829368	-	1.829334	= 0.000034
#157#	1.833221	-	1.833218	= 0.000003
#158#	1.837216	-	1.837206	= 0.000010
#159#	1.841322	-	1.841310	= 0.000012
#160#	1.845343	-	1.845348	= -0.000005
#161#	1.849955	-	1.849910	= 0.000045
#162#	1.854460	-	1.854478	= -0.000018
#163#	1.859156	-	1.859112	= 0.000044
#164#	1.863901	-	1.863923	= -0.000022
#165#	1.868427	-	1.868442	= -0.000015
#166#	1.873587	-	1.873590	= -0.000003
#167#	1.878743	-	1.878750	= -0.000007
#168#	1.883880	-	1.883891	= -0.000011
#169#	1.889216	-	1.889197	= 0.000019
#170#	1.894433	-	1.894407	= 0.000026
#171#	1.900042	-	1.900075	= -0.000033
#172#	1.905732	-	1.905747	= -0.000015
#173#	1.911313	-	1.911328	= -0.000015

#174#	1.917087	-	1.917107	= -0.000020
#175#	1.922574	-	1.922570	= 0.000004
#176#	1.928661	-	1.928684	= -0.000023
#177#	1.934542	-	1.934545	= -0.000003
#178#	1.940348	-	1.940345	= 0.000003
#179#	1.946320	-	1.946319	= 0.000001
#180#	1.951838	-	1.951821	= 0.000017
#181#	1.957963	-	1.957965	= -0.000002
#182#	1.963660	-	1.963638	= 0.000022
#183#	1.969398	-	1.969405	= -0.000007
#184#	1.975006	-	1.975003	= 0.000003
#185#	1.980145	-	1.980162	= -0.000017
#186#	1.985787	-	1.985839	= -0.000052
#187#	1.990993	-	1.990945	= 0.000048
#188#	1.996001	-	1.996016	= -0.000015
#189#	2.000767	-	2.000773	= -0.000006
#190#	2.005057	-	2.005027	= 0.000030
#191#	2.009641	-	2.009653	= -0.000012
#192#	2.013746	-	2.013743	= 0.000003
#193#	2.017552	-	2.017562	= -0.000010
#194#	2.020953	-	2.020947	= 0.000006
#195#	2.023771	-	2.023783	= -0.000012
#196#	2.026855	-	2.026830	= 0.000025
#197#	2.029262	-	2.029242	= 0.000020
#198#	2.031235	-	2.031224	= 0.000011
#199#	2.032908	-	2.032903	= 0.000005
#200#	2.033801	-	2.033805	= -0.000004
#201#	2.034849	-	2.034859	= -0.000010
#202#	2.035373	-	2.035357	= 0.000016
#203#	2.035269	-	2.035310	= -0.000041
#204#	2.034869	-	2.034866	= 0.000003
#205#	2.033651	-	2.033640	= 0.000011
#206#	2.032700	-	2.032697	= 0.000003
#207#	2.031036	-	2.031068	= -0.000032
#208#	2.028843	-	2.028843	= 0.000000
#209#	2.026144	-	2.026125	= 0.000019
#210#	2.023439	-	2.023471	= -0.000032
#211#	2.020277	-	2.020234	= 0.000043
#212#	2.016571	-	2.016532	= 0.000039
#213#	2.012762	-	2.012715	= 0.000047
#214#	2.007993	-	2.008054	= -0.000061
#215#	2.003675	-	2.003642	= 0.000033
#216#	1.998760	-	1.998763	= -0.000003
#217#	1.993490	-	1.993480	= 0.000010
#218#	1.988066	-	1.988039	= 0.000027
#219#	1.981932	-	1.981928	= 0.000004
#220#	1.976339	-	1.976300	= 0.000039
#221#	1.970074	-	1.970083	= -0.000009
#222#	1.963635	-	1.963660	= -0.000025
#223#	1.957149	-	1.957105	= 0.000044
#224#	1.950048	-	1.950043	= 0.000005
#225#	1.943461	-	1.943435	= 0.000026
#226#	1.936346	-	1.936385	= -0.000039
#227#	1.929277	-	1.929284	= -0.000007
#228#	1.922020	-	1.922013	= 0.000007
#229#	1.914261	-	1.914291	= -0.000030
#230#	1.907317	-	1.907235	= 0.000082
#231#	1.899526	-	1.899566	= -0.000040
#232#	1.892145	-	1.892082	= 0.000063
#233#	1.884350	-	1.884418	= -0.000068

```
#234#          1.876539 -          1.876537 =          0.000002
#235#          1.869050 -          1.869040 =          0.000010
#236#          1.861408 -          1.861358 =          0.000050
#237#          1.853622 -          1.853618 =          0.000004
#238#          1.845893 -          1.845882 =          0.000011
#239#          1.837926 -          1.837898 =          0.000028
#240#          1.830362 -          1.830342 =          0.000020
#241#          1.822674 -          1.822713 =         -0.000039
Largest : 2.035373 @ line #202#
Smallest: 1.441534 @ line #  1#
Error   : 0.000082 @ line #230#
Error Ratio: 0.00%
```

Appendix A2: FSP vs. MP Pseudo Inverse Timing Data

The output is recorded from the UNIX shell function time. Each row contains real time (wall time), user time (time of the process), and system time (time the system used). Each time is in seconds. Each Jacobian has two rows of times associated with them. The first row is FSP, while the second row is MP Pseudo Inverse. The data in Figure Three was constructed by dividing the user time of the FSP by the user time of the Pseudo Inverse. One obvious question to be answered is why the FSP took longer than the 1.7 average for the 6x10 cases.

The data can be constructed as follows by using IKOR's command line interface[10]. Be sure to comment the macro DEBUG in general.h. This is necessary because FSP prints out a lot more diagnostic information than is only needed during debugging.

ex.

```
% time IKOR N M <fastest FSP is type 4>  <x
% time IKOR 3 7 4 <x
```

Output is printed in terminal window. The above command would run FSP on a 3x7 with obstacle and joint limit avoidance bypassed (type 4). Input from a file called x is also used, this file contains answers to questions that are normally input interactively.

The data gathered for the two timing experiments presented in section 3.16 and 3.23 comprises the remainder of Appendix A2. First the timing results gathered from FSP vs. Pseudo Inverse without obstacle and joint limit avoidance are presented. Lastly, the timing results gathered from FSP one and two step methods vs. Pseudo Inverse with obstacle and joint limit avoidance are presented.

[10] A complete description of the IKOR driver is found by entering the program name with a -help parameter.

FSP vs. Pseudo Inverse: No obstacle or Joint Limit Avoidance

```
:::::::::::::: Traj :#1a ::::::::::::::::::
3x10     2.4 real          2.4 user          0.1 sys
         1.8 real          1.4 user          0.0 sys
3x7      1.7 real          1.5 user          0.1 sys
         1.1 real          0.9 user          0.0 sys
6x7      2.1 real          2.0 user          0.0 sys
         1.3 real          1.1 user          0.0 sys
6x10     4.5 real          4.0 user          0.0 sys
         1.8 real          1.6 user          0.1 sys
:::::::::::::: Traj :#1b ::::::::::::::::::
3x10    20.7 real         20.4 user          0.1 sys
        12.2 real         11.9 user          0.1 sys
3x7     14.2 real         13.1 user          0.1 sys
         8.2 real          8.0 user          0.1 sys
6x7     18.1 real         18.4 user          0.0 sys
        10.2 real          9.9 user          0.1 sys
6x10    42.7 real         42.2 user          0.1 sys
        15.8 real         15.5 user          0.1 sys


:::::::::::::: Traj :#2a ::::::::::::::::::
3x10     1.7 real          1.6 user          0.0 sys
         1.1 real          0.9 user          0.0 sys
3x7      1.2 real          1.4 user          0.0 sys
         0.9 real          0.8 user          0.0 sys
6x7      1.5 real          1.4 user          0.0 sys
         1.0 real          0.8 user          0.0 sys
6x10     4.1 real          3.5 user          0.0 sys
         1.4 real          1.1 user          0.1 sys
:::::::::::::: Traj :#2b ::::::::::::::::::
3x10    14.8 real         14.6 user          0.1 sys
         8.8 real          8.5 user          0.1 sys
3x7      8.7 real          9.7 user          0.0 sys
         5.9 real          5.7 user          0.1 sys
6x7     12.8 real         12.5 user          0.0 sys
         7.3 real          7.0 user          0.1 sys
6x10    30.9 real         38.8 user          0.1 sys
        11.1 real         10.8 user          0.1 sys


:::::::::::::: Traj :#3a ::::::::::::::::::
3x10     1.4 real          1.2 user          0.0 sys
         0.9 real          0.7 user          0.0 sys
3x7      1.0 real          0.7 user          0.0 sys
         0.7 real          0.5 user          0.0 sys
6x7      1.3 real          1.0 user          0.0 sys
         0.9 real          0.6 user          0.0 sys
6x10     2.5 real          2.3 user          0.0 sys
         1.1 real          0.9 user          0.0 sys
:::::::::::::: Traj :#3b ::::::::::::::::::
3x10     5.4 real          5.5 user          0.0 sys
         3.3 real          3.1 user          0.0 sys
3x7      3.8 real          3.4 user          0.0 sys
         2.3 real          2.0 user          0.1 sys
6x7      4.7 real          4.9 user          0.1 sys
         2.8 real          2.6 user          0.0 sys
6x10    11.3 real         10.9 user          0.1 sys
```

```
4.1 real         3.9 user         0.0 sys
```

FSP vs. Pseudo Inverse: Obstacle and Joint Limit Avoidance

```
:::::::::::::: Traj :#1 ::::::::::::::::
Pseudo  9.9 real        10.0 user         0.0 sys
FSP-1s 19.6 real        19.7 user         0.1 sys
FSP-2s 19.8 real        19.9 user         0.0 sys


:::::::::::::: Traj :#3 ::::::::::::::::
Pseudo  9.9 real        10.0 user         0.0 sys
FSP-1s 19.6 real        19.7 user         0.0 sys
FSP-2s 35.9 real        36.0 user         0.1 sys


:::::::::::::: Traj :#3 ::::::::::::::::
Pseudo  9.9 real        10.0 user         0.0 sys
FSP-1s 19.6 real        19.7 user         0.0 sys
FSP-2s 30.8 real        30.9 user         0.1 sys
```

Appendix B : User's Guide

This is the user's guide. Its purpose is to instruct and enable the user on how to specify trajectories and otherwise execute IKOR/FSP. IKOR is currently implemented on a SPARC station platform as well as Silicon Graphics.

The SPARC Station platform contains the full and latest version of IKOR/FSP as well as a graphical driver, and the manipulator simulator, CheezyIGRIP. Appendix B1 should show the USER how to create trajectories, find solutions for them, and display them in a simulator.

The SG platform contains an LLTI interface to TELEGRIP® as well as a joystick interface. The version of IKOR on this system is limited to the AIRARM manipulator and platform control is not enabled. A brief discussion on the execution of the system as well as information on how to port to the SG platform sums up Appendix B2.

Appendix B3 is a consortium of tips, tricks, and guides that may help keep the hair on the modifier's head. A brief discussion on how the data was gathered for this paper as well as understanding and using the documentation standard, kicks off this portion of the Appendix. Since memory leaks were found that resulted from the matrix code, a brief section describing its correct usage is given. Perhaps the most useful portion of the entire paper for a new system's analyst is given next. Compiling Files and Procedures takes the analyst through the compilation process, changing manipulators, and understanding the MACROS. Finally a Versions List is given which attempts to show the progression of the system so that an administrator can quickly locate the beginnings of a design.

Appendix B1: SPARC station

The SPARC station system has two complementary software packages for the IKOR/FSP system. One is a driver that allows users to access all the parameters of IKOR, while the other is a three dimensional graphical simulator.

Directory Structure v2.0

General knowledge of the filesystem is required for use outside of the GUI. The following displays the structure of the system as well as a brief explanation of each part. At the top level, all but one of the directories are for running the system. The other directory, COMPILE, contains everything that is needed to compile the system.

.alias	: Contains some helpful aliases and paths for the system
AIRARM	: Next Generation Munitions Handler (NGHM)
COMPILE	: all compilation is here. current compiler (gcc)
driver	: xtpanel scripts for SPARCstation GUI
HERMIES	: Hostile Environment Remote Manipulator ...
ONETIME	: Solution Generator tester
ikor	: SPARCstation GUI executable
README	: A brief description of the file system

TWOD: <not yet implemented>

General Information for **AIRARM** and **HERMIES** directories:

Each directory has executables. One for the simulator CheezyIGRIP, and one for the Inverse Kinematics on Redundant Systems (IKOR) program. Additionally there is a run-time file called ROBOT.dat. If these data files get corrupt they can also be found in COMPILE/MANIPULATORS/<ROBOT>/. Joint limits can be changed in this file as well as the general kinematics. Each of theses directories have the following fundamental files:

CheezyAngles ->	From IKOR, the joint values for each time step enabling the completion of the requested motion.
CheezyIGRIP ->	The simulator. Takes command line parameters similar to IKOR's.
EulerAngles ->	This file is provided by CheezyIGRIP to describe a motion of the manipulator-

. IKOR uses this file in conjunction with TrajectoryPts to describe a motion.

IKOR -> The Inverse Kinematics program. Type IKOR -help to get a listing of the command-line options.

ROBOT.dat -> Manipulator Descriptor file read during run-time. Allows kinematic alteration as well as joint limit alterations.

TrajectoryPts -> This file is provided by CheezyIGRIP to describe a motion of the manipulator. IKOR uses this file in conjunction with EulerAngles to describe a motion.

data -> If IKOR or CheezyIGRIP is giving a 'd' option, this file will be generated.

sphere_data -> contains data on current obstacles in the system. Can be altered in a text editor or during run time with a 'c' option.

Graphical System Driver

The driver is located in the directory: /rpsd/uf/project/pine/IKOR/v2.0/driver. The file is called IKOR_driver.xtp. It must be executed with xtpanel located in: /rpsd/uf/project/pine/bin/X11. Once this is in your path, the following brings the system up:

```
% xtpanel < IKOR_driver.xtp > IKOR_data
```

Upon successful execution, the user will see the diagram in Figure B1a. The top row consists of four buttons: QUIT, HELP, ABOUT, and GO. Quit exits the driver, while HELP does its best to explain what the USER's choices are as well as assist in explaining the entire system. ABOUT pops up a window that contains the names and email addresses of many who have worked on the system, from when it was an infant until now.

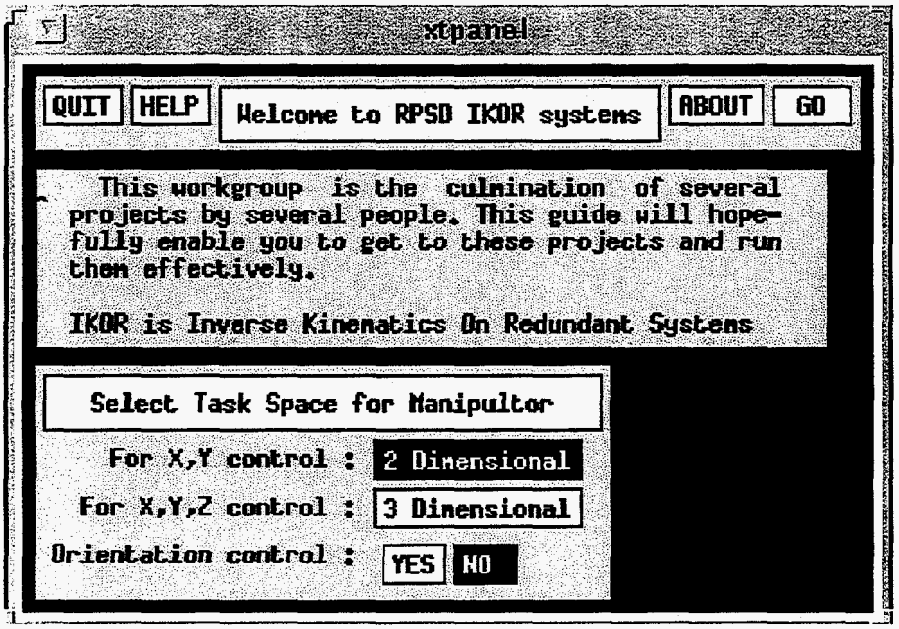


Figure B1a: First Screen of SPARC driver

The GO button executes IKOR based on the USER's selections. If there is a current trajectory, IKOR will apply the selected constraints to that trajectory and then run the simulator. By choosing < load file > inside of the simulator window and then typing an x in the load window, the solution to the trajectory will be loaded. Then by pressing < simulate >, the user can view the chosen manipulator executing the motion. CheezyIGRIP is discussed following the graphical driver's description. Feel free to skip ahead if you are tinkering inside of the simulator.

The first major choice the USER possesses is whether to run the two or three dimensional system and whether or not orientation control is desired. If the two dimensional system is chosen along with the desired orientation control, pressing GO will run these choices. However, if the three dimensional system is chosen, more options are available. These are shown in Figure B1b below.

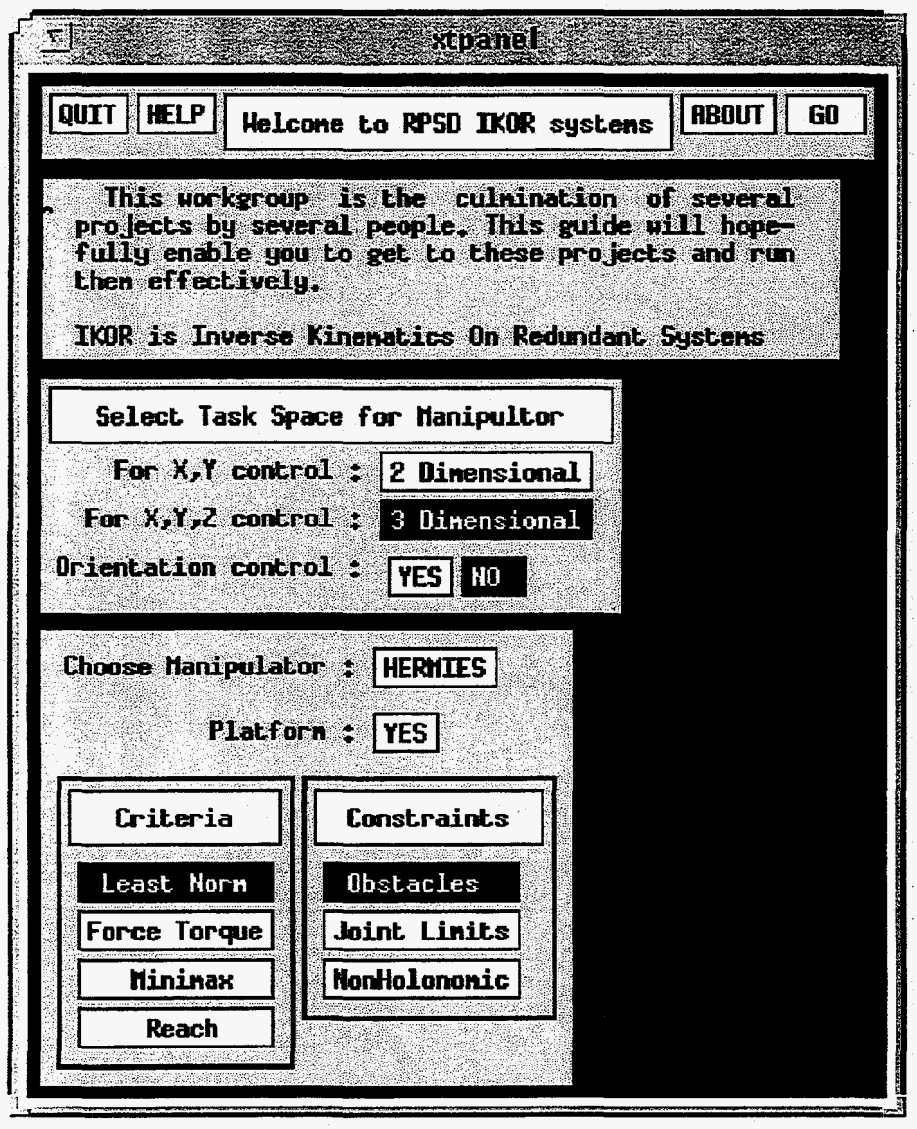


Figure B1b: Three Dimensional System Choices

The USER's first option is the manipulator. Currently two manipulators are known to the system; see Appendix B3 to learn how to integrate more. Simply click the name HERMIES to change manipulators; see Figure B1c. Upon clicking on the name, a small menu appears. Use the mouse to select the desired manipulator. In a similar manner, the user can choose either to integrate a platform or not.

Now its time to choose a criterion, however only the Least Norm is currently implemented. Finally, the USER can choose the type of constraints that will be placed on the system. Although Obstacle Avoidance is the default, it is also possible to include Joint Limit Avoidance, only one of these, or none. By de-selecting Obstacle Avoidance in the Figure below, there will be no constraints are selected and therefore only the

criteria will be adhered to. Once all options are selected, select the GO button. The current trajectory will be constrained and the solution will be calculated for the current criteria. The USER may then view the motion within the simulator.

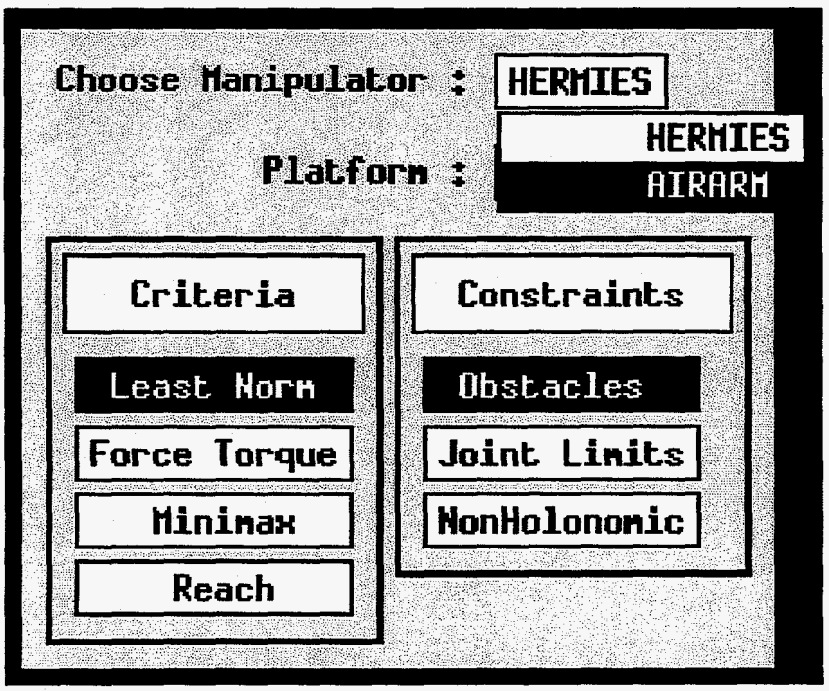


Figure B1c: Choosing A Manipulator

By choosing GO again, another instance will begin with the same configurations. By altering the options and choosing GO, another trajectory will be calculated using the new options.

CheezyIGRIP: SPARCstation Simulator

CheezyIGRIP is the simulator for the SPARC station IKOR/FSP platform. It has been co-developed with IKOR, but contains many incompatibilities. First an overview, and then a discussion on how it can become more integrated is presented. This simulator was first developed by Derek Carlson to simulate the seven degrees of freedom HERMIES manipulator. Kristi Morgansen was able to add a platform since FSPv1.0 could handle more degrees of redundancy. Allow for different manipulators, patch up memory leaks and remove orientation control, and the current version of the new simulator appears. Both changes by Kristi and I added more power to the simulator and at the same time reduced backwards compatibility.

Once the simulator has started, the screen, as shown in Figure B1d, will be placed on the screen. The middle of the screen that contains a picture of the chosen manipulator along with a coordinate axis and obstacles, will be referred to as the WORLD. There are four information control windows in each corner of the simulator.

The window in the top left hand side of the simulator has many options, some that have grown out of use with time and others that are used readily. Using the tracers are an effective way of showing the motion of the platform in papers and slides. The program can be exited in four ways: one from this window, another by pressing the space bar while inside of the WORLD, yet another by pressing the mouse inside the top bar of the program, and finally by using <CTRL-C> inside of the c-shell that spawned the program. Using the top bar creates a display interrupt error, and pressing the spacebar accidentally can create a loss of speech etiquette.

The window in the top right hand side of the simulator shows details such as current position and orientation of the end effector. This window allows the user to reset the world orientation, useful when the axes have twisted into an unrecognizable position due to a currently unlocated bug. The world's coordinate axis can be changed by pressing the right mouse button and then moving the mouse inside of the WORLD.

The window in the bottom right hand side of the simulator allows the user to change the joint values for the manipulator. There are three ways of choosing the motion. One way is selecting the joint with the left mouse button, then by pressing the left button within the WORLD while moving the mouse up and down. That joint's value will increase or decrease accordingly. Another way is to select the joint with the mouse and then hit any key while inside the joint window. This will prompt the USER for a joint value. Finally by selecting the joint and then pressing the right button while still inside the joint window, that joint will move in minute positive increments (.025 feet or degrees) for every click of the right button.

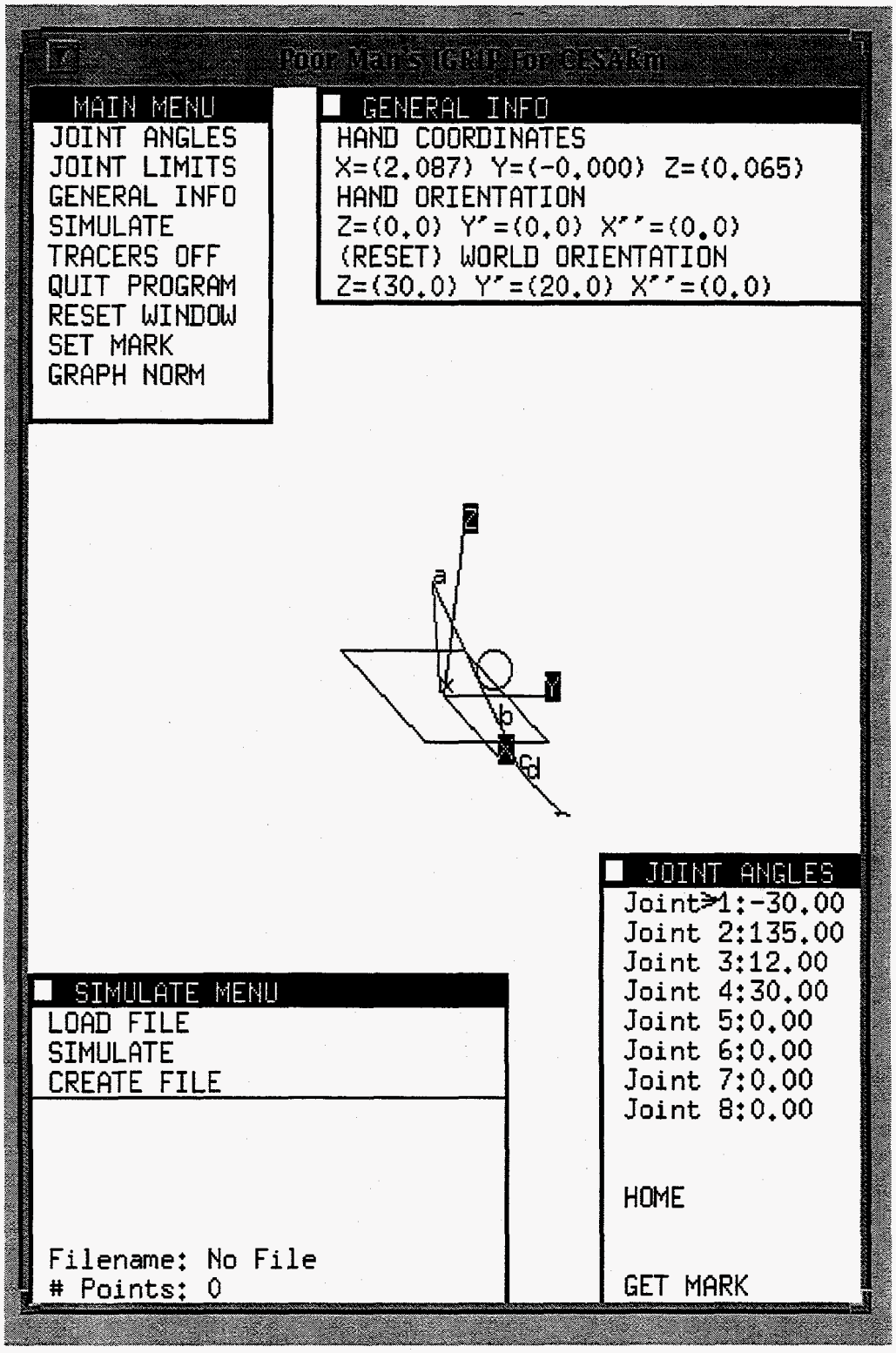


Figure B1d: CheezyIGRIP Simulator

Finally the bottom left menu, the simulate menu, demonstrates the real power of the system. Here we can < Load > or < Simulate > solutions to a trajectory, or < Create > a new trajectory. The bottom of this menu shows details about the loaded file, if any.

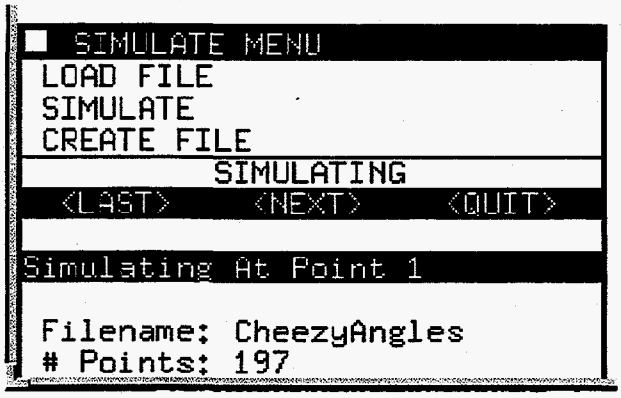


Figure B1e: After pressing SIMULATE

To load a file of solutions produced by IKOR/FSP simply click on the < Load File > button. A window will pop up prompting for the file's name. Entering < x > and then < Return > at this time will prompt the system to load the file CheezyAngles, the default file name produced by IKOR. However, other files can also be entered. Enter the path and the filename of the solutions file and then press < Return >. The name of the file and the number of points in that file will be displayed in the bottom portion of the window.

To simulate a loaded file, simply press the left mouse button while over the word < Simulate >. From here, the bottom portion of the simulate menu displays information that allows for a step wise movement through the solutions, see Figure B1e. Holding the left mouse button down while over any part of the Simulate Menu and meanwhile sliding the mouse up or down allows for an accelerated view of the trajectory. Press < Quit > when finished viewing the trajectory.

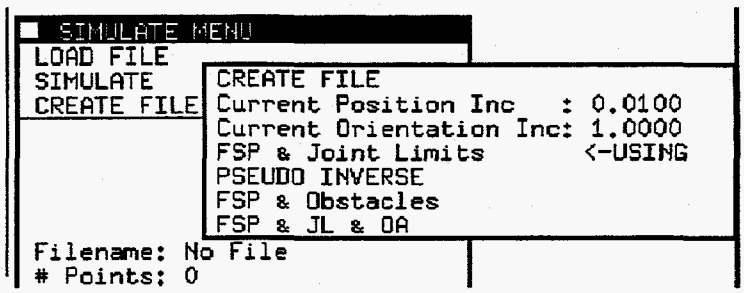


Figure B1f: Creating a trajectory

Finally, CheezyIGRIP enables the user to create straight line trajectories. By pressing < Create > with the mouse button, a window pops up as shown in Figure B1f. First choose one of the four methods by selecting it with the mouse button (The word

and arrow : <-USING will appear next to the selected constraints). Then press < CREATE FILE >. The two constants named position and orientation determine the size of the increments from one position to in the trajectory to the next.

Now the system prompts for a beginning and ending point for the trajectory. Using the method described above, move the manipulator's joints until the end effector is in the desired orientation and position; remember that IKOR/FSP uses the initial joint values as the initial condition to the differential equation. Once the starting position is found, select < Begin >. Now select the end position and orientation of the manipulator. The configuration of the manipulator is of no consequence to the IKOR/FSP system. Only its position is of any use unless orientation control is selected. Now the simulator calculates a trajectory file and runs IKOR.

CheezyIGRIP has the following command line options:

```
CheezyIGRIP s 565 720 o p d homer
```

The 's' and 'p' options allow the user to specify the size of the simulator window as well as its position on the screen. The 'd' options tell the system to use another machine to display the window. Note that no orientation control is the default. Orientation control is enabled by the 'o' option. It says make the size of the task space equal to 6. For orientation control, this must be equal to six so that the simulator will produce a Euler Angle Trajectory as well. The only way to make CheezyIGRIP generate orientation or non-orientation control is by sending the 'o' option. Additionally if a platform was constructed for the manipulator and its use is desired, a 'p' must be given as a parameter.

There are further incompatibilities between this simulator and IKOR. Using CheezyIGRIP by itself does not allow the USER to remove the activity of the platform or orientation control dynamically. To do this, the user must either use the driver or use IKOR's command line options. For instance, if a trajectory was created and a comparison was desired between the platform and arm versus just the arm, the following series of commands are necessary. This example assumes the trajectory is in the file TrajectoryPts and was made for the HERMIES arm and IKOR must have been made for the HERMIES arm:

```
% IKOR t0 p
% cp CheezyAngles Soln_with_platform
% IKOR t0
% cp CheezyAngles Soln_without_platform
```

CheezyIGRIP is capable of displaying both motions as long as the USER simply types in the name of the files. Recall that IKOR takes the task space, joint space, and method as command line options. A task space of three means no orientation control, while six means orientation control. If a trajectory was made using CheezyIGRIP without

orientation control (that is, -n3 as a parameter), then sending a six to IKOR means nothing because the file EulerAngles will contain all zeros. On the other hand, if 'o' was given to CheezyIGRIP, then either a three or a six will work with IKOR.

Getting back to the platform/ no platform example above, the last three joint values in the kinematics routine are specified to be the platform, therefore using a 'm7' will tell IKOR to disregard the platform, and a 'm10' will tell IKOR to use it. The option 'p' is recommended however.

Appendix B2: Silicon Graphics Station

A Silicon Graphics Station is designed for high speed video and multimedia power house. It is an ideal architecture for software packages such as TELEGRIP, which is capable of real-time animation and simulation.

To run this simulator with the IKOR controller, find a silicon graphics machine with the two sticks, two buttons, and one pot joy stick box. It is possible to move this box to other machines, see the systems administrator. Log into an account and run TELEGRIP (at RPSD, this is usually done by entering tg, tg2, or tg3 in one of the shell tools). Next, in another shell tool change directory:

```
% cd /rpsd/u4/people/hacker/Robots/TESTBED
```

Now run the program < joy >. Joy has one command line argument that defaults to 2010 if no argument is given. This argument represents the port number and must be greater than 2000.

First the manipulator must be loaded into the workcell. Go to the TELEGRIP program window and select Layout from the Main Menu (see Figure B2a).

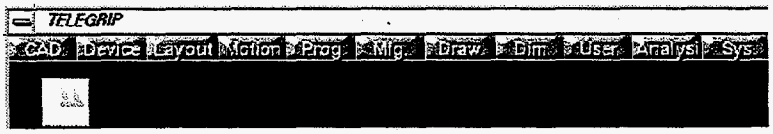


Figure B2a: TELEGRIP's Main Menu

Once the Layout option is selected two submenus are shown on the right side of the window. In the top right submenu, choose Workcell. Figure B2b should now be seen on the right side of the software's window. Choose Retrieve Workcell. A box will pop up with selections, choose: `hacker/HACKERlib/WORKCELLS`. Another box will pop up, this time choose: `AIRARM`. After a few moments, the AIRARM should be displayed inside the Workcell. If the AIRARM is not displayed or the options stated above do not exist, see the administrator.

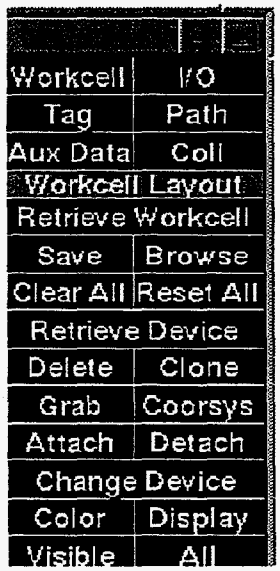


Figure B2b: Layout/Workcell Menu

Once the AIRARM is loaded into the workcell, the next thing to do is to hook it up to IKOR. By choosing < Sys > from the main menu, as shown in Figure B2a, the menus shown in Figure B2c should appear. If different, then choose the < File > submenu from the upper right hand corner of the window. At the bottom the Figure B2c, the LLTI menu can be seen. Simply press the add button.

A pop up box should appear as shown in Figure B2d. The first field should read port. If it does not, click on the word until port appears, or simply type the word port. The port address is the same address that joy takes as a command line argument. Enter the argument given to the joy program or the value 2010 if no argument was given. All TeleGRIP LLTI ports must be greater than 2000.

Next, hit < done > in the upper right hand portion of the pop up box. The pop up will disappear and the AIRARM will be under joystick control.

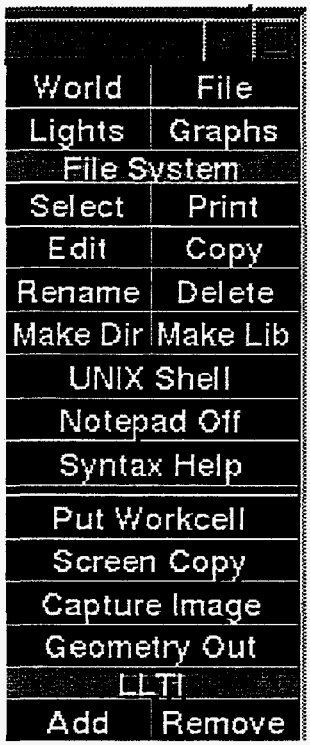


Figure B2c: System/ File Menu

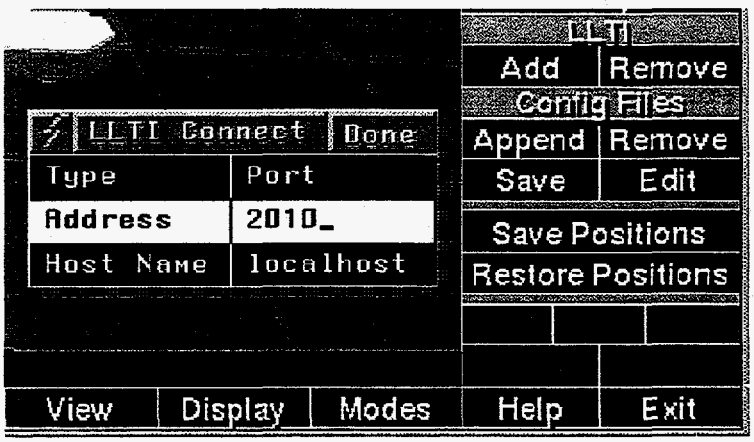


Figure B2d: Connecting to the LLTI Interface

Once the LLTI connection has been established, the TELEGRIP mouse and keyboard interface will slow dramatically. If camera angles, lighting, or world orientation needs to be changed, the USER must take an extra amount of time.

The joystick is relatively simple to use. It has two sticks, two buttons, and one pot. Joystick one will either control the end effector's orientation or the platform motion dependent on the position of switch one. Joystick two always controls the position of the arm. In fact, joystick 2 is the controlling inputs into the IKOR system since the mathematics of the orientation control for the AIRARM have yet to be developed.

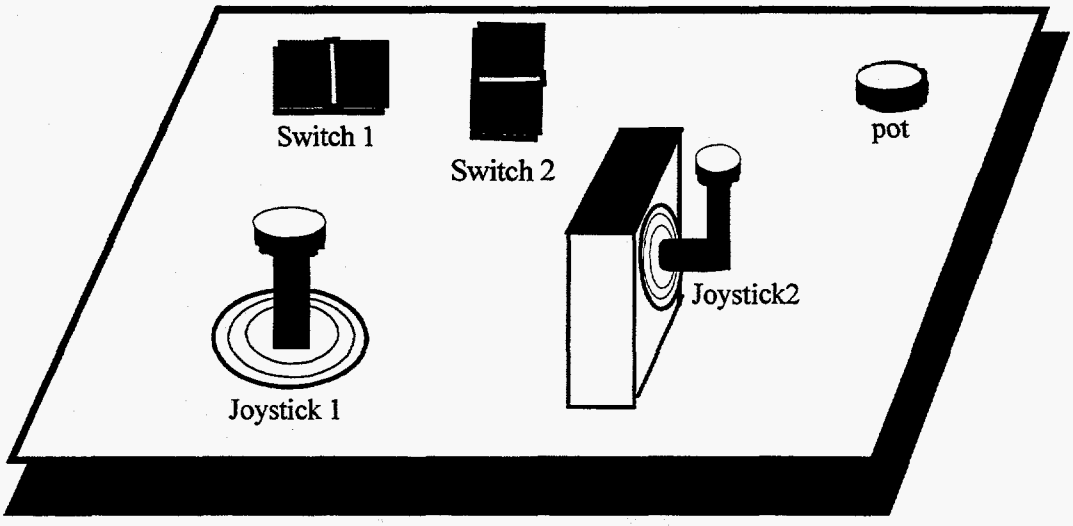


Figure B2e: Joystick Configuration

The control of the platform is somewhat awkward since its rotation is based on the world coordinated axis and not its own. The following chart summarizes how the joystick is interfaced with the AIRARM manipulator.

Device	Button #1	Value
Joystick #1	on	Controls yaw, pitch, roll of EE
	off	Controls x, y, and theta of platform
Joystick #2		Controls x, y, and z of the EE
pan		Controls pitch of EE
Button #2		Shuts down the joy program

That about does it, have fun.

Appendix B3: Modification of IKOR

Documentation

Because of the special continuity requirement discussed in §2.2. A lot of work was put into documenting, and drafting system information. Appendix C shows the flow of information in the system, however once this general outline is understood it becomes necessary to venture within the software itself. It is therefore important to able to find detailed information within the code as to what the subprocedures accomplish, their inputs and outputs, and a modification list.

The coded included in the paper in Appendix D contains such documentation. It is important when developing new code or modifying the existing code to describe variables, procedures, as well as what is being accomplished.

PROCEDURES:

For procedures, each procedure should have a header like the one shown below. The header should contain the procedure's name, a brief description, inputs, outputs. There should also be a section discussing when the procedure was modified and what was done during the modification.

```
/*_____________________________________________________
  | name:                                               |
  | description:                                        |
  |                                                     |
  | inputs:                                             |
  | outputs:                                            |
  |_____________________________________________________|
  | Modifier      Date          Description             |
  |_____________________________________________________|*/
```

VARIABLES:

Each variable should contain a brief description of its use when it is declared or passed into a function as shown below:

```
int dog,  /* 4leged mammal created to lick toes.  */
    cat,  /* 4leged mammal created tobe indignant.*/
    rat;  /* 4leged mammal created tobechasedbycat*/
```

These sometimes seem self-evident in coding, but they can become very helpful after a couple weeks pass and we completely forget what a cat or a dog is.

Appendix B3: Modification of IKOR
Using MATRIX Code

Throughout the code, matrix routines created by Ole Dorum are called. The structure Matrix consists of two integers and a row pointer that points to columns:

```
int N;          /* Number of Elements in a row    */
int M;          /* Number of Elements in a column */
float **p;      /* Pointer to Pointer             */
```

When using any of these routines, a pointer of type MATRIX must be declared. In previous versions of the code, it used to be necessary to have the word struct in front of all MATRIX declarations:

```
MATRIX *mymatrix;  /* Allocates 2 bytes of memory  */
```

In order to use this matrix, memory must be allocated for the structure. Ole Dorum allowed two mechanisms by which this can be accomplished. The memory leaks found in the code are reflective to the misuse of the two techniques. The first, more traditional mechanism is that of a mat_malloc function. When the function mat_malloc(int N, int M) is called, memory is allocated for the two integers as well as the pointer to a pointer (p). Next a row pointer is allocated to be of size MAX(N, M). Mr. Dorum used the MAX function to facilitate one of his transpose matrix subroutines. Finally each row pointer is allocated space for a column vectors of size M that will hold all of the data for the matrix. Finally, the matrix pointer in the original structure is assigned to the pointer that points to the row of pointers.

To undo this, a call to mat_free() is necessary. Without freeing the matrix, every time the function is called, more memory is allocated and the previous memory is lost. As long as mat_free() is used, correct use of the first mechanism is guaranteed.

However, another mechanism exists that provides for more complexity. In this mechanism, some of the functions call mat_malloc() for you. These functions are recognized by the presence of a 2 in their names (see examples below). Therefore, if the matrix is mat_malloc'ed by the programmer and one of these functions is called, a memory leak results since the initial mat_malloc() can never be undone until the program stops execution. This is shown in the example one below. Even though these functions call mat_malloc() for you, they DO NOT mat_free() them for you. Therefore if a variable has already been allocated and a function with a '2' is called, a memory leak necessarily results.

ex: 1

```
MATRIX   *mymatrix,/* Pointer to mymatrix        */
         *x,       /* Pointer to matrix x        */
         *y;       /* Pointer to matrix y        */

mymatrix = mat_malloc (3,4); /* Allocate mem for matrix */
x        = mat_malloc (4,3); /* Allocate memory for x   */

y = mat_tra2(mymatrix);      /* y is the transpos matrix */
                             /* y is allocated by this   */
                             /*    function*/
mat_tra(x, mymatrix)         /* OK too, both already     */
                             /*    allocated */
x = mat_tra2(mymatrix)       /* BAD, x is alloc'ed twice */

mat_free(x);
mat_free(y);
mat_free(mymatrix);          /* Don't forget, free matrix*/
```

Also be aware of this situation:

ex: 2

```
MATRIX *x, *y; *z;
z = mat_malloc (30, 45);

    <<<<  insert any code that fills matrix z >>>>

x = mat_tra2(z);   /* OK, x unallocated previously */
y = mat_cp2 (z);   /* OK, y unallocated previously */
y = mat_cp2 (x,z); /* BAD, allocates y twice     */
```

One way to detect these leaks is with the UNIX shell command:

ex: 3

```
    % ps -aux | grep <program>
```

If this is run several times during the execution, and the memory usage is growing rapidly, there is a memory leak.

Appendix B3: Modification of IKOR

Compiling Files and Procedures

This appendix details how to compile the software, how to add files and procedures, and how to add new manipulators. The system currently uses gcc (that is, the gnu 'c' compiler). Assuming that the compiler is in the systems analyst's path (that's you), the following should work.

Compiling the System

To compile a manipulator, go to the make directory, COMPILE/make (see the beginning of this Appendix for a description of the file system.) Each manipulator is compiled differently:

```
HERMIES      :  make TOP=$(HOME) ROBOT=HERMIES
AIRARM       :  make TOP=$(HOME) ROBOT=AIRARM
TWOD         :  make TOP=$(HOME) ROBOT=TWOD
ONETIME      :  <see below>
```

The makefile actually assumes that TOP=$(HOME), however if the location of the directory kinematics is not ~/$(HOME)/kinematics then this can be overridden. Additionally if Robots is not the directory off of kinematics, it can be overridden by the variable VER. For instance, if the manipulator i needed to compile was the AIRARM and the system was located in /rpsd/u1/hacker/kinematcs/IKORv2.0, then the following flags would enable make to compile the manipulator:

```
make TOP=/rpsd/u1/hacker VER=IKORv2.0 ROBOT=AIRARM
```

The makefile assumes that Jacob_$(ROBOT).c is located in :

```
~$(HOME)/kinematics/Robots/COMIPLE/MANIPULATORS/$(ROBOT)/Jacob_$(ROBOT).c
```

Make then makes the executables for IKOR and CheezyIGRIP and places them in:

```
~$(HOME)/kinematics/Robots/$(ROBOT)/IKOR
~$(HOME)/kinematics/Robots/$(ROBOT)/CheezyIGRIP
```

To make the ONETIME version of IKOR, which is used for testing the solution generator on problematic jacobians. The macro in general.h, called FSP_DEBUG, must be uncommented. The resulting IKOR executable must then be moved to IKOR_ONETIME, so that the link from the ONETIME subdirectory can be resolved. The TWOD is not complete, it will not compile properly, nor is it supposed too... yet.

Understanding MACROS

There are several macros that guide compilation. These macros are important to get the desired executable. They can be found in general.h and are shown below. They can also be given at the command line for the gcc compiler using -D (this is not advisable).

MACRO	Value	Description
DEBUG	defined	The file < data > will be produced and will contain run time data of the system. This data both long and detailed yields almost 200 lines of output STEP. The system is very slow when this is activated.
	undefined	The file < data > will not be produce.
DEBUG_OUT	defined	The user will receives the minimal output to <stderr> (usually the screen) as to what step the system is currently executing and what constraints are being avoided
	undefined	The minimal output to the screen will not be produced.
FSP_DEBUG	defined	When FSPv1.0 was first designed it was desired to execute only one user-defined Jacobian. When defined, the Jacobian found in FSP_data will be used by the system. The size N and M must be given on IKOR's command line. One limitation is that N and M must be within the limits of the manipulator. Therefore, choose the HERMIES system to use the most M's (1- 10).
	undefined	The system will get the Jacobian from the GET_JACOB() subroutine for the specified manipulator.
TWOSTEP	defined	As discussed in §3.2.3, when defined the system will find the full space motion in one step and avoid constraints in the null space.
	undefined	The system will avoid constraints as well as perform the motion at the same time.

As can be seen, numerous possibilities based on these combinations are feasible, not all of the combinations necessarily make sense together.

New Manipulators

Preparing the command line Executables

New manipulators should be installed in the following way. First make a directory parallel with the IKOR, HERMIES, and AIRARM directories. This is demonstrated in figure B3a; the new manipulator will be referred to as NEWARM. Any reference to the word NEWARM below, should be replaced with then desired name. Assuming that the system is in the pine/IKOR subdirectory, directory can be created by using the following command (The '$' is the system prompt):

```
$ mkdir NEWARM
$ mkdir COMPILE/MANIPULATORS/NEWARM
```

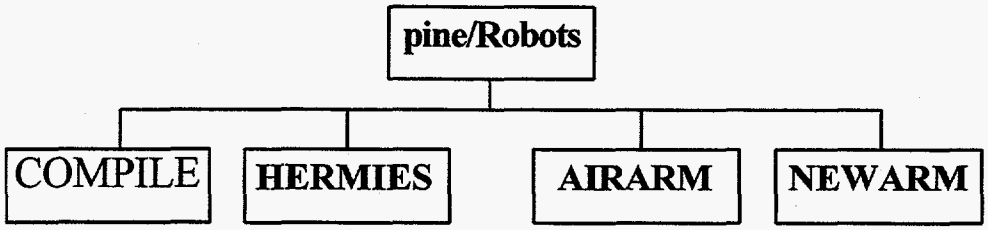


Figure B3f: System Directory Structure

Once the directory is made, copy the robot specifications file into the directory and name it ROBOT.dat. Also copy the file that contains the robot's Forward Kinematic and Jacobian subroutines into the COMPILE/MANIPULATORS/NEWARM directory and name it Jacob_NEWARM.c:

```
$ cp NEWARM_datafile
     /pine/Robots/NEWARM/ROBOT.dat
$cp NEWARM_Kinematicsfile
     COMPILE/MANIPULATORS/NEWARM/Jacob_NEWARM.c
```

Now by executing make, as described above, the system should be ready to execute. Simply change directories to the newarm directory and run IKOR or CheezyIGRIP. If the manipulator is not working correctly, the kinematics or the data file could have some trouble. If this occurs, give ikor the 'd' option. If the error was caused by a segmentation fault or bus error, try uncommenting the line 'data = stderr;' in the file IKOR_driver.c[11].

[11] If there are more than five links, CheezyIGRIP may have trouble displaying the manipulator. However, IKOR should work.

Updating the Graphical Driver

The graphical driver consists of two files. One is a script file and the other is a 'c' program called go.c. The script file writes a data file which go.c interprets. When a new manipulator, constraint, or criteria is introduced, both the script file and go.c must be altered.

To alter the script file, bring it into an editor. Do a search on either AIRARM or HERMIES. Copy the block of code for either existing manipulator and rename all occurrences of the existing manipulator to the new manipulator's name. Also be sure that a unique identifier is printed to the output file.

The changes to go.c are minimal as well. Create a macro at the beginning of the file for the unique arm identifier. Add a branching routine based on the new manipulator. Copy the system call from one of the other manipulators and change that manipulator's name to the newarm's name.

Backing Up and Restoring Systems

These directions should enable you to do everything I have done as far as backing up and restoring projects. The target for backups are floppy disks, while the target for restoring is a directory structure beneath the current directory.

TO RESTORE TO RESTORE TO RESTORE TO RESTORE TO RESTORE

---

1. insert the disk into the diskdrive /* Easy enough */

2. at the prompt type:

```
sudo mount /pcfs        /* PC file system      */
ls /pcfs                /* get listing         */
cp /pcfs/<name> ~/      /* copy file -> home*/
uncompress ~/<filename>.tar.Z  /* uncompress*/
tar xvpf <filename>.tar        /* unpack     */

3. sudo umount /pcfs       /* unmount pcfs   */

4. eject                   /* get disk out   */
```

A few notes: uncompress will remove the .Z extension. Tar will unpack all the files into a directory named appropriately after the tar file.

TO BACKUP TO BACKUP TO BACKUP TO BACKUP TO BACKUP TO BA

Backing up is very similar, in this example AIRARM system is being backed up.

1. insert the disk into the diskdrive /* Easy enough */
2. at the prompt type:

```
sudo mount /pcfs        /* PC file system     */
ls -l                   /* output listed below */
total 2839
drwxr-xr-x  4 hacker      512 Feb 16 15:03 ./
drwxr-xr-x 12 hacker     1536 Feb 16 13:14 ../
drwxr-xr-x  2 hacker     2048 Jan  3 10:01 AIRARM/
drwxr-xr-x  2 hacker     2048 Feb  7 15:20 HERMIES/
```

3. first pack up all directories with tar:

```
tar cvpf AIRARM.tar AIRARM /* destinatn file */
ls  -l                     /* output listed*/
total 2839
drwxr-xr-x  4 hacker     512 Feb 16 15:03 ./
drwxr-xr-x 12 hacker    1536 Feb 16 13:14 ../
drwxr-xr-x  2 hacker    2048 Jan  3 10:01 AIRARM/
drwxr-xr-x  2 hacker  145430 Feb  7 15:33 AIRARM.tar
drwxr-xr-x  2 hacker    2048 Feb  7 15:20 HERMIES/

compress AIRARM.tar       /* Now compress it*/
ls -l                     /* output listed  */
total 2839
drwxr-xr-x  4 hacker  512 Feb 16 15:03 ./
drwxr-xr-x 12 hacker 1536 Feb 16 13:14 ../
drwxr-xr-x  2 hacker 2048 Jan  3 10:01 AIRARM/
drwxr-xr-x  2 hacker75430 Feb  7 15:33 AIRARM.tar.Z
drwxr-xr-x  2 hacker 2048 Feb  7 15:20 HERMIES/

mv AIRARM.tar.Z /pcfs /* move to disk       */
sudo umount /pcfs     /* unmount pcfs       */
eject                 /* get disk back      */
```

Compressed files on a SPARC station usually have either '.Z' or '.gz' as an extension. '.Z' is for compress/uncompress and '.gz' is for gnuzip.

Appendix B3: Modification of IKOR

Version List

Many versions of the system are currently available. Each version has different capabilities and scope. Figure B3g shows time chart of each version's development. Following this is a description of each version.

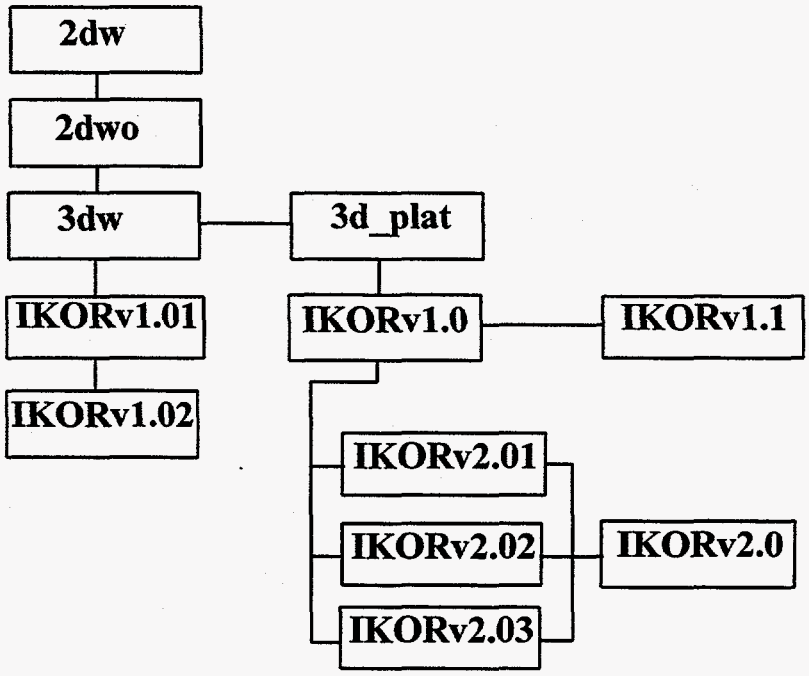


Figure B3g: Development of IKOR over time.

There is a good chance that there were versions before '2dw'. However, this is a reasonable place to begin since I have been unable to locate any software prior to this version. Each of these versions have been backed up and stored in the archive directory under IKOR. See Appendix B3 backup and restore for information on expanding these systems.

System Name	Originator	Description
2dw	Boozer, Tulloch Lupo	A two dimensional, four joint manipulator with orientation control. Able to avoid obstacles and joint limits while producing a least norm motion. Obstacles included spheres and polygons. Trajectories included arcs and straight lines.
2dwo	Tulloch	Same as 2dw, but orientation control was removed.
3dw	Carlson	A three dimensional, seven joint manipulator with orientation control. Able to avoid obstacles and joint limits while producing a least norm motion. Only sphere obstacles implemented and trajectories were circular or straight line trajectories.
3d_plat	Morgansen	Same as 3dw but implemented with a platform as well as the enhanced version of FSP, FSPv1.0. However, obstacle or joint limit avoidance were not available.
IKORv1.01	Hacker	Same as 3dw but with enhanced version of FSP, FSPv1.0.
IKORv1.02	Hacker	Same as IKORv1.01 but orientation was removed.
IKORv1.0	Hacker	Same as 3d_plat, but orientation control was allowed to be turned on and off as well as the platform. Preliminary portability was established. Also both joint limit and obstacle avoidance were made available.
IKORv1.1	Hacker	Same basic backbone of IKORv1.0, except this version is a real-time system and implemented on the AIRARM. Also only joint limit avoidance was available, and platform control is based on joystick control and not inverse kinematics.
IKORv2.01	Fries	Same as IKORv1.0, except this version includes basic version of FSPv2.0.
IKORv2.02	Hacker	Same as IKORv1.0, except the portability problems were solved.
IKORv2.03	Hacker	Same as IKORv1.0, except the analytical structure was changed to include the one-shot method.
IKORv2.0	Hacker	This incorporates IKORv2.01, v2.02, and v2.03. Contains the full FSPv2.0 enhanced code as well as full documentation

Appendix C: System Data Management

The data flow diagrams show the basic flow of data among four entries: USER, IKOR, FSP, Simulators, and the Manipulator Design Engineers. This Appendix includes data flow diagrams that explain the inner workings of IKORv2.0 as well as a data element dictionary:

The data flow diagrams are intended to give the modifier of the system a general idea of the data flows and processes involved. A Data Element Dictionary is necessary to decrease the learning curve of the system. It is the responsibility of the modifiers to expand and update this dictionary.

Appendix C1: Data Flow Diagrams and Descriptions

This part of Appendix C describes the entire system from its boundary perspective down into its primary and secondary levels. The explosions of each successive level are intended to aid the next system's analyst in determining the structure and composition of the system. The data flow diagrams in this appendix contain four basic symbols as shown in figure C1.[12]

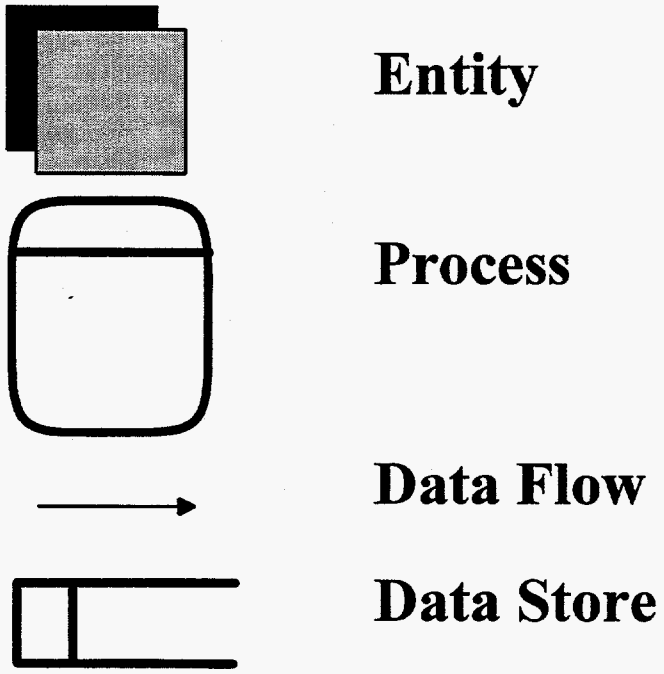


Figure c1 The four basic symbols used in Appendix C1 Data Flow Diagrams

Entities are sources of data such as type of manipulator or manipulator specifications. The rounded square is representative of a process that transforms or changes data. The data flow is representative of data that flows from one process, data store, or entity to another. The destination of the data is shown by the arrow. Data flows may carry several data elements as well as structures. The last symbol in this figure is the Data Store. The Data store is an open ended rectangle that is representative of physical storage such as a disk.

Detailed information on all Data Stores, Processes, Data Elements, and Flows can be found in Appendix C2, the data element dictionary.

---

[12] Data Flow Diagrams are modeled after C. Gane and T. Sarson, Structured Systems Analysis and Design Tools and Techniques (Englewood Cliffs, NJ: Prentice-Hall, Inc., 1979)

Context Data Flow Diagrams

Figure C1a is the Context Level Diagram of the system. It shows the boundaries of the system. The system is broken into four major subprocesses: 1 IKOR, 2 FSP, 3 Simulators, and 4 Kinematics Production. Each of these processes is described in detail below. The entity USER is anyone who operates the system, whether a system's analyst or laymen, the goal was to make the interface to IKOR relatively simple (see Appendix B1 for push-button instructions on how to run the system). The USER is responsible for supplying the type of manipulator to use with the system, what constraints they wish to have FSP adhere to, as well as the size of the joint and task space for the manipulator. Also the USER can choose what kind of criteria to use such as Least Norm or strength optimization. In return for the USER's efforts in determining controlling parameters, IKOR can provide either detailed or simple output dependent on the compiling options chosen. Also the USER will have access to the data store Joint Values, which they can then feed into a simulator and witness IKORv2.0's true glory.

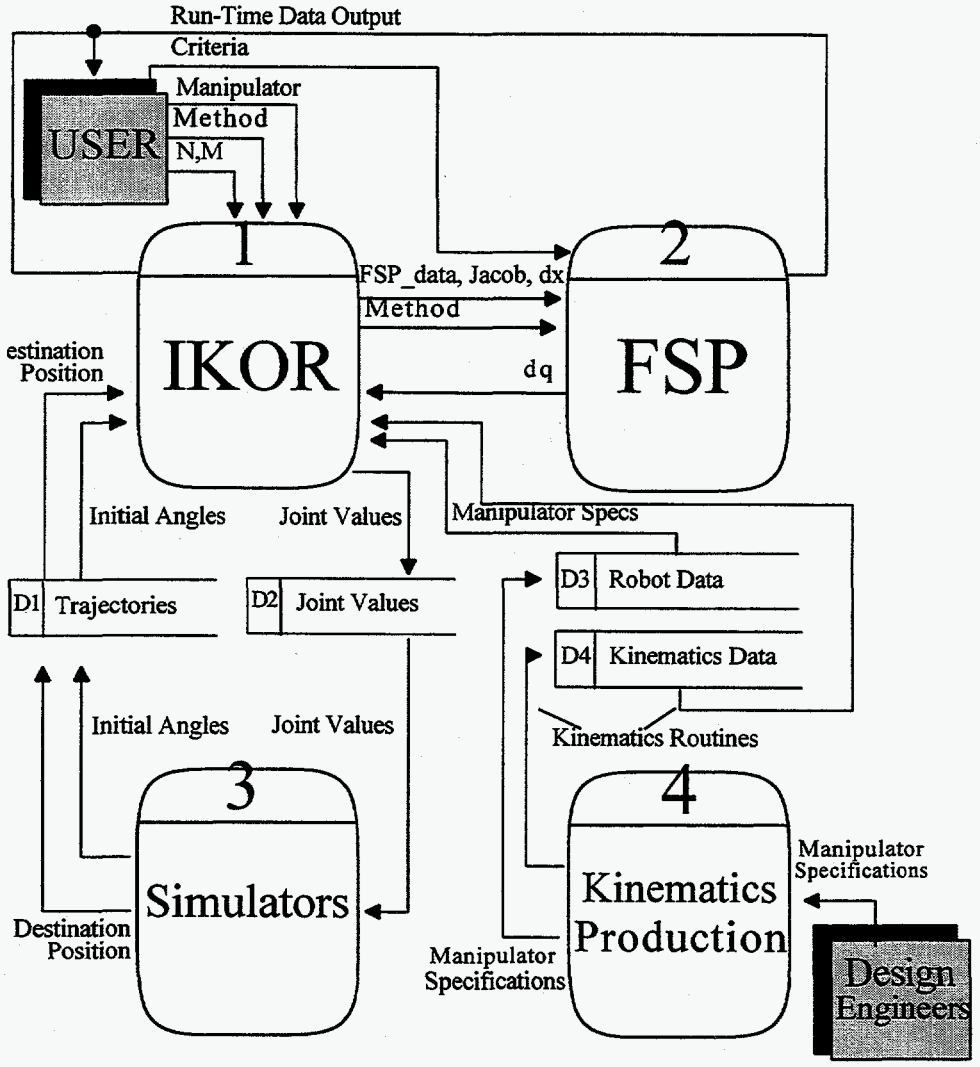
If it is desired to integrate another manipulator, the entity Design Engineers must provide pertinent information about specifications such as number of links, link lengths, type of joints and their limits, as well as the type and size of a platform if desirable. In return for the designer's efforts, IKOR will solve and trajectories for the new manipulator (see §3.2.4 for manipulator construction and porting and Appendix B3).

Two processes, 3-Simulators and 4-Kinematics Production, are not exploded past the context level diagram. The subprocess, 3-Simulators, allows the USER to view output of IKOR/FSP on a device that resembles the manipulator either as a stick figure or realistic 3D imaging. There are two simulators that are currently used with the system. The first simulator is CheezyIGRIP; this simulator has been developed in parallel with IKOR/FSP. In this case, all parameters given by the USER in the context diagram are now provided by the simulator. However, CheezyIGRIP can not access all the powers of IKOR/FSP. Another Simulator, TeleGRIP/IGRIP® developed by DENEB©, can also be used. This simulator has compatibility with PRO-E a computer aided drafting package. This allows the Design Engineers to view a realistic 3D image of the manipulator being designed. It is also possible to run the IKOR/FSP system from this simulator. However, they are not fully integrated at this time. For more information on this simulator, see Appendix B1: SG system. Whichever simulator is chosen, it must be able to provide the initial angles, trajectory, and number of points to the data store, D1-Trajectories. In return, it must be able to accept the new joint values that IKOR/FSP provides.

The other process not exploded is 4-Kinematics Production. This process is responsible for creating forward kinematics and jacobian routines the manipulator. There are strict guidelines for this subroutine creation. Also 4-Kinematics Production is responsible for converting the Design Engineers Manipulator Specifications to a Robot data file that can be understood by IKOR. There are fairly strict guidelines for this also. Finally, this process is responsible for writing a small script executable that will copy these

files into the system files and compile the system. This is described in process 1.1 System Compilation.

Figure C1a: Context Data Flow Diagram



Level 1 Data Flow Diagrams

IKOR : see Figure C1b: Level 1 Data Flow Diagram

IKOR compiles, executes, and drives FSP. Based on information provided by the user (type of manipulator desired, size of joint and task space, and method), IKOR will compile itself during 1.1-System Compilation with procedures created by 4-Kinematics Production. Next IKOR initializes the manipulator with the data store D3-ROBOT.dat, as well as several global and local variables during 1.2-System Execution and Initialization. Finally the system begins 1.3-Feedback loop where IKOR determines the change in end effector motion (dx) based on a forward kinematics routine and a simulator defined trajectory. IKOR also creates the jacobian for the time step and passes these as well as the user defined method to 2-FSP. A more detailed description of this subprocess can be found in the Level 2 Data Flow Diagrams. However, first a general discussion of FSP is in order.

FSP : see Figure C1c: Level 1 Data Flow Diagram

The process 2-FSP has three well-defined subprocesses. The subprocess 2.1-Solution Generator has been optimized by both K. Morgansen (FSPv1.0) and G. Fries (FSPv2.0). More information on this subprocess should be attained from either of the aforementioned TMs. Its primary responsibility is to produce enough solution vectors to span the entire space of solutions. It consists of special case detection, blocking, and collecting solution vectors. The subprocess, 2.2-Initialize Criteria and Calculate Betas for Constraints, is exploded further in the Level 2 Diagrams. This subprocess is responsible for applying USER requested constraints, initializing the criteria, such as Least Norm, and calling the most efficient Analytical Solution process. Finally, dependent on whether constraints were discovered, the proper subprocess 2.3-Find Analytical Solution and Calculate dq will be called. Although only one process is necessary, two or more are desired to decrease average time step run-time. The inner workings of this process can be found in Pin's paper [1]. This process uses all acquired betas to calculate a specific solution dq to achieve the requested motion, dx. B and H are determined by the criteria chosen. Currently only the Least Norm Criterion is implemented.

Figure C1b: Level 1 Data Flow Diagram

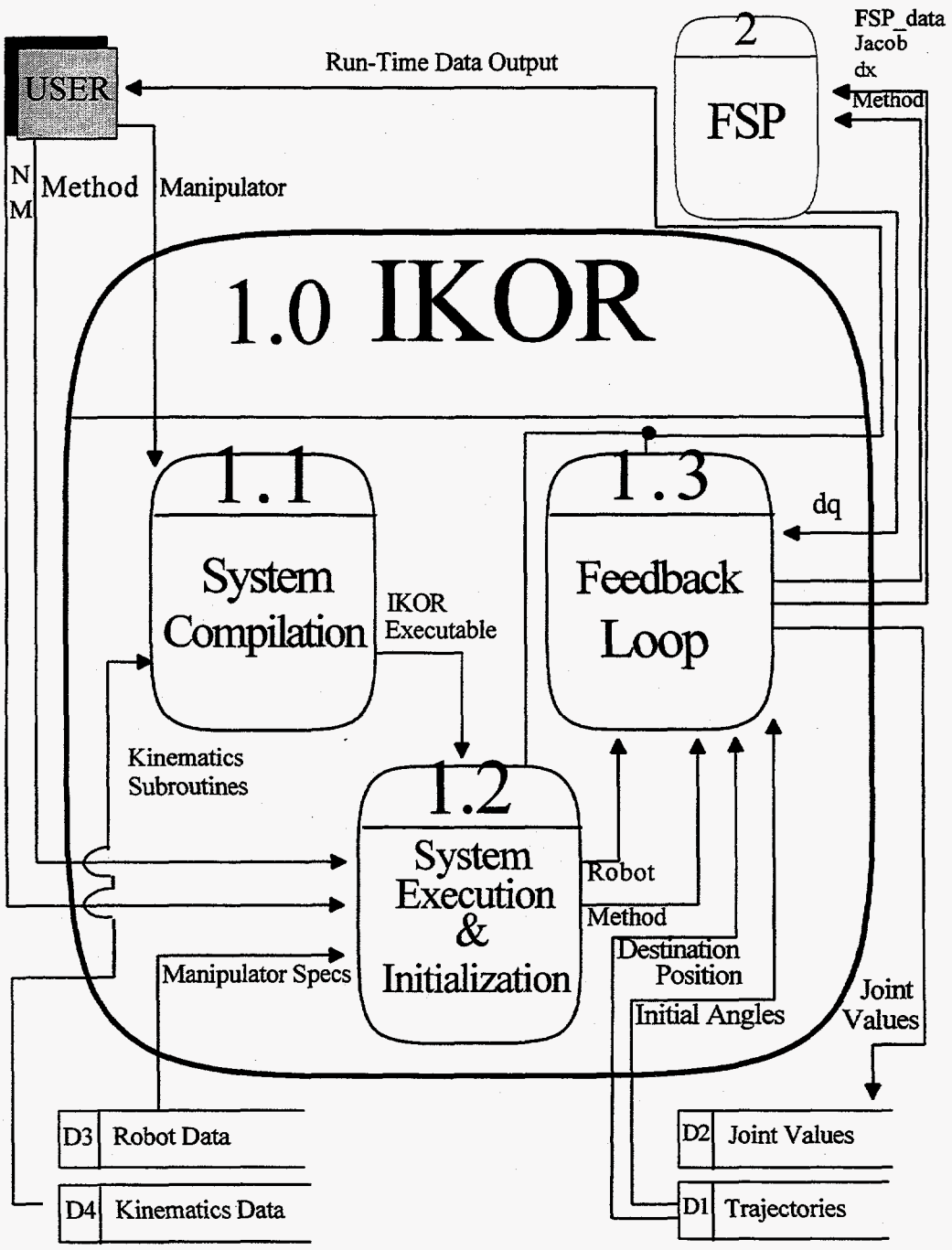
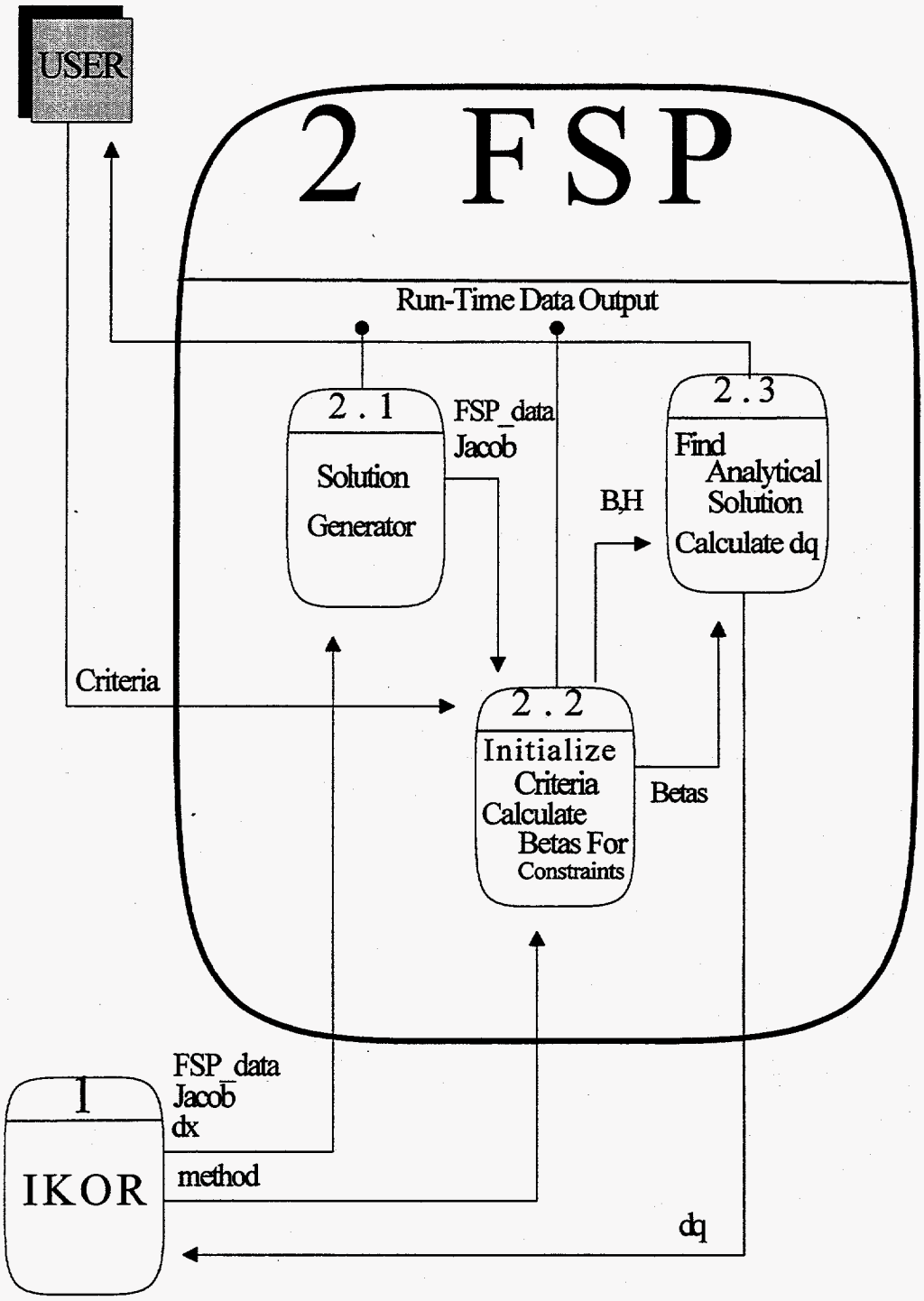


Figure C1c: Level 1 data Flow Diagram



Level 2 Data Flow Diagrams

1.0 IKOR: Level 2 Diagrams

The following three level 2 Data Flow Diagrams are the explosion of the Level 1 IKOR system. They include the following elements:

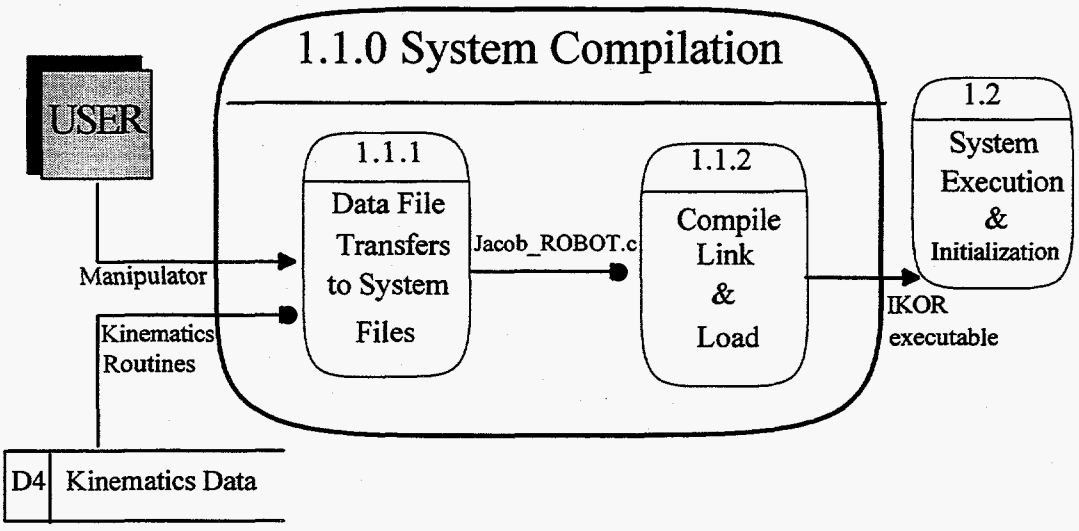
1.1.0 System Compilation
1.2.0 System Execution and Initialization
1.3.0 Feedback Loop

Each diagram also has a brief text based description.

1.1.0 System Compilation

System Compilation is required when the USER desires to change manipulators. Otherwise, this stage is bypassed. Once the USER chooses a different manipulator, a routine is called that transfers the source file that contains the forward kinematics and jacobian into the system, process 1.1.1. Two makefiles are then called which compile and link the new source code into the system. Finally, the executable is loaded into memory with the USER's options; this is subprocess 1.1.2.

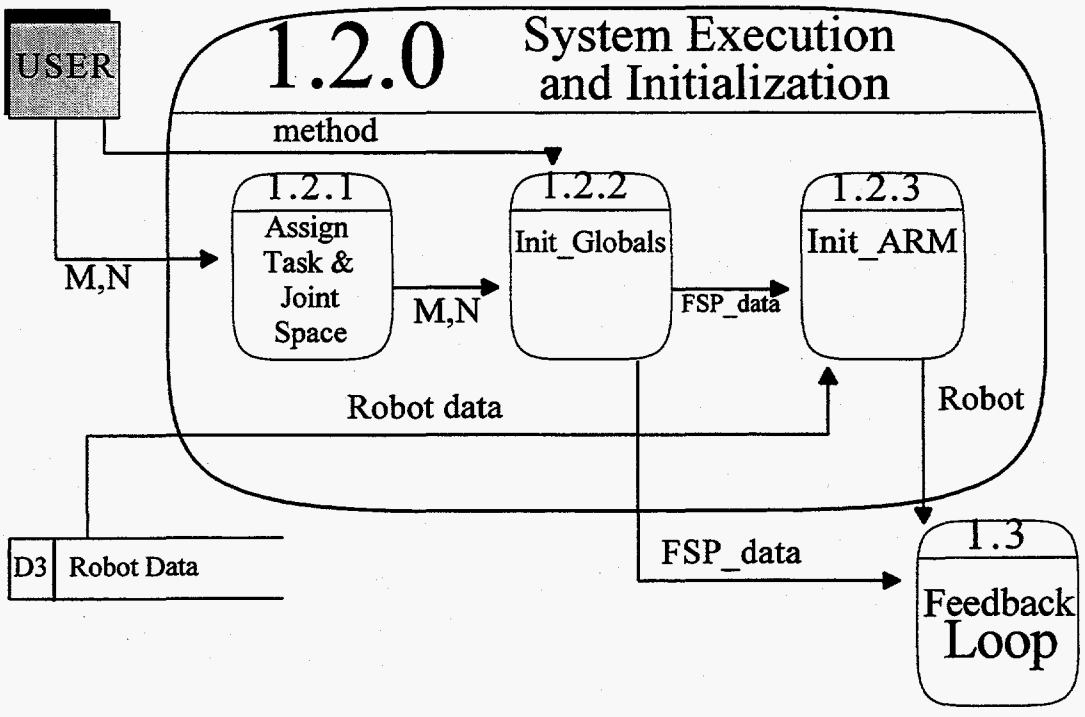
Figure C1d: Level 2 Data Flow Diagram



1.2.0 System Execution and Initialization

Once the system is compiled and loaded, it assigns the USER's requested task and joint space values to the system's joint and task space values, process 1.2.1. Next, local variables and global variables are initialized. Local variable initializations include the systems initial angles, the reading of the Trajectory file, and the reading of the obstacle file. Many of the global variables are matrixes used by CheezyIGRIP to draw the manipulators; they were included to maintain compatibility. Additionally, these values will become useful in the future when the platform or the end effector will be made to avoid obstacles. Currently neither of these are used by IKOR/FSP. The process Kinematics Production must use these matrixes if CheezyIGRIP compatibility is desired. This is process 1.2.2. Finally the data file specific to the manipulator must by read in from data store D3-Robot Data. The system uses the data flow Robot, initialized by process 1.2.3 for the joint limit, obstacle avoidance, forward kinematics, and jacobian subprocedures (see 1.3.0 Feedback Loop for more details). 1.2.0 completes by sending the data flows Robot and FSP_data, to 1.3.0 Feedback Loop.

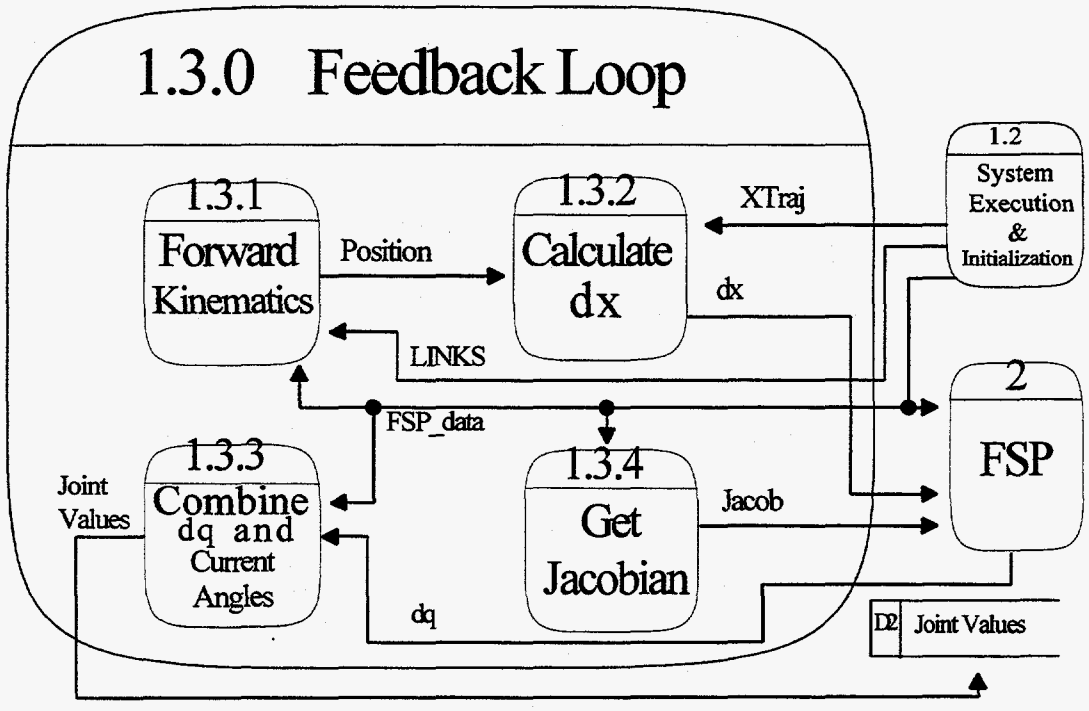
Figure C1e: Level 2 Data Flow Diagram



1.3.0 Feedback Loop

This process is responsible for most of the workings of the entire system. First the 1.3.1-Forward Kinematics routine is called using the values in Qarray contained within data flow FSP_data and the length of the links (LINKS is contained within the data flow Robot). This determines the end effector's position that is then subtracted from the desired end effector position contained in data flow XTraj, process 1.3.2. The 1.3.4-Get Jacobian subprocess is also called, which computes the derivative of the forward kinematics based on the current position. The Jacobian, represented by the data flow Jacob, desired positional change, data flow dx, and FSP_data are all sent to 2-FSP, which returns the change in joint values to 1.3.3-Combine dq and Current Angles. These values are written to the Data Store Joint Values.

Figure C1f: Level 2 Data Flow Diagram



2.0 FSP Level 2 Diagrams

The Level 1 Data Flow Diagram for FSP contains three subprocesses:

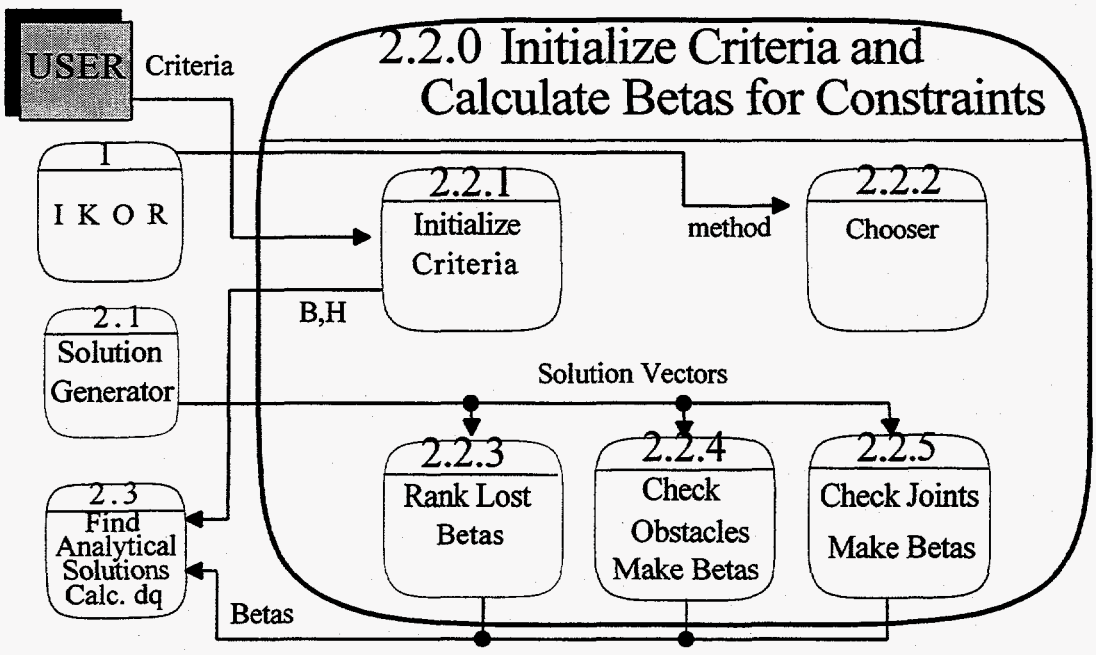
2.1.0 Solution Generator
2.2.0 Initialize Criteria and Calculate Betas
2.3.0 Find Analytical Solution and Calculate dq

However, both 2.1.0 and 2.3.0 are described in detail in three technical memorandums. For 2.1.0, see G. Fries and K. Morgansen. For 2.3.0, see F.G. Pin. Subprocess 2.2.0 is described below.

2.2.0 Initialize Criteria and Calculate Betas

After solution vectors are generated and 2.2.1 initializes the criteria, the system runs the chooser. Based on the value of method, the chooser calls the appropriate constraint detection routines and if violated, calls the necessary procedures to produce the beta vectors. Both the solution vectors and Betas are sent on to 2.3 Find Analytical Solutions and Calculate dq.

Figure C1g: Level 2 Data Flow Diagram

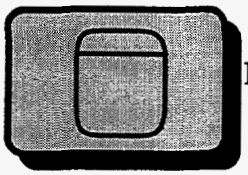


Appendix C2: Data Element Dictionary

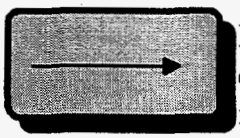
The Data Element Dictionary should be maintained to ease the learning curve of modifiers of the system. It is the responsibility of the modifier to expand and update this dictionary. The dictionary is divided into five sub categories.

Processes
Data Flows
Data Stores
Data Structures
Data Elements

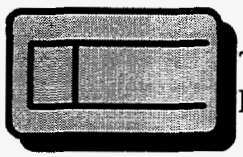
Each data element type has a separate form. Each form provides the name, a short description, and a symbol relating what data element type it belongs. Additionally, the forms also contain information pertinent to that data type.



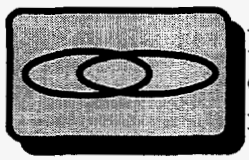
Processes contain inputs, a description of the process, and its outputs.



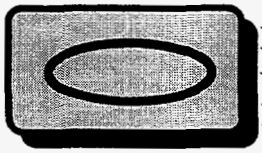
Data Flows on the other hand, contain the source and destination of the data flows as well as any structures that flow with it. Volume is also included to demonstrate the frequency of this transfer.



The Data Stores form includes the contents of the store as well as the Data Flows that flow into and out of the store.



Data Structures contain an additional description field that describes the data elements contained in the structure. Additionally, there is a field for related Data Flows and Volume.



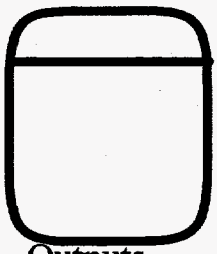
Finally Data Elements contain an alias field that shows other names used in the system to represent this element. The type of element, length, range, and discrete values is also included.

Appendix C2: Data Element Dictionary

Processes

IKOR_driver

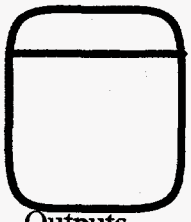
Description: Drives entire sytem, branches to Initializations, and performs feedback loop.



Inputs	Description of Process	Outputs
N M method	Calls Solutions_init() for FSP_data Calls Init_Globals() Feedback Loop: GET_ACTUAL_X() Finds dx GET_JACOBIAN() Calculate dq w/ CALC_PSEUDO() or FSP() Add dq to Qarray end	FSP_data dq dx Jacob method Joint_Values

CALC_PSEUDO

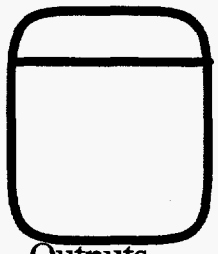
Description: Given a Jacobian and dx, calculates necessary change in Joint Values by a call to mat_pseudoinv.



Inputs	Description of Process	Outputs
Jacob dx	Pads Jacobian with zeros to make it a square matrix Calls mat_pseudoinv() Calculates dq = Jacobian_inverse*dx Returns dq	dq

FSP

Description Calculates changes in joint values necessary to acomplish desired dx. Calls constraint avoidance and criteria.

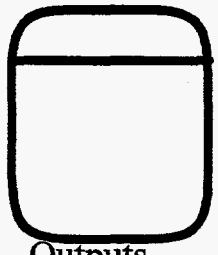


Inputs	Description of Process	Outputs

FSP_data
Jacob
dx
method
position
Robot
	Call Solution_generator()
	Call Criteria.. Least_Norm()
	Dependent on Method
	  Detect Constraint obstructions
	  Calculate Betas
	Find analytical solutions:
	  findt_with_Betas_Holonomic
	  findt_without_Betas_Holonomic
		dq

OPEN_FILES

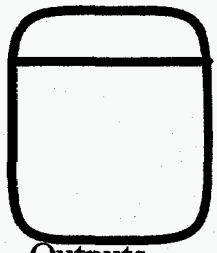
Description Opens most system files, initializes Qarray and XTraj and Number Of Points.



Inputs	Description of Process	Outputs
Trajectory Joint_Values	Request Name for data store: Joint_Values Opens data, Joint_Values, Trajectory Initailizes Qarray to Initial Angles Initializes XTraj to Trajectory Initializes Number of Points	Qarray XTraj Number of Points

Init_Globals

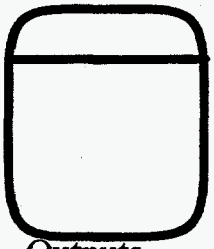
Description: Initializes Global Variables associated with the CheezyIGRIP simulator. Also initializes Robot



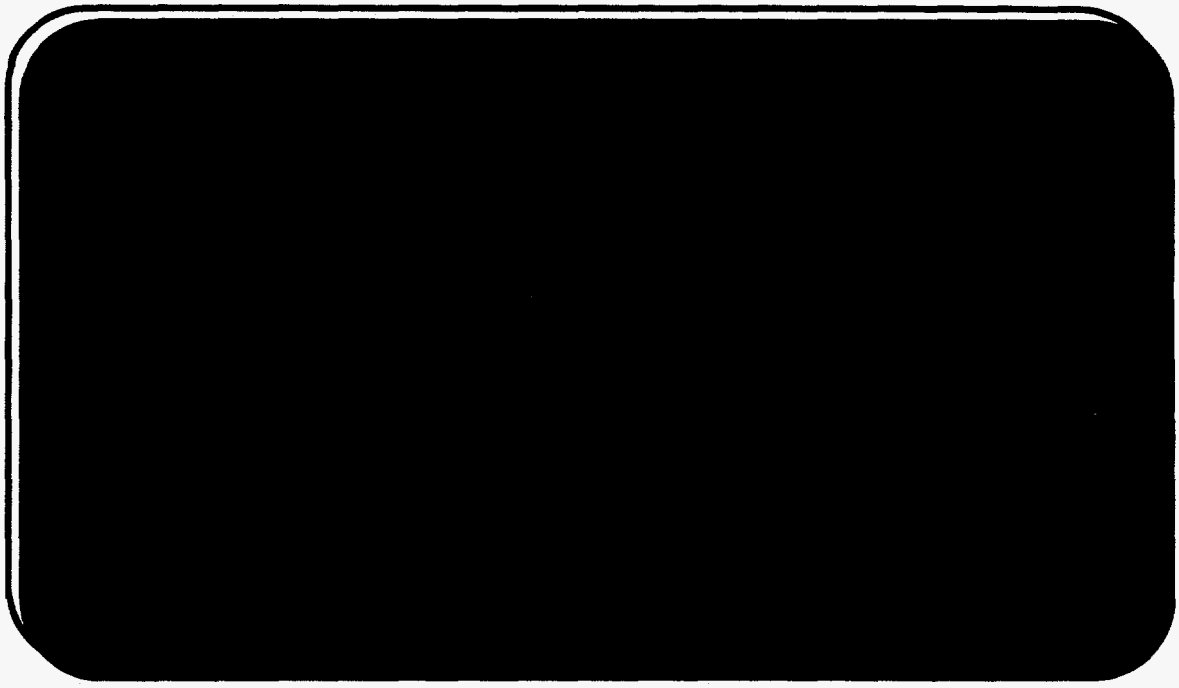
Inputs	Description of Process	Outputs
	Calls init_ARM() Allocates memory for Transformation matrixes Initializes matrixes that describe the end effector, and platform.	Robot

GET_JACOBIAN

Description: Calls GET_JACOB with the full manipulator link lengths.

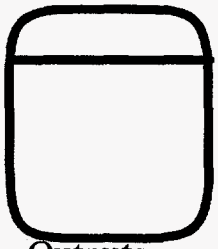


Inputs	Description of Process	Outputs
LINKS NL Qarray	Initializes link length variable LL used by GET_JACOB() to be equal to the original manipulator lengths. (see GET_JACOBIAN_ALTERED)	LL



getT2

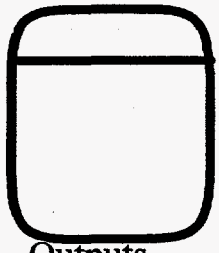
Description Calculates transformation matrix for given three rotations and three translations, rZ rY rX tX tY tZ



Inputs	Description of Process	Outputs
rot: Z Y X trans: X Y Z	Calculates Transformation Matrix	Transformed Matrix

ExtractRPY2

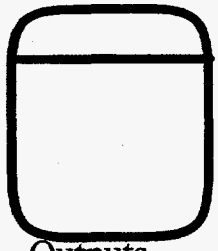
Description Extracts Euler Angles from input matrix which is the homogenous transfrom matrix from base to end effector



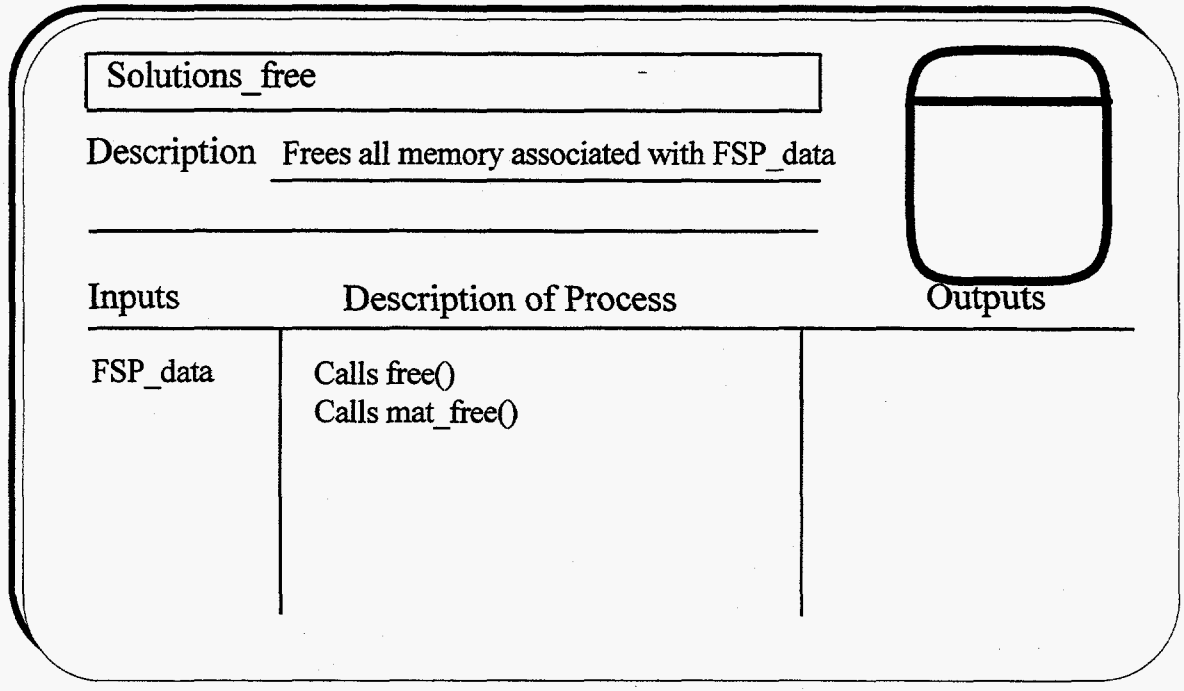
Inputs	Description of Process	Outputs
Transformation Matrix	Extracts Euler Angles	Euler angles

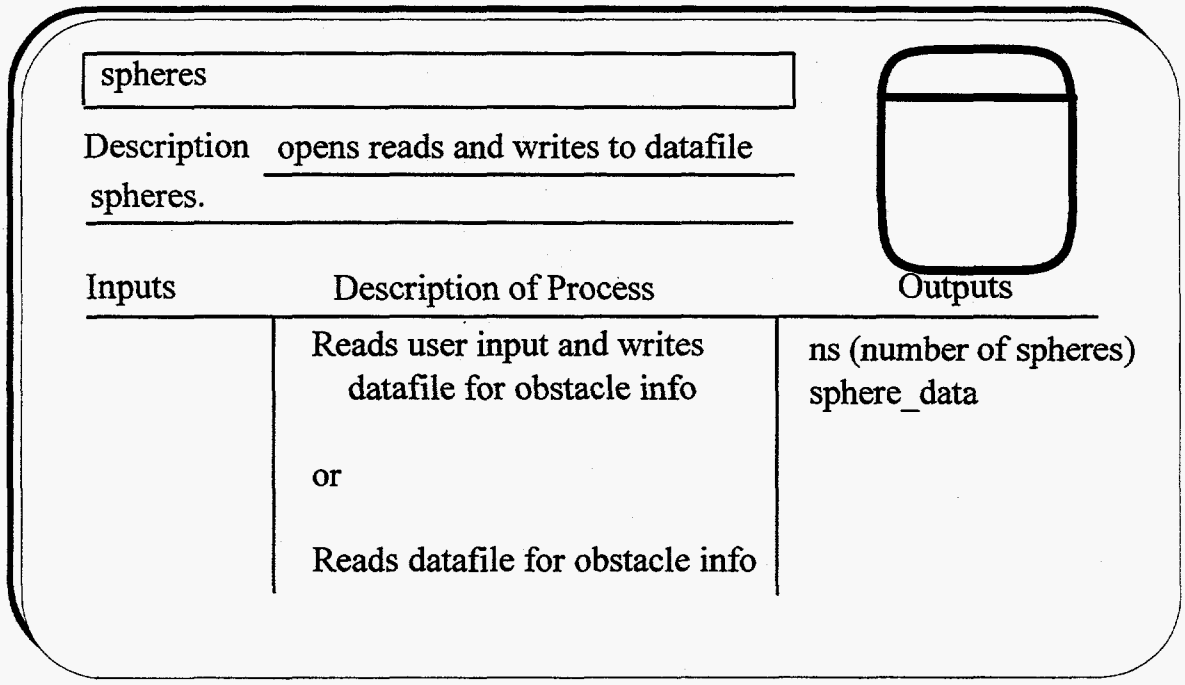
Solutions_init

Description Initializes data structure Solutions



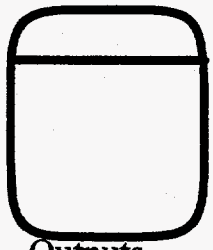
Inputs	Description of Process	Outputs
M N	Allocates memory for FSP_data Initializes FSP_data's M = Mred = M FSP_data's N = Nred = N Initializes cn (# of constraints) to ZERO	FSP_data


Solutions_free
Description Frees all memory associated with FSP_data
Inputs
Description of Process
Outputs
FSP_data
Calls free()
Calls mat_free()


spheres
Description opens reads and writes to datafile spheres.
Inputs
Description of Process
Outputs
Reads user input and writes datafile for obstacle info
or
Reads datafile for obstacle info
ns (number of spheres)
sphere_data

fmat_pr / fmat_prf

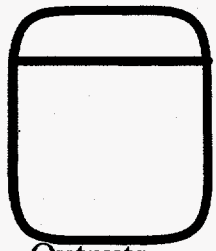
Description Prints matrix description, contents, and size to requested data file



Inputs	Description of Process	Outputs
File Pointer description matrix	Writes description and size of matrix to data file Writes contents of matrix with four or eight decimal places (pr or prf respectively)	File Pointer contains matrix information.

fprint_norm

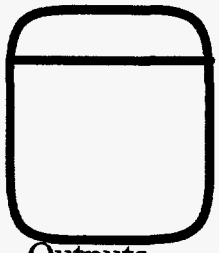
Description Prints the norm of the change in angles to the requested file.



Inputs	Description of Process	Outputs
File Pointer dq	Calculates norm = sum of the squares of each element.	Output file contains the value of dq for the current time step.

avoid_limits

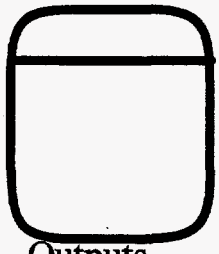
Description Joint limit detection routine and calls find_jl_betas if outside of range.



Inputs	Description of Process	Outputs
Qarray Robot	Determines if current value in Qarray is outside of operating parameters determined by Angles Max and Min Limits. If outside, calculates beta values by a call. find_jl_betas()	betas for each joint value in excess of limit.

avoid_obstacles

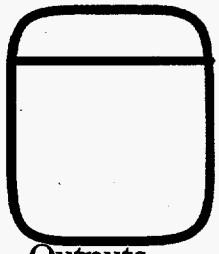
Description Drives routine that checks for intersection with obstacles, also calculates corresponing betas if needed.



Inputs	Description of Process	Outputs
FSP_data position sphere_data Robot	Cycles through each link and calls find_intersection_sphere() If intersection occurs, betas are calculated: find_obs_beta()	betas for collided link

find_intersection_sphere

Description Determines if a given link has intersected any known spheres. If so, returns information necessary to calculate betas,

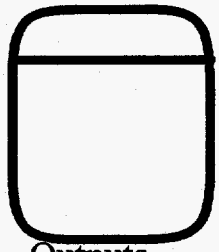


Inputs	Description of Process	Outputs
link sphere_data	Determines if link, or elbow intersects with a given sphere.	delta(distance link needs to move) newl(point on link of farthest intrusion) normal(vector from object center and point newl on link)

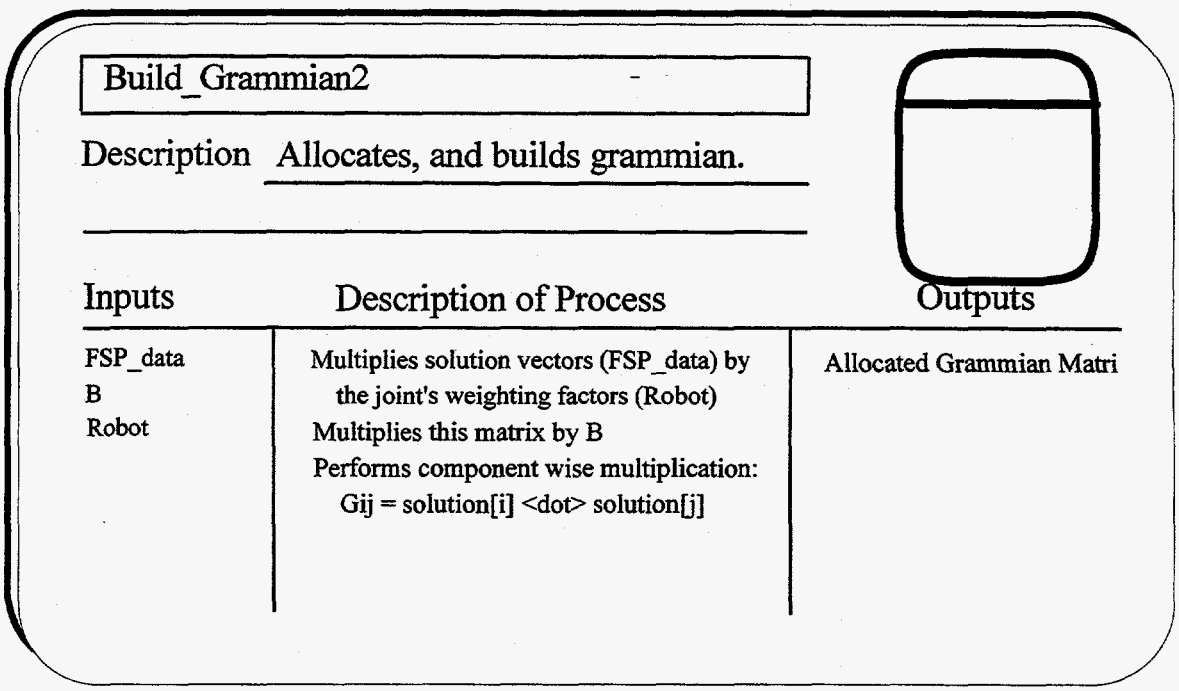
Least_Norm

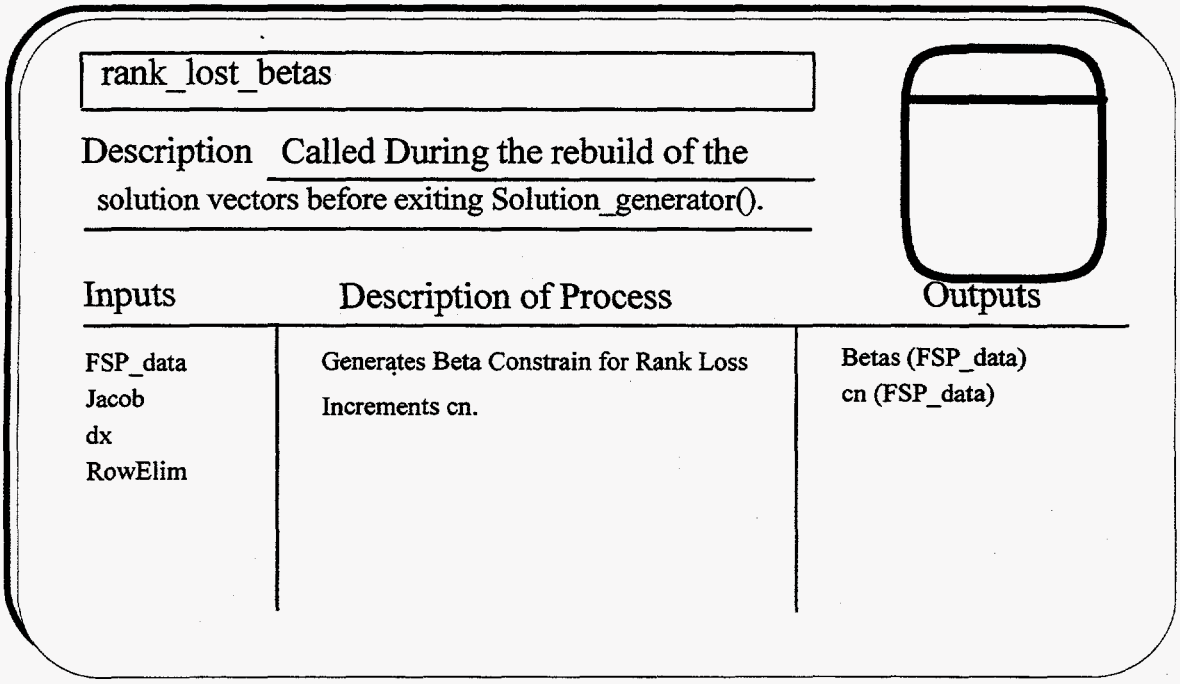
Description Initializes B, H for Least_Norm

Criteria



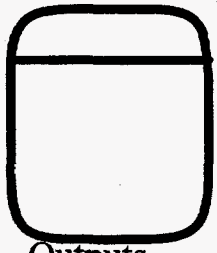
Inputs	Description of Process	Outputs
	B is the zero vector H is the zero matrix	B H


Build_Grammian2
Description Allocates, and builds grammian.
Inputs
Description of Process
Outputs
FSP_data
B
Robot
Multiplies solution vectors (FSP_data) by the joint's weighting factors (Robot)
Multiplies this matrix by B
Performs component wise multiplication:
Gij = solution[i] <dot> solution[j]
Allocated Grammian Matri


rank_lost_betas
Description Called During the rebuild of the solution vectors before exiting Solution_generator().
Inputs
Description of Process
Outputs
FSP_data
Jacob
dx
RowElim
Generates Beta Constrain for Rank Loss
Increments cn.
Betas (FSP_data)
cn (FSP_data)

find_jl_beta

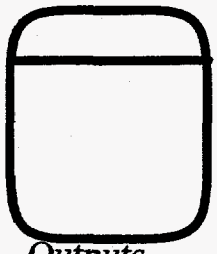
Description: Finds beta due to a joint being a radian distance beyond its limit.



Inputs	Description of Process	Outputs
distance link identifier	Creates Beta equal to the column identified by link divided by the distance needed to be moved. Increments cn (number of constraints)	Beta (FSP_data) cn (FSP_data)

find_obs_beta

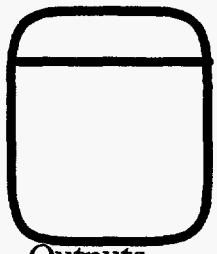
Description: Determines Beta vector based on information provided by find_intersection().



Inputs	Description of Process	Outputs
newl (pt on link of deepest intersection) delta (distance to move) normal(obs cntr to pt newl)	Calls GET_JACOBIAN_ALTERED() based on a newl terminated manipulator. Calculates Alphas[i,j]= sum of dot products of Jacob and solution vector Calculates Beta = Alpha * normal increments cn (number of constraints)	FSP_data's Betas cn

findt_with_Betas_Holonomic()

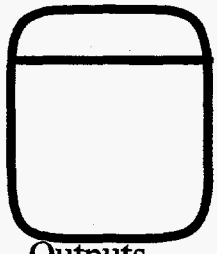
Description Calculates dq based on t's calculated for a robot on a holonomic platform and where some constraint was encountered.



Inputs	Description of Process	Outputs
B H Betas cn Solution Vectors	Calculates constants a, b, c, d, A, mu, and nu. Calculates t's Calculates dq	dq

findt_without_Betas_Holonomic

Description Finds t's and then dq's for a Holonomic platform or arm only when no constraints were encountered.



Inputs	Description of Process	Outputs
Betas cn Solutions B H	Quicker calculation of dq's when no constraints are encountered see findt_with_Betas_Holonomic() for more details.	dq

Appendix C2: Data Element Dictionary

Data Flows

Run-Time Data Output

Description This flow can produce either simple or detailed informatio of the system dependent if the macro DEBUG is defined (see general.h)

Source	Destination
Almost Every Process	USER or data file

Data Structure Traveling with the Flow	Volume
simple: Time step, and constraint avoidance detailed: Matrixes, system variable values.	simple: 3/STEP detailed: 100*STEP

Criteria

Description Determines if Least Norm or other criteria will be used, currently only Least Norm is available

Source	Destination
USER	2.2.1 FSP

Data Structure Traveling with the Flow	Volume
Criteria, and integer	One/STEP

Manipulator

Description Determines what manipulator is to be used with the system, if it changes then 1.1 System Compilation must be performed.

Source	Destination
USER	1.1 System Compilation 1.2 System Initialization

Data Structure Traveling with the Flow	Volume
Manipulator, an integer	Once/Execution

N, M

Description User requested task and joint space sizes,

Source	Destination
USER	1.2 System Initialization

Data Structure Traveling with the Flow	Volume
M, an integer N, an integer	Once/Execution

Method

Description Allows user to choose obstacle avoidance, joint limit avoidance, or any constraint.

Source	Destination
USER	2.2.2 FSP Chooser

Data Structure Traveling with the Flow	Volume
method, an integer	Once/Execution

FSP_data, Jacob, dx

Description System information: contains many goodies such as betas current joint values, size of FSP reduced matrix...

Source	Destination
1 IKOR	2 FSP
2 FSP	1 IKOR

Data Structure Traveling with the Flow	Volume
Structure: FSP_data Matrix: Jacob Matrix: dx	2 to 4/STEP

dq

Description Change in Angles necessary to effect the motion requested by dx.

Source	Destination
1 FSP	2 IKOR

Data Structure Traveling with the Flow	Volume
Matrix: dq	Once/Step

Destination Position

Description Contains the entire trajectory, aliases include XTraj.

Source	Destination
3 Simulators D1 Trajectories	D1 Trajectories 1 IKOR

Data Structure Traveling with the Flow	Volume
Data File: TrajectoryPts, and Euler Angles integer: Number_Of_Points	Once/Execution

Initial Angles

Description Contains the starting angles for all joint values (i.e. the initial condition for the differentail equation).

Source	Destination
3 Simulators	D1 Trajectories
D1 Trajectories	1 IKOR

Data Structure Traveling with the Flow	Volume
Array: Qarray	Once/Execution

Joint Values

Description The values of the Joint for the time step. This is an addition of the previous time steps angles plus dq.

Source	Destination
1 IKOR	D2 Joint Values
D2 Joint Values	3 Simulators

Data Structure Traveling with the Flow	Volume
Data File: Joint Values	Once/STEP or Once/Execution

Manipulator Specs

Description Data given to the Kinematics Production about a given Manipulator

Source	Destination
Design Engineers	4 Kinematics Production
4 Kinematics Production	D3 Robot data & 1 IKOR

Data Structure Traveling with the Flow

Data File : ROBOT.dat

Volume

Once/Execution

Kinematics Routines

Description Procedures for GET_JACOB() and GET_ACTUAL_X() the forward kinematics and jacobian calculation subroutines.

Source	Destination
4 Kinematics Production	D4 Kinematics Data
D4 Kinematics Data	1.1 System Compilation

Data Structure Traveling with the Flow

Data File: Jacob_ROBOT.c

Volume

Once/Execution if change of manipulator

B, H

Description Define type of criteria.

Source	Destination
2.2 Initialize Criteria	2.3 Find Analytical Soultions

Data Structure Traveling with the Flow	Volume
Reference Vector: B Weighting Matrix: H	Once/Step

Betas

Description Vectors which enable generated change in angles to avoi constraints.

Source	Destination
2.2 Calculate Betas for Constraints	2.3 Find Analytical Solutions

Data Structure Traveling with the Flow	Volume
Structure Solutions: FSP_data	see FSP_data

XTraj

Description Holds entire trajectory given by the data store.

Source	Destination
1.2 System Execution & Init	1.3.2 Calculate dx

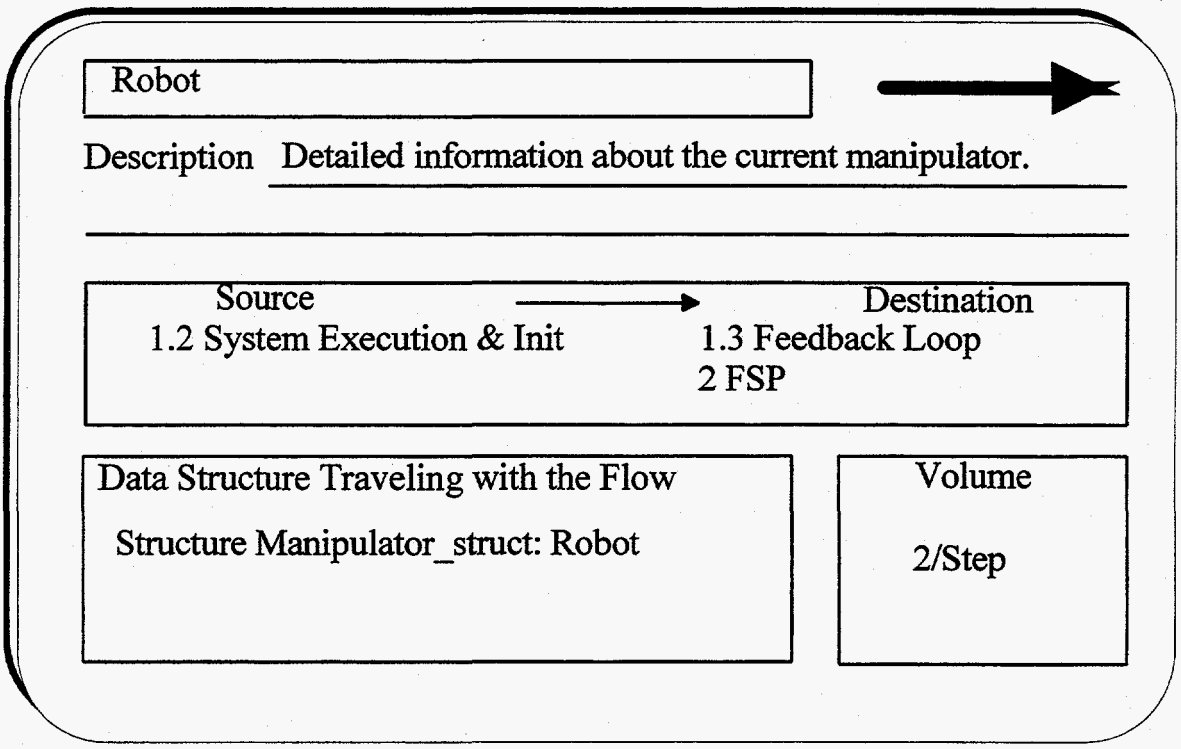
Data Structure Traveling with the Flow	Volume
Double Array: XTraj	Once/Step

Solution Vectors

Description Vectors that span the space given by dx

Source	Destination
2.1 Solution Generator	2.2.3-5 Beta Calculations 2.3 Find Analytical Solutions

Data Structure Traveling with the Flow	Volume
Structure Solutions: FSP_data	see FSP_data


Robot
Description Detailed information about the current manipulator.
Source
1.2 System Execution & Init
Destination
1.3 Feedback Loop
2 FSP
Data Structure Traveling with the Flow
Structure Manipulator_struct: Robot
Volume
2/Step

Appendix C2: Data Element Dictionary

Data Stores

D1 Trajectories

Description Contains information relative to the path planning of the current trajectory.

Contents
- Number of Points
- Trajectory (X,Y,Z, yaw, pitch, roll)

Data Flows In	Data Flows Out
3 Simulators	1.2 System Initialization

D2 Joint Values

Description Contains the result of IKOR/FSP. Namely, the joint values for each time step that accomplish the trajectory.

Contents
- A Value in radians or meters of each joint for each step.

Data Flows In	Data Flows Out
1.3 IKOR's Feedback Loop	3 Simulators

D3 Robot Data

Description Conains necessary specifications of the current manipulator.

Contents
File ROBOT.dat, # links, limits, home, link lengths, platform info.

Data Flows In	Data Flows Out
4 Kinematics Production	1.2 IKOR System Initialization

D4 Kinematics Data

Description Holds the subprocedures for the calculation of the forward kinematic and jacobian

Contents
"C" Procedures: GET_JACOB()
GET_ACTUAL_X()

Data Flows In	Data Flows Out
4 Kinematics Production	1.1 IKOR's System Compilation

Appendix C2: Data Element Dictionary

Data Structures

ANGLE

Short Description For each joint, max & min limit, home, translational/rotational

Description

Prism	If == Y, then translational else rotational
Max_limit	Maximum joint value in degrees
Min_limit	Minimum joint value in degrees
Home	Initial Position of Manipulator

Related Flows

Robot

Angles

Volume

One instance:

Substructure of

Manipulator_struct

Platform

Short Description Contains Platform Information

Description

Active	if ==Y, then platform exists
Holonomic	if ==Y, then Holonomic
Length	Length of platform
Width	Width of platform
Thick	Thickness of platform
Z_off	Offset of first joint from plat
L_off	Offset of arm from plat cntr
ANG_off	Angle offset of first joint

Related Flows

Robot

PLAT

Volume

One Instance:

substructure of

Manipulator_struct

Manipulator_struct

Short Description Contains all manipulator Information

Description

NA	Number of Angles in Manipulator
NL	Number of Links in Manipulator
Angles	Contains Pointer ANGLES[NA]
PLAT	Contains Pointer to Platform
Weights	Activity of Joints
LINKS	Values of each link

Related Flows

Robot

Volume

One Instance

MATRIX

Short Description See matrix.c, matrix.h

Description

p	double pointer to data
rows	Number of Rows in MATRIX
cols	Number of Columns in MATRIX

Related Flows

All MATRIXS
Jacob, dx, dq,
x_of_link ...

Volume

hundreds of
instances

Solutions

Short Description Holds a congloumarate of data used by FSP

Description

M	User requested Joint Space Size
N	User requested Task Space Size
Mred	FSP reduces Jacobian Joint Space
Nred	FSP reduces Jacobian Task Space
cn	Contains number of constraints
g	Contains solution vectors
Qarray	Contains current Joint_values
betall	Contains all Betas for time step

Related Flows

FSP_data

Volume

Altered in Every Time Step

Appendix C2: Data Element Dictionary

Data Elements

Prism

Short Description: Tells system if joint is prismatic or rotational

Aliases (Contexts): Robot->Angles[i].Prism

Alphabetic	X
Alphanumeric	
Numeric	

If Continuous:

Length

Range

If Discrete

Value	Meaning
Y	Translational/Prismatic
y	
Any Other	Rotational

Max_Limit

Short Description: The Maximum_Limit (includes Buffer zone)

Aliases (Contexts): Robot->Angles[i].Max_Limit

Alphabetic	
Alphanumeric	
Numeric	X

If Continuous:

Length: dependent

Range: -inf to inf

If Discrete

Value	Meaning

Min_Limit

Short Description Contains the Minimum Limit for desired Joint

Aliases (Contexts) Robot->Angles[i].Min_Limit

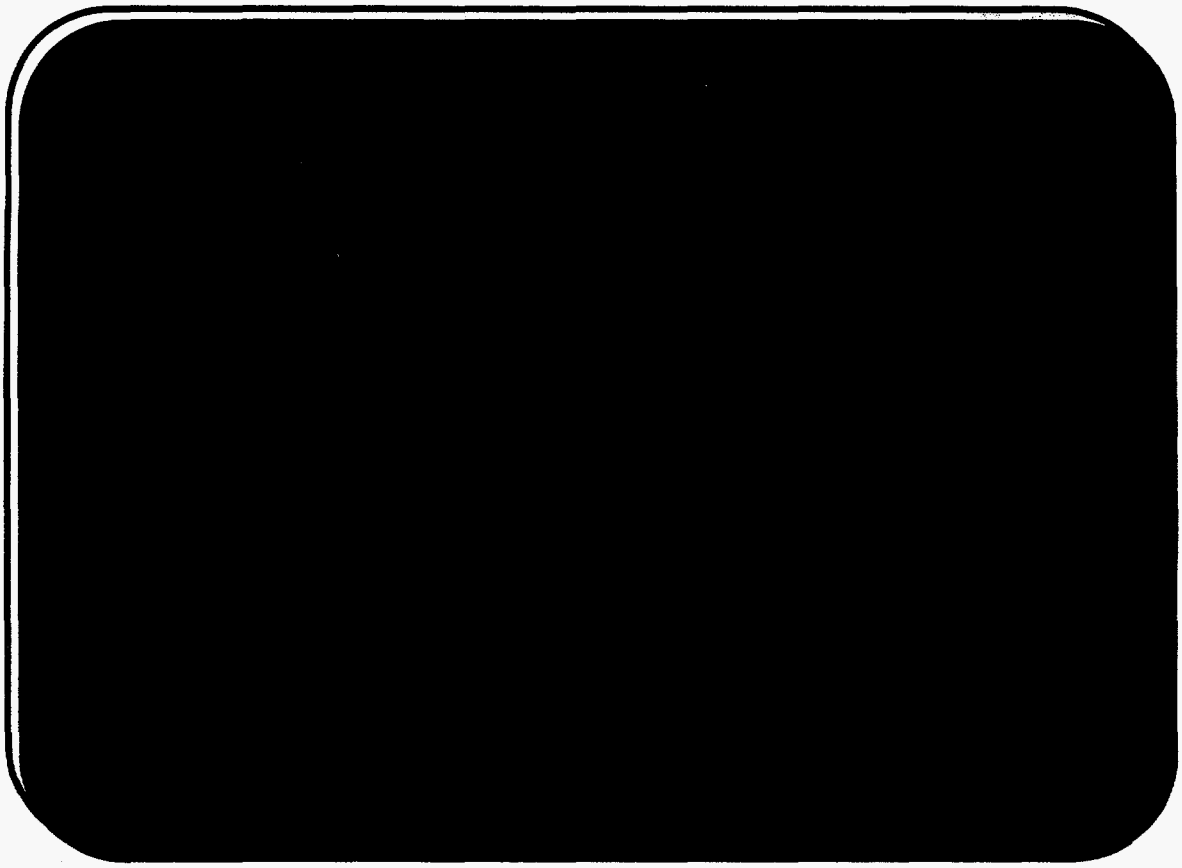
Alphabetic	___
Alphanumeric	___
Numeric	X

If Continuous:

Length: dependent

Range: -inf to inf

If Discrete	
Value	Meaning



Active

Short Description Determines if Platform is Active

Aliases (Contexts) Robot->PLAT.Active

Alphabetic	X
Alphanumeric	
Numeric	

If Continuous:

Length

Range

If Discrete

Value	Meaning
Y y	Platform is present
Any other value	No Platform present

Holonomic

Short Description Determines if given platform is Holonomic

Aliases (Contexts) Robot->PLAT.Holonomic

Alphabetic	X
Alphanumeric	
Numeric	

If Continuous:

Length

Range

If Discrete

Value	Meaning
Y y	Platform is Holonomic
Any Other Value	Patform is Non-Holonomic

Length

Short Description Length of Platform

Aliases (Contexts) Robot->PLAT.Length

Alphabetic	___
Alphanumeric	___
Numeric	X

If Continuous:

Length meters

Range 0 to size

If Discrete	
Value	Meaning

Width

Short Description Width of Platform

Aliases (Contexts) Robot->PLAT.Width

Alphabetic	___
Alphanumeric	___
Numeric	X

If Continuous:

Length meters

Range 0 to size

If Discrete	
Value	Meaning

Thick

Short Description Thickness of Platform

Aliases (Contexts) Robot->PLAT.Thick

Alphabetic	
Alphanumeric	
Numeric	X

If Continuous:

Length meters

Range 0 to size

If Discrete	
Value	Meaning

Z_OFF

Short Description Distance of first link from Manipulator

Aliases (Contexts) Robot->PLAT.Z_OFF

Alphabetic	
Alphanumeric	
Numeric	X

If Continuous:

Length meters

Range 0 to size

If Discrete	
Value	Meaning

L_OFF

Short Description Offset of arm from platform center along its Length

Aliases (Contexts) Robot->PLAT.L_OFF

Alphabetic	___
Alphanumeric	___
Numeric	X

If Continuous:

Length meters

Range 0 to Length/2

If Discrete	
Value	Meaning

ANG_OFF

Short Description Angle offset of first joint from platform

Aliases (Contexts) Robot->PLAT.ANG_OFF

Alphabetic	___
Alphanumeric	___
Numeric	X

If Continuous:

Length radians

Range -inf to inf

If Discrete	
Value	Meaning

M

Short Description: Size of Joint Space

Aliases (Contexts): M, FSP_Data->M, FSP_data->Mred

Alphabetic	___
Alphanumeric	___
Numeric	X

If Continuous:

Length

Range

If Discrete

Value	Meaning
0 - NL	Can be modified real-time to allow for dynamic removal of the platform, if the platform is the last three elements in Jacobian.

N

Short Description: Size of Task Space

Aliases (Contexts): Nred, FSP_data->N, FSP_data->Nred

Alphabetic	___
Alphanumeric	___
Numeric	X

If Continuous:

Length

Range

If Discrete

Value	Meaning
1-3	X,Y,Z
4-6	Orientation Control Yaw,Pitch,Roll Can be removed real time by reducing N to 3

cn

Short Description Contains number of constraints currently active

Aliases (Contexts) FSP_data->cn

Alphabetic	
Alphanumeric	
Numeric	X

If Continuous:

Length

Range

If Discrete	
Value	Meaning
1	One constraint
2	Two constraints
3	Three constraints
.	.
.	.
.	.
M-N+constant	Up to degrees of Redundancy

g

Short Description Contains all solution vectors for the system

Aliases (Contexts) FSP_data->g

Alphabetic	
Alphanumeric	
Numeric	X

If Continuous:

Length: -inf to inf

Range : -inf to inf

If Discrete	
Value	Meaning

Qarray

Short Description Contains current values of all joints

Aliases (Contexts) FSP_data->Qarray

Alphabetic	___
Alphanumeric	___
Numeric	X

If Continuous:

Length radians

Range Max_Limit to Min_Limit

If Discrete	
Value	Meaning

Betall

Short Description Contains all Betas currently active

Aliases (Contexts) FSP_data->Betas

Alphabetic	___
Alphanumeric	___
Numeric	X

If Continuous:

Length cn x span

Range -inf to inf

If Discrete	
Value	Meaning

PoorMansFile

Short Description: Global Variable, name of Joint_Values data store

Aliases (Contexts): CheezyFile

Alphabetic	
Alphanumeric	X
Numeric	

If Continuous:

Length

Range

If Discrete

Value	Meaning
Name of the system file that CheezyFile points to.	

Method

Short Description: Array of descriptions that decribe data element method.

Aliases (Contexts):

Alphabetic	
Alphanumeric	X
Numeric	

If Continuous:

Length

Range

If Discrete

Value	Meaning
	method has this value:
FSP & Joint Limits	0
PSEUDO INVERSE	1
FSP & Obstacles	2
FSP & JL & OA	3
FSP only	4

method

Short Description User's option of which constraints are to be used.

Aliases (Contexts)

Alphabetic	___
Alphanumeric	___
Numeric	X

If Continuous:

Length

Range

If Discrete

Value	Meaning
0	FSP LN with JL
1	PSEUDO INVERSE
2	FSP LN with OA
3	FSP LN w/ JL & OA
4	FSP LN only

ARM_file

Short Description Character string that contains system file name.

Aliases (Contexts)

Alphabetic	___
Alphanumeric	X
Numeric	___

If Continuous:

Length

Range

If Discrete

Value	Meaning
Robot.dat	This file is opened and read when initializing manipulator.

LL

Short Description Contains lengths to be used in GET_JACOB()

Aliases (Contexts)

Alphabetic	___
Alphanumeric	___
Numeric	X

If Continuous:

Length meter

Range 0 to real length

If Discrete	
Value	Meaning

Num_Of_Pts

Short Description Contains number of points in current trajectory

Aliases (Contexts)

Alphabetic	___
Alphanumeric	___
Numeric	X

If Continuous:

Length

Range

If Discrete	
Value	Meaning
-inf to 0	No Trajectory
1 to MAX_PTS	Number in Trajectory
> MAX_PTS	MAX_PTS in Trajectory

quit

Short Description: Describes Termination Status

Aliases (Contexts):

Alphabetic	___
Alphanumeric	___
Numeric	X

If Continuous:

Length

Range

If Discrete

Value	Meaning
0	OK, continue to next time step
-1	Exit, Partial Trajectory
-2	Exit, File Trouble

XTraj

Short Description: Holds Trajectory found in Trajectory data store

Aliases (Contexts):

Alphabetic	___
Alphanumeric	___
Numeric	X

If Continuous:

Length meters/ radians

Range -inf to inf

If Discrete

Value	Meaning

spheredata

Short Description 4x4 matrix containing position(x,y,z) and radius

Aliases (Contexts)

Alphabetic	___
Alphanumeric	___
Numeric	X

If Continuous:

Length meters

Range radius>0

If Discrete	
Value	Meaning

ns

Short Description Number of Spheres

Aliases (Contexts)

Alphabetic	___
Alphanumeric	___
Numeric	___

If Continuous:

Length

Range

If Discrete	
Value	Meaning
0	No spheres
1 - 4	One to Four Spheres
>4	Four Spheres

Jacob

Short Description Holds Jacobian Matrix size NxM

Aliases (Contexts) In FSP, it is called Matrix A

Alphabetic	___
Alphanumeric	___
Numeric	X

If Continuous:

Length radian*meters

Range -inf to inf

If Discrete	
Value	Meaning
Rows 0-2	X,Y,Z positional control
Rows 3-5	Orienataion Control part of Jacobian
Last 3 columns	If platform specified, these columns are interpreted as X,Y,Angle

dx

Short Description Holds desired Change in Position

Aliases (Contexts) In FSP, refered to as Matrix b

Alphabetic	___
Alphanumeric	___
Numeric	X

If Continuous:

Length meters/radians

Range -inf to inf

If Discrete	
Value	Meaning
Elements: 0 - 2	Change in X,Y,Z
3-5	Changes in Yaw, Pitch, Roll.

dq

Short Description Holds Change in Angles

Aliases (Contexts)

Alphabetic	___
Alphanumeric	___
Numeric	X

If Continuous:

Length radians

Range -inf to inf

If Discrete	
Value	Meaning

x_of_link

Short Description Holds position of begining and ending of all links.

Aliases (Contexts) Position

Alphabetic	___
Alphanumeric	___
Numeric	X

If Continuous:

Length meters

Range -inf to inf

If Discrete	
Value	Meaning

FSP_data

Short Description Solution Type, holds pertinent FSP information

Aliases (Contexts)

Alphabetic	
Alphanumeric	X
Numeric	

If Continuous:

Length

Range

If Discrete

Value	Meaning
cn <0..M-N+constant>	number of constraints
Betall	Beta vectors for all cn
g	Solution Vectors
Qarray	Current Position of all Joint Values

Appendix D : Code Listing

This appendix is subdivided into three sections. Each section involves different aspects of the software. Additionally, each section includes a breakdown of all the files present within the section as well as a brief description of its contents. At the beginning of each and every file listed inside a section is a header. If the listed file is a 'c' program, then the header will also include the names of all procedures listed in the file.

The first section is **IKOR Programs**. The first file listed is general.h. This file must be included in every 'c' file. It in turn, includes headers.h and structures.h. headers.h includes every function in the entire system as well as what 'c' file they belong to. This file can help the analyst find a procedure easily. The last header file displayed is Globals.h. This file contains the declaration of all global variables, and is included only once from IKOR.c. This section closes by listing the c files that comprise of IKOR.

The second section is **FSP Programs**. This section includes FSP.c that contains the solution generator, criteria.c that includes optimization criteria (only Least Norm currently), constraints.c that determines if a requested configuration is illegal, and lastly, analytical.c that moves the manipulator to a legitimate location.

The last section is **Manipulator Programs**. This section includes two example files for both the Kinematics file and the manipulator data file. Note that while both kinematics files include platform control, currently only the HERMIES' system contains a jacobian that provides for orientation control. Each example file is listed under the system's filename headings (that is when compiled, the example file Jacob_HERMIES.c must take the name of the system file Jacob_ROBOT.c).

IKOR Programs

The following files are listed in this section. Header files are listed first, then the 'c' files and finally utility programs. Although the files located in the section relating to Manipulator Programs are an integral part to IKOR, they have been separated for the mere purpose of readability and locatability.

HEADERS:

general.h	Macros for debugging and a plethora of system information.
headers.h	Contains headers for all functions.
structures.h	Holds all major data structures, FSP_data & Robot.
Globals.h	Most of the system's global variables.

CORE FILES:

short.c	Driver Routine for IKOR, provides command line operation.
IKOR.c	Main loop that calls FSP, obstacles, etc.. for every time step.
Manipulators.c	Reads manipulator datafile.

UTILITES:

useful_UTILS.c	Numerous oddball procedures, get spheres and more.
matrix.h	Matrix Routines description headers.
matrix.c	Matrix functions, pseudoinverse, svdcmp.
nrutil.c	'Numerical Recipes in C' base routines.

HEADERS

GENERAL.H

```
/*
   ┌──────────────────────────────────────────────────────┐
   │ Program Name:   general.h                            │
   ├──────────────────────────────────────────────────────┤
   │ description:  Contains defines that are used by many │
   │     of the modules in the IKOR system.  Every file that │
   │     gets compiled seperately must contain this file  │
   │     for adequate linkage to external variables.      │
   └──────────────────────────────────────────────────────┘ */

#ifndef _GENERAL_H
#define _GENERAL_H

#define DEBUG_OUT       /** uncomment this line when DEBUGGING  **/
/*#define FSP_DEBUG /** FSP run on Jacob,dx in FSP_data **/
/*#define TWOSTEP   /** uncomment when using two step    **/
/* The MACRO DEBUG was removed, use commandline to see output*/

#include <math.h>
#include <stdio.h>
#include "matrix.h"
#include "structures.h"

/*************************************************************
 *                                                           *
 *                      Global External References           *
 *                                                           *
 *************************************************************/

extern char PoorMansFile[40],                /** Globals.h **/
            Method[5][30],                   /** Globals.h **/
            ARM_file[15];                    /** Globals.h **/
                        /** Globals.h **/
extern MATRIX   *AW,*BW,*CW,*DW,*EW,*RW,*XW,
             *A,*B,*C,*D,*E,*R,*X,
             *PW1, *PW2, *PW3, *PW4, *EW1, *EW2, *EW3, *EW4,
             *P1,*P2,*P3,*P4,*E1,*E2,*E3,*E4;

extern int N, M, DEBUG;             /** Globals.h  **/
extern Manipulator_struct *Robot; /** Globals.h  **/
extern double *LL;              /** Globals.h  **/

/*************************************************************
 *                                                           *
 *                      Macro Definitions                    *
 *                                                           *
 *************************************************************/

#define MAX_PTS         3000 /** Sets Maximum Points in Trajectory
**/
#define VSpan           (M-N+1)
#define SPAN            (FSP_data->g->rows)
#define SPAN2           (FSP_data->g->rows - NumSpg)
#define SQUARE(a)       ((a)*(a));
#define rad(x)          ((x)*PI/180) /*Convert to Radians*/
#define deg(x)          ((x)*180/PI) /*Convert to Degrees*/
#define SWAP(a,b)       {double temp; temp=a;a=b;b=temp;}

#define ZERO            0.0
#define PI              3.14159265
#define     TRUE        1
#define     FALSE       0
#define OBSTACLES       2
#define JN_LIMITS       0
#define BOTH            3
#define COMPLETE        2
#define NOT_COMPLETE    -1
#define RESTRICTED      -2
#define SMALL     5.0e-05       /* user-defined value for FSP
*/
#define      K2_BND    1.0e+01    /* user-defined value for FSP
*/

/*************************************************************
 *                                                           *
 *                      FUNCTION PROTOTYPES                  *
 *  function prototypes for every function in headers.h)     *
 *                                                           *
 *************************************************************/
#include "headers.h"


#endif      /* end !_GENERAL_H */
```

HEADERS.H

```
/*
    ┌──────────────────────────────────────────────────────┐
    │ Program Name:   headers.h                            │
    ├──────────────────────────────────────────────────────┤
    │ description: Should contain headers for all functions│
    │    in the system referenced by the system file name  │
    │                                                      │
    └──────────────────────────────────────────────────────┘ */

/****************** IKOR_driver.c ************************/

int IKOR_driver (int IKMethod, int N_use, int M_use);
MATRIX *CALC_PSEUDO (MATRIX *Jacob, MATRIX *dx);
MATRIX *FSP (Solutions *FSP_data, MATRIX *Jacob, MATRIX *dx,
        int method, int ns, double spheredata[4][4], MATRIX
*x_of_link,
        FILE *datafp);
int OPEN_FILES  (double XTraj[MAX_PTS][6], char PoorMansFile[40],
        MATRIX *Qarray);
int Init_Globals();


/******************** Jacob_UTILS.c ***********************/
void   GET_JACOBIAN_ALTERED(MATRIX* Jacob, MATRIX *Qarray);
void   GET_JACOBIAN         (MATRIX* Jacob, MATRIX *Qarray);
void   ExtractRPY2 (MATRIX *T, MATRIX *x_of_link);
MATRIX *getT2      (double ZA,double YB,double XG,
                    double tx,double ty,double tz);


/******************** Jacob_ROBOT.c ***********************/
void   GET_JACOB     (MATRIX *Jacob,  MATRIX *Qarray);
void   GET_ACTUAL_X  (MATRIX *Qarray, MATRIX *x_of_link);


/********************** FSP.c *****************************/
int    Solution_generator (Solutions *FSP_data, MATRIX *Jacob,
                   MATRIX *dx, FILE *datafp);
void   RestofSoln (int Mred, int Nred, int NextToFind, MATRIX *block,
                   MATRIX *g, MATRIX *bred, MATRIX *Ared, int *Tackon,
                   int *FirstOK, FILE *check);
int    BLOCK_COL_FIND_X(int *Tackon, float *g, float *block,
                   MATRIX *b, MATRIX *A,
                   int Mred, int Nred, FILE *check);
int    ReduceA     (Solutions *FSP_data, MATRIX *Aorig, MATRIX *Ared,
                   MATRIX *borig, MATRIX *bred, MATRIX *Xelim,
                   int *ColElim, int *RowElim,
                   int *NumSpg, MATRIX *Specialg);
int    CheckB      (MATRIX *b, int m);
int    CheckRange  (MATRIX * b, MATRIX * Aorig, MATRIX * g,
                   int *RowElim, int Mred);
void   Rebuild_gs  (Solutions *FSP_data, int *ColElim,
                    MATRIX *Specialg, int NumSpg);
int    GetData     (MATRIX *A,  MATRIX *b);


/****************** useful_UTILS .c ***********************/
Solutions  *Solutions_init(int M, int N);
void   Solutions_free(Solutions *Temp );
int    spheres     (double spheredata[4][4], FILE *datafp);
void   fmat_pr     (FILE *checkfile, char *string, MATRIX *a);
void   fmat_prf    (FILE *checkfile, char *string, MATRIX *a);
void   fprint_norm (FILE *checkfile, MATRIX *dq);


/********************** constraints.c *********************/
int    avoid_limits    (Solutions *FSP_data, MATRIX *Jacob, MATRIX
*dx,
                   int *got_gs, FILE *datafp);
int    avoid_obstacles (Solutions *FSP_data, MATRIX *Jacob, MATRIX
*dx,
                   MATRIX *x_of_link, int ns, double spheredata[4][4],
                   int *got_gs, FILE *datafp);
int    find_intersection_sphere(double pt1[3], double pt2[3], int ws,
                   int ns, double spheredata[4][4], int *elbow_check,
                   double *newl, double *delta, double normal[3],
                   FILE *datafp);


/********************** criteria.c ************************/
int    Least_Norm (MATRIX *B, MATRIX *H, FILE *datafp);


/********************** analytical.c **********************/
MATRIX *Build_Grammian2 (Solutions *FSP_data, MATRIX *alphas, FILE
*datafp);
void    rank_lost_betas (Solutions *FSP_data, MATRIX *Jacob, MATRIX
*dx,
                   int *RowElim, FILE *datafp);
void    find_jl_beta    (Solutions *FSP_data, int chk,
                   double limit, FILE *datafp);
void    find_obs_beta   (Solutions *FSP_data, int chk,
                   double *newl, double delta, double normal[3],
                   FILE *datafp);
MATRIX *findt_with_Betas_Holonomic  (Solutions *FSP_data, MATRIX *B,
                   MATRIX *H, FILE *datafp);
MATRIX *findt_without_Betas_Holonomic  (Solutions *FSP_data, MATRIX
*B,
                   MATRIX *H, FILE *datafp);
```

STRUCTURES.H

```
/*
   |-------------------------------------------------------|
   | Program Name:   structures.h                          |
   |-------------------------------------------------------|
   | description: Contains data structures for system as   |
   |    well as a brief description of each. For more info |
   |    try the user's manual.                             |
   |-------------------------------------------------------| */
/* Angles is an array containing containing several instances*/
/* of the following structure: limits, home, weight, etc...  */
typedef struct {
  int   Prism;        /* ==Y then translates ==N then rotates */
  float Max_limit,    /* Max Limit for specific joint         */
        Min_limit,    /* Min Limit for specific joint         */
        Home;         /* Home position of joint               */
} ANGLE;

/* PLAT is an instance of the following structure which con- */
/* tains information about a specific platform.              */
typedef struct {
   int   Active;      /* Determines if Platform is active     */
   int   Holonomic;   /* ==Y Holonomic, ==N Automobile-like   */
   float Length,      /* Length of Platform                   */
         Width,       /* Width  of Platform                   */
         Thick,       /* Thickness of Platform                */
         Z_OFF,       /* offset of arm from platform Z        */
         L_OFF,       /* offset of arm from platform center   */
         ANG_OFF;     /* angle offset of first joint of Manip */
} Platform;

/* Robot is an instance of the following structure which con-*/
/*     tains information of the Manipulator.                 */
typedef struct {
   int   NA,            /* Maximum number of Angles        */
         NL;            /* Number of Links in Manipulator  */
   ANGLE     *Angles;   /* see Angles_struct above.        */
   Platform   PLAT;     /* see Platform_struct above.      */
   MATRIX    *Weights;  /* How active the joint is 0<x<= 1 */
   double    *LINKS;    /* length of the links of the robot*/
} Manipulator_struct;

/* FSP_data is an instance of the following structure type.  */
/* It contains info vital to FSP and IKOR's integration.     */
typedef struct {
    int   M, N,          /* (unused) may be used to replace globals */
          Mred,         /* number of columns in reduced Jacobian   */
          Nred,         /* number of rows    in reduced Jacobian   */
          cn;           /* number of constraints on system (betas) */
    MATRIX *g, /* array of solution vectors                  */
           *Qarray,     /* current angles positions                */
           *betall;     /* Contains All Betas (Beta all)           */
} Solutions;
```

GLOBALS.H

```
/*
    ┌──────────────────────────────────────────────────────┐
    │ Program Name:   Globals.h                            │
    ├──────────────────────────────────────────────────────┤
    │ description:  Contains Global variables. These should │
    │     be eliminated when ever the chance arrives. Good  │
    │     luck, because most of these variables are used by │
    │     CheezyIGRIP.c                                     │
    └──────────────────────────────────────────────────────┘ */

/* Globals N, and M can be removed since FSP_data contains them*/
/*     variables as well and contain correct values.  I have   */
/*     not had the time to convert all N,M to FSP_data->N ect. */
int     N,                      /*  Holds size of Task Space       */
        M,                      /*  Holds size of Joint Space      */
        DEBUG=0;                /* Tells system to print out info  */
FILE   *datafp,             /* datafile: Hardly used globally   */
       *CheezyFile;         /* angle output file: Hardly global */
char    PoorMansFile[40],   /*Name set OPEN_FILES()             */
        Method[5][30]={  "FSP & Joint Limits", "PSEUDO INVERSE",
              "FSP & Obstacles"   , "FSP & JL & OA ", "FSP only"  };

/* These globals are used in Init_Globals, Jacob_ROBOT.c, and  */
/* CheezyIGRIP.  They are fairly well imbedded into CheezyIGRIP*/
/*   , however not so much in Jacob_ROBOT.c                    */
MATRIX  *AW,*BW,*CW,*DW,*EW,*RW,*XW,
        *A,*B,*C,*D,*E,*R,*X,
        *PW1, *PW2, *PW3, *PW4, *EW1, *EW2, *EW3, *EW4,
        *P1,*P2,*P3,*P4,*E1,*E2,*E3,*E4;

char ARM_file[15] = "ROBOT.dat";/* File to read manipulator  */
Manipulator_struct *Robot;  /* Robot Information, see structures.h */
double   *LL;  /* get_jacobain uses values for obstacle avoid */
               /* these are values from the robot             */
```

CORE FILES

SHORT.C

```
/*
    ┌──────────────────────────────────────────────────────┐
    │ Program Name:   short.c                              │
    ├──────────────────────────────────────────────────────┤
    │ description: Launches system independently of other  │
    │              programs.                               │
    │                                                      │
    │ Procedures:                                          │
    │    main             :                                │
    └──────────────────────────────────────────────────────┘ */

#include "general.h"       /* Header File to link all others */

/*
    ┌──────────────────────────────────────────────────────┐
    │ name:     main                                       │
    │ description:  launches system with as little as poss-│
    │     possible.  If it is possible to launch with less,│
    │     make it so.  This is a short driver that enables │
    │     a command-line approach to the IKOR-FSP system.  │
    │     see help file for more information.              │
    │                                                      │
    │ inputs:  The file info which contains help.          │
    │ outputs: Calls IKOR_driver with proper parameters.   │
    ├──────────────────────────────────────────────────────┤
    │ Authors:     Date:     Modifications:                │
    │  c hacker     10/94    Created                       │
    └──────────────────────────────────────────────────────┘ */

void main (int argc, char **argv)
{
  int   i,          /* counter variable                          */
        IKMethod,   /* 0-LN&JL, 1-Pseudo, 2-LN&OB,               */
                    /* 3-LN&JL&OB, 4-LN ONLY                     */
        IKstatus,       /* return status from IKOR_driver        */
        ck1,            /* flag to see if joint space was given  */
        ck2,            /* flag to see if task space was given   */
        ck3;            /* flag to see if a method was given     */

  if  (argc%2 == 0)
      { fprintf(stderr,"Unmatched arguments! \n"); help(); }

  for (i=1; i<argc; i++)
  {
    switch (argv[i][0]) {
         case 'm':
         case 'M': ck1++; M = atoi(argv[i+1]); i++; break;
         case 'n':
         case 'N': ck2++; N = atoi(argv[i+1]); i++; break;
         case 't':
         case 'T': ck3++; IKMethod = atoi(argv[i+1]); i++; break;
         case 'D': DEBUG = atoi(argv[i+1]); i++; break;
      default : fprintf(stderr,"Unknown argument! \n"); help();
                break;
    }
  }
```

```
  if ((!ck1)||(!ck2)||(!ck3))
     { fprintf(stderr,"Argument Missing!\n"); help(); }

  Init_Globals(); /* Mostly for CheezyIGRIP    */

  IKstatus = IKOR_driver(IKMethod, N, M);  /* begin driver */

  switch (IKstatus) {
    case -2 :   fprintf(stderr,"\nSome kind of file error, sorry!\n");
                break;
    case -1 :   fprintf(stderr,"\nFile complete. Partial Traj
Only\n");
                break;
    default :   fprintf(stderr,"\nFile complete. Full Trajectory\n");
                break;

  } /* end switch */
} /* end short */


/*
    name:     help
    description:  prints the information file to stdout.

    inputs:  none.
    outputs: All text in file info displayed on stdou

    Authors:      Date:     Modifications:
     c hacker      10/94    Created
                                                              */
int help( void )
{
  FILE *temp;       /* used to point to help file             */
  char  one;        /* used to read help file                 */


  if ((temp = fopen("../IKOR/info", "r")) == NULL)
     fprintf(stderr, "Info File GONE. Use at own RISK!\n");
  else
     while (!feof(temp)) fputc(fgetc(temp), stderr);
  if (!temp) fclose (temp);

  exit(-1);
}
```

IKOR_DRIVER.C

```
/*
    Program Name:   IKOR_driver.c

    description:  Driver for FSP, Psuedo Inverse, Manipul-
        ator porting, obstacle detection, and utilities.
        IKOR_driver does not contain a main, instead it can
        be called by the main in short.c or CheezyIGRIP.c
        or any other.

    Procedures:
        IKOR_driver     :
        CALC_Pseudo     :
        OPEN_FILES      :
        init_globals    :
                                                              */

#include "general.h"        /* Generic Constants              */
#include "Globals.h"        /* Global Variables used by system */

/*
    name:     IKOR_driver
    description:  Drives entire system, although main is
        not here (see short.c or CheezyIGRIP.c for main).
        Initializes global variables, reads in manipulator
        trajectory files, calls Jacobian, FSP, and con-
        straint avoidance routines, if wanted. Also allows
        for Moore-Penrose Pseudo Inverse solutions, however
        the Pseudo is not equipped for constraint avoidance

    inputs:  N, M, method.  Manipulator, trajectory
    outputs: Alters two variables within FSP data.
        the beta vector for the joint is added to betall,
        and cn, the number of beta vectors present, is
        incremented.

    Authors:      Date:     Modifications:
     F Tulloch     4/94     Algorithm Creation for 2 g's
     c hacker     10/94     Ported from 2 g's to FSP
                   2/95     Ported to Solutions struct
                   3/95     Altered FSP, Pseudo calls
                                                              */
int IKOR_driver(IKMethod, N_use, M_use)
int IKMethod,       /* Psuedo-Inverse, or FSP (LN, jl, obs)  */
    M_use,          /* Size of Joint Space (!> num angles)   */
    N_use;          /* Size of Task Space (x,y,z,yw,ptch,rll)*/
{
  int    i, j, k,      /* Generic counters                    */
         ns,          /* Number of Spheres                   */
         STEP,  /* Current Step Along Trajectory  */
         Num_Of_Pts,    /* Number of points in Trajectory */
         quit = 0;      /* Exit_Status:  -2 File Trouble  */
                        /*               -1 Partial Traj  */
  double XTraj[MAX_PTS][6], /* End Effector (EE) Coordinates  */
         dPHI  ,  phi,      /* Euler Angles with deltas       */
```

```
      dTHETA,  theta,     /* Euler Angles with deltas       */
      dPSI   ,  psi,      /* Euler Angles with deltas       */
      spheredata[4][4];   /* Holds sphere info              */
MATRIX *Jacob,        /* Holds Forward Kinematic Deriv. */
       *dx,           /* Holds change in "EE"           */
       *dq,        /* Holds change in ANGLES         */
      *x_of_link;          /* Holds end points of each link  */
Solutions *FSP_data; /* see structure.h for explanation*/

/*-------------------------------------------------------------*
 * GLOBAL and LOCAL INITIALIZATIONS  (Order of decls important) *
 *-------------------------------------------------------------*/
N=N_use;
M=M_use;
dx    = mat_malloc(N, 1);      /* memory for change in position  */
Jacob = mat_malloc(N,M);       /* memory for Forw.Kin. derivatve */
                               /* memory for x,y,z of each link  */
x_of_link=mat_malloc(Robot->NL+1,N);
FSP_data=Solutions_init(M,N); /* memory for solutions,betas, etc*/
                              /* initaillize trajectory, data... */
if (!(Num_Of_Pts=OPEN_FILES(XTraj, PoorMansFile, FSP_data->Qarray)))
      return (-2);
ns=spheres(spheredata,datafp);/* initialize obstacles in system */

/* correct size of the Weight MATRIX, see structures.h,        */
Robot->Weights->rows = Robot->Weights->cols = M;

/*-------------------------------------------------------------*
 * Uncomment, to trace bus errors or segmentation faults       *
 *-------------------------------------------------------------*/
/*datafp = stderr;              /* When in trouble, uncomment     */


/*-------------------------------------------------------------*
 *       HEADER INFORMATION for Standard Output                *
 *-------------------------------------------------------------*/

#ifdef DEBUG_OUT
  fprintf (stderr, "\n\n\n"
                   "*********************************************\n"
                   "  INVERSE KINEMATIC AND REDUNDANCY RESOLVER  \n"
                   "  For HERMIES CESARm, %d D.O.F, %d D.O.R.    \n"
                   "*********************************************",
                     M,M-N);
  fprintf (stderr, "\n\n\n"
                   "      < NUMBER OF POINTS IN TRAJECTORY >  %d  \n"
                   "---- Current Step for %s Method ----      \n"
                   "         << Time Step >>                  \n"
                   "         ---------------------            \n"
                   "\n<< -1>>", Num_Of_Pts-2, Method[IKMethod]);
#endif

/*-------------------------------------------------------------*
 *                          MAIN-LOOP                          *
 *                                                             *
 *    Cycles through all points on trajectory until done doing *
 *    Least Norm  and possibly Joint and Obstacle Avoidance until *
 *    an unavoidable problem occurs.                           *
 *-------------------------------------------------------------*/
for (STEP = 0; STEP < Num_Of_Pts-2 ; STEP++)
{

  /*-----------------------------------------------------------*
   * Print Angles for simulation programs (IGRIP, Cheeze)  *
   *    and print step number to the data and stderr files *
   *-----------------------------------------------------------*/

  for (i=0;i< Robot->NA;i++)
    if (i<M)
      fprintf (CheezyFile, "%f ", FSP_data->Qarray->p[i][0]);
    else fprintf (CheezyFile, "0.0 ");
  fprintf (CheezyFile,"\n");
  #ifdef DEBUG_OUT
    fprintf (stderr, "\b\b\b\b\b\b\b\b\b<<%3d>>", STEP+1);
  #endif
  if (DEBUG)
    fprintf (datafp, "\n******** STEP <%d> **********\n", STEP+1);

  /*-----------------------------------------------------------*
   * Based on Angles, Calculates Position of End Effector  *
   *-----------------------------------------------------------*/
  #ifndef FSP_DEBUG
    GET_JACOBIAN(Jacob, FSP_data->Qarray);
    GET_ACTUAL_X(FSP_data->Qarray, x_of_link);
    if (DEBUG) {
      fmat_pr(datafp,"Current Position", x_of_link);
      fprintf(datafp,"\nDestination Position\n");
    }

    /*----------------------GET DX'S-----------------------*
     *  Based on actual position, Calculates Change in X nec *
     *  This feedback loop is necessary because this system  *
     *  is highly nonlinear                                  *
     *-----------------------------------------------------------*/

    for (i = 0; i <  N; i++)
    {
      if (DEBUG) {
        fprintf(datafp,"Element [%d] : %f\n", i, XTraj[STEP+1][i]);
      }
     dx -> p[i][0] = XTraj[STEP + 1][i] -
x_of_link->p[Robot->NL][i];
      if (fabs(dx -> p[i][0]) < SMALL) dx -> p[i][0] = ZERO;
    }

    /*-----------------------------------------------------------*
     * CONVERT INCREMENTAL EULER ANGLES INTO INCREMENTAL      *
     * ANGLES ABOUT EACH AXIS OF BASE FRAME                   *
     *-----------------------------------------------------------*/

    if (N==6)
    {
      phi    = x_of_link->p[Robot->NL][3];     dPHI   = dx->p[3][0];
      theta  = x_of_link->p[Robot->NL][4];     dTHETA = dx->p[4][0];
      psi    = x_of_link->p[Robot->NL][5];     dPSI   = dx->p[5][0];

       /* wx *
      dx -> p[3][0] = dPHI*cos(theta)*cos(psi) + dTHETA*sin(psi);/
      /* wy */
```

```
      dx -> p[4][0] =-dPHI*sin(theta)          + dPSI;
      /* wz */
      dx -> p[5][0] =-dPHI*cos(theta)*sin(psi) + dTHETA*cos(psi);
    }
  #else
    GetData(Jacob, dx);
    STEP = Num_Of_Pts;
  #endif

  /*-------------------------------------------------------*
   * CALCULATE dq'S USING ALGORITHM Selected by IKMethod   *
   *-------------------------------------------------------*/

  FSP_data->cn = 0; /* intialize num of constraints for STEP */
  if (IKMethod != 1)
    dq = FSP (FSP_data, Jacob, dx, IKMethod,
              ns, spheredata, x_of_link, datafp);
  else
    dq = CALC_PSEUDO (Jacob, dx);
  if (dq== (MATRIX *) -1) break;

  /*-------------------------------------------------------*
   *   Now add the necessary changes in angles to the current  *
   *   Angles and keep values withing 4 PI                 *
   *-------------------------------------------------------*/

  for (i = 0; i < M; i++)
  {
    FSP_data->Qarray->p[i][0] += dq -> p[i][0];
    if ((Robot->Angles[i].Prism == 'N') ||
        (Robot->Angles[i].Prism == 'n'))
    {
      if (FSP_data->Qarray->p[i][0] >  2*PI)
          FSP_data->Qarray->p[i][0] -= 2*PI;
      if (FSP_data->Qarray->p[i][0] < -2*PI)
          FSP_data->Qarray->p[i][0] += 2*PI;
    }       /* end if ! prism */
  }  /* end for i=1 to M */

  mat_free(dq);           /* free up dq for next step or end  */

  if (quit < 0) break;    /* If unavoidable trouble quit      */

}  /* END MAIN FOR LOOP*/



/*-------------------------------------------------------*
 *                        FINAL SUMMARY                  *
 *-------------------------------------------------------*/
fprintf (datafp, "\n\n ----FINAL SUMMARY------\n"
                 " --- Exit Status: %d ---\n"
                 " Errors in final point:  \n", quit);

fprintf (datafp, "\nStarting Coordinates:\n");
for (i = 0; i < N; i++)
  fprintf (datafp, "Element [%d] >%f\n", i, XTraj[0][i]);

fprintf (datafp, "\nRequested Ending Coordinates\n");
for (i = 0; i < N; i++)
   fprintf (datafp, "Element [%d] >%f\n", i, XTraj[Num_Of_Pts-1][i]);

  fmat_pr (datafp, "Actual Ending Coordinates:", x_of_link);

  #ifdef DEBUG_OUT
    fprintf (stderr,"\n DATA FILE %s HAS BEEN CREATED AND CONTAINS
THE"
                    "\n ANGLES FOR THE MANIPULATOR FOR EACH TIME STEP"
                    "\n EXAMINE, OR RUN IT THROUGH A SIMULATOR.\n",
                       PoorMansFile);
  #endif

  /*-------------------------------------------------------------*
   * Clean up Global variables and Close Files                   *
   *-------------------------------------------------------------*/
  fclose(CheezyFile);
  fclose(datafp);

  Solutions_free(FSP_data);
  mat_free(x_of_link);
  mat_free(Jacob);
  mat_free(dx);
  free (LL);

  return (quit);
}                          /* END IKOR_driver */


/*
  +-------------------------------------------------------------+
  | name:     CALC_PSEUDO                                       |
  | description: Calculate Change in Angles using Pseudo        |
  |     Inverse. The procedure prepares the Jacobian and        |
  |     calls the routine mat_pseudoinv() found in the          |
  |     file matrix.c                                           |
  |                                                             |
  | inputs:  Jacobian and change in x                           |
  | outputs: Returns dq necessary to complete motion.           |
  +-------------------------------------------------------------+
  | Authors:     Date:     Modifications:                       |
  |  F Tulloch    4/94     Algorithm Creation for 2 g's         |
  |  c hacker     3/95     rearranged calling structure         |
  |               3/95     took struct from "struct MATRIX"     |
  +-------------------------------------------------------------+ */

MATRIX *CALC_PSEUDO (MATRIX *Jacob, MATRIX *dx)
{
  int    i, j;       /* counter variables                 */
  MATRIX *Jacobinv,     /* Holds Jacobian Psuedo-inverse  */
         *dq;       /* Holds delta angle changes needed */

  Jacobinv = mat_malloc(M,M);

  mat_cp(Jacob, Jacobinv);

  for (i=0;i<M;i++)       /* Must be padded with zeros        */
    for (j=N; j< M; j++)
      Jacobinv->p[j][i] = 0.0;

  mat_pseudoinv(Jacobinv); /* JacobTemp holds jacob's inverse */
```

```
  Jacobinv->cols = N;
  dq = mat_mul2(Jacobinv, dx);
  Jacobinv->cols = M;

  if (DEBUG) {
     fmat_pr(datafp,"*** Jacobian ***",  Jacob);
     fmat_pr(datafp,"***     dx    ***",     dx);
     fmat_pr(datafp,"***     dq    ***",     dq);
  }

  mat_free(Jacobinv);
  return (dq);
}
```

```
 name:    FSP
 description:  This procedure calculates the changes in
     angles via the new algorithm (FSP). It also calls
     routines to check for obstacles, limits, ect...
     Finally, a procedure to find a particular para-
     metric solution based on constraints, criteria.
     The reason for several find_t functions is due to
     speed considerations, so that, the find_t that gets
     called should be the fastest.

 inputs:  Jacobian, change in x, IKmethod, sphere data,
     and position.
 outputs: Returns dq necessary to complete motion.

 Authors:     Date:     Modifications:
  F Tulloch    4/94     Algorithm Creation for 2 g's
  c hacker     3/95     rearranged calling structure
               3/95     took struct from "struct MATRIX"
               3/95     converted to ONE STEP Method
*/
MATRIX *FSP(FSP_data, Jacob, dx, method, ns, spheredata, x_of_link,
            datafp )
FILE   *datafp;              /* data file declaration          */
int     method;              /* 0-JL,2-OBS,3-BOTH              */
int     ns;                  /* sphere_data (# of spheres)     */
double  spheredata[4][4];    /* sphere_data (rad,x,y,z)        */
MATRIX *dx,                  /* change in EE position          */
       *Jacob,          /* derivative of Forward Kinematic*/
       *x_of_link;      /* position of all links          */
Solutions *FSP_data;    /* see structures.h               */
{
  int     i, j, k,      /* counter variables                 */
          got_gs=TRUE,  /* for TWOSTEP, if g's generated     */
          Terminate;
  MATRIX *dq,           /* final solution for jnt space move */
         *B,
         *H;

  B = mat_malloc (  M,  1 );
  H = mat_malloc (  M,  M );

  Terminate = Solution_generator(FSP_data, Jacob, dx, datafp);
  Least_Norm (B, H, datafp);

  if (Terminate != NOT_COMPLETE)
  {
    #ifndef TWOSTEP
      if ((method == OBSTACLES) || (method == BOTH))
        avoid_obstacles (FSP_data, Jacob, dx, x_of_link,
                         ns, spheredata, &got_gs, datafp);
      if ((method == JN_LIMITS) || (method == BOTH))
        avoid_limits     (FSP_data, Jacob, dx, &got_gs, datafp);
    #endif

    if (Robot->PLAT.Holonomic)
      if (FSP_data->cn)
        dq = findt_with_Betas_Holonomic(FSP_data, B, H, datafp);
      else
        dq = findt_without_Betas_Holonomic(FSP_data, B, H, datafp);
    else {;}

    #ifdef TWOSTEP

      /*************************************************************
       *   Now add the necessary changes in angles to the current  *
       *   Angles and keep values withing 4 PI                     *
       *************************************************************/
      got_gs = FALSE;

      for (i = 0; i < M; i++)
      {
        FSP_data->Qarray->p[i][0] += dq -> p[i][0];

       dq -> p[i][0] = ZERO;  /* Reset dq's if no constraints, */

        if ((Robot->Angles[i].Prism == 'N') ||
            (Robot->Angles[i].Prism == 'n'))
        {
          if (FSP_data->Qarray->p[i][0] >  2*PI)
              FSP_data->Qarray->p[i][0] -= 2*PI;
          if (FSP_data->Qarray->p[i][0] < -2*PI)
              FSP_data->Qarray->p[i][0] += 2*PI;
        }              /* end if ! prism */
      }  /* end for i=1 to M */

      /*  for (i=0;i<N;i++) dx->p[i][0] = 10;*/

      if (( method == OBSTACLES ) || ( method == BOTH ))
        avoid_obstacles (FSP_data, Jacob, dx, x_of_link,
                         ns, spheredata, &got_gs, datafp);
      if ((method == JN_LIMITS) || (method == BOTH))
        avoid_limits     (FSP_data, Jacob, dx, &got_gs, datafp);

      if (got_gs)
      {
        mat_free(dq);   /* dq has been allocated already by findt */
        dq = findt_with_Betas_Holonomic(FSP_data, B, H, datafp);
      }
    #endif
```

```
  } /* end if != NOT_COMPLETE (This is a correct Double-negative 8^)
*/
  else fprintf(stderr, "\n FSP didn't complete!\n") ;

  if (DEBUG) {
    fmat_pr(datafp,"Q's after Least Norm (Q + dq)",FSP_data->Qarray);
    fmat_pr(datafp,"Position",x_of_link);
  }               /* if DEBUGGING print out  (see users guide) */

  mat_free (B);
  mat_free (H);

  return(dq);
}/*END ROUTINE*/



/*
  |-------------------------------------------------------------------|
  | name:    OPEN_FILES                                               |
  | description: This procedure opens most of the system              |
  |     files (spheredata is in spheres() in useful_UTILS.c           |
  |     Also the initial condition to the differential                |
  |     equation (notably Qarray, initial Angles) is def-             |
  |     ined, and a trajectory file is read into XTraj.               |
  |                                                                   |
  | inputs:  XTraj, angle output file, Qarray                         |
  | outputs: XTraj contains trajectory, Qarray has initial            |
  |     angle values.                                                 |
  |-------------------------------------------------------------------|
  | Authors:     Date:     Modifications:                             |
  |  F Tulloch    2/92     Who knows who started this one             |
  |  c hacker    10/94     restructured, removed globals              |
  |-------------------------------------------------------------------|
*/
int OPEN_FILES(XTraj, PoorMansFile, Qarray)
char    PoorMansFile[40];
double  XTraj[MAX_PTS][6];
MATRIX *Qarray;
{
  FILE  *XYZfile, *EAfile;
  int    i, Num_Of_Pts, flag = -1;
  char   tempname[20];
  float  temp_Q;

  /*********************FILE INITIALIZATION*************************/
  #ifdef DEBUG_OUT
    fprintf (stderr, "\nANGLES PRINTED TO CheezyAngles.\n");
  #endif
  strcpy (PoorMansFile, "CheezyAngles");

  if ((XYZfile    = fopen("TrajectoryPts","r")) == NULL) flag = 0;
  if (N>3)
  if ((EAfile     = fopen("EulerAngles",  "r")) == NULL) flag = 1;
  if ((CheezyFile = fopen(PoorMansFile,   "w")) == NULL) flag = 2;
  if ((datafp     = fopen("data",         "w")) == NULL) flag = 3;

  switch (flag) {
    case 0:  fprintf (stderr, "'TrajectoryPts' doesn't exist,\n");
               return 0;
    case 1:  fprintf (stderr, "'EulerAngles'   doesn't exist,\n");
               return 0;
    case 2:  fprintf (stderr, "'%s' not Created.\n",PoorMansFile);
               return 0;
    case 3:  fprintf (stderr, "Couldn't open file:  data \n"    );
               return 0;
    default: fprintf (datafp, "Traj..., Euler..., %s, and "
                              "data Opened\n", PoorMansFile);
  }
  /*-------------------------------------------------------------*/
  /* READ TRAJECTORY INTO ARRAY XTraj, MAX 3000 POINTS (makepts.h) */
  /*-------------------------------------------------------------*/
  fscanf(XYZfile, "%d", &Num_Of_Pts);
  if (Num_Of_Pts > MAX_PTS) Num_Of_Pts = MAX_PTS;

  /*-------------------------------------------------------------*/
  /* Print first line of Q angles at t=0                         */
  /* terminated with a '\n'.                                     */
  /*-------------------------------------------------------------*/
  for (i = 0; i < Robot->NA; i++)
  {
    fscanf ( XYZfile, "%f", &temp_Q);
    if (i < M) Qarray->p[i][0] = rad(temp_Q);
  }
  fmat_pr (datafp, "Initial Q Array", Qarray);

  for (i = 0; i < Num_Of_Pts; i++)
  {
    fscanf(XYZfile, "%lf %lf %lf",
                      &XTraj[i][0], &XTraj[i][1], &XTraj[i][2]);
    if (N>3)
    fscanf(EAfile,  "%lf %lf %lf",
                      &XTraj[i][3], &XTraj[i][4], &XTraj[i][5]);
  }
  fclose(XYZfile);
  if (N>3)
  fclose(EAfile);

  /*-------------------------------------------------------*/
  /* Print Header Line For IGRIP Simulation Angles File    */
  /* Which is in turn read in through Dave Reister's file  */
  /* which puts the points to the IGRIP model every so many */
  /* seconds to view the motion.                           */
  /*-------------------------------------------------------*/

  fprintf (CheezyFile, "Angles For IGRIP to read to animate arm.\n");

  fflush (datafp);
  return (Num_Of_Pts);
}                       /* END OPEN_FILES */
```

```
/*
   ┌──────────────────────────────────────────────────────────────────┐
   │ name:    init_Globals                                            │
   │ description: This, I'm not too proud of.  In order to            │
   │     CheezyIGRIP work with different manipulators I had           │
   │     to incorporate this, but unfortuneately CheezyIGRIP          │
   │     is indeed Cheezy in many ways, one being its thirst          │
   │     for global variables.  A,B,C,D,E,R,X and others              │
   │     are needed to be global for the forward kinematic            │
   │     which CheezyIGRIP uses.  Someone should eventually           │
   │     remove these globals into a structure before it              │
   │     gets even more out of hand, but alas... tis not I.           │
   │ inputs:  none                                                    │
   │ outputs: As named, also initializes manipulator                  │
   ├──────────────────────────────────────────────────────────────────┤
   │ Authors:      Date:     Modifications:                           │
   │  c hacker      3/95     created sadly                            │
   └──────────────────────────────────────────────────────────────────┘ */

int Init_Globals()
{
  int i;
  float HandWidth   =0.05,
        HandLength  =0.05,
        FrontLength =1.22,
        SideLength  =1.52;



/*-------------------------------------------------------------------*/
  /* READ Users Manipulator's Parameters into System
(Manipulators.c)*/

/*-------------------------------------------------------------------*/
    init_ARM();       /* Initialize User_Selected Arm     */

    if (M > Robot->NA) {
      fprintf(stderr,"\n%s%d  %s%d \n%s\n",
                     "Requested M: ", M,
                     "larger than the number of joints: ", Robot->NA,
                     "exiting to system....\n");
      sleep(5); exit(0);
    }

  /* E1-E4 are vectors to locate the 4 points to draw the EE */

  E1 = mat_malloc(4,1);
  E2 = mat_malloc(4,1);
  E3 = mat_malloc(4,1);
  E4 = mat_malloc(4,1);

  /* EW1-EW4 are locations of 4 points of EE wrt world coordinates */
  EW1 = mat_malloc(4,1);
  EW2 = mat_malloc(4,1);
  EW3 = mat_malloc(4,1);
  EW4 = mat_malloc(4,1);

  /* P1-P4 are vectors to locate 4 points used to draw the platform */
  P1 = mat_malloc(4,1);
  P2 = mat_malloc(4,1);
  P3 = mat_malloc(4,1);
  P4 = mat_malloc(4,1);

  /* Also see Jacob_ROBOT.c for the use of A... */
  A = mat_malloc(4,1);      /* vector to locate end of link1 */
  B = mat_malloc(4,1);      /* vector to locate end of link2 */
  C = mat_malloc(4,1);      /* vector to locate end of link3 */
  D = mat_malloc(4,1);      /* vector to locate end of link4 */
  E = mat_malloc(4,1);      /* vector to locate end of link5 */
  R = mat_malloc(4,1);      /* vector to locate base of manipulator */
  X = mat_malloc(4,1);      /* vector to locate manip on platform */

  AW = mat_malloc(4,1); /* location of end of link1 wrt world coords*/
  BW = mat_malloc(4,1); /* location of end of link2 wrt world coords*/
  CW = mat_malloc(4,1); /* location of end of link3 wrt world coords*/
  DW = mat_malloc(4,1); /* location of end of link4 wrt world coords*/
  EW = mat_malloc(4,1); /* location of end of link5 wrt world coords*/
  RW = mat_malloc(4,1); /* location of manip base wrt world coords  */
  XW = mat_malloc(4,1); /* location of manip on plat wrt worldcoords*/


    /* initialize all location vectors and transformation matrices */
  for (i=0;i<=3;i++)
  {
    A->p[i][0]=0;  B->p[i][0]=0;  C->p[i][0]=0;  D->p[i][0]=0;
    E->p[i][0]=0;  R->p[i][0]=0;  X->p[i][0]=0;

    P1->p[i][0]=0;P2->p[i][0]=0;P3->p[i][0]=0;P4->p[i][0]=0;
    E1->p[i][0]=0;E2->p[i][0]=0;E3->p[i][0]=0;E4->p[i][0]=0;
  }

   /* drawing vectors for the manipulator */
  E1->p[0][0]=Robot->LINKS[4]+HandLength; E1->p[2][0]=-HandWidth;
                                          E1->p[3][0]=1.0;
  E2->p[0][0]=Robot->LINKS[4]           ; E2->p[2][0]=-HandWidth;
                                          E2->p[3][0]=1.0;
  E3->p[0][0]=Robot->LINKS[4]           ; E3->p[2][0]= HandWidth;
                                          E3->p[3][0]=1.0;
  E4->p[0][0]=Robot->LINKS[4]+HandLength; E4->p[2][0]= HandWidth;
                                          E4->p[3][0]=1.0;

   /* drawing vectors for the platform */
  P1->p[0][0]=-FrontLength/2; P1->p[1][0]=-SideLength/2;
  P1->p[3][0]=1.0;
  P2->p[0][0]=-FrontLength/2; P2->p[1][0]= SideLength/2;
  P2->p[3][0]=1.0;
  P3->p[0][0]= FrontLength/2; P3->p[1][0]= SideLength/2;
  P3->p[3][0]=1.0;
  P4->p[0][0]= FrontLength/2; P4->p[1][0]=-SideLength/2;
  P4->p[3][0]=1.0;

  /* manip base offset is drawn along a local y axis */
  R->p[2][0]=0; R->p[3][0]=1.0;
  X->p[2][0]=0; X->p[3][0]=1.0;
}
```

MANIPULATORS.C

```
/*
   ┌──────────────────────────────────────────────────────────┐
   │ Program Name:   Manipulators.c                           │
   ├──────────────────────────────────────────────────────────┤
   │ description: Integrates different manipulator's into     │
   │    the system by reading ARM_file, "ROBOT.dat". This     │
   │    allows a data file, such as ROBOT_AIRARM.dat, to be   │
   │    copied into ROBOT.dat and the system will now emul-   │
   │    the new manipulator. See User's Guide or sample a     │
   │    sample data file for creation of a new manipulator.   │
   │                                                          │
   │ Procedures:                                              │
   │    go_next          :                                    │
   │    rm_blanks        :                                    │
   │    init_arm         :                                    │
   └──────────────────────────────────────────────────────────┘ */

#include "general.h"

/*
   ┌──────────────────────────────────────────────────────────┐
   │ name:              go_next                               │
   │ description:  Finds an Exclamation Mark in requested     │
   │     file and then continues to read the rest of the      │
   │     line containing that delimiter.  Prepares file       │
   │     read pointer to get input on the line immediately    │
   │     following the excamation mark.                       │
   │           33 is the exclamation '!' character            │
   │           10 is the newline     '\n' character           │
   │                                                          │
   │ inputs:  File to find the delimiter                      │
   │ outputs: File read pointer points to beginning of next   │
   │     line.                                                │
   ├──────────────────────────────────────────────────────────┤
   │ Authors:     Date:     Modifications:                    │
   │ c hacker      3/95     Created                           │
   └──────────────────────────────────────────────────────────┘ */
int go_next(FILE *ARM)
{
  char CHAR = 'x';

  while ((!feof(ARM)) && (CHAR != 33)) CHAR = fgetc(ARM);
  if (!feof(ARM))
    while ((!feof(ARM)) && (CHAR != 10)) CHAR = fgetc(ARM);

  if (feof(ARM))
  { fprintf(stderr,"\nManipulator File Integrity Lost"); exit(0); }
}

/*
   ┌──────────────────────────────────────────────────────────┐
   │ name:              rm_blanks                             │
   │ description:  Reads past all blanks and tabls in a       │
   │     line and returns the first character found.          │
   │                                                          │
   │ inputs:  File to find the character                      │
   │ outputs: Returns first non blank or non tab character    │
   ├──────────────────────────────────────────────────────────┤
   │ Authors:     Date:     Modifications:                    │
   │ c hacker      3/95     Created                           │
   └──────────────────────────────────────────────────────────┘ */
int rm_blanks(FILE *ARM)
{
  char CHAR=' ';

  while ((!feof(ARM)) && ((CHAR == 32) || (CHAR == 9))) CHAR =
fgetc(ARM);
  if (feof(ARM))
  { fprintf(stderr,"\nManipulator File Integrity Lost"); exit(0); }

  return CHAR;
}


/*
   ┌──────────────────────────────────────────────────────────┐
   │ name:              init_ARM                              │
   │ description:  Reads file determined by global character  │
   │     constant defined in Globals.h, and then builds a     │
   │     manipulator based on that data.                      │
   │                                                          │
   │ inputs:  char constant ARM_file                          │
   │ outputs: Global variables PLAT, Angles, LINKS, and LL    │
   │     are built.                                           │
   ├──────────────────────────────────────────────────────────┤
   │ Authors:     Date:     Modifications:                    │
   │ c hacker      3/95     Created                           │
   └──────────────────────────────────────────────────────────┘ */
init_ARM()
{
  int   i;       /* Loop counter variable                    */
  FILE *ARM;     /* File pointer that will point to ARM_file */
  double temp_f; /* Temporary float variable for reading     */
  int    temp_i; /* Temporary int   variable for reading     */

  if (( ARM = fopen(ARM_file, "r") )==NULL) {
    fprintf(stderr, "\nCouldn't Open Manipulator Description File");
    exit(0);
  }
  Robot = (Manipulator_struct *) malloc ( sizeof(Manipulator_struct));

  go_next(ARM);          /* find next exclamation mark       */
  fscanf(ARM,"%d",&temp_i);
  Robot->NL = temp_i;    /* Read in Number of Links          */
                         /* Allocate memory for LINKS and LL */
  Robot->LINKS = (double *) malloc( Robot->NL * sizeof(double) );
  LL           = (double *) malloc( Robot->NL * sizeof(double) );
  go_next(ARM);          /* find next exclamation mark       */
                         /* Assign Link Lengths to LINKS     */
  for (i=0; i<Robot->NL; i++)
  {
    fscanf(ARM,"%lf", &temp_f);
    Robot->LINKS[i] = temp_f;
  }

  go_next(ARM);          /* find next exclamation mark       */
  fscanf(ARM,"%d",&temp_i);
```

```
  Robot->NA = temp_i;    /* Read in Number of Angles          */
                         /* Allocate memory for Angles struct*/
  Robot->Angles = (ANGLE *) malloc (Robot->NA * sizeof(ANGLE));
  Robot->Weights= mat_malloc (Robot->NA, Robot->NA);
  go_next(ARM);          /* find next exclamation mark        */
                         /* Assign Angle info to Angles       */
  for (i=0; i<Robot->NA; i++) {
    fscanf(ARM,"%f", &(Robot->Angles[i].Max_limit));
    fscanf(ARM,"%f", &(Robot->Angles[i].Min_limit));
    Robot->Angles[i].Prism = rm_blanks(ARM);
    fscanf(ARM,"%f", &(Robot->Weights->p[i][i]));
    fscanf(ARM,"%f", &(Robot->Angles[i].Home));
  }

  go_next(ARM);          /* find next exclamation mark        */
                         /* Determine if Robot has platform   */
  Robot->PLAT.Active = rm_blanks(ARM);
  go_next(ARM);          /* find next exclamation mark        */
                         /* If Platform exists, Assing info   */
  if (Robot->PLAT.Active=='Y') {
     fscanf( ARM,"%f", &( Robot->PLAT.Length  ) );
     fscanf( ARM,"%f", &( Robot->PLAT.Width   ) );
     fscanf( ARM,"%f", &( Robot->PLAT.Thick   ) );
     go_next(ARM);
     fscanf( ARM,"%f", &( Robot->PLAT.Z_OFF   ) );
     fscanf( ARM,"%f", &( Robot->PLAT.L_OFF   ) );
     fscanf( ARM,"%f", &( Robot->PLAT.ANG_OFF ) );
  }
  Robot->PLAT.Holonomic = 1; /*Currently all plats assumed holonomic*/
  /*********************************************************
   *                                                       *
   *  Print out all data attained from Manipulator file    *
   *                                                       *
   *********************************************************/
  #ifndef FSP_DEBUG
    if (DEBUG) {
    fprintf(stderr, "\n NL: %d", Robot->NL);
    for (i=0; i<Robot->NL; i++)
      fprintf(stderr, "\n LINK[%d]: %f", i, Robot->LINKS[i]);
    fprintf(stderr, "\n NA: %d", Robot->NA);
    for (i=0; i<Robot->NA; i++)
      fprintf(stderr, "\n%3d %s %s %9.2f %s %9.2f %s %c %s %f %s %f",
        i, ":",
        "Max:"   ,Robot->Angles[i].Max_limit,
        "Min:"   ,Robot->Angles[i].Min_limit,
        "Prism:" ,Robot->Angles[i].Prism    ,
        "Weight:",Robot->Weights->p[i][i]   ,
        "Home:"  ,Robot->Angles[i].Home      );
    fprintf(stderr, "\n PLAT: %c", Robot->PLAT.Active);
    fprintf(stderr, "\n   Length   : %f", Robot->PLAT.Length);
    fprintf(stderr, "\n   Width    : %f", Robot->PLAT.Width);
    fprintf(stderr, "\n   Thickness: %f", Robot->PLAT.Thick);
    fprintf(stderr, "\n   Z_OFF    : %f", Robot->PLAT.Z_OFF);
    fprintf(stderr, "\n   L_OFF    : %f", Robot->PLAT.L_OFF);
    fprintf(stderr, "\n   ANG_OFF  : %f\n\n", Robot->PLAT.ANG_OFF);
    }
  #endif
  fclose(ARM);
}
```

UTILITIES

USEFUL_UTILS.C

```
/*
  ┌──────────────────────────────────────────────────────────┐
  │ Program Name:   useful_UTILS.c                           │
  ├──────────────────────────────────────────────────────────┤
  │ description: This file contains stuff that I'm not sure  │
  │    where to put it in the system.  Some of them like     │
  │    Solutions_init really don't belong here, others       │
  │    might. I recommend that if you too have procedures    │
  │    to add and not quite sure where they should go,       │
  │    leave them here_(8^>).                                │
  │                                                          │
  │ Procedures:                                              │
  │    Solutions_init   :                                    │
  │    Solutions_free   :                                    │
  │    spheres          :                                    │
  │    fmat_pr/fmat_prf :                                    │
  │    fprint_norm      :                                    │
  └──────────────────────────────────────────────────────────┘ */

#include "general.h"       /* Header File to link all others */

/*
  ┌──────────────────────────────────────────────────────────┐
  │ name:    Solutions_init                                  │
  │ description: The Solutions structure is explained in     │
  │     structures.h, this procedure is used to declare      │
  │     and allocate memory for the structure.               │
  │                                                          │
  │ inputs:  M (joint space) and N (Task space)              │
  │ outputs: Allocates memory for the three Matrixes as      │
  │     well as the structure itself. Also initializes       │
  │     M, N, Mred, Nred (Mred and Nred are the current      │
  │     delienators of the space, they are changed in        │
  │     the file FSP.c in the function ReduceA.              │
  ├──────────────────────────────────────────────────────────┤
  │ Authors:     Date:     Modifications:                    │
  │ c hacker     3/95      Created                           │
  └──────────────────────────────────────────────────────────┘ */

Solutions *Solutions_init(int M, int N)
{
  Solutions *Temp;

  Temp = (Solutions *) malloc( sizeof( Solutions ) );
  Temp->g       = mat_malloc( M, M);          /* Soln   */
  Temp->Qarray  = mat_malloc( M, 1);          /* Angles */
  Temp->betall  = mat_malloc( M - N + 5, M); /* Betas  */

  if(!Temp || !Temp->betall)
  {  fprintf(stderr, "Matrix allocation failed\n"); return(NULL);  }

  Temp->M = Temp->Mred = M;     /* init  space size */
  Temp->N = Temp->Nred = N;
  Temp->cn = 0;                 /* init # constrnts */

  return (Temp);
```

```
}

/*
   name:    Solutions_free
   description: The Solutions structure is explained in
       structures.h, this procedure is used to free
       any allocated memory for the structure.

   inputs:  M (joint space) and N (Task space)
   outputs: Deallocates memory for the three Matrixes as
       well as the structure itself.

   Authors:     Date:     Modifications:
   c hacker     3/95      Created
                                                        */
void Solutions_free( Solutions *Temp )
{
  mat_free(Temp->betall);
  mat_free(Temp->Qarray);
  mat_free(Temp->g);
  free ( (char *) Temp);
}


/*
   name:    spheres
   description: spheres reads a datafile, or user's input
       for data concerning spherical objects.  However, I
       rarely use this in an windows interface becuase it
       is easier to just change the sphere_data file it-
       self, but to each his own.
   inputs:  spheredata a 2D array
   outputs: spheredata contains spherical obstacles.

   Authors:     Date:     Modifications:
   D Carlson    2/92      Created
   c hacker    10/94      removed spheredata as global
                                                        */
int spheres(spheredata, datafp)
FILE  *datafp;
double spheredata[4][4];
{
  int   ns, sphere_no, i;
  FILE *sphere_file;
  char  question='n';

  #ifdef DEBUG_OUT
    fprintf(stdout, "\nDo you want to change the spheres? ");
    scanf(" %c", &question);
  #endif

  if ((sphere_file = fopen("sphere_data", "r+")) == NULL)
  {
    fprintf(stderr, "Cannot open: sphere_data\n");
    fclose(sphere_file);
    return 0;
  }

  if ((question != 'Y') && (question != 'y'))
  {
    fscanf(sphere_file, "%d", &ns);
    for (sphere_no = 0; sphere_no < ns; sphere_no++)
      for (i = 0; i < 4; i++)
        fscanf(sphere_file, "%lf", &spheredata[sphere_no][i]);
  }
  else
  {
    fprintf(stdout, "How many spheres? (4 max): ");
    scanf("%d", &ns);
    if (ns > 4)  ns = 4;

    for (i = 0; i < ns; i++)
    {
      fprintf(stdout, "xcenter of sphere[%d]: ", i);
      fscanf(stdin, "%f", &spheredata[i][0]);
      fprintf(stdout, "ycenter of sphere[%d]: ", i);
      fscanf(stdin, "%f", &spheredata[i][1]);
      fprintf(stdout, "zcenter of sphere[%d]: ", i);
      fscanf(stdin, "%f", &spheredata[i][2]);
      fprintf(stdout, " radius of sphere[%d]: ", i);
      fscanf(stdin, "%f", &spheredata[i][3]);
    }

    fprintf(sphere_file, "%i\n", ns);
    for (sphere_no = 0; sphere_no < ns; sphere_no++)
    {
      for (i = 0; i < 4; i++)
        fprintf(sphere_file, "%f ", spheredata[sphere_no][i]);
      fprintf(sphere_file, "\n");
    }
  }                          /******* end else ********/

  fprintf(datafp,"\n Spheres read into the system...");
  for (i=0; i< ns; i++)        /* confirm proper initialization  */
    fprintf(datafp,"\n%i: <%f, %f, %f> %f",i,
       spheredata[i][0], spheredata[i][1], spheredata[i][2],
       spheredata[i][3]);

  fclose(sphere_file);
  return (ns);
}                            /* end spheres... */
```

```
/*
 name:    fmat_pr/ fmat_prf
 description: Disqusted with the inability to print the
     MATRIX to a file which is currently the way IKOR
     communicates with the outside world, I have alter-
     ed the mat_pr function in matrix.c. The function
     that ends with an 'f' means more decimal places
     will be printed out.
 inputs:  Datafile, description string, and a MATRIX
 outputs: The checkfile contains the input string and
     MATRIX.

 Authors:     Date:     Modifications:
 c hacker    10/94      Created
              3/95      took struct from "struct MATRIX"
*/
void fmat_pr (FILE *checkfile, char *string, MATRIX *a)
{
  int i, j;

  fprintf(checkfile, "\n   %s ROWS: %d, COLS: %d\n",
       string, a->rows, a->cols);
  for (i = 0; i < a->rows; i++)
  {
    for (j = 0; j < a->cols; j++)
      fprintf(checkfile, "%8.4f ", a->p[i][j]);
    fprintf(checkfile, "\n");
  }
  fprintf(checkfile, "\n");
}

void fmat_prf(FILE *checkfile, char *string, MATRIX *a)
{
  int i, j;

  fprintf(checkfile, "\n   %s Rows: %d, Cols: %d\n",
       string, a->rows, a->cols);
  for (i = 0; i < a->rows; i++)
  {
    for (j = 0; j < a->cols; j++)
    {
      fprintf(checkfile,"   %16.12f", a->p[i][j]);
    }
    fprintf(checkfile, "\n");
  }
  fprintf(checkfile, "\n");
}
```

```
/*
 name:       fprint_norm
 description: Prints out the Norm of the change in
     angles.  This has been used mostly to compare the
     psuedo inverse against the FSP.
 inputs:  Datafile, change in angles
 outputs: The datafile contains the least norm.

 Authors:     Date:     Modifications:
 D Carlson     2/92     Created
 c hacker     10/94     Removed Squarroot function
               3/95     took struct from "struct MATRIX"
*/
void fprint_norm(outfile,dq)
FILE  *outfile;
MATRIX *dq;
{
  int i;
  long double norm = 0.0;

  for (i = 0; i < M; i++)
  {
    norm = norm + (deg(dq -> p[i][0]) * deg(dq -> p[i][0]));
  }

  fprintf (outfile, " %lf\n", norm);
}
```

MATRIX.H

```
/*
   ┌──────────────────────────────────────────────────────┐
   │ Program Name:   matrix.h                             │
   ├──────────────────────────────────────────────────────┤
   │ description: Contains MATRIX data structure and func-│
   │    tion declarations and descriptions for matrix.c   │
   └──────────────────────────────────────────────────────┘ */
struct MATRIX_struct
{
  float **p;     /* Pointer to array of pointers to matrix rows. */
  int   rows;    /* Row dimension of the matrix.                 */
  int   cols;    /* Column dimension of the matrix.              */
};
typedef struct MATRIX_struct MATRIX;

extern void free_vector(float *,int,int);
extern float *vector(int,int);

void ludcmp();         /*   LU decomposition                     */
void lubksb();         /*   LU backsubstitution                  */
void svdcmp();         /*   Singular Value Decomposition         */
void svbksb();         /*   SVD backsubstitution                 */
void mat_free();       /*   Free matrix pointer                  */
void mat_pr();         /*   Print matrix (not full precision)    */
void mat_prf();        /*   Print matrix (full precision)        */
void mat_mul();        /*   Multiplicate two matrices            */
void mat_add();        /*   Add two matrices                     */
void mat_sub();        /*   Subtract a matrix from another       */
void mat_sca();        /*   Scale a matrix by a factor           */
void mat_LU_inv();     /*   Invert a matrix using LU-dec.        */
void mat_pseudoinv(); /*   Inverse or pseudoinverse using SVD   */
double mat_det();      /*   Determinant using SVD                */
double mat_eigen();    /*    max eigen value of AtA              */
void mat_tra();        /*   Transpose a matrix                   */
void mat_cp();         /*   Copy a matrix into another           */
  /* Dot product two vectors: rows0 of the 2 matrices     */
float mat_vec_dot();
  /* Cross product two vectors: rows 0 of the 2 matrices  */
void  mat_vec_cross();
  /* Length (absolute value) of a vector: rows 0 a matrix */
float mat_vec_abs();

/* All these functions *mat_xxx2() are equivalent to the mat_xxx() *
 * function, except that they automatically creates the return     *
 * matrix by allocating space for them, and returning the pointer  *
 * to the MATRIX structure.                                        */
MATRIX *mat_malloc(); /*   Matrix allocation                   */
MATRIX *mat_mul2();
MATRIX *mat_cp2();
MATRIX *mat_add2();
MATRIX *mat_sub2();
MATRIX *mat_tra2();   /* B = mat_tra2(A); is:  B = transpose of A */
```

MATRIX.C

```
/*************************************************************
*
* File          : matrix.c
* Author        : Ole Henry Dorum
* Created       : January 28'th 1992
* Purpose       : Perform matrix operations.
*
*************************************************************/

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

#include "matrix.h"

#define SWAP(a,b) {float temp = (a); (a) = (b); (b) = temp;}
#define MAX(a,b)  ((a) > (b) ? (a) : (b))

#define SVD_THRESHOLD 1.0e-6
#define PrintIfBiggerThan 0

/*************************************************************
 *************************************************************
 **                                                         **
 **   Num Rec in C:    LU-decomp        &   LU-backsubstitution  **
 **                    SV-decomposition &   SVD-backsubstitution **
 **                                                         **/

#define TINY 1.0e-20
#define SV_SMALL 1.0e-6

static float at,bt,ct;

#define PYTHAG(a,b) ((at=fabs(a)) > (bt=fabs(b)) ? \
(ct=bt/at,at*sqrt(1.0+ct*ct)) : (bt ? (ct=at/bt,bt*sqrt(1.0+ct*ct)):
0.0))
/*
static float maxarg1,maxarg2;
#define MAX(a,b) (maxarg1=(a),maxarg2=(b),(maxarg1) > (maxarg2) ?\
        (maxarg1) : (maxarg2))
*/

#define SIGN(a,b) ((b) >= 0.0 ? fabs(a) : -fabs(a))

              /*************************************************/


     /*************************************************/
     /* Return the null space of a matrix using svd.    */
     /*************************************************/
void mat_null(a, n_rank, n, K2)
MATRIX *a, *n;
int *n_rank;
float *K2;
{
  MATRIX *a_sqr,
                *v,
                *S_temp;
  float **U, *S, **V,
        *vector(),
        **conv_2_nric_ptr();
  float s_min, s_max, temp;
  void free_vector(),
       free_ivector();
```

```
int i, j, R, C, *order;
int *ivector();

R = a->rows;
C = a->cols;

S_temp = mat_malloc(C+1,1);
v      = mat_malloc(C,C);
V      = conv_2_nric_ptr(v);
S      = vector(1,C);
order  = ivector(1,C);

if (R < C)
   {
   a_sqr = mat_malloc(C,C);
   for (i=0; i<R; i++)
      for (j=0; j<C; j++)
         a_sqr->p[i][j]=a->p[i][j];
   for (i=R; i<C; i++)
      for (j=0; j<C; j++)
         a_sqr->p[i][j]=0.0;
   }
else
   {
   a_sqr = mat_malloc(R,C);
   mat_cp(a, a_sqr);
   }

U = conv_2_nric_ptr(a_sqr);

svdcmp(U, C, C, S, V);

for (i=1; i<(C+1); i++)
   {
   S_temp->p[i][0] = S[i];
   order[i] = i;
   }
for (i=C; i>1; i--)
   for (j=i-1; j>=1; j--)
      if (fabs(S_temp->p[i][0])>fabs(S_temp->p[j][0]))
         {
         temp = S_temp->p[i][0];
         S_temp->p[i][0] = S_temp->p[j][0];
         S_temp->p[j][0] = temp;
         temp = order[i];
         order[i] = order[j];
         order[j] = temp;
         }
if (R<=C)
   s_min = fabs(S_temp->p[R][0]);
else
   s_min = fabs(S_temp->p[C][0]);
s_max = fabs(S_temp->p[1][0]);

*n_rank = 0;
while (fabs(S_temp->p[C-*n_rank][0]) < SV_SMALL)
   {
   for (j=1; j<=C; j++)
      n->p[j-1][*n_rank] = V[j][order[C-*n_rank]];
   *n_rank = *n_rank + 1;
   }

  if ((fabs(s_min)>=SV_SMALL) && (fabs(s_min/s_max) < SV_SMALL))
     {
     for (j=1; j<=C; j++)
        n->p[j-1][*n_rank] = V[order[j]][C-*n_rank];
     *n_rank = *n_rank+1;
     }

  *K2 = s_max/s_min;


  mat_free(v);
  mat_free(a_sqr);
  mat_free(S_temp);
  free_vector(S,1,C);
  free_ivector(order,1,C);
  free( (char *) V);
  free( (char *) U);
}



void ludcmp(a,n,indx,d)
int n,*indx;
float **a,*d;
{
       int i,imax,j,k;
       float big,dum,sum,temp;
       float *vv,*vector();
       void nrerror(),free_vector();

       vv=vector(1,n);
       *d=1.0;
       for (i=1;i<=n;i++) {
              big=0.0;
              for (j=1;j<=n;j++)
                     if ((temp=fabs(a[i][j])) > big) big=temp;
              if (big == 0.0) nrerror("Singular matrix in routine
LUDCMP");
              vv[i]=1.0/big;
       }
       for (j=1;j<=n;j++) {
              for (i=1;i<j;i++) {
                     sum=a[i][j];
                     for (k=1;k<i;k++) sum -= a[i][k]*a[k][j];
                     a[i][j]=sum;
              }
              big=0.0;
              for (i=j;i<=n;i++) {
                     sum=a[i][j];
                     for (k=1;k<j;k++)
                            sum -= a[i][k]*a[k][j];
                     a[i][j]=sum;
                     if ( (dum=vv[i]*fabs(sum)) >= big) {
                            big=dum;
                            imax=i;
                     }
              }
              if (j != imax) {
```

```
                        for (k=1;k<=n;k++) {
                                dum=a[imax][k];
                                a[imax][k]=a[j][k];
                                a[j][k]=dum;
                        }
                        *d = -(*d);
                        vv[imax]=vv[j];
                }
                indx[j]=imax;
                if (a[j][j] == 0.0) a[j][j]=TINY;
                if (j != n) {
                        dum=1.0/(a[j][j]);
                        for (i=j+1;i<=n;i++) a[i][j] *= dum;
                }
        }
        free_vector(vv,1,n);
}


void lubksb(a,n,indx,b)
float **a,b[];
int n,*indx;
{
        int i,ii=0,ip,j;
        float sum;

        for (i=1;i<=n;i++) {
                ip=indx[i];
                sum=b[ip];
                b[ip]=b[i];
                if (ii)
                        for (j=ii;j<=i-1;j++) sum -= a[i][j]*b[j];
                else if (sum) ii=i;
                b[i]=sum;
        }
        for (i=n;i>=1;i--) {
                sum=b[i];
                for (j=i+1;j<=n;j++) sum -= a[i][j]*b[j];
                b[i]=sum/a[i][i];
        }
}


void svbksb(u,w,v,m,n,b,x)
float **u,w[],**v,b[],x[];
int m,n;
{
        int jj,j,i;
        float s,*tmp,*vector();
        void free_vector();

        tmp=vector(1,n);
        for (j=1;j<=n;j++) {
                s=0.0;
                if (w[j]) {
                        for (i=1;i<=m;i++) s += u[i][j]*b[i];
                        s /= w[j];
                }
                tmp[j]=s;
        }
        for (j=1;j<=n;j++) {
                s=0.0;
                for (jj=1;jj<=n;jj++) s += v[j][jj]*tmp[jj];
                x[j]=s;
        }
        free_vector(tmp,1,n);
}


void svdcmp(a, m, n, w, v)
  float **a, *w, **v;
  int m, n;
{
  int flag, i, its, j, jj, k, l, nm;
  float c, f, h, s, x, y, z;
  float anorm = 0.0, g = 0.0, scale = 0.0;
  float *rv1, *vector();
  void nrerror(), free_vector();
  if (m < n)
    nrerror("SVDCMP: You must augment A with extra zero rows");
  rv1 = vector(1, n);
  for (i = 1; i <= n; i++)
  {
    l = i + 1;
    rv1[i] = scale * g;
    g = s = scale = 0.0;
    if (i <= m)
    {
      for (k = i; k <= m; k++)
       scale += fabs(a[k][i]);
      if (scale)
      {
       for (k = i; k <= m; k++)
       {
         a[k][i] /= scale;
         s += a[k][i] * a[k][i];
       }
       f = a[i][i];
       g = -SIGN(sqrt(s), f);
       h = f * g - s;
       a[i][i] = f - g;
       if (i != n)
       {
         for (j = l; j <= n; j++)
         {
           for (s = 0.0, k = i; k <= m; k++)
             s += a[k][i] * a[k][j];
           f = s / h;
           for (k = i; k <= m; k++)
             a[k][j] += f * a[k][i];
         }
       }
       for (k = i; k <= m; k++)
         a[k][i] *= scale;
      }
    }
    w[i] = scale * g;
    g = s = scale = 0.0;
    if (i <= m && i != n)
    {
```

```
      for (k = l; k <= n; k++)
        scale += fabs(a[i][k]);
      if (scale)
      {
        for (k = l; k <= n; k++)
        {
          a[i][k] /= scale;
          s += a[i][k] * a[i][k];
        }
        f = a[i][l];
        g = -SIGN(sqrt(s), f);
        h = f * g - s;
        a[i][l] = f - g;
        for (k = l; k <= n; k++)
          rv1[k] = a[i][k] / h;
        if (i != m)
        {
          for (j = l; j <= m; j++)
          {
            for (s = 0.0, k = l; k <= n; k++)
              s += a[j][k] * a[i][k];
            for (k = l; k <= n; k++)
              a[j][k] += s * rv1[k];
          }
        }
        for (k = l; k <= n; k++)
          a[i][k] *= scale;
      }
    }
    anorm = MAX(anorm, (fabs(w[i]) + fabs(rv1[i])));
  }
  for (i = n; i >= 1; i--)
  {
    if (i < n)
    {
      if (g)
      {
        for (j = l; j <= n; j++)
          v[j][i] = (a[i][j] / a[i][l]) / g;
        for (j = l; j <= n; j++)
        {
          for (s = 0.0, k = l; k <= n; k++)
            s += a[i][k] * v[k][j];
          for (k = l; k <= n; k++)
            v[k][j] += s * v[k][i];
        }
      }
      for (j = l; j <= n; j++)
        v[i][j] = v[j][i] = 0.0;
    }
    v[i][i] = 1.0;
    g = rv1[i];
    l = i;
  }
  for (i = n; i >= 1; i--)
  {
    l = i + 1;
    g = w[i];
    if (i < n)
      for (j = l; j <= n; j++)
        a[i][j] = 0.0;
    if (g)
    {
      g = 1.0 / g;
      if (i != n)
      {
        for (j = l; j <= n; j++)
        {
          for (s = 0.0, k = l; k <= m; k++)
            s += a[k][i] * a[k][j];
          f = (s / a[i][i]) * g;
          for (k = i; k <= m; k++)
            a[k][j] += f * a[k][i];
        }
      }
      for (j = i; j <= m; j++)
        a[j][i] *= g;
    }
    else
    {
      for (j = i; j <= m; j++)
        a[j][i] = 0.0;
    }
    ++a[i][i];
  }
  for (k = n; k >= 1; k--)
  {
    for (its = 1; its <= 30; its++)
    {
      flag = 1;
      for (l = k; l >= 1; l--)
      {
        nm = l - 1;
        if (fabs(rv1[l]) + anorm == anorm)
        {
          flag = 0;
          break;
        }
        if (fabs(w[nm]) + anorm == anorm)
          break;
      }
      if (flag)
      {
        c = 0.0;
        s = 1.0;
        for (i = l; i <= k; i++)
        {
          f = s * rv1[i];
          if (fabs(f) + anorm != anorm)
          {
            g = w[i];
            h = PYTHAG(f, g);
            w[i] = h;
            h = 1.0 / h;
            c = g * h;
            s = (-f * h);
            for (j = 1; j <= m; j++)
            {
              y = a[j][nm];
              z = a[j][i];
```

```
        a[j][nm] = y * c + z * s;
        a[j][i] = z * c - y * s;
      }
    }
  }
}
z = w[k];
if (l == k)
{
 if (z < 0.0)
 {
   w[k] = -z;
   for (j = 1; j <= n; j++)
     v[j][k] = (-v[j][k]);
 }
 break;
}
if (its == 30)
 nrerror("No convergence in 30 SVDCMP iterations");
x = w[l];
nm = k - 1;
y = w[nm];
g = rv1[nm];
h = rv1[k];
f = ((y - z) * (y + z) + (g - h) * (g + h)) / (2.0 * h * y);
g = PYTHAG(f, 1.0);
f = ((x - z) * (x + z) + h * ((y / (f + SIGN(g, f))) - h)) / x;
c = s = 1.0;
for (j = l; j <= nm; j++)
{
 i = j + 1;
 g = rv1[i];
 y = w[i];
 h = s * g;
 g = c * g;
 z = PYTHAG(f, h);
 rv1[j] = z;
 c = f / z;
 s = h / z;
 f = x * c + g * s;
 g = g * c - x * s;
 h = y * s;
 y = y * c;
 for (jj = 1; jj <= n; jj++)
 {
   x = v[jj][j];
   z = v[jj][i];
   v[jj][j] = x * c + z * s;
   v[jj][i] = z * c - x * s;
 }
 z = PYTHAG(f, h);
 w[j] = z;
 if (z)
 {
   z = 1.0 / z;
   c = f * z;
   s = h * z;
 }
 f = (c * g) + (s * y);
 x = (c * y) - (s * g);
 for (jj = 1; jj <= m; jj++)
 {
   y = a[jj][j];
   z = a[jj][i];
   a[jj][j] = y * c + z * s;
   a[jj][i] = z * c - y * s;
 }
}
rv1[l] = 0.0;
rv1[k] = f;
w[k] = x;
  }
}
free_vector(rv1, 1, n);
}


#undef SIGN
#undef PYTHAG
#undef TINY

/*************************************************************************
*******/


MATRIX *mat_malloc(rows, cols)
int rows, cols;
{
  int    i;
  float  *mat, **row;
  MATRIX *matrix;

  /* Allocate space for structure, elements and pointers.
   *
   * Note, that the allocated number of row pointers is MAX(row, cols)
   * because it facilitates transposing rectangular matrices.
   */
  mat = (float * ) malloc(rows * cols * sizeof(float));
  row = (float **) malloc((MAX(rows, cols)) * sizeof(float *));
  matrix = (MATRIX *) malloc( sizeof( MATRIX ) );

  if(!mat || !row || !matrix)
    {
      fprintf(stderr, "Matrix allocation failed\n");
      return(NULL);
    }

  matrix->p = row;
  matrix->rows = rows;
  matrix->cols = cols;

  /* The Nth element of the array row points to the 1st element
   * on the Nth row. Thus, **m = *m[0] = m[0][0]
   *
   * Calculate the addresses of the pointers pointing to the
   * rows of the matrix
   */

  for(i = 0; i < rows; i++)
  {
```

```
    row[i] = mat;
    mat += cols;
  }

  return(matrix);
}


void mat_free(m)
MATRIX *m;
{
  free( (char *) *m->p);
  free( (char *)  m->p);
  free( (char *)  m);
}


void mat_pr(a)
MATRIX *a;
{
  int i, j;

  printf("\n");
  for (i = 0; i < a->rows; i++)
  {
    for (j = 0; j < a->cols; j++)
      {
       if (fabs(a->p[i][j]) >= PrintIfBiggerThan)
         printf("\t% .8f", a->p[i][j]);
       else
         printf("\t  - ");
      }
    printf("\n");
  }
  printf("\n");
}


void mat_prf(a)
MATRIX *a;
{
  int i, j;

  printf("\n");
  for (i = 0; i < a->rows; i++)
  {
    for (j = 0; j < a->cols; j++)
    {
      printf("\t% f", a->p[i][j]);
    }
    printf("\n");
  }
  printf("\n");
}


/* The inverted matrix will still reside in a */
void mat_LU_inv(a)
MATRIX *a;
{
  MATRIX *y;
  float  **A, **Y, d, *col;
  int    N, i, j, *indx;
  float  **conv_2_nric_ptr();

  N = a->rows;

  /* Allocate space for y matrix to hold result, column vector col,
   * and indx array.
   */
  y    = mat_malloc(a->rows, a->cols);
  col  = vector(1, N);
  indx = (int *) malloc(N * sizeof(float));

  /* Create pointers to a and y conforming with Num-Rec-in-C format.
   * A and Y is from [1..N][1..N]
   */
  A = conv_2_nric_ptr(a);
  Y = conv_2_nric_ptr(y);

  /* LU decompose A
   */
  ludcmp(A, N, indx, &d);

  /* Find inverse by columns. (There is no better way to do it.)
   */
  for (j = 1; j <= N; j++)
    {
      for (i = 1; i <= N; i++)
       col[i] = 0.0;

      col[j] = 1.0;
      lubksb(A, N, indx, col);

      for (i = 1; i <= N; i++)
       Y[i][j] = col[i];
    }

  /* Inverted matrix is now in y. Copy y into a and free matrix y
   */
  mat_cp(y, a);

  mat_free(y);
  free_vector(col, 1, N);
  free( (char *) indx);
  free( (char *) A);
  free( (char *) Y);
}


/* The pseudoinverted matrix will still reside in a. In the case of a
 * square matrix, the result will actually be the inverted matrix. If
 * the matrix is rectangular, the pseudoinverse will have the correct
 * dimension.
 */
void mat_pseudoinv(a)
MATRIX *a;
{
```

```
  MATRIX *q2, *z, *tmp;
  float  **A, **Q2, *Z;
  float  z_min, z_max;
  int    M, N, i, j;
  float  **conv_2_nric_ptr();

  M = a->rows; N = a->cols;

  /* Allocate space for q2 matrix, vector Z[1..N], z and tmp
   */
  q2 = mat_malloc(N, N);
  z  = mat_malloc(N, N);
  Z  = vector(1, N);

  /* Create pointers to a and q2 conforming with Num-Rec-in-C format.
   */
  A  = conv_2_nric_ptr(a);
  Q2 = conv_2_nric_ptr(q2);

  /* Compute A[1..M][1..N]`s singular value decomposition (SVD): A =
Q1*Z*Q2_tra
   *
   * Q1  will replace A, and the diagonal value of singular values Z
is output
   * as a vector Z[1..N]. The matrix Q2 (not the transpose Q2_tra) is
output
   * as Q2[1..N][1..N]. M must be greater than or equal to N; If it is
smaller,
   * then A should be filled up to square with zero rows.
   */
  svdcmp(A, M, N, Z, Q2);

  /* (Singular values = squareroot of the eigenvalues), find maximum.
   */
  z_max = 0.0;
  for (i = 1; i <= N; i++) if (Z[i] > z_max) z_max = Z[i];

  /* Set threshold value of the minimum singular value allowed
   * to be nonzero.
   */
  z_min = z_max*SVD_THRESHOLD;

  /* Invert while copying from the Z vector into the z+ matrix, and
weed out
   * the too small singular values.
   */
  for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
      z->p[i][j] = ((i == j) && Z[i+1] > z_min) ? 1.0/Z[i+1] : 0.0;
     /* z->p[i][j] = 1.0/Z[i+1];*/


  /*
   * A_pseudoinv = Q2 * Z_pseudoinv * Q1_tra
   *
   * Returned matrix A from svdcmp() is actually Q1, therefore:
   */

/*
  printf("Testing the inverse:\n");
  q1t = mat_tra2(a);
  qq = mat_mul2(a,q1t);
  printf("Q1t*q1 = \n");
  mat_pr(qq);
  mat_free(qq);
  mat_free(q1t);
*/


  mat_tra(a);             /* Transpose Q1 to Q1_tra                    */
  tmp = mat_mul2(z, a);   /* tmp = Z_pseudoinv * Q1_tra                */
  mat_mul(q2, tmp, a);    /* A_pseudoinv=Q2*tmp=Q2*Z_pseudo * Q1_tra */

/*
  q2t = mat_tra2(q2);
  qq2 = mat_mul2(q2,q2t);
  printf("Q2t*Q2 = \n");
  mat_pr(qq2);
  mat_free(qq2);
  mat_free(q2t);
*/



  mat_free(q2);
  mat_free(z);
  mat_free(tmp);
  free_vector(Z, 1, N);
  free( (char *) A);
  free( (char *) Q2);

}


/* The pseudoinverted matrix will still reside in a. In the case of a
 * square matrix, the result will actually be the inverted matrix. If
 * the matrix is rectangular, the pseudoinverse will have the correct
 * dimension.
 */


double mat_det(a)
MATRIX *a;
{
  MATRIX *q2, *cp;
  float  **A, **Q2, *Z;
  int    M, N, i, j;
  float  **conv_2_nric_ptr();
  double det;

  M = a->rows; N = a->cols;

  /* Allocate space for q2 matrix, vector Z[1..N], z and tmp */
  cp = mat_cp2(a);
  q2 = mat_malloc(N, N);
  Z  = vector(1, N);

  /* Create pointers to a and q2 conforming with Num-Rec-in-C format.
   */
  A  = conv_2_nric_ptr(cp);
```

```
  Q2 = conv_2_nric_ptr(q2);

  /* Compute A[1..M][1..N]`s singular value decomposition (SVD): A =
Q1*Z*Q2_tra
   *
   * Q1 will replace A, and the diagonal value of singular values Z is
output
   * as a vector Z[1..N]. The matrix Q2 (not the transpose Q2_tra) is
output
   * as Q2[1..N][1..N]. M must be greater than or equal to N; If it is
smaller,
   * then A should be filled up to square with zero rows.
   */
  svdcmp(A, M, N, Z, Q2);

  /* (Singular values = squareroot of the eigenvalues of AtA), find
maximum. */

  for(det=1.0,i=1;i<=N;i++)
       det *= (double) Z[i];

  mat_free(q2);
  mat_free(cp);
  free( (char *) A);
  free( (char *) Q2);
  free_vector(Z, 1, N);

  return(det);
}


double mat_eigen(a)
MATRIX *a;
{
  MATRIX *q2;
  float  **A, **Q2, *Z;
  float  z_min, z_max;
  int    M, N, i, j;
  float  **conv_2_nric_ptr();

  M = a->rows; N = a->cols;

  /* Allocate space for q2 matrix, vector Z[1..N]
   */
  q2 = mat_malloc(N, N);
  Z  = vector(1, N);

  /* Create pointers to a and q2 conforming with Num-Rec-in-C format.
   */
  A  = conv_2_nric_ptr(a);
  Q2 = conv_2_nric_ptr(q2);

  /* Compute A[1..M][1..N]`s singular value decomposition (SVD): A =
Q1*Z*Q2_tra
   *
   * Q1  will replace A, and the diagonal value of singular values Z
is output
   * as a vector Z[1..N]. The matrix Q2 (not the transpose Q2_tra) is
output
   * as Q2[1..N][1..N]. M must be greater than or equal to N; If it is
smaller,
   * then A should be filled up to square with zero rows.
   */
  svdcmp(A, M, N, Z, Q2);

  /* (Singular values = squareroot of the eigenvalues), find maximum.
   */
  z_max = 0.0;
  for (i = 1; i <= N; i++) if (Z[i] > z_max) z_max = Z[i];

  mat_free(q2);
  free_vector(Z, 1, N);
  free( (char *) A);
  free( (char *) Q2);

  return(z_max);
}



/* The 'c' matrix must already be declared float of size: arows x
bcols.
 */
void mat_mul(a, b, c)
MATRIX *a, *b, *c;
{
  int i, j, k;

  for (k = 0; k < b->cols; k++)
    for (i = 0; i < a->rows; i++)
    {
      c->p[i][k] = 0;
      for (j = 0; j < a->cols; j++)
       c->p[i][k] += a->p[i][j]*b->p[j][k];
    }
}


/* The function will automatically create the result matrix of correct
 * dimension. The pointer to this matrix structure is returned.
 */
MATRIX *mat_mul2(a, b)
MATRIX *a, *b;
{
  MATRIX *c;

  c = mat_malloc(a->rows, b->cols);

  mat_mul(a, b, c);

  return(c);
}


/* The 'c' matrix must already be declared float of size: arows x
bcols.
 *
 * c = a + b;
```

```
 */
void mat_add(a, b, c)
MATRIX *a, *b, *c;
{
  float *p, *q, *r;
  int   i;

  p = *a->p; q = *b->p; r = *c->p;

  for (i = a->rows*a->cols; i--;)
    *r++ = *p++ + *q++;
}


/* The function will automatically create the result matrix of correct
 * dimension. The pointer to this matrix structure is returned.
 *
 * c = a + b;
 */
MATRIX *mat_add2(a, b)
MATRIX *a, *b;
{
  MATRIX *c;

  c = mat_malloc(a->rows, a->cols);

  mat_add(a, b, c);

  return(c);
}


/* The 'c' matrix must already be declared float of size: arows x
acols.
 *
 * c = a - b;
 */
void mat_sub(a, b, c)
MATRIX *a, *b, *c;
{
  float *p, *q, *r;
  int   i;

  p = *a->p; q = *b->p; r = *c->p;

  for (i = a->rows*a->cols; i--;)
    *r++ = *p++ - *q++;
}



/* The function will automatically create the result matrix of correct
 * dimension. The pointer to this matrix structure is returned.
 *
 * c = a - b;
 */
MATRIX *mat_sub2(a, b)
MATRIX *a, *b;
{
  MATRIX *c;

  c = mat_malloc(a->rows, a->cols);

  mat_sub(a, b, c);

  return(c);
}



/* The matrix 'a' is scaled by the factor 'c'
 */
void mat_sca(a, c)
MATRIX *a;
float  c;
{
  float *p;
  int   i;

  p = *a->p;

  for (i = a->rows*a->cols; i--;)
    *p++ *= c;
}



/* The transposed matrix still resides in 'a' after transposition
 */
void mat_tra(a)
MATRIX *a;
{
  int   i, j, temp;
  float *p;

  if (a->rows == a->cols)               /* Square matrix */
  {
    for (i = a->rows-1; i--;)
      for (j = a->cols-1; j > i; j--)
       SWAP(a->p[i][j], a->p[j][i]);
  }
  else                                  /* Rectangular matrix */
  {
    temp = a->rows;
    a->rows = a->cols;
    a->cols = temp;

    /* Recompute pointers to the new rows of the matrix
     */
    p = *a->p;
    for(i = 0; i < a->rows; i++)
    {
      a->p[i] = p;
      p += a->cols;
    }

    /* No need to swap elements if the matrix has only one row or
column.
     */
    if (!((a->rows == 1) || (a->cols == 1)))
```

```
    {
      MATRIX *m;

      /*...otherwise put elements in temporary matrix and
       * copy from it into the columns of the transposed matrix.
       */
      m = mat_malloc(a->rows, a->cols);
      mat_cp(a,m);
      p = *m->p;

      for (j = 0; j < a->cols; j++)
       for (i = 0; i < a->rows; i++)
         a->p[i][j] = *p++;

      mat_free(m);
    }
  }
}


/* The function will automatically create the result matrix of correct
 * dimension which will be the transpose of A. The pointer to this
matrix
 * structure is returned.
 */
MATRIX *mat_tra2(A)
MATRIX *A;
{
  int   i, j;
  MATRIX *At;

  At = mat_malloc(A->cols, A->rows);

  for (i = A->rows-1; i--;)
    for (j = A->cols-1; j > i; j--)
      At->p[i][j] = A->p[j][i];

  return(At);
}


/* The copied matrix 'cpy' must already be declared float of
 * same size as 'a'.
 */
void mat_cp(a, cpy)
MATRIX *a, *cpy;
{
  float *p, *q;
  int   i;

  p = *a->p; q = *cpy->p;

  for (i = a->rows*a->cols; i--;)
    *q++ = *p++;
}


/* The function will automatically create the result matrix of correct
 * dimension . The pointer to this matrix structure is returned.
 */
MATRIX *mat_cp2(a)
MATRIX *a;
{
  int   i;
  float *p, *q;
  MATRIX *cpy;

  cpy = mat_malloc(a->rows, a->cols);

  mat_cp(a, cpy);

  return(cpy);
}


/* Returns an pointer which points to an array of pointers to matrix
row
 * elements in a way so that the matrix a->p[0..n][0..m] can be
accessed
 * from [1..n-1][1..m-1] which is required by the Numerical Rec. i C.
 */
float **conv_2_nric_ptr(a)
MATRIX *a;
{
  float **new_ptr_2_row;
  int   i;

  /* Allocate space for an array of OFFSET pointers.
   */
  new_ptr_2_row = (float **) malloc((a->rows + 1) * sizeof(float *));

  /* The array of pointers to row must be offset by -1
   */
  for (i = 0; i < a->rows; i++)
    new_ptr_2_row[i+1] = a->p[i] - 1;

  /* To not have element[0][] dangling, let it point to a->p[0][0].
   */
  new_ptr_2_row[0] = a->p[0];

  return( new_ptr_2_row );
}
```

NRUTIL.C

```
/* Free pointer pointing to the matrix in Numerical-Rec.-in-C-format
 */
void free_nric_ptr(p)
float **p;
{
  free( (char *)  p);
}



/* Returns vector dot product of the two vectors: row 0 of matrix a
and b.
 */
float mat_vec_dot(a, ar, b, br)
MATRIX *a, *b;
int ar, br;
{
  int i;
  float d = 0;

  for (i = 0; i < a->cols; i++)
    d += a->p[ar][i] * b->p[br][i];

  return( d );
}



/* Returns vector cross product of the two vectors: row ra, rb of
matrix a and b.
 * Result vector matrix c must be of size [1][m] or [n][m].
 */
void mat_vec_cross(a, ar, b, br, c)
MATRIX *a, *b, *c;
int ar, br;
{
  c->p[0][0] = a->p[ar][1] * b->p[br][2] - a->p[ar][2] * b->p[br][1];
  c->p[0][1] = a->p[ar][2] * b->p[br][0] - a->p[ar][0] * b->p[br][2];
  c->p[0][2] = a->p[ar][0] * b->p[br][1] - a->p[ar][1] * b->p[br][0];
}



/* returns length (absolute value) of the vector: row r of matrix a.
 */
float mat_vec_abs(a, r)
MATRIX *a;
int r;
{
  int i;
  float d = 0;

  for (i = 0; i < a->cols; i++)
    d += a->p[r][i] * a->p[r][i];

  return( sqrt(d) );
}
```

```
#include <malloc.h>
#include <stdio.h>

void nrerror(error_text)
char error_text[];
{
        void exit();

        fprintf(stderr,"Numerical Recipes run-time error...\n");
        fprintf(stderr,"%s\n",error_text);
        fprintf(stderr,"...now exiting to system...\n");
        exit(1);
}



float *vector(nl,nh)
int nl,nh;
{
        float *v;

        v=(float *)malloc((unsigned) (nh-nl+1)*sizeof(float));
        if (!v) nrerror("allocation failure in vector()");
        return v-nl;
}

int *ivector(nl,nh)
int nl,nh;
{
        int *v;

        v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
        if (!v) nrerror("allocation failure in ivector()");
        return v-nl;
}

double *dvector(nl,nh)
int nl,nh;
{
        double *v;

        v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
        if (!v) nrerror("allocation failure in dvector()");
        return v-nl;
}



float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
        int i;
        float **m;

        m=(float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m -= nrl;
```

```
	for(i=nrl;i<=nrh;i++) {
		m[i]=(float *) malloc((unsigned)
(nch-ncl+1)*sizeof(float));
		if (!m[i]) nrerror("allocation failure 2 in matrix()");
		m[i] -= ncl;
	}
	return m;
}

double **dmatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
	int i;
	double **m;

	m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
	if (!m) nrerror("allocation failure 1 in dmatrix()");
	m -= nrl;

	for(i=nrl;i<=nrh;i++) {
		m[i]=(double *) malloc((unsigned)
(nch-ncl+1)*sizeof(double));
		if (!m[i]) nrerror("allocation failure 2 in dmatrix()");
		m[i] -= ncl;
	}
	return m;
}

int **imatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
	int i,**m;

	m=(int **)malloc((unsigned) (nrh-nrl+1)*sizeof(int*));
	if (!m) nrerror("allocation failure 1 in imatrix()");
	m -= nrl;

	for(i=nrl;i<=nrh;i++) {
		m[i]=(int *)malloc((unsigned) (nch-ncl+1)*sizeof(int));
		if (!m[i]) nrerror("allocation failure 2 in imatrix()");
		m[i] -= ncl;
	}
	return m;
}



float **submatrix(a,oldrl,oldrh,oldcl,oldch,newrl,newcl)
float **a;
int oldrl,oldrh,oldcl,oldch,newrl,newcl;
{
	int i,j;
	float **m;

	m=(float **) malloc((unsigned) (oldrh-oldrl+1)*sizeof(float*));
	if (!m) nrerror("allocation failure in submatrix()");
	m -= newrl;

	for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+oldcl-newcl;

	return m;
}


void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
	free((char*) (v+nl));
}

void free_ivector(v,nl,nh)
int *v,nl,nh;
{
	free((char*) (v+nl));
}

void free_dvector(v,nl,nh)
double *v;
int nl,nh;
{
	free((char*) (v+nl));
}


void free_matrix(m,nrl,nrh,ncl,nch)
float **m;
int nrl,nrh,ncl,nch;
{
	int i;

	for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
	free((char*) (m+nrl));
}

void free_dmatrix(m,nrl,nrh,ncl,nch)
double **m;
int nrl,nrh,ncl,nch;
{
	int i;

	for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
	free((char*) (m+nrl));
}

void free_imatrix(m,nrl,nrh,ncl,nch)
int **m;
int nrl,nrh,ncl,nch;
{
	int i;

	for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
	free((char*) (m+nrl));
}


void free_submatrix(b,nrl,nrh,ncl,nch)
```

```
float **b;
int nrl,nrh,ncl,nch;
{
        free((char*) (b+nrl));
}


float **convert_matrix(a,nrl,nrh,ncl,nch)
float *a;
int nrl,nrh,ncl,nch;
{
        int i,j,nrow,ncol;
        float **m;

        nrow=nrh-nrl+1;
        ncol=nch-ncl+1;
        m = (float **) malloc((unsigned) (nrow)*sizeof(float*));
        if (!m) nrerror("allocation failure in convert_matrix()");
        m -= nrl;
        for(i=0,j=nrl;i<=nrow-1;i++,j++) m[j]=a+ncol*i-ncl;
        return m;
}


void free_convert_matrix(b,nrl,nrh,ncl,nch)
float **b;
int nrl,nrh,ncl,nch;
{
        free((char*) (b+nrl));
}
```

FSP Programs

The following files are listed in this section. Note that the file FSP.c is not the entire FSP system, it is only the solution generator. The other files include the necessary information to parameterize the solutions according to constraints and criteria.

FSP.c	FSP, ReduceA among many: does all blocking and inverting.
criteria.c	Optimization Criteria: Currently only the Least Norm
constraints.c	Finds objects in path, joints beyond limits.
analytical.c	Calculates Betas, Grammian, and parameters t_i's.

FSP.C

```
/*
    Program Name:   FSPv2.0.c

    description:  Given the Jacobian and dx the procedure
        finds a a set of g vectors which span the solution
        space of the equation to the equation J*q=dx, for
        all values of the vector q.


    Procedures:
        Solution_generator  :
        RestofSoln          :
        Dependency          :
        BLOCK_COL_FIND_X    :
        ReduceA             :
        CheckB              :
        CheckRange          :
        Rebuild_gs          :
*/

#include "general.h"

int  SystemComplete; /* marks when all necessary vecs have been
found*/

/*
    name:    Solution_generator
    description:  Main FSP procedure which sets up all
       initialization of variables, and starts production
       of g vectors.  After calling reduce it checks for
       SpecialCase2 then finds the first valid blocking
       pattern.  After which it calls RestofSoln and
       depending on the outcome it decides which method to
       determine the solution vector.


    inputs:  Aorig, borig, M, N
    outputs: Ared, bred, Mred, Nred, Xelim, ColElim,
             RowElim, g

    Authors:      Date:     Modifications:
     g fries      3/95      Created
*/

int Solution_generator(FSP_data, Aorig, borig, check)
FILE      *check;
MATRIX    *Aorig,
          *borig;
Solutions *FSP_data;
{
MATRIX *Ared,   /* reduced A                                        */
       *Asub,   /* submatrix from square submatrix is found         */
       *Asqr,   /* square submatrix                                 */
       *bred,   /* reduced b                                        */
       *block,  /* locations of blocked columns for each soln       */
       *Xelim,  /* comps of final X that were elim. in reduction */
       *n,      /* null space vectors                               */
       *Specialg;  /* null vectors created by SpecialCase1          */
float  K2,      /* matrix condition number:  sv_max/sv_min          */
       N_BND;   /* min acceptable value for nonzero nspace comp    */
int    i, j, k, I,  /* loop counters                                */
       bcheck,      /* check for completely zero bred              */
       *ColElim,    /* marks columns eliminated from original A    */
       *RowElim,    /* which rows are to be eliminated from the    */
       move,        /* which of 4 possible blocks is being moved   */
       NextToFind,  /* vector number being searched for            */
       *Tackon,     /* columns r indexed for effic, soln finding   */
       *FirstOK,    /* if OK to sub in for firstsoln column        */
       pos,         /* used with the Ordering[] variable           */
       nullity,     /* dimension of null space                     */
       binrange,    /* marks if b is in the range of the A         */
       Exit_Status, /* tells calling routine if OK                 */
       NumSpg,      /* # of g vectors created by SpecialCase1      */
       Stop;        /* loop flag to end                            */

  /* allocate memory space for variables */
  Tackon  = ( int *) malloc ( (M - N) * sizeof( int ) );
  FirstOK = ( int *) malloc (    N    * sizeof( int ) );
  ColElim = ( int *) malloc (    M    * sizeof( int ) );
  RowElim = ( int *) malloc (    N    * sizeof( int ) );
  Ared       = mat_malloc(N, M);
  bred       = mat_malloc(N, 1);
  Xelim       = mat_malloc(M,1);
  Specialg= mat_malloc((M-1),M);

  /* initialize SystemComplete and FirstOK */
  SystemComplete = FALSE;
  for (i=0; i< N; i++)
    FirstOK[i] = FALSE;

  /*check = stderr;              /* uncomment when in trouble   */
  /* Eliminate all the nonredundancies from the A and b        */
  ReduceA(FSP_data, Aorig, Ared, borig, bred, Xelim,ColElim,RowElim,
                                              &NumSpg, Specialg);

  /* Setup and initialization of g vectors */
  mat_free(FSP_data->g);
  FSP_data->g=mat_malloc((FSP_data->Mred-FSP_data->Nred+1+NumSpg), M);
  for (i=0; i<(SPAN); i++)
    for (j=0; j<M; j++) FSP_data->g->p[i][j] = 0.0e0;

  n = mat_malloc(FSP_data->Mred, FSP_data->Nred);

if (DEBUG) {
  fprintf(check,"\n ____________________________ \n");
  fprintf(check,"\n          FSP               \n");
  fprintf(check,  " ____________________________ \n\n");
  fmat_pr(check, " *** A Original *** ", Aorig);
  fmat_pr(check, " *** b Original *** ", borig);
  fmat_pr(check, " *** A Reduced *** ", Ared);
  fmat_pr(check, " *** b Reduced *** ", bred);
  fprintf(check, " M   : %d, N   : %d\n Mred: %d, Nred: %d",
                 M, N, FSP_data->Mred, FSP_data->Nred);
  fprintf (check, "\nRowElim: ");
  for (i=0; i< N; i++) fprintf (check,"%d ", RowElim[i]);
  fprintf (check, "\nColElim: ");
```

```
  for (i=0; i< M; i++) fprintf (check,"%d ", ColElim[i]);
}            /* if DEBUGGING, include code  (see file info) */

bcheck=CheckB(bred, FSP_data->Nred);
if (bcheck == 0)
    {
    fprintf(check, "GIVEN MATRIX IS COMPLETELY RESTRICTED\n\n");
    Asub = mat_malloc(FSP_data->Nred, FSP_data->Mred);
    for (i=0; I < FSP_data->Nred; i++)
       for (j=0; j < FSP_data->Mred; j++)
           Asub->p[i][j] = Ared->p[i][j];

    mat_null(Asub, &nullity, n, &K2);

    /* set the solution vectors for a completely restricted system  */
    for (i=0; i < FSP_data->Mred; i++)
       for (j=0; j< SPAN2 - 1; j++)
           FSP_data->g->p[j][i] = n->p[i][j];
    for (i=0; I < FSP_data->Mred; i++)
       FSP_data->g->p[FSP_data->Mred-FSP_data->Nred][i] = 0.0;
    SystemComplete = TRUE;

    Exit_Status = RESTRICTED;
    if (DEBUG)
      fmat_pr(check, "solutions", FSP_data->g);
    }
else
    {
    if (FSP_data->Nred != N)
        {
            fprintf(check, "GIVEN MATRIX IS RESTRICTED FOR %d ",
                N-FSP_data->Nred);
            fprintf(check, "OUT OF %d VECTOR COMPONENTS\n\n", N);
        }

    block= mat_malloc(SPAN2,FSP_data->Nred);
    Asub = mat_malloc(FSP_data->Nred,FSP_data->Nred+1);
    Asqr = mat_malloc(FSP_data->Nred,FSP_data->Nred);
    NextToFind = 0;

    /* initialize the blocking positions for the 1st solution vector
*/
    if (DEBUG)
       for (i=0; i<(FSP_data->Nred); i++)
            for (j=0; j<(FSP_data->Mred-FSP_data->Nred+1); j++)
              block->p[j][i] = 0;

    for (i=0; i<(FSP_data->Nred); i++)
       block->p[0][i] = i;
    k=0;
    for (i=0; i<FSP_data->Mred; i++)
      if (block->p[0][k] == i)
          {
          for (j=0; j<FSP_data->Nred; j++)
              Asqr->p[j][k] = Ared->p[j][i];
         k++;
          }

    mat_null(Asqr, &nullity, n, &K2);
```

```
if (DEBUG) {
   fmat_pr (check, "block", block);
   fmat_pr (check, "Asqr", Asqr);
   fprintf(check,"nullity = %d, K2 = %f\n", nullity, K2);
}     /* if DEBUGGING, include code  (see file info) */

/* loop until an acceptable well-conditioned 1st solution found */
while ((nullity != 0) && (block->p[0][0]<N))
   {
   pos = FSP_data->Nred-1;
   while (block->p[0][pos] == SPAN2 - 1 +pos)
       pos--;
   block->p[0][pos]++;
   move=pos+1;
   while (move<FSP_data->Nred)
       {
       block->p[0][move]=block->p[0][(move-1)]+1;
       move++;
       }
   k=0;
   for (i=0; i<FSP_data->Mred; i++)
       if (block->p[0][k] == i)
           {
           for (j=0; j<FSP_data->Nred; j++)
               Asqr->p[j][k] = Ared->p[j][i];
           k++;
           }
   if (block->p[0][0] <= FSP_data->Nred)
       mat_null(Asqr, &nullity, n, &K2);
   if (DEBUG) {
       fmat_pr(check, "block", block);
        fmat_pr(check, "Asqr", Asqr);
        fprintf(check,"nullity = %d, K2 = %f\n", nullity, K2);
   }
   }
k=0; I=0;
for (i=0; i<FSP_data->Mred; i++)
   if (block->p[0][k] == i)
       k++;
   else
       {
       Tackon[I]=i;
       I++;
       }

BLOCK_COL_FIND_X(Tackon,FSP_data->g->p[NextToFind],
          block->p[NextToFind],
          bred,Ared,FSP_data->Mred,FSP_data->Nred,check);

if (DEBUG) {
   fprintf(check, "first solution \n");
   fmat_pr(check, "block", block);
    fmat_pr(check, "Asqr", Asqr);
    for (i=0; i<FSP_data->Mred-FSP_data->Nred;i++)
       fprintf(check, "here %d", Tackon[i]);
}
if (FSP_data->Nred==FSP_data->Mred)
   SystemComplete=TRUE;
else
   RestofSoln(FSP_data->Mred, FSP_data->Nred, NextToFind, block,
```

```
                FSP_data->g, bred, Ared, Tackon, FirstOK, check);

     if (DEBUG) {
       fmat_pr(check, "block", block);
       fmat_pr(check, "solutions", FSP_data->g);
       }
     }


    if (DEBUG)
      fmat_pr(check, "Xelim", Xelim);

        /* check if the eliminated b elements in bred can be       */
        /* produced from the g vectors and the reduced A matrix    */
        /* ie check that b is in the range of A                    */
    if (!SystemComplete)
    {
        fprintf(stderr,"System did not complete!\n");
        fprintf(check,"System did not complete!\n");
         binrange = 0;
    }
    else
       binrange = CheckRange(borig, Aorig, FSP_data->g, RowElim,
                         FSP_data->Mred);

    /* check that the original b was not all zeros (special case) */
    bcheck = CheckB(borig, M);

        /* check whether the original A had dependent rows  */
    if ((binrange) && (bcheck != 0) && (Exit_Status != RESTRICTED))
    {
      Rebuild_gs (FSP_data, ColElim, Specialg, NumSpg);
      Exit_Status = COMPLETE;
      for (i=0; (( i < N ) && (FSP_data->cn == 0)); i++)
      {
        if (RowElim[i] != 2)
          i++;
        else
          rank_lost_betas(FSP_data, Aorig, borig, RowElim, check);
      }
    }
    else if (Exit_Status != RESTRICTED)
    {
      Exit_Status = NOT_COMPLETE;
      if (bcheck != 0)
        fprintf(check, "\n\nB NOT IN RANGE OF A\n");
      else
        fprintf(check, "\n\n%s\n%s", "SPECIAL CASE\nB IS ZERO VECTOR",
                "CONSTRAINED OPTIMIZATION MUST BE USED\n");
    }

/* free up all variable memory space */
mat_free(Ared);
mat_free(Asub);
mat_free(bred);
mat_free(n);
mat_free(Specialg);
free(Tackon);
free(FirstOK);
```

```
return (Exit_Status);
}



/*
   ┌──────────────────────────────────────────────────────────────┐
   │ name:    RestofSoln                                          │
   │ description:  Finds the remaining solution vectors           │
   │               after the first has been selected.  It is      │
   │               recursive and thus calls itself when a         │
   │               valid solution is found, if this solution      │
   │               leads to a dead end, it will pop back to       │
   │               last solution and build from there.  This      │
   │               insures that all possible combinations of      │
   │               blocking patterns are available, given         │
   │               first one, in pattern which follows the        │
   │                algoritm presented in the article.            │
   │                                                              │
   │ inputs:  Mred, Nred, NextToFind, block, g, bred, Ared,       │
   │          Tackon, FirstOK                                     │
   │ outputs: block, g                                            │
   ├──────────────────────────────────────────────────────────────┤
   │ Authors:     Date:     Modifications:                        │
   │  g fries     2/95      Created                               │
   │  g fries     3/95      Addition of SpecialCase2 case         │
   └──────────────────────────────────────────────────────────────┘ */
void RestofSoln (int Mred, int Nred, int NextToFind, MATRIX *block,
                 MATRIX *g, MATRIX *bred, MATRIX *Ared, int *Tackon,
                 int *FirstOK, FILE *check)
{
int     i, j, k, /*Loop counters                                      */
        X, Y,    /*Position markers, keeping track of the current   */
                 /*column to be subbed in for and which one to sub  */
        nullity, /*Nullity sent to mat_null for dependent rows      */
       *Newtack, /*Temp array for tackon, sent to the next level of */
                 /*recursion                                        */
       *Changed; /*Temp array for FirstOK, sent to the next level of*/
                 /*recursion                                        */
MATRIX *n,       /*null space vectors                               */
       *Asub;    /*submatrix from square submatrix is found         */
float K2;

Newtack = (int *) malloc ( (M-N) * (sizeof( int )) );
Changed = (int *) malloc (  Nred * (sizeof( int )) );

/*Increment NextToFind so looking for next solution */
NextToFind++;
X = NextToFind -1;

/*Change FirstOK if a previously 0 value of g for a column in first
*/
/*blocking pattern has become nonzero
*/
for (i=0; i<Nred; i++)
  if ((!FirstOK[i]) &&
```

```
  (fabs(g->p[NextToFind-1][(int) block->p[NextToFind-1][i]]) > SMALL))
  {
     FirstOK[i] = TRUE;
  }

for (j=0; j<Nred; j++)
{
  if (DEBUG) {
     if (j==0) fprintf(check,"\nFIRST OK:  ");
     fprintf(check," %d ", FirstOK[j]);
     if (j==Nred-1) fprintf(check,"\n\n");
  }
  Changed[j] = FirstOK[j];
}

while ((X<(Mred-Nred))&&(!SystemComplete))
    {
    if (DEBUG)
      fprintf(check,"\n Next Solution to find is: %d \n", NextToFind);


    /*Set up next asub comparing the next remaining column to be
*/
    /*subbed in with the blocking pattern for that level of
recursion*/
    Asub = mat_malloc(Nred,Nred+1);
    for (i=0; i<Nred; i++)
       for (j=0; j<Nred; j++)
         Asub->p[j][i] = Ared->p[j][(int)block->p[(NextToFind-1)][i]];
    for (j=0; j<Nred; j++)
      Asub->p[j][Nred] = Ared->p[j][Tackon[X]];
    if (DEBUG) fmat_pr(check, "Asub", Asub);

    n = mat_malloc(Mred,Nred);
    mat_null(Asub, &nullity, n, &K2);
    if (DEBUG) fmat_pr(check, "n", n);
    Y=0;

    /*Loop which compares column to be subbed in w/ current blocking
*/
    /*pattern for dependency, if dependent it completes swap & valid
*/
    while ((Y<(Nred))&&(!SystemComplete))
       {
        if (DEBUG) {
           fprintf(check,"\n Compare column %d with %d \n", Tackon[X],
                                 (int)block->p[(NextToFind-1)][Y]);
         fprintf(check, "\n it is %f \n",fabs(n->p[Y][0]));
       }
       if ((fabs(n->p[Y][0]) > SMALL) && (FirstOK[Y]))
           {
           if (DEBUG)
             fprintf(check, "*************** %d *************** \n",
                                                    NextToFind);
           for (i=0; i<Nred;i++)
              block->p[NextToFind][i]=block->p[(NextToFind-1)][i];
           Newtack[0]=(int)block->p[NextToFind][Y];
           block->p[NextToFind][Y] =Tackon[X];
           for (i=0; i<X; i++)
              Newtack[i+1]=Tackon[i];
           for (j=X+1; j<(Mred-Nred); j++)
              Newtack[j]=Tackon[j];

BLOCK_COL_FIND_X(Newtack,g->p[NextToFind],block->p[NextToFind],
                                              bred,Ared,Mred,Nred,check);
           if (DEBUG)
             fprintf(check, "\n it will be %f \n",
                                 fabs(g->p[(NextToFind)]
                            [(int)block->p[NextToFind][Y]]));

           /*Check to ensure the created g vector is independent */
           /*of previously created ones in branch                */
           if (fabs(g->p[(NextToFind)][(int) block->p[NextToFind][Y]])
>SMALL)
              if (NextToFind <(Mred-Nred))
                  {
                  if (DEBUG) {
                     fmat_pr(check, "block", block);
                     fmat_pr(check, "solutions", g);
                         for (i=0; i<(Mred-Nred); i++)
                         fprintf(check, "Ord %d",Newtack[i]);
                   }
                  /*if more g vectors are needed,goes to next level*/
                  RestofSoln(Mred, Nred, NextToFind,block,g,bred,Ared,
                             Newtack, Changed, check);
                  }
              else
                  SystemComplete = TRUE;
              if (!SystemComplete)
                  block->p[NextToFind][Y]=Newtack[0];
              if (DEBUG) fprintf(check, "\n ******************** \n");
              }
        Y++;
        }
     X++;
     }

  free (Tackon);
  free (Changed);
}
```

```
/*
 ---------------------------------------------------------------------
 name:     Dependency
 description:  Returns whether or not the given two
               columns are dependent upon each other or
               not.  Essential for detection of an
               occurance of SpecialCase1.


 inputs:  Aorig, SLRow, Nred, first, second
 outputs: whether columns SLRow[first] and SLRow[second]
          are dependent
 ---------------------------------------------------------------------
 Authors:      Date:     Modifications:
  g fries      3/95      Created
 ---------------------------------------------------------------------
*/

int Dependency (MATRIX *Atemp, int *SLRow, int Nred, int first,
                                int second)
{
int i,STOP;
float devidor;
i=0;
devidor=0;
STOP=FALSE;
while (i<Nred && !STOP)
    if ((fabs(Atemp->p[SLRow[i]][first])<SMALL) &&
                            (fabs(Atemp->p[SLRow[i]][second])<SMALL))
       i++;
    else if ((fabs(Atemp->p[SLRow[i]][first])>SMALL) &&
                            (fabs(Atemp->p[SLRow[i]][second])>SMALL))
       if (devidor==0)
           {
           devidor= (Atemp->p[SLRow[i]][first]/
                        Atemp->p[SLRow[i]][second]);
           i++;
           }
       else if (fabs(Atemp->p[SLRow[i]][first]-
                      (Atemp->p[SLRow[i]][second]*devidor))<SMALL)
           i++;
       else
           STOP=TRUE;
    else
       STOP=TRUE;
if (STOP)
    return(FALSE);
else
    return(TRUE);
}
```

```
/*
 ---------------------------------------------------------------------
 name:     BLOCK_COL_FIND_X
 description:  This procedure calculates a g vector for
               a given square blocking pattern.


 inputs:  Tackon, g, block, bred, Ared, Mred, Nred
 outputs: new g vector
 ---------------------------------------------------------------------
 Authors:      Date:     Modifications:
  k morganson  8/94      Created
  g fries      2/95      Adaption for new blocking
                         strategy (keeping blocked col)
 ---------------------------------------------------------------------
*/

BLOCK_COL_FIND_X        (int *Tackon, float *g, float *block,
                         MATRIX *b, MATRIX *A,
                         int Mred, int Nred, FILE *check)
{
int r, I, i, j,k;
float blocktemp[Nred], temp;
MATRIX *Atemp, *gtemp, *btemp;

Atemp = mat_malloc(Nred,Nred);
gtemp= mat_malloc(Nred,1);
btemp = mat_malloc(Nred,1);

for (i=0; i<Nred; i++)
   btemp->p[i][0]=b->p[i][0];
/* the columns blocked need to be listed from smallest to largest */
for (i=0; i<(Nred); i++)
   blocktemp[i]=(int)block[i];

for (i=(Nred-1); i>0; i--)
   for (j=i-1; j>=0; j--)
      if (blocktemp[i]<blocktemp[j])
         {
         temp=blocktemp[i];
         blocktemp[i]=blocktemp[j];
         blocktemp[j]=temp;
         }

   k=0; I=0;
   for (i=0; i<Mred; i++)
      if (blocktemp[k] == i)
         {
         for (j=0; j<Nred; j++)
            Atemp->p[j][k] = A->p[j][i];
       k++;
         }
      else
       I++;
if (DEBUG) fmat_pr(check, "asqr", Atemp);

mat_LU_inv(Atemp);

gtemp = mat_mul2(Atemp, btemp);

if (DEBUG) {
  fprintf(check, "\n gvector");
```

```
  for (i=0; i<Nred; i++)
    fprintf(check, "%f    ", gtemp->p[i][0]);
}

/*Mow add a zero to where column was blocked */

j=0;
I=0;
for (i=0; i<Nred; i++)
   g[(int)blocktemp[i]]=gtemp->p[i][0];
for (j=0; j<(Mred-Nred); j++)
     g[Tackon[j]]=0.0e0;

mat_free(Atemp);
mat_free(gtemp);
mat_free(btemp);
}


/*
   name:     ReduceA
   description:  Restricted work space motions can be
                 identified by rows of the A which only
                 have one nonzero element.  Since the
                 corresponding column must be present in
                 all final joint space solutions, the
                 appropriate joint space motion will be
                 calculated before any redundancy
                 resolution is performed, and the
                 appropriate motions and joints will be
                 removed from the work space and A
                 respectively.  Also such cases as
                 dependent rows, and SpecialCase1 must be
                 identified and dealt with.


   inputs:  Tackon, g, block, bred, Ared, Mred, Nred
   outputs: new g vector

   Authors:      Date:     Modifications:
    k morganson 8/94       Created
    g fries     2/95       Adaption for new blocking
                           strategy and handling of dep row
    g fries     3/95       Recognition and handling of
                           SpecialCase1 and cont reduction
*/

ReduceA(Solutions *FSP_data, MATRIX *Aorig, MATRIX *Ared, MATRIX
*borig,
      MATRIX *bred, MATRIX *Xelim, int* ColElim,int* RowElim,
       int *NumSpg, MATRIX *Specialg)

{
int i,j,k,m,r,p,    /*loop counters                                */
    StillChecking, /*flag to mark when all nonredundancies are go*/
    Stop,Stop2,    /*Used as loop flag n detection 4 SpecialCase1*/
    ClassANum,     /* # of ClassificationA columns, for SpecialCa*/
    reference,     /*Reference position used for SpecialCase2 row*/
    LastNred,      /*Check if all rows have been looked at       */
    nullity,       /*Used in mat_null when determining depend row*/
    nul_count,     /*Loop counter for dependent rows             */
    Restriction,   /*num of joints which contrib to a work space */
    DoneRed,       /*Keeps track if reduction is done loop execut*/
                   /*without any reduction                       */
    *SLRow,        /*Stands for Still Left Row, stores remaining */
                   /*eliminated rows in sequence                 */
    *SLCol,        /*Stands 4 Still Left Column, stores remaining*/
                   /*eliminated columns in sequence              */
    *ClassA,       /*Array used to store ClassificationA columns */
    count,count2,  /*Position markers when copying over SLCol whe*/
    position,      /*  SpecialCase1 has been found               */
    nonzerocol,    /* # of nonzero col when check for SpecialCase*/
    *Nonzero,      /*Stores values of nonzero col when check for */
    NonZeroCol,    /*Column which has nonzero element for give ro*/
    Flag,          /*Used in SC2 for an undefined beta value     */
    Goin;          /*"Go in" for entance into ClassA for SC2     */
double btemp[N];   /*Holds the b vector as it is modified by back*/
float K2,          /*Matrix condition number:  sv_max/sv_min     */
      *Constant,   /*Constant values used when comparing Classifi*/
                   /*columns                                     */
       beta;       /*Value used for determining SpecialCase2     */
MATRIX *n,         /*null space vectors                          */
       *Atrans,    /*Transpose matrix used when looking 4 depend r*/
       *AElim;     /*Matrix used 2 find null vectors when dealing */
                   /* SpecialCase1                               */

     /* allocation of memory */
SLRow   = (int   *) malloc ( (N) * (sizeof( int )) );
SLCol   = (int   *) malloc ( (M) * (sizeof( int )) );
Nonzero = (int   *) malloc ( (M) * (sizeof( int )) );
ClassA  = (int   *) malloc ( (M) * (sizeof( int )) );
Constant= (float *) malloc ( (N) * (sizeof( int )) );

n = mat_malloc(M,M);

     /* initialize variables */
FSP_data->Nred = N;     /* number of rows in the reduced A       */
FSP_data->Mred = M;     /* number of columns in the reduced A       */
*NumSpg=0;
for (i=0; i<N; i++)
   {
   SLRow[i]=i;
   RowElim[i]=0;
   btemp[i]=borig->p[i][0];
   }
for (i=0; i<M; i++)
   {
   SLCol[i]=i;
   Xelim->p[i][0]=0;
   ColElim[i]=0;
   }
DoneRed=FALSE;


     /* start reduction  */
while (!DoneRed)
    {
    DoneRed=TRUE;
    StillChecking = 1;
```

```
/* Check each row for the number of nonzero elements.  If only one  */
/* element is nonzero, solve for the joint space motion, and         */
/* modify the b vector so that the appropriate row and column        */
/* can be eliminated from the A.  After a row is eliminated          */
/* the remaining A must be rechecked for any new restrictions        */
    while (StillChecking)
        {
        LastNred = FSP_data->Nred;
        for (i=0; i<FSP_data->Nred; i++)
            {
            j=-1;
            Restriction=0;

            /* check for nonzero row elements */
            while ((j<(FSP_data->Mred-1)) && (Restriction < 2))
                {
                j++;
                if (fabs(Aorig->p[SLRow[i]][SLCol[j]]) > SMALL)
                    {
                    Restriction++;
                    NonZeroCol = j;
                    }
                }

            /* if a row only has 1 nonzero element, eliminate it from
*/
            /* the A                                                 */
            if ((Restriction == 1) && (RowElim[SLRow[i]] != 1))
                {
                Xelim->p[SLCol[NonZeroCol]][0]=
                  btemp[SLRow[i]]/
Aorig->p[SLRow[i]][SLCol[NonZeroCol]];
                for (k=0; k<FSP_data->Nred; k++)
                  btemp[SLRow[k]] =  btemp[SLRow[k]] -
                                        Aorig->p[SLRow[k]][SLCol
                      [NonZeroCol]]*Xelim->p[SLCol[NonZeroCol]][0];
                ColElim[SLCol[NonZeroCol]]=1;
                RowElim[SLRow[i]]=1;
                for (j=NonZeroCol; j<(FSP_data->Mred-1); j++)
                    SLCol[j]=SLCol[j+1];
                for (j=i; j<(FSP_data->Nred-1); j++)
                    SLRow[j]=SLRow[j+1];
                FSP_data->Nred--;
                FSP_data->Mred--;
                }
            else            /* row of all zeros, also eliminated */
            if ((Restriction == 0) && (RowElim[SLRow[i]] != 1))
                {
                RowElim[SLRow[i]]=1;
                for (j=i; j<(FSP_data->Nred-1); j++)
                    SLRow[j]=SLRow[j+1];
                FSP_data->Nred--;
                }
            }
        if (FSP_data->Nred == LastNred)
        StillChecking = 0;
        }


     /* check for columns of zeros */
    for (i=0; i<FSP_data->Mred; i++)
        {
        j=0;
        while ((j<FSP_data->Nred)&&(fabs(Aorig->p[SLRow[j]][SLCol[i]])
<                                                                SMALL))
            j++;
        if (j==FSP_data->Nred)
            {
            ColElim[SLCol[i]]=1;
            for (j=i; j<(FSP_data->Mred-1); j++)
               SLCol[j]=SLCol[j+1];
            FSP_data->Mred--;
            i--;
            }
        }

     /* check for dependent rows */
    if (FSP_data->Mred !=0)
    {Atrans = mat_malloc(FSP_data->Mred,FSP_data->Mred);
    for (i=0; i<FSP_data->Nred; i++)
       for (j=0; j<FSP_data->Mred; j++)
           Atrans->p[j][i] = Aorig->p[SLRow[i]][SLCol[j]];
    for (i=FSP_data->Nred; i<FSP_data->Mred; i++)
       for (j=0; j<FSP_data->Mred; j++)
           Atrans->p[j][i] = 0;
     for (i=0; i<FSP_data->Nred; i++)
       for (j=FSP_data->Mred; j<FSP_data->Mred; j++)
           Atrans->p[j][i] = 0;
    mat_null(Atrans, &nullity, n, &K2);
    nullity=nullity+FSP_data->Nred-FSP_data->Mred;
    nul_count=0;
    while (nul_count < nullity)
        {
        i=FSP_data->Mred-FSP_data->Nred +nul_count;
        j=0;
        while (fabs(n->p[j][i]) < SMALL)
            j++;
        if (RowElim[SLRow[j]] != 1)
            {
            RowElim[SLRow[j]] = 2;
            for (k=j; k<(FSP_data->Nred-1); k++)
               SLRow[k]=SLRow[k+1];
            FSP_data->Nred--;
            }
            nul_count++;
            }
    mat_free(Atrans);}


     /* check for SpecialCase1 */
    for (i=0;(i<FSP_data->Nred);i++)
       if (fabs(btemp[SLRow[i]])<SMALL)
            {
            nonzerocol=0;
            j=0;
            Stop =FALSE;
            while (j<(FSP_data->Mred-1) && !Stop)
               if (fabs(Aorig->p[SLRow[i]][SLCol[j]])>SMALL)
                    {
```

```
            k=j+1;
            while (!Stop && (k<FSP_data->Mred))
               if (fabs(Aorig->p[SLRow[i]][SLCol[k]])>SMALL)
                  if (Dependency(Aorig, SLRow, FSP_data->Nred,
                                 SLCol[j], SLCol[k]))
                     k++;
                  else
                     Stop=TRUE;
               else
                   k++;
            Nonzero[nonzerocol]=SLCol[j];
            nonzerocol++;
            j++;
            }
         else
            j++;
      if (fabs(Aorig->p[SLRow[i]][SLCol[(FSP_data->Mred-1)]])
>SMALL)
         {
         Nonzero[nonzerocol]=SLCol[(FSP_data->Mred-1)];
         nonzerocol++;
         }
      if (nonzerocol==1)
         Stop=TRUE;
      if (!Stop) /* Discovered a SpecialCase1 */
         {
         DoneRed=FALSE;
         for (r=0;r<nonzerocol;r++)
            ColElim[Nonzero[r]]=2;
         count=0;
         count2=0;
         for (position=0; position<FSP_data->Mred; position++)
            if (SLCol[position]!=Nonzero[count2])
               {
               SLCol[count]=SLCol[position];
               count++;
               }
            else
               count2++;
         /* Construct new g matrix */
         AElim = mat_malloc(1, nonzerocol);
         for (r=0; r<nonzerocol;r++)
            AElim->p[0][r]=Aorig->p[SLRow[i]][Nonzero[r]];
         fmat_pr(stderr, "AElim", AElim);
         mat_null(AElim, &nullity, n, &K2);
         fmat_pr(stderr, "n", n);
         mat_free(AElim);
         FSP_data->Mred=FSP_data->Mred-nonzerocol;
         RowElim[SLRow[i]]=1;
         for (r=i; r<(FSP_data->Nred-1); r++)
            SLRow[r]=SLRow[r+1];
         FSP_data->Nred--;
         for (m=0; m<(nonzerocol-1);m++)
            {
            for (r=0; r<M; r++)
               Specialg ->p[*NumSpg][r]=0;
            for (r=0; r<nonzerocol;r++)
               Specialg ->p[*NumSpg][Nonzero[r]]=n->p[r][m];
            *NumSpg=*NumSpg+1;
            }
```

```
            }
         }
      }

   /*Check for SpecialCase2*/
   i=0;
   Stop2=FALSE;
   while ((i<(FSP_data->Nred-1)) && (!Stop2))
      {
      j=i+1;
      while ((j<FSP_data->Nred) && (!Stop2))
         {
         Stop=FALSE;
         Flag=FALSE;
         if (fabs(btemp[SLRow[i]])>SMALL)
            beta= btemp[SLRow[j]]/ btemp[SLRow[i]];
         else
            Flag=TRUE;
         ClassANum=0;
         k=0;
         reference=i;
         while ((!Stop) && (k<FSP_data->Mred))
            {
            Goin=FALSE;
            if (fabs(Aorig->p[SLRow[i]][SLCol[k]])<SMALL)
               if (!Flag)
                  Goin=TRUE;
               else;
            else
               {
               if (fabs((Aorig->p[SLRow[j]][SLCol[k]]/
                     Aorig->p[SLRow[i]][SLCol[k]])-beta)>SMALL)
                  Goin=TRUE;
               if (Flag)
                  Goin=TRUE;
               }
            if (Goin)
               {
               ClassA[ClassANum]=SLCol[k];
               if (ClassANum==0)
                  {
                  if (fabs(Aorig->p[SLRow[reference]][ClassA[0]])
<SMALL)
                     {
                     reference=j;
                     if (fabs
(Aorig->p[SLRow[reference]][ClassA[0]])
                                                        <SMALL)
                        Stop=TRUE;
                     }
                  if (!Stop)
                     for (r=0;r<FSP_data->Nred;r++)
                        Constant[r]=Aorig->p[SLRow[r]][ClassA[0]]/
                            Aorig->p[SLRow[reference]][ClassA[0]];
                  }
               else if
```

```
(fabs(Aorig->p[SLRow[reference]][ClassA[ClassANum]])
                                                        <SMALL)
                    Stop=TRUE;
                 else
                    {
                    r=0;
                    while ((r<FSP_data->Nred) && (!Stop))
                        {
                        if
                        (fabs((Aorig->p[SLRow[r]][ClassA[ClassANum]]/
Aorig->p[SLRow[reference]][ClassA[ClassANum]])
-Constant[r])>SMALL)
                            Stop=TRUE;
                        r++;
                        }
                    }
                 ClassANum++;
                 }
              k++;
              }
           if (!Stop)
              Stop2=TRUE;
           j++;
           }
        i++;
        }
     if (Stop2)
        {
        DoneRed=FALSE;
        fprintf(stderr,"\nYes we have SC2 in rows %d and %d\n",
          SLRow[i-1], SLRow[j-1]);
        fprintf(stderr,"\n Eliminate row %d \n", SLRow[reference]);
        /* Construct new g matrix */
        AElim = mat_malloc(1, ClassANum);
        for (p=0; p<ClassANum;p++)
            AElim->p[0][p]=Aorig->p[SLRow[reference]][ClassA[p]];
        fmat_pr(stderr, "AElim", AElim);
        mat_null(AElim, &nullity, n, &K2);
        fmat_pr(stderr, "n", n);
        free(AElim);
        for (p=0; p<(ClassANum-1);p++)
            {
            for (r=0; r<FSP_data->M; r++)
               Specialg ->p[*NumSpg][r]=0;
            for (r=0; r<ClassANum;r++)
               Specialg ->p[*NumSpg][ClassA[r]]=n->p[r][p];
            *NumSpg=*NumSpg+1;
            }
        FSP_data->Nred--;
        RowElim[SLRow[reference]]=1;
        for (r=0;r<ClassANum;r++)
            ColElim[ClassA[r]]=3;
        for (r=reference;r<FSP_data->Nred;r++)
            SLRow[r]=SLRow[(r+1)];
        count=0;
        count2=0;
        for (position=0; position<FSP_data->Mred; position++)
            if (SLCol[position]!=ClassA[count2])
               {
               SLCol[count]=SLCol[position];
               count++;
               }
            else
               count2++;
          FSP_data->Mred=FSP_data->Mred-ClassANum;
          }
     }


     /* store the reduced A in the variable Ared, and the       */
     /* modified b in bred                                      */
j=-1;
for (i=0; i<N; i++)
  if (RowElim[i] == 0)
     {
     j++;
     bred->p[j][0]=btemp[i];
     m=-1;
     for (k=0; k<M; k++)
       if (ColElim[k] == 0)
          {
          m++;
          Ared->p[j][m] = Aorig->p[i][k];
          }
     }
mat_free(n);
free (SLRow);
free (SLCol);
free (Nonzero);
}


/*
   ┌──────────────────────────────────────────────────────────┐
   │ name:    CheckB                                          │
   │ description:  Checks that after reducing the A, not all  │
   │               requested workspace motions are zero       │
   │               (trivial motion).                          │
   │                                                          │
   │ inputs:  b, corresponding N                              │
   │ outputs: If it is all zeros                              │
   ├──────────────────────────────────────────────────────────┤
   │ Authors:     Date:     Modifications:                    │
   │  k morganson 8/94      Created                           │
   └──────────────────────────────────────────────────────────┘ */
int CheckB       (MATRIX *b, int m)
{
int i;

i=0;
      /* stop checking b when a nonzero element is found */
while ((i<m) && (fabs(b->p[i][0]) < SMALL))
   i++;

if (i==m)
   return(0);
```

```
else
   return(m);
}

/*
   ┌──────────────────────────────────────────────────────────────┐
   │ name:     CheckRange                                         │
   │ description:  Checks that the original b is in the           │
   │               range of A.                                    │
   │                                                              │
   │ inputs:  borig, Aorig, g, RowElim, Mred                      │
   │ outputs: If it is in range                                   │
   ├──────────────────────────────────────────────────────────────┤
   │ Authors:      Date:      Modifications:                      │
   │  k morganson 8/94        Created                             │
   │  c hacker    10/94       Changed from scaled to constant     │
   └──────────────────────────────────────────────────────────────┘ */
int CheckRange (MATRIX * b, MATRIX * Aorig, MATRIX * g,
                int   *RowElim,    int    Mred )
{
  int i, j, k;
  double CheckValue, CheckTemp, CheckTemp2;

  for (i = 0; i < (Aorig -> rows); i++)
  {
    if (RowElim[i] == 2)
    {
      for (j = 0; j < (g -> rows); j++)
      {
        CheckValue = 0.0;
       for (k = 0; k< (Aorig -> cols); k++)
         CheckValue += Aorig -> p[i][k] * g -> p[j][k];
        CheckTemp  = fabs (b->p[i][0] - CheckValue);
        CheckTemp2 = .01;
       if (CheckTemp > CheckTemp2)
         return (0);
      }
    }
  }
  return (1);
}

/*
   ┌──────────────────────────────────────────────────────────────┐
   │ name:     Rebuild_gs                                         │
   │ description:  Given the created g vectors for the            │
   │               reduced Jacobian, this procedure adds          │
   │               zeros in eliminated columns and tacks on       │
   │               the g vectors created by SpecialCase1          │
   │                                                              │
   │ inputs:  g, ColElim, Mred, Nred, M, NumSpg, Specialg         │
   │ outputs: g                                                   │
   ├──────────────────────────────────────────────────────────────┤
   │ Authors:      Date:      Modifications:                      │
   │  c hacker     3/95       Created                             │
   │  g fries      3/95       Additional ability handling         │
   │                          SpecialCase1 (ie Specialg)          │
   └──────────────────────────────────────────────────────────────┘ */

void Rebuild_gs (Solutions *FSP_data, int *ColElim,
                 MATRIX *Specialg,int NumSpg)
{
  int    i,j,k;
  MATRIX *gtemp;

  gtemp = mat_cp2(FSP_data->g);

  k = 0;
  for (j = 0; j < FSP_data->M; j++)
  {
    if (ColElim[j] != 0)
    {
       for (i=0;i<FSP_data->g->rows;i++)
        FSP_data->g->p[i][j] = ZERO;
    }
    else
    {
       for (i=0;i<FSP_data->g->rows;i++)
        FSP_data->g->p[i][j] = gtemp->p[i][k];
       k++;
    }
  }

  /*Addition of null vectors created by SpecialCase1*/
  for (i=(FSP_data->Mred-FSP_data->Nred+1);i<
                    (FSP_data->Mred-FSP_data->Nred+1+NumSpg);i++)
    for(j=0;j<FSP_data->M; j++)
       if (ColElim[j] !=0)
          FSP_data->g->p[i][j]=

Specialg->p[(i-FSP_data->Mred+FSP_data->Nred-1)][j];
          else
             FSP_data->g->p[i][j]=FSP_data->g->p[0][j];

  mat_free(gtemp);
}

/*
   ┌──────────────────────────────────────────────────────────────┐
   │ name:     GetData                                            │
   │ description:    read in A and b vector to be used for        │
   │     testing the code with a single jacobian and dx. The      │
   │     macro DEBUG_FSP must be defined in the file:             │
   │                    general.h.                                │
   │                                                              │
   │ inputs:  A file called : FSP_data                            │
   │ outputs: Aorig, borig                                        │
   ├──────────────────────────────────────────────────────────────┤
   │ Authors:      Date:      Modifications:                      │
   │  K Morgansn   6/94       Created                             │
   │  c hacker     3/95       Removed struct from header          │
   └──────────────────────────────────────────────────────────────┘ */
GetData       ( MATRIX *A,  MATRIX *b)
{
  FILE *fpin;
  int i, j;
```

```
  fpin = fopen("FSP_data", "r");

  for (i=0; i<N; i++)
    for (j=0; j<M; j++)
      fscanf(fpin, "%f", &A->p[i][j]);

  for (i=0; i<N; i++)
    fscanf(fpin, "%f", &b->p[i][0]);

  fclose(fpin);
}
```

CRITERIA.C

```
/*
 ------------------------------------------------------------
 Program Name:   Criteria.c
 ------------------------------------------------------------
 description: This file contains the declarations for
    different criteria, such as Least Norm, Force,
    Torque, or strength optimization.

 Procedures:
    Least_Norm() : H is ZERO, and B is zero.
 ------------------------------------------------------------ */

#include "general.h"        /* Header File to link all others */

/*
 ------------------------------------------------------------
 name:    Least_Norm()
 description:  Form Least Norm Motion, the reference
     vector Zr is zero and therefore H is zero.  The B
     vector is also zero.

 inputs:  none
 outputs: H is the zero matrix, B is the zero vector
 ------------------------------------------------------------
 Authors:      Date:     Modifications:
 c hacker       3/95    Created
 ------------------------------------------------------------ */

int Least_Norm (MATRIX *B, MATRIX *H, FILE* datafp)
{
  int i, j;

  for (i=0; i<B->rows; i++) B->p[i][0] = ZERO;


  if (DEBUG) {
    fmat_pr(datafp,"Least Norm B", B);
    fmat_pr(datafp,"Least Norm H", H);
  }
}
```

CONSTRAINTS.C

```
/*
 ------------------------------------------------------------
 Program Name:   constraints.c
 ------------------------------------------------------------
 description: Determines if constraints are necessary
    for the system and if so calculates necessary betas.

 Procedures:
    avoid_limits     : finds limit breach & calls beta
    avoid_obstacles  : finds obs   breach & calls beta
    find_intersection_sphere : As named
 ------------------------------------------------------------ */
```

```
#include "general.h"      /* Header File to link all others */

/*
   ┌──────────────────────────────────────────────────────────┐
   │ name:    avoid_limits                                    │
   │ description:  Loops through all joint limits to find     │
   │     out if a joint has reached either an upper or a      │
   │     a lower limit.  If one has been reached and          │
   │     solution vectors have yet to be found, solution      │
   │     vectors are calculated and betas produced, other-    │
   │     wise, if solutions have already been found, the      │
   │     find beta function is called.                        │
   │                                                          │
   │ inputs:  See Declarations below                          │
   │ outputs: Calls functions to calculate betas.             │
   ├──────────────────────────────────────────────────────────┤
   │ Authors:     Date:     Modifications:                    │
   │  F Tulloch     4/94    Algorithm Creation for 2 g's      │
   │  c hacker     10/94    Ported from 2 g's to FSP          │
   │                2/95    Ported to Solutions struct        │
   └──────────────────────────────────────────────────────────┘ */

int avoid_limits(FSP_data, Jacob, dx, got_gs, datafp)
FILE   *datafp;          /* data file declaration            */
int    *got_gs;          /* get_gs or not, used for  TWOSTEP */
MATRIX *Jacob,           /* needed to find soln's if TWOSTEP */
       *dx;              /* needed to find soln's if TWOSTEP */
Solutions *FSP_data;     /* see structures.h                 */
{
  int    qchk, ii,
         bound = FALSE;
  double distance_2_move;

  if (DEBUG) {
    fprintf(datafp,"\n                                        \n");
    fprintf(datafp,"\n      JOINT LIMIT AVOIDANCE             \n");
    fprintf(datafp,  "                                        \n\n");
  }

  for (qchk = 0; qchk < M; qchk++)
  {
            /*  limit_upper... */
    if  (rad(Robot->Angles[qchk].Min_limit) >
FSP_data->Qarray->p[qchk][0])
    {
      bound = TRUE;
      distance_2_move =
        rad(Robot->Angles[qchk].Min_limit) -
          FSP_data->Qarray->p[qchk][0];
    }                          /*  limit_upper... */

    else      /*  limit_lower... */
    if (rad(Robot->Angles[qchk].Max_limit) <
          FSP_data->Qarray->p[qchk][0])
    {
      bound = TRUE;
      distance_2_move =
       rad(Robot->Angles[qchk].Max_limit) -
                FSP_data->Qarray->p[qchk][0];
    }                          /*  limit_lower... */

    if (bound)
    {
      bound = FALSE;
      #ifdef TWOSTEP
        if (!(*got_gs))
        {
          GET_JACOBIAN(Jacob, FSP_data->Qarray);
          Solution_generator ( FSP_data, Jacob, dx, datafp );
        }
        *got_gs = TRUE;
      #endif

      #ifdef DEBUG_OUT
        fprintf(stderr, "   joint(1-%d) #%i limited\n   ", M, qchk+1);
      #endif
      if (DEBUG) {
        fprintf(datafp, "\n\njoint # %i out of range by %lf ",
                qchk+1, deg(distance_2_move));
      }
      find_jl_beta(FSP_data, qchk, distance_2_move, datafp);
    }                          /* end if(bound)        */
  }                            /* end for(qchk...)     */
}                    /* end avoid_limits()       */

/*
   ┌──────────────────────────────────────────────────────────┐
   │ name:    avoid_obstacles                                 │
   │ description:  Loops through all obstacles and calls      │
   │     routines to determine intersection. If an obstacle   │
   │     has been breached and solution vectors have yet to   │
   │     be found, solution vectors are calculated and        │
   │     betas are produced, otherwise, if solutions have     │
   │     already been found, only the find beta function is   │
   │     called.                                              │
   │                                                          │
   │ inputs:  See Declarations below                          │
   │ outputs: Calls Function to Calculate Betas               │
   ├──────────────────────────────────────────────────────────┤
   │ Authors:     Date:     Modifications:                    │
   │  F Tulloch     4/94    Algorithm Creation for 2 g's      │
   │  c hacker     10/94    Ported from 2 g's to FSP          │
   │                2/95    Ported to Solutions struct        │
   └──────────────────────────────────────────────────────────┘ */

int avoid_obstacles(FSP_data, Jacob, dx, x_of_link, ns, spheredata,
                    got_gs, datafp)
int     ns;                    /* circ info (# of circles) */
double  spheredata[4][4];      /* circ info (rad, center)  */
FILE   *datafp;          /* data file declaration          */
int    *got_gs;          /* get_gs or not, used for  TWOSTEP */
MATRIX *Jacob,           /* needed to find soln's if TWOSTEP */
       *dx,              /* needed to find soln's if TWOSTEP */
       *x_of_link;       /* link endpts                      */
Solutions *FSP_data;     /* see structures.h                 */
{
  int    i, kk, is, ii, elbow_check;
  double newl, delta,
         pt1[3], pt2[3], Xj[3], normal[3];

  #ifdef TWOSTEP
```

```
  GET_ACTUAL_X   ( FSP_data->Qarray, x_of_link );
#endif

if (DEBUG) {
  fprintf(datafp,"\n ____________________________________ \n");
  fprintf(datafp,"\n          OBSTACLE AVOIDANCE          \n");
  fprintf(datafp,  " ____________________________________ \n\n");
  fmat_pr(datafp," Position of Each Link ", x_of_link);
}

for (kk = 0; kk < Robot->NL; kk++)
{
  for (i=0; i<3; i++)
  {
    pt1[i] = x_of_link->p[kk][i];
    pt2[i] = x_of_link->p[kk+1][i];
  }
  if (DEBUG) {
    fprintf (datafp,"\nChecking LINK(0-4) %d\n", kk);
    fprintf (datafp," pt1 = < %f, %f, %f >\n pt2 = < %f, %f, %f >",
            pt1[0],pt1[1],pt1[2],pt2[0],pt2[1],pt2[2]);
  }

  for (is = 0; is < ns; is++)
  {
    if (find_intersection_sphere(pt1, pt2, is, ns, spheredata,
            &elbow_check, &newl, &delta, normal, datafp))
    {
      #ifdef TWOSTEP
        if (!(*got_gs))
        {
          GET_JACOBIAN(Jacob, FSP_data->Qarray);
          Solution_generator( FSP_data, Jacob, dx, datafp );
        }
        *got_gs = TRUE;
      #endif
      for (i=0; i<3; i++)
        Xj[i]= x_of_link->p[kk][i] + newl*x_of_link->p[kk+1][i];


      #ifdef DEBUG_OUT
        fprintf (stderr,"   LINK %d COLLIDED\n       ", kk);
      #endif
      if (DEBUG) fprintf (datafp,"\tXj  = < %f,  %f,  %f   >\n",
                                        Xj[0],  Xj[1],  Xj[2]);

      find_obs_beta(FSP_data, kk, &newl, delta, normal, datafp);

    }                           /* end if (intersection) */
  }                         /* end for(is=0 to ns)  */
}             /* end kk loop */
}     /* end avoid_obstacles() */
/*
 ____________________________________________________
| name:    find_intersection_sphere                  |
| description:  For a given obstacle and link, determines
|     if that link is within the obstacles buffer zone.
|     If so, returns closest point on the link to the
|     center of the object, the normal vector from the
|     center to that point, and the distance needed in
|     to move said point to the edge of the buffer zone.
|     This information is critical to the formation of
|     of beta values.
|
| inputs:  link (pt1,pt2), sphere
| outputs: newl, delta, normal
|
| Authors:     Date:     Modifications:
|  F Tulloch     4/94    Algorithm Creation for 2 g's
|  c hacker      2/95    Ported to Solutions struct
|____________________________________________________*/

int find_intersection_sphere(pt1, pt2, ws, ns, spheredata,
       elbow_check, newl, delta, normal, datafp)
FILE   *datafp;           /* data file declaration         */
int     ws;               /* which sphere                  */
int    *elbow_check;      /* check if elbow of link inside */
double  normal[3],        /* normal from center of sphere  */
       *newl,             /* closest pnt j to cntr of obs  */
       *delta,            /* distance pnt j is in boundary */
        pt1[3], pt2[3];   /* endpoints of link             */
int     ns;               /* sphere info (# spheres)       */
double  spheredata[4][4];/* sphere info (rad, cntr)        */
{
  int    i, j;
  double  x_temp,  y_temp,  z_temp,
          x_mid,   y_mid,   z_mid,
          xcenter, ycenter, zcenter, radius,
          deltax,  deltay,  deltaz,
          b_check1, b_check2, root_chk,
          coeff_t_sq, coeff_t_f,
          t_sol_init, t_sol1, t_sol2,
          xsol1, xsol2,  ysol1, ysol2,  zsol1, zsol2,
          constant1, constant2, constant3, constantf1, constantf;

  xsol1 =ysol1 =zsol1 =x_temp =y_temp =z_temp =0.0;

  xcenter = spheredata[ws][0];
  ycenter = spheredata[ws][1];
  zcenter = spheredata[ws][2];
  radius  = spheredata[ws][3];

  /* defining constants needed in forming equations */
  deltax  = pt2[0] - pt1[0];
  deltay  = pt2[1] - pt1[1];
  deltaz  = pt2[2] - pt1[2];

  coeff_t_sq = (deltax * deltax) + (deltay * deltay) +
                 (deltaz * deltaz);


  coeff_t_f = 2 * ( deltax * (pt1[0] - xcenter) +
                    deltay * (pt1[1] - ycenter) +
                    deltaz * (pt1[2] - zcenter) );

  constant1  = pt1[0] - xcenter;
  constant2  = pt1[1] - ycenter;
  constant3  = pt1[2] - zcenter;
  constantf1 =  (constant1 * constant1) +
               (constant2 * constant2) +
               (constant3 * constant3);
```

```
  constantf = constantf1 - SQUARE(radius);

  /* now that coefficients of the quadratric are defined,solve it */

  b_check1 = SQUARE(coeff_t_f);
  b_check2 = coeff_t_sq * constantf * 4;
  root_chk = b_check1 - b_check2;

        /* now to ascertain what situation we have */
  if (root_chk <  0.0) return FALSE;            /* no intersection */
  if (root_chk == 0.0) return FALSE;            /* graze */


       /* root_chk > 0.0 intersection occurs */
  t_sol_init = sqrt(root_chk);
  t_sol1 = (-(coeff_t_f) + t_sol_init) / (2 * coeff_t_sq);
  t_sol2 = (-(coeff_t_f) - t_sol_init) / (2 * coeff_t_sq);

  if (DEBUG) fprintf(datafp,"\tt_sol1:%4f
t_sol2:%4f\n",t_sol1,t_sol2);

  /************      now check t's for validity      ************/

          /* Does pt of inters. lie on link? */
  if (((0.0 > t_sol1) || (t_sol1 > 1.0)) &&
      ((0.0 > t_sol2) || (t_sol2 > 1.0)))
  {
    if (DEBUG) fprintf(datafp,"\tIntersection not on link \n");
    return FALSE;
  }

          /* Does pt of inters. elbow sit. */
  if (((0.0 > t_sol1) || (t_sol1 > 1.0)) ||
      ((0.0 > t_sol2) || (t_sol2 > 1.0)))
  {
    if (DEBUG) fprintf(datafp, "\tElbow Intersection!\n");
   *elbow_check = *elbow_check + 1;

    if (0.0 <= t_sol1 <= 1.0)
    {
      xsol1 = pt1[0] + (deltax * t_sol1);
      ysol1 = pt1[1] + (deltay * t_sol1);
      zsol1 = pt1[2] + (deltaz * t_sol1);
    }

    if (0.0 <= t_sol2 <= 1.0)
    {
      xsol1 = pt1[0] + (deltax * t_sol2);
      ysol1 = pt1[1] + (deltay * t_sol2);
      zsol1 = pt1[2] + (deltaz * t_sol2);
    }

    if (*elbow_check == 1)
    {
      x_temp = xsol1;           /* end of link becomes point j */
      y_temp = ysol1;
      z_temp = zsol1;
      return FALSE;
    }
    else if (*elbow_check % 2 == 0)
    {
      x_mid = (xsol1 + x_temp) / 2;
      y_mid = (ysol1 + y_temp) / 2;
      z_mid = (zsol1 + z_temp) / 2;

      *delta = radius - sqrt(((zcenter - z_mid) * (zcenter - z_mid))
                           + ((ycenter - y_mid) * (ycenter - y_mid))
                           + ((xcenter - x_mid) * (xcenter - x_mid)));

      normal[0] = x_mid - xcenter;
      normal[1] = y_mid - ycenter;
      normal[2] = z_mid - zcenter;

      *newl = 0.0;
    }/* end if 2nd inter.    */
  }  /* end elbow sit.       */
  else                          /* there are two inter. */
  {
    xsol1 = pt1[0] + (deltax * t_sol1);
    ysol1 = pt1[1] + (deltay * t_sol1);
    zsol1 = pt1[2] + (deltaz * t_sol1);

    xsol2 = pt1[0] + (deltax * t_sol2);
    ysol2 = pt1[1] + (deltay * t_sol2);
    zsol2 = pt1[2] + (deltaz * t_sol2);

    x_mid = (xsol1 + xsol2) / 2;
    y_mid = (ysol1 + ysol2) / 2;
    z_mid = (zsol1 + zsol2) / 2;

    *delta = radius - sqrt(((zcenter - z_mid) * (zcenter - z_mid))
                         + ((ycenter - y_mid) * (ycenter - y_mid))
                         + ((xcenter - x_mid) * (xcenter - x_mid)));

    normal[0] = x_mid - xcenter;
    normal[1] = y_mid - ycenter;
    normal[2] = z_mid - zcenter;

    *newl = sqrt((z_mid - pt1[2]) * (z_mid - pt1[2])
               + (y_mid - pt1[1]) * (y_mid - pt1[1])
               + (x_mid - pt1[0]) * (x_mid - pt1[0]));
  } /* end 2 inter. */

  if (DEBUG) {
    fprintf (datafp, "\n\tCollision with #%d circle has occured!",
ws);
    fprintf (datafp, "\n\tnormal = <%lf,%lf,%lf>", normal[0],
                                                   normal[1],
                                                   normal[2]);
    fprintf (datafp, "\n\tCircle Delta = %lf\n", *delta);
  }

  return  TRUE;
}                               /* end find inter sphere */
```

ANALYTICAL.C

```
/*
   ┌──────────────────────────────────────────────────────┐
   │ Program Name:   Analytical.c                         │
   ├──────────────────────────────────────────────────────┤
   │ description: Aquires Betas for constraints and det-  │
   │     ermine the Analytical solution for each time step.│
   │                                                      │
   │ Procedures:                                          │
   │    Build_Grammian2  :                                │
   │    rank_lost_betas  :                                │
   │    find_j1_beta     :                                │
   │    find_obs_beta    :                                │
   │    findt            :                                │
   └──────────────────────────────────────────────────────┘ */

#include "general.h"        /* Header File to link all others */

/*
   ┌──────────────────────────────────────────────────────┐
   │ name:    Build_Grammian2                             │
   │ description:  The Grammian is a multiplication of one│
   │     each solution vector with each solution vector.  │
   │     Each component is a dot product of two of the    │
   │     vectors.  The B values are used as a weighting   │
   │     factor.  If B is the diagonal matrix for the     │
   │     link weights, as the value goes from 1 to zero the│
   │     joint that it relates to becomes more active. B  │
   │     is the same as the matrix B (not Beta) in F.G.   │
   │     Pin's paper.                                     │
   │                                                      │
   │ inputs:  Solution Vectors (g), and alphas            │
   │ outputs: Allocates memory (the 2 in Build_Grammian2, │
   │     means that it allocates memory which must be freed│
   │     by the calling routine), builds, and return the  │
   │     structure to the calling routine.                │
   ├──────────────────────────────────────────────────────┤
   │ Authors:      Date:     Modifications:               │
   │  c hacker      10/94    Created                      │
   │                 1/95    Alphas implemented           │
   └──────────────────────────────────────────────────────┘ */
MATRIX *Build_Grammian2 (FSP_data, B, datafp)
FILE      *datafp;
MATRIX    *B;
Solutions *FSP_data;
{
  int i,j,k;
  MATRIX *Grammian,
         *gtemp;


  Grammian = mat_malloc(SPAN, SPAN);

  gtemp = mat_mul2(FSP_data->g, Robot->Weights);

  if (DEBUG) {
    fprintf(datafp,"\n  _____________________________  \n");
    fprintf(datafp,"\n          Building Grammian          \n");
    fprintf(datafp,"  _____________________________  \n\n");
    fmat_pr(datafp,"g",FSP_data->g);
    fmat_pr(datafp,"Angle Weights", Robot->Weights);
    fmat_pr(datafp,"B Vector", B);
    fmat_pr(datafp,"g*Weights*B",gtemp);
  }

  for (i = 0; i < gtemp->rows; i++)
    for (j = 0; j < gtemp->rows; j++)
    {
      Grammian-> p[i][j] = 0;
      for (k = 0; k < gtemp->cols; k++)
      {
        Grammian -> p[i][j] += gtemp -> p[i][k] *  gtemp -> p[j][k];
      }
    }

  mat_free(gtemp);

  return (Grammian);
}


/*
   ┌──────────────────────────────────────────────────────┐
   │ name:    rank_lost_betas                             │
   │ description:  finds beta values for cases where the  │
   │     optimized FSP Jacobian has lost rank.            │
   │                                                      │
   │ inputs:  See Declarations below                      │
   │ outputs: Alters two variables within FSP_data.       │
   │     the beta vector for the joint is added to betall,│
   │     and cn, the number of beta vectors present, is   │
   │     incremented.                                     │
   ├──────────────────────────────────────────────────────┤
   │ Authors:      Date:     Modifications:               │
   │  K Morgansn    4/94     Algorithm Creation for 1 loss│
   │  G Fries       3/95     Altered from 1 loss to infinite│
   │  C Hacker      3/95     Ported to Solutions struct   │
   └──────────────────────────────────────────────────────┘ */
void rank_lost_betas(FSP_data, Jacob, dx, RowElim, datafp)
FILE   *datafp;          /* data file declaration           */
int    *RowElim;         /* determines which rows eliminated */
MATRIX *Jacob,           /* Forward Kinematic Derivative    */
       *dx;              /* Requested change in EE          */
Solutions *FSP_data;     /* see structures.h                */
{
  int    i, j, k;
  double num;

  for (j=0; j<M; j++)
    if (RowElim[j]==2)
    {
      for (i=0; i < SPAN; i++)
      {
        num = 0;
        for (k=0; k<M; k++)
          num += Jacob->p[j][k] * FSP_data->g->p[i][k];
       FSP_data->betall->p[i][FSP_data->cn] = num/dx->p[j][0];
```

```
    }
    FSP_data->cn++;
  }
}                /* end find_jl_beta */

/*
  ┌──────────────────────────────────────────────────────────────────┐
  │ name:    find_jl_beta                                            │
  │ description: finds beta values for the joint defined             │
  │    by chk.  Restricts said joint by confining it to              │
  │    remain at a maximum or minimum limit.  The variable           │
  │    distance is the amount that the joint must move to            │
  │    get to the max or the min.                                    │
  │                                                                  │
  │ inputs:  See Declarations below                                  │
  │ outputs: Alters two variables within FSP_data.                   │
  │     the beta vector for the joint is added to betall,            │
  │     and cn, the number of beta vectors present, is               │
  │     incremented.                                                 │
  ├──────────────────────────────────────────────────────────────────┤
  │ Authors:     Date:     Modifications:                            │
  │  F Tulloch     4/94    Algorithm Creation for 2 g's              │
  │  c hacker     10/94    Ported from 2 g's to FSP                  │
  │                2/95    Ported to Solutions struct                │
  └──────────────────────────────────────────────────────────────────┘ */
void find_jl_beta(FSP_data, chk, distance, datafp)
FILE  *datafp;          /* data file declaration                */
int    chk;             /* link number and collision flag       */
double distance;        /* distance over low/up joint limit     */
Solutions *FSP_data;    /* see structures.h                     */
{
  int i;

  if (chk > (FSP_data->g->cols -1));
  else
    for( i=0; i < FSP_data->g->rows; i++ )
      FSP_data->betaII->p[i][FSP_data->cn] =
                          FSP_data->g->p[i][chk]/(distance);

  FSP_data->cn++;
}                /* end find_jl_beta */

/*
  ┌──────────────────────────────────────────────────────────────────┐
  │ name:    find_obs_beta                                           │
  │ description: finds beta values for a link[chk] at a              │
  │   distance, newl from its beginning, that has entered            │
  │   an obstacle's buffer zone.  A Jacobian is created              │
  │   for up to that portion of the link, and the beta               │
  │   vector is calculated such that the point pushed back           │
  │   to the edge of the zone.                                       │
  │                                                                  │
  │ inputs:  See Declarations below                                  │
  │ outputs: Alters two variables within FSP_data. The               │
  │   beta vector for the joint is added to betall, and              │
  │   cn, the number of beta vectors present, is                     │
  │   incremented.                                                   │
  ├──────────────────────────────────────────────────────────────────┤
  │ Authors:     Date:     Modifications:                            │
  │  F Tulloch     4/94    Algorithm Creation for 2 g's              │
  │  c hacker     10/94    Ported from 2 g's to FSP                  │
  │                2/95    Ported to Solutions struct                │
  └──────────────────────────────────────────────────────────────────┘ */
void find_obs_beta(FSP_data, chk, newl, delta, normal, datafp)
  FILE   *datafp;       /* data file declaration                */
  int     chk;          /* link that has moved into buffer      */
  double *newl,         /* distance on chk link of most contact */
         delta,         /* distance needed to move along normal */
         normal[3];     /* normal guide vector                  */
  Solutions *FSP_data; /* See structures.h                      */
{
  int    i, j, p, k;
  MATRIX *alpha,        /* Holds alphas        */
         *Jacob;        /* Holds Jacobian      */

  alpha = mat_malloc( 3, SPAN );
  Jacob = mat_malloc( N, M );

  /*************************************************************
   *                                                           *
   *        get new jacob using using new links based on inters    *
   *************************************************************/

  for (i=0; i<Robot->NL; i++) LL[i] = Robot->LINKS[i];

  LL[chk] = *newl;
  for (i=chk+1; i<Robot->NL; i++ )  LL[i] = 0.0;

  GET_JACOBIAN_ALTERED(Jacob, FSP_data->Qarray);

  /*********************************************************
   *        Calculate Alphas and then betas necessary        *
   *********************************************************/

  for (i = 0;i<3;i++) for (p = 0;p<SPAN;p++) alpha -> p[i][p] = 0.0;

  /* Calculate Alpha: J * g = (3 x VSpan) */
  for (i = 0; i < 3; i++)
    for (j = 0; j < SPAN; j++)
      for (k = 0; k < M; k++)
        alpha->p[i][j] += Jacob->p[i][k] * FSP_data->g->p[j][k];

  if (DEBUG) {
    fmat_pr(datafp,"*** Jacobian @ Xj ***", Jacob);
    fmat_pr(datafp,"*** Alpha ***", alpha);
  }

  /* Calculate Beta: Alpha * normal = (VSpan x 1) */
  for (i = 0; i < SPAN; i++)
  {
    FSP_data->betall->p[i][FSP_data->cn] = 0;
    for (j = 0; j < 3; j++)
      FSP_data->betall->p[i][FSP_data->cn] +=
                alpha->p[j][i] * normal[j];
    FSP_data->betall->p[i][FSP_data->cn] /= fabs(delta);
  }
  FSP_data->cn++;
```

```
  mat_free(alpha);
  mat_free(Jacob);
}                   /* end find_obs_beta */


/*
   ---------------------------------------------------------------
   | name:    findt_with_Betas_Holonomic                         |
   | description:  find the parameterization for the             |
   |    solution vectors found using FSP blocking.  The          |
   |    linear cmbination produced will either be a              |
   |    full space or null space motion based on the             |
   |    values of nu, and mu.  A macro is used to                |
   |    determine which method to use.  TWOSTEP means            |
   |    that a motion is first produced, and then                |
   |    obstacles and joint limits are avoided in the            |
   |    null space.  Otherwise, everything is done at            |
   |    once in the full space.                                  |
   |                                                             |
   | inputs:  FSP_data's betall and g vectors                    |
   | outputs: dq contains the new delta angles for step          |
   ---------------------------------------------------------------
   | Authors:      Date:     Modifications:                      |
   |  F Tulloch     4/94     Algorithm Creation for 2 g's        |
   |  c hacker     10/94     Ported from 2 g's to FSP            |
   |                2/95     Ported to Solutions struct          |
   |                3/95     Implemented ONE & TWO step          |
   ---------------------------------------------------------------
*/
MATRIX *findt_with_Betas_Holonomic(FSP_data, B, H, datafp)
FILE      *datafp;          /* data file declaration       */
MATRIX    *B,               /* B, not Beta, in Pins paper  */
          *H;               /* H in Pins paper             */
Solutions *FSP_data;        /* See structures.h            */
{
  int     i, j, k, a, b;
  double  sum1 = 0, sum2 = 0,
         afinal;
  MATRIX *bfinal,
         *cfinal,
         *dfinal,
         *A,  *Ainv,
         *nu, *mu,
         *e, *eT,
         *t, *temp, *temp2, *temp3,
         *G, *Ginv, *iden,
        *dq;

  dq   = mat_malloc(M   , 1);
  e    = mat_malloc(SPAN, 1);
  eT   = mat_malloc( 1,   SPAN);

  #ifdef DEBUG_OUT
    fprintf(stderr, "  # of constraints %d \n", FSP_data->cn);
  #endif
  if (DEBUG) fprintf(datafp, "\n\n# of constraints %d \n",
FSP_data->cn);

  if (FSP_data->cn > (SPAN - 1))
  {
    fprintf(stderr, "\nSORRY!...more constraints than D.O.R.\n");
    fprintf(datafp, "\nSORRY!...more constraints than D.O.R.\n");
    return ( (MATRIX *) -1);
  }

  /* initialize the vectors of ones */
  for (i = 0; i < SPAN; i++)
  { e -> p[i][0] = eT -> p[0][i] = 1.0e0;}

  /* See Methods.c, malloc's G, multiplies g's  and returns */
  G = Build_Grammian2(FSP_data, B, datafp);

  Ginv = mat_cp2(G);
  mat_LU_inv(Ginv);
  iden = mat_mul2(Ginv, G);

  if (DEBUG) {
    fmat_pr(datafp,"*** Identity Matrix Check (G * Ginv) ***", iden);
    fmat_pr(datafp,"*** betas ***", FSP_data->betall);
  }

  /*************************************************************
   *                                                           *
   *    a,b,c,d, A, mu, and nu are based on F.G. Pin's TM      *
   *                                                           *
   *************************************************************/


  /********* Calculate a's *********/
  afinal = 0;
  for (i = 0; i < SPAN; i++)
    for (j = 0; j < SPAN; j++)
      afinal += Ginv -> p[i][j];

  /********* Calculate b's *********/

  temp   = mat_malloc (SPAN,1);
  bfinal = mat_malloc (1, FSP_data->cn);
  for (k = 0; k < FSP_data->cn; k++)
  {
    for (i = 0; i < SPAN; i++)
    {
      temp->p[i][0] = 0.0;
      for (j = 0; j < SPAN; j++)
        temp->p[i][0] += Ginv->p[i][j] * FSP_data->betall->p[j][k];
    }
    bfinal->p[0][k] = 0.0;
    for (i = 0; i < SPAN; i++)
      bfinal->p[0][k] += temp->p[i][0];
  }                 /* end k for */
  mat_free(temp);

  /********* Calculate c's ********/

  temp = mat_malloc (SPAN, 1);
  cfinal = mat_malloc (FSP_data->cn , 1);
  for (a = 0; a < FSP_data->cn; a++)
  {
    for (i = 0; i < SPAN; i++)
    {
      temp->p[i][0] = 0.0;
```

```
    for (j = 0; j < SPAN; j++)
  temp->p[i][0] += Ginv->p[i][j];
  }
  cfinal ->p[a][0] = 0.0;
  for (j = 0; j < SPAN; j++)
    cfinal->p[a][0] += FSP_data->betall->p[j][a] * temp->p[j][0];
 }                  /* end a for */
mat_free(temp);

/******** Calculate d's ********/
temp   = mat_malloc(SPAN, 1);
dfinal = mat_malloc(FSP_data->cn,FSP_data->cn);
for (k = 0; k < FSP_data->cn; k++)
  for (b = 0; b < FSP_data->cn; b++)
  {
    dfinal->p[k][b] = 0.0;
    for (i = 0; i < SPAN; i++)
    {
      temp->p[i][0] = 0.0;
      for (j = 0; j < SPAN; j++)
        temp->p[i][0] += Ginv->p[i][j] * FSP_data->betall->p[j][k];
    }
    for (j = 0; j < SPAN; j++)
    dfinal->p[k][b] += FSP_data->betall->p[j][b] * temp->p[j][0];
  }                 /* end k for */
mat_free(temp);

/******** Calculate A  ********/

temp = mat_malloc(FSP_data->cn,FSP_data->cn);
for(a=0;a<FSP_data->cn;a++)
  for(b=0;b<FSP_data->cn;b++)
    temp ->p[a][b]=dfinal->p[a][b]*afinal;

temp2= mat_malloc(FSP_data->cn,FSP_data->cn);
  for (i = 0; i < FSP_data->cn; i++)
    for (j = 0; j < FSP_data->cn; j++)
      temp2->p[i][j]=cfinal->p[i][0]*bfinal->p[0][j];

A    = mat_malloc(FSP_data->cn, FSP_data->cn);
Ainv = mat_malloc(FSP_data->cn, FSP_data->cn);

for (i = 0; i < FSP_data->cn; i++)
  for (j = 0; j < FSP_data->cn; j++)
    A ->p[i][j]=temp2->p[i][j]-temp->p[i][j];

mat_free(temp);
mat_free(temp2);

mat_cp(A, Ainv);
mat_LU_inv(Ainv);

/*************************************************************
 *                                                           *
 *    Find mu and nu Lagrange multipliers for Null or Full   *
 *                                                           *
 *************************************************************/
#ifdef TWOSTEP

  /************<< find nu >>************ NULLSPACE ********/
  temp = mat_malloc (FSP_data->cn, 1);
  for(i=0;i<FSP_data->cn;i++)
    temp->p[i][0]=afinal;
  nu=mat_mul2(Ainv,temp);
  mat_free(temp);

  /************<< find mu >>************ NULLSPACE ********/
  mu = mat_mul2(bfinal, nu);
  mu->p[0][0] /= -afinal;

#else  /* This is the ONE SHOT Method, It is default      */

   /************<< find nu >>*********** FULLSPACE ********/
  temp = mat_malloc (FSP_data->cn, 1);
  for(i=0;i<FSP_data->cn;i++)
    temp->p[i][0]=afinal - cfinal->p[i][0];
  nu=mat_mul2(Ainv,temp);
  mat_free(temp);

  /************<< find mu >>*********** FULLSPACE ********/
  mu = mat_mul2(bfinal, nu);
  mu->p[0][0] = -1 * (mu->p[0][0] + 1 )/ afinal;

#endif

/*************************************************************
 *                                                           *
 *    Find the parametric Analytical Solutions 't'           *
 *                                                           *
 *************************************************************/
temp2 = mat_malloc(SPAN, 1);    /* temp2= Ge     */
for(i=0;i<SPAN;i++)
{
  temp2->p[i][0] = 0.0;
  for(j=0;j<SPAN;j++)
    temp2->p[i][0]+=Ginv->p[i][j];
}

temp3 = mat_malloc(SPAN, 1);    /* temp3= -muGe   */
for(i=0;i<SPAN;i++)
  temp3->p[i][0]= -(mu->p[0][0])*temp2->p[i][0];
mat_free(temp2);

temp = mat_malloc (SPAN, 1);   /* temp = bNu     */
for(i=0;i<SPAN;i++)
{
  temp->p[i][0] = 0.0;
  for(a=0;a<FSP_data->cn;a++)
    temp->p[i][0]+=FSP_data->betall->p[i][a]*nu->p[a][0];
}

temp2= mat_malloc (SPAN, 1);   /* temp2= GbNu    */
for(i=0;i<SPAN;i++)
{
  temp2->p[i][0] = 0.0;
  for(j=0;j<SPAN;j++)
    temp2->p[i][0]+=Ginv->p[i][j]*temp->p[j][0];
}

t = mat_malloc (SPAN, 1);
```

```
  for(i=0;i<SPAN;i++)
    t->p[i][0] = temp3->p[i][0] - temp2->p[i][0];


  mat_free(temp );
  mat_free(temp2);
  mat_free(temp3);

  if (DEBUG) {
     fprintf(datafp, "  a= %9.4f\n", afinal);
     fmat_pr(datafp, "b= ", bfinal);
     fmat_pr(datafp, "c= ", cfinal);
     fmat_pr(datafp, "d= ", dfinal);
     fmat_pr(datafp,"A = ", A);
     fmat_pr (datafp, "Nu =", nu);
     fprintf (datafp, "Mu = %9.4f \n", mu->p[0][0]);
     fmat_pr(datafp,"Term1", temp3);
     fmat_pr(datafp,"Term2", temp2);
     fmat_pr (datafp, "*** t's for Collision ***", t);

     sum1 = 0;
     for (i = 0; i < SPAN; i++)
        sum1 += t -> p[i][0];
     fprintf(datafp, "\nsum ti:  %7.4f\n", sum1);

     for (j=0; j<FSP_data->cn; j++)
     {
        sum1 = 0;
        for (i = 0; i < SPAN; i++)
          sum1 += FSP_data->betall -> p[i][j] * t -> p[i][0];
        fprintf(datafp, "\nsum betai*ti:  %7.4f \n", sum1);
     }
  }

  for (i = 0; i < FSP_data->M; i++)
  {
    dq->p[i][0]=0.0;
    for (j = 0; j < SPAN; j++)
      dq->p[i][0] += (t -> p[j][0]) * (FSP_data->g->p[j][i]);
  }

  if (DEBUG) fmat_pr(datafp, "New dq reflecting constraints (t * g)",
dq);

  mat_free(t);
  mat_free(nu);
  mat_free(mu);

  mat_free(iden);
  mat_free(Ginv);
  mat_free(bfinal);
  mat_free(cfinal);
  mat_free(dfinal);
  mat_free(A);
  mat_free(Ainv);
  mat_free(G);
  mat_free(e);
  mat_free(eT);
  return (dq);
}                 /* end findt */
```

```
/*
   ┌──────────────────────────────────────────────────────────┐
   │ name:    findt_without_Betas_Holonomic                   │
   │ description:  find the parameterization for the          │
   │    solution vectors found using FSP blocking.  The       │
   │    linear cmbination produced will be a full space       │
   │    leastnorm motion.  This is called when there are      │
   │    no constraints on the system.                         │
   │                                                          │
   │ inputs:  g vectors                                       │
   │ outputs: dq contains the new delta angles for step       │
   ├──────────────────────────────────────────────────────────┤
   │ Authors:     Date:     Modifications:                    │
   │  K Morgensn    6/94    Algorithm Creation for 2 g's      │
   │  c hacker      3/95    Implemented for speed             │
   └──────────────────────────────────────────────────────────┘ */

MATRIX *findt_without_Betas_Holonomic(FSP_data, B, H, datafp)
FILE      *datafp;        /* data file declaration       */
MATRIX    *B,             /* B, not Beta, in Pins paper  */
          *H;             /* H in Pins paper             */
Solutions *FSP_data;      /* See structures.h            */
{
  int    i, j, k;
  float  sum=0.0;
  MATRIX *dq,             /* Change in Angles for requested dx     */
         *G,*Gtemp,       /* Grammian formed of solution vectors   */
         *t,              /* weighting factors, one for each vector*/
         *x,*y,*iden,     /* dummy variable for calculations       */
         *e,              /* vertical vector of ones               */
         *eT;             /* horizontal vector of ones             */


  dq    = mat_malloc (M    , 1);
  e     = mat_malloc (SPAN,    1);
  eT    = mat_malloc (   1, SPAN);
  Gtemp = mat_malloc (SPAN, SPAN);

  /* initialize the vectors of ones */
  for (i = 0; i < (SPAN); i++)
  { e  -> p[i][0] = eT -> p[0][i] = 1.0e0; }

  Gtemp = Build_Grammian2 (FSP_data, B, datafp);

  G = mat_cp2(Gtemp);
  mat_LU_inv(G);

  if (DEBUG) {
    fprintf(datafp,"\n ____________________________ \n");
    fprintf(datafp,"\n       LEAST NORM MOTION      \n");
    fprintf(datafp,  " ____________________________ \n\n");

    iden = mat_mul2(G,Gtemp);  /* This code verifies Grammian
Inversion */
    fmat_pr(datafp,"*** G  ***",Gtemp);
    fmat_pr(datafp,"*** Ginv ***", G);
    fmat_pr(datafp,"*** iden ***", iden);
    mat_free(iden);
  }               /* if DEBUGGING print out  (see file: info) */
```

```
  t = mat_mul2(G, e);
  x = mat_mul2(eT, G);
  y = mat_mul2(x, e);

  mat_sca(t, 1 / y->p[0][0]);  /* t's now complete */

  for (i = 0; i < M; i++)
  {
    dq->p[i][0] = 0.0e0;
    for (j = 0; j < SPAN; j++)
      dq->p[i][0] += (t -> p[j][0]) * (FSP_data->g -> p[j][i]);
  }

  if (DEBUG) {              /** print out t's, sums and New dq's **/
    fmat_pr(datafp, "*** t's ***", t);
    for (i = 0, sum = 0; i < SPAN; i++)
      sum += t -> p[i][0];
    fprintf(datafp, "\nsum :  %7.4f \n", sum);
    for (i = 0, sum = 0; i < M; i++)
      sum += dq->p[i][0];
    fprintf(datafp, "\nsum t*g:  %7.4f\n", sum);
    fprintf(datafp, "\n\nNew dq LEAST NORM (t * g): degrees\n");
    for (j = 0; j < M; j++)
      fprintf(datafp," %14.8f\n", deg(dq->p[j][0]));
  }             /* if DEBUGGING, include code  (see file info) */

  mat_free(Gtemp);
  mat_free(G);
  mat_free(eT);
  mat_free(e);
  mat_free(t);
  mat_free(x);
  mat_free(y);

  return (dq);
}
```

Manipulator Porting Programs

The example files in this section are listed corresponding to the names of their system file counterparts:

Jacob_ROBOT.c	Code for Forward Kinematics and Jacobian.
Jacob_HERMIES.c	Jacob_ROBOT.c for HERMIES arm.
Jacob_AIRARM.c	Jacob_ROBOT.c for AIRARM arm.
ROBOT.dat	Manipulator specifications
ROBOT_HERMES.dat	Manipulator Specs for HERMIES arm.
ROBOT_AIRARM.dat	Manipulator Specs for AIRARM arm.
Jacob_UTILS.c	Drivers for Jacob_ROBOT.c and other utilities

```
JACOB_ROBOT.c
JACOB_HERMIES.c
/*
   ┌──────────────────────────────────────────────────────────┐
   │ Program Name:   Jacob_HERMIES.c                          │
   ├──────────────────────────────────────────────────────────┤
   │ description: This file must contain compatible versions  │
   │    of GET_JACOB and GET_ACTUAL_X for the Manipulator of   │
   │    the same name (i.e. ROBOT_HERMIES.dat)                │
   │                                                          │
   │ Procedures:                                              │
   │    GET_JACOB       :  Derivative of Forward Kinematic    │
   │    GET_ACTUAL_X    :  Forward Kinematic                  │
   └──────────────────────────────────────────────────────────┘ */

#include "general.h"       /* Header File to link all others */

/*
   ┌──────────────────────────────────────────────────────────┐
   │ name:    GET_JACOB                                       │
   │ description:  This file computes the jacobian matrix     │
   │    which transforms the seven joint velocities of        │
   │    CesARM to the Cartesian translational velocities      │
   │    of the end effector at its centerpoint in the         │
   │    stationary base coordinates.                          │
   │                                                          │
   │    For Obstacle Avoidance, the Jacobian must be comput-  │
   │    ed at the point of intersection.  Functions in        │
   │    Jacob_UTILS.c accomplish this assuming the follow-    │
   │    ing hold:                                             │
   │      1. The links should be labeled as LL[0] - LL[4]     │
   │      2. Change constants such as Link Lengths ect..      │
   │         into a data file (see ROBOT_HERMIES.dat and      │
   │         the Users Guide)                                 │
   │                                                          │
   │ inputs:  Initial Qarray and LL[0]-LL[4]                  │
   │ outputs: Returns the Jacobian                            │
   ├──────────────────────────────────────────────────────────┤
   │ Authors:      Date:     Modifications:                   │
   │  c hacker      12/94    Created                          │
   │                3/95     Platform Included                │
   └──────────────────────────────────────────────────────────┘ */

void GET_JACOB(MATRIX *J, MATRIX *Qarray)
{
MATRIX *Jmanip;
double s1,s2,s3,s4,s5,s6,s7,s10,
       c1,c2,c3,c4,c5,c6,c7,c10,
       xx, yy, zz, ww,
       xx1,yy1,zz1,zz4,yy3,
       D1, D2, D3,
       C11,C12,C13,
       C21,C22,C23,
       X_ARM,     /* x displacement of EE from base of manipulator */
                  /* in platform coords */
       Y_ARM,     /* y displacement of EE from base of manipulator */
                  /* in platform coords */
       X_OFFSET, /* x displacement of base of manip from cntr of  */
                 /* platform in platform coords*/
       Y_OFFSET; /* y displacement of base of manip from cntr of  */
                 /* platform in platform coords*/

Jmanip=mat_malloc(2,7);/* components of rows 1&2 of orig jacobian*/

/*-------------------------------------------------------------*/
/* The calculations for the additional three columns of the Jacobi*/
/* corresponding to x displacement, y displacement, and rotation  */
/* of the center of the platfrom can be found with FG Pin.  The   */
/* component corresponding to the platform rotation is a one in   */
/* the tenth column of row six.  The entire first and second rows */
/* have been rewritten using the original two rows as well as the */
/* x and y positions of the end effector in the manipulator base  */
/* coords. (these displacements were found using the forward      */
/* kinematics from the original program)                          */
/*-------------------------------------------------------------- --*/

s1 = sin(Qarray->p[0][0]);c1 = cos(Qarray->p[0][0]);
s2 = sin(Qarray->p[1][0]);c2 = cos(Qarray->p[1][0]);
s3 = sin(Qarray->p[2][0]);c3 = cos(Qarray->p[2][0]);
s4 = sin(Qarray->p[3][0]);c4 = cos(Qarray->p[3][0]);
s5 = sin(Qarray->p[4][0]);c5 = cos(Qarray->p[4][0]);
s6 = sin(Qarray->p[5][0]);c6 = cos(Qarray->p[5][0]);
s7 = sin(Qarray->p[6][0]);c7 = cos(Qarray->p[6][0]);
s10 = sin(Qarray->p[9][0]);c10 = cos(Qarray->p[9][0]);

/** FORWARD KINEMATICS FOR X AND Y POSITION OF END EFFECTOR *******/
/*********** IN MANIPULATOR BASE FRAME COORDS ********************/

yy  = c1*s2;       zz  = c1*c2;      xx  = zz*c3-s1*s3;
yy1 = s1*s2;       zz1 = s1*c2;      xx1 = c1*s3+zz1*c3;
yy3 = c5*c6;       zz4 = s5*s6;
D1  = yy3*LL[3];     D2  = zz4*LL[3];     D3  = c6*LL[3];

C11 = xx*c4-yy*s4;     C12 = -xx*s4-yy*c4;     C13 = zz*s3+s1*c3;
C21 = xx1*c4-yy1*s4;   C22 = -xx1*s4-yy1*c4;   C23 = zz1*s3-c1*c3;

X_OFFSET = -Robot->PLAT.L_OFF*sin(Robot->PLAT.ANG_OFF);
Y_OFFSET =  Robot->PLAT.L_OFF*cos(Robot->PLAT.ANG_OFF);
X_ARM    = C11*D1+C12*D2+C13*D3+(LL[2]*(-yy *s4+xx *c4)+LL[4]*xx
+LL[1]*yy +LL[0]*s1);
Y_ARM    =
C21*D1+C22*D2+C23*D3+(LL[2]*(-yy1*s4+xx1*c4)+LL[4]*xx1+LL[1]*yy1-LL[0]
*c1);


/******************ORIGINAL JACOBIAN ROW 1*******************/
ww = LL[3]*(c4*s5+s4*c5)*s6+LL[2]*s4-LL[1];
zz = LL[3]*(-s4*s5+c4*c5)*s6+LL[2]*c4;
Jmanip->p[0][0] =
(c1*(-s3*zz+LL[3]*c3*c6-LL[4]*s3+LL[0])-s1*(c2*(c3*zz+LL[3]*s3*c6+LL[4
]*c3)-s2*ww));
Jmanip->p[0][1] = (c1*(-s2*(c3*zz+LL[3]*s3*c6+LL[4]*c3)-c2*ww));
Jmanip->p[0][2] =
(c1*c2*(-s3*zz+LL[3]*c3*c6-LL[4]*s3)+s1*(-c3*zz-LL[3]*s3*c6-LL[4]*c3))
;

xx = zz-LL[2]*c4;
```

```
yy = -LL[3]*(c4*s5*s6+s4*c5*s6);
zz = yy-LL[2]*s4;
Jmanip->p[0][3] = (c1*(c2*c3*zz-s2*(xx+LL[2]*c4))-s1*s3*zz);
Jmanip->p[0][4] = (c1*(c2*c3*yy-s2*xx)-s1*s3*yy);

xx = LL[3]*(c4*c5-s4*s5)*c6;
Jmanip->p[0][5] =
(c1*(c2*(c3*xx-LL[3]*s3*s6)-s2*LL[3]*c6*(c4*s5+s4*c5))+s1*(-LL[3]*c3*s
6-s3*xx));
Jmanip->p[0][6] = ZERO;


/*****************ORIGINAL JACOBIAN ROW 2**********************/

xx = LL[3]*(-s4*s5+c4*c5)*s6+LL[2]*c4;
Jmanip->p[1][0] =
(c1*(c2*(c3*xx+LL[3]*s3*c6+LL[4]*c3)+s2*(zz+LL[1]))+s1*(-s3*xx+LL[3]*c
3*c6-LL[4]*s3+LL[0]));
Jmanip->p[1][1] =
(s1*(-s2*(c3*xx+LL[3]*s3*c6+LL[4]*c3)+c2*(zz+LL[1])));
Jmanip->p[1][2] =
(s1*c2*(-s3*xx+LL[3]*c3*c6-LL[4]*s3)-c1*(-c3*xx-LL[3]*s3*c6-LL[4]*c3))
;

yy = -LL[3]*(c4*s5+s4*c5)*s6;
zz = yy-LL[2]*s4;
Jmanip->p[1][3] = (s1*(c2*c3*zz-s2*xx)+c1*s3*zz);
Jmanip->p[1][4] = (s1*(c2*c3*yy-s2*LL[3]*s6*(c4*c5-s4*s5))+c1*s3*yy);

zz = LL[3]*(c4*c5-s4*s5)*c6;
Jmanip->p[1][5] =
(s1*(c2*(c3*zz-LL[3]*s3*s6)-s2*LL[3]*c6*(c4*s5+s4*c5))-c1*(-LL[3]*c3*s
6-s3*zz));
Jmanip->p[1][6] = ZERO;


/**********************NEW JACOBIAN ROW 1**********************/
J->p[0][0] = Jmanip->p[0][0]*c10-Jmanip->p[1][0]*s10;
J->p[0][1] = Jmanip->p[0][1]*c10-Jmanip->p[1][1]*s10;
J->p[0][2] = Jmanip->p[0][2]*c10-Jmanip->p[1][2]*s10;
J->p[0][3] = Jmanip->p[0][3]*c10-Jmanip->p[1][3]*s10;
J->p[0][4] = Jmanip->p[0][4]*c10-Jmanip->p[1][4]*s10;
J->p[0][5] = Jmanip->p[0][5]*c10-Jmanip->p[1][5]*s10;
J->p[0][6] = Jmanip->p[0][6]*c10-Jmanip->p[1][6]*s10;

if (M>7) {
J->p[0][7] = 1.0e0;
J->p[0][8] = ZERO;
J->p[0][9] = -(X_ARM+X_OFFSET)*s10-(Y_ARM+Y_OFFSET)*c10;
}
/***************************New Row 2***************************/
J->p[1][0] = Jmanip->p[0][0]*s10+Jmanip->p[1][0]*c10;
J->p[1][1] = Jmanip->p[0][1]*s10+Jmanip->p[1][1]*c10;
J->p[1][2] = Jmanip->p[0][2]*s10+Jmanip->p[1][2]*c10;
J->p[1][3] = Jmanip->p[0][3]*s10+Jmanip->p[1][3]*c10;
J->p[1][4] = Jmanip->p[0][4]*s10+Jmanip->p[1][4]*c10;
J->p[1][5] = Jmanip->p[0][5]*s10+Jmanip->p[1][5]*c10;
J->p[1][6] = Jmanip->p[0][6]*s10+Jmanip->p[1][6]*c10;

if (M>7) {
J->p[1][7] = ZERO;
J->p[1][8] = 1.0e0;
J->p[1][9] = (X_ARM+X_OFFSET)*c10-(Y_ARM+Y_OFFSET)*s10;
}
/***************************ROW 3*****************************/

J->p[2][0] = ZERO;

zz = LL[3]*(-s4*s5+c4*c5)*s6+LL[2]*c4;
J->p[2][1] = (c2*(c3*zz+LL[3]*s3*c6+LL[4]*c3)-s2*ww);
J->p[2][2] = s2*(-s3*zz+LL[3]*c3*c6-LL[4]*s3);
J->p[2][3] = (c2*zz-s2*c3*(ww+LL[1]));

yy = ww-LL[2]*s4+LL[1];
zz = zz-LL[2]*c4;
J->p[2][4] = (c2*zz-s2*c3*yy);
J->p[2][5] =
(s2*(c3*(LL[3]*(c4*c5-s4*s5)*c6)-LL[3]*s3*s6)+c2*(LL[3]*(c4*s5+s4*c5)*
c6));
J->p[2][6] = ZERO;

if (M>7) {
J->p[2][7] = ZERO;
J->p[2][8] = ZERO;
J->p[2][9] = ZERO;
}
/************************BOTTOM 3 ROWS************************/
if (N == 6)                 /* Execute only if there are 3 rows at the
                             * bottom */
{
  /*************************************************************/
  /* The lower three rows of the Jacobian are determined using the*/
  /* method discussed in the textbook Robotics-Control, Sensing,  */
  /* Vision and Intelligence by Fu, Gonzalez, and Lee.  Appendix  */
  /* B pp544-547.                                                 */
  /* see my printout "jacoblow", a macsyma session which derived  */
  /* expressions for CESARm-- dated 11/21/91--MAU                 */
 /*************************************************************/
  J -> p[3][0] = ZERO;
  J -> p[4][0] = ZERO;
  J -> p[5][0] = 1.0e0;
  J -> p[3][1] = s1;
  J -> p[4][1] = -c1;
  J -> p[5][1] = ZERO;
  J -> p[3][2] = -c1 * s2;
  J -> p[4][2] = -s1 * s2;
  J -> p[5][2] = c2;

  xx = c1 * c2 * s3 + s1 * c3;
  yy = s1 * c2 * s3 - c1 * c3;
  zz = s2 * s3;
  J -> p[3][3] = xx;
  J -> p[4][3] = yy;
  J -> p[5][3] = zz;
  J -> p[3][4] = xx;
  J -> p[4][4] = yy;
  J -> p[5][4] = zz;

  xx = c1 * c2 * c3 - s1 * s3;
  yy = c1 * s3 + s1 * c2 * c3;
```

```
  zz = c1 * s2;
  J -> p[3][5] =  (-xx * s4 - zz * c4) * c5 - (xx * c4 - zz * s4) *
s5;
  J -> p[3][6] = ((-xx * s4 - zz * c4) * s5 + (xx * c4 - zz * s4) *
c5) * s6
                                          - (-c1 * c2 * s3 - s1 * c3) *
c6;

  zz = s1 * s2;
  J -> p[4][5] = (-yy * s4 - zz * c4) * c5 - (yy * c4 - zz * s4) * s5;
  J -> p[4][6] = ((-yy * s4 - zz * c4) * s5 + (yy * c4 - zz * s4) *
c5) * s6
                                          - (c1 * c3 - s1 * c2 * s3) *
c6;
  zz = s2 * c3;
  J -> p[5][5] = (c2 * c4 - zz * s4) * c5 - (c2 * s4 + zz * c4) * s5;
  J -> p[5][6] = ((c2 * c4 - zz * s4) * s5 + (c2 * s4 + zz * c4) * c5)
* s6
                                                    + s2 * s3 *
c6;
  if (M>7) {
  J -> p[3][7] = ZERO;  J -> p[4][7] = ZERO;  J -> p[5][7] = ZERO;
  J -> p[3][8] = ZERO;  J -> p[4][8] = ZERO;  J -> p[5][8] = ZERO;
  J -> p[3][9] = ZERO;  J -> p[4][9] = ZERO;  J -> p[5][9] = 1.0e0;
  }

}
  mat_free(Jmanip);
} /* End of function JACOBCALC */


/*-------------------------------------------------------------------*/
/*                  Forward Kinematics to get X                      */
/* Given Joint Angles (in Qarray) it returns in Xactual the X,Y,Z*/
/* and the Z,Y',X'' Euler Angles of the end effector.                */
/*                                                                   */
/*                                                                   */
/*-------------------------------------------------------------------*/
void GET_ACTUAL_X(MATRIX *Qarray, MATRIX *x_of_link)
{
  int i,j, not_first_time;  /* loop counters */
  MATRIX *TempFrame, *CurrentFrame,
    *TOC  ,     /* from world to center of platform                 */
    *TCR  ,     /* from center of platform to rot, about cntr plat  */
    *TRX  ,     /* transform from center to base of manip */
    *TXB  ,
    *TRB  ,     /* from rotated to base of manipulator              */
    *TOPO ,     /* to joint 1 frame                                 */
    *TAOP ,     /* to joint 2 frame                                 */
    *TBA  ,     /* to joint 3 frame                                 */
    *TBPB ,     /* to joint 4 frame                                 */
    *TCBP ,     /* to joint 5 frame                                 */
    *TCPC ,     /* to joint 6 frame                                 */
    *TDCP ,     /* to joint 7 frame                                 */
    *TDTT ;     /* to end effector frame                            */

  /* links 1-4 are drawn on local y axes */
  A->p[1][0]=Robot->LINKS[0] ; A->p[3][0]=1.0;
  B->p[1][0]=Robot->LINKS[1] ; B->p[3][0]=1.0;
  C->p[1][0]=Robot->LINKS[2] ; C->p[3][0]=1.0;
  D->p[1][0]=Robot->LINKS[3] ; D->p[3][0]=1.0;
  E->p[0][0]=Robot->LINKS[4] ; E->p[3][0]=1.0;

  R->p[2][0]=0.0; R->p[3][0]=1.0;  /* manip base offset drwn on local
y axis */
  X->p[2][0]=0.0; X->p[3][0]=1.0;

  if (not_first_time)
  {
    mat_free(XW);
    mat_free(RW);
    mat_free(AW);
    mat_free(BW);
    mat_free(CW);
    mat_free(DW);
    mat_free(EW);
  }

  /* shift 2 x,y plt cntr */
  TOC           = getT2(0,0,0,Qarray->p[7][0],Qarray->p[8][0],0);
  TCR           = getT2(Qarray->p[9][0],0,0,0,0,0);   /* rotate about
plt cntr*/
  CurrentFrame = mat_mul2(TOC,TCR);

  PW1=mat_mul2(CurrentFrame,P1);  /* locate the four corners of the
plat    */
  PW2=mat_mul2(CurrentFrame,P2);
  PW3=mat_mul2(CurrentFrame,P3);
  PW4=mat_mul2(CurrentFrame,P4);

  /* rot about platform center to dir of base of manip */
  TRX           =
getT2(0,0,0,-Robot->PLAT.L_OFF*sin(Robot->PLAT.ANG_OFF),
   Robot->PLAT.L_OFF*cos(Robot->PLAT.ANG_OFF),0);
  TempFrame     = mat_mul2(CurrentFrame,TRX); mat_free(CurrentFrame);
  XW            = mat_mul2(TempFrame,X);

  TXB           = getT2(0,0,0,0,0,Robot->PLAT.Z_OFF);
  CurrentFrame = mat_mul2(TempFrame,TXB);     mat_free(TempFrame);
  RW            = mat_mul2(CurrentFrame,R);   /* locate the base of
maniptr */

  /* shift to joint 1 frame */
  TOPO          = getT2(-PI,-PI/2,Qarray->p[0][0],0,0,0);
  TempFrame     = mat_mul2(CurrentFrame,TOPO);
mat_free(CurrentFrame);
  AW            = mat_mul2(TempFrame,A);        /* locate end of link 1
*/

  /* shift to jnt 2 frame */
  TAOP          = getT2(PI/2,0,Qarray->p[1][0],0,Robot->LINKS[0],0);
  CurrentFrame = mat_mul2(TempFrame,TAOP);    mat_free(TempFrame);
  BW            = mat_mul2(CurrentFrame,B);         /* locate end of
link 2 */

  /* shift to joint 3 frame */
  TBA           =
getT2(-PI/2,0,PI/2+Qarray->p[2][0],0,Robot->LINKS[1],0);
  TBPB          = getT2(0,PI/2,Qarray->p[3][0],0,0,0);    /* shift to
joint 4 frame */
```

```
  TempFrame    = mat_mul2(CurrentFrame,TBA); mat_free(CurrentFrame);
  CurrentFrame = mat_mul2(TempFrame,TBPB);   mat_free(TempFrame);
  CW           = mat_mul2(CurrentFrame,C);   /* locate end of link 3
*/

  /* shift to joint 5 frame */
  TCBP       = getT2(0,0,Qarray->p[4][0],0,Robot->LINKS[2],0);
  /* shift to jnt 6 frme */
  TCPC       = getT2(PI/2,-PI/2,-PI+Qarray->p[5][0],0,0,0);
  TempFrame    = mat_mul2(CurrentFrame,TCBP);
mat_free(CurrentFrame);
  CurrentFrame = mat_mul2(TempFrame,TCPC);   mat_free(TempFrame);
  DW           = mat_mul2(CurrentFrame,D);       /* locate end of link4
*/

  /* shift to EE frame */
  TDCP         = getT2(-PI/2,-PI,Qarray->p[6][0],0,Robot->LINKS[3],0);
  TempFrame    = mat_mul2(CurrentFrame,TDCP);
mat_free(CurrentFrame);

  EW1 = mat_mul2(TempFrame,E1);  /* locate the 4 drawing pnts of the
EE */
  EW2 = mat_mul2(TempFrame,E2);
  EW3 = mat_mul2(TempFrame,E3);
  EW4 = mat_mul2(TempFrame,E4);

  TDTT         = getT2(0,PI/2,0,0,0,0);
  CurrentFrame = mat_mul2(TempFrame,TDTT);
  ExtractRPY2(CurrentFrame, x_of_link);

  EW  = mat_mul2(TempFrame,E);

  mat_free(CurrentFrame);
  mat_free(TempFrame);
  not_first_time++;

  for (i=0; i<3; i++)
  {
    x_of_link->p[0][i] = RW->p[i][0];
    x_of_link->p[1][i] = AW->p[i][0];
    x_of_link->p[2][i] = BW->p[i][0];
    x_of_link->p[3][i] = CW->p[i][0];
    x_of_link->p[4][i] = DW->p[i][0];
    x_of_link->p[5][i] = EW->p[i][0];
  }

} /* GET_ACTUAL_X */
```

JACOB_AIRARM.C

```
/*
 ┌──────────────────────────────────────────────────────────────┐
 │ Program Name:   Jacob_AIRAM.c                                │
 ├──────────────────────────────────────────────────────────────┤
 │ description: This file must contain compatible versions      │
 │    of GET_JACOB and GET_ACTUAL_X for the Manipulator of      │
 │    the same name (i.e. ROBOT_AIRARM.dat)                     │
 │                                                              │
 │ Procedures:                                                  │
 │    GET_JACOB       :  Derivative of Forward Kinematic        │
 │    GET_ACTUAL_X    :  Forward Kinematic                      │
 └──────────────────────────────────────────────────────────────┘ */

#include "general.h"       /* Header File to link all others */

/*
 ┌──────────────────────────────────────────────────────────────┐
 │ name:    GET_JACOB                                           │
 │ description:  This file computes the jacobian matrix         │
 │    which transforms the seven joint velocities of            │
 │    ATD AIRARM to the Cartesian translational velocities      │
 │    of the end effector at its centerpoint in the             │
 │    stationary base coordinates.                              │
 │                                                              │
 │    For Obstacle Avoidance, the Jacobian must be comput-      │
 │    ed at the point of intersection.  Functions in            │
 │    Jacob_UTILS.c accomplish this assuming the follow-        │
 │    ing hold:                                                 │
 │      1. The links should be labeled as LL[0] - LL[4]         │
 │      2. Change constants such as Link Lengths ect..          │
 │         into a data file (see ROBOT_AIRARM.dat and           │
 │         the Users Guide)                                     │
 │                                                              │
 │ inputs:  Initial Qarray and LL[0]-LL[4]                      │
 │ outputs: Returns the Jacobian                                │
 ├──────────────────────────────────────────────────────────────┤
 │ Authors:     Date:     Modifications:                        │
 │  c hacker     12/94    Created                               │
 │               3/95    Platform Included                      │
 └──────────────────────────────────────────────────────────────┘ */
void GET_JACOB(MATRIX *Jacob, MATRIX *Qarray)
{
int    i,j;       /* used to pad orientation rows with zeros,
*/
MATRIX *Jman;     /* temporary variable to hold original arm For Kine
*/
double s10, c10, link1,
       X_ARM,     /* x displacement of end effector from base of manip
*/
                  /* in platform coords */
       Y_ARM,     /* y displacement of end effector from base of manip
*/
                 /* in platform coords */
       X_OFFSET, /* x displacement of base of manip from center of
*/
                 /* platform in platform coords*/
```

```
      Y_OFFSET; /* y displacement of base of manip from center of
*/
                /* platform in platform coords*/

s10 = sin(Qarray->p[7][0]);c10 = cos(Qarray->p[7][0]);
Jman= mat_malloc(2, 5); /* compnts of rows 1 and 2 of original
jacobian */

link1 = LL[1] + (Qarray->p[2][0]);

X_OFFSET = -Robot->PLAT.L_OFF*sin(Robot->PLAT.ANG_OFF);
Y_OFFSET =  Robot->PLAT.L_OFF*cos(Robot->PLAT.ANG_OFF);
X_ARM    = LL[0]*sin(Qarray->p[0][0]) +
         link1*sin(Qarray->p[0][0]+Qarray->p[1][0]) +
          LL[3]+LL[4]*cos(Qarray->p[4][0]);
Y_ARM    =-LL[4]   *  sin(Qarray->p[4][0]);

/************************ORIGINAL JACOBIAN ROW
1************************/
Jman->p[0][0] = LL[0]*cos(Qarray->p[0][0]) +
              link1 * cos(Qarray->p[0][0] + Qarray->p[1][0]);
Jman->p[0][1] =link1 * cos(Qarray->p[0][0] + Qarray->p[1][0]);
Jman->p[0][2] =        sin(Qarray->p[0][0] + Qarray->p[1][0]);
Jman->p[0][3] = 0;
Jman->p[0][4] =-LL[4] * sin(Qarray->p[4][0]);

/************************ORIGINAL JACOBIAN ROW
2************************/
Jman->p[1][0] = 0;
Jman->p[1][1] = 0;
Jman->p[1][2] = 0;
Jman->p[1][3] = 0;
Jman->p[1][4] =-LL[4] * cos(Qarray->p[4][0]);

/************************NEW JACOBIAN ROW 1************************/
Jacob->p[0][0] = Jman->p[0][0]*c10-Jman->p[1][0]*s10;
Jacob->p[0][1] = Jman->p[0][1]*c10-Jman->p[1][1]*s10;
Jacob->p[0][2] = Jman->p[0][2]*c10-Jman->p[1][2]*s10;
Jacob->p[0][3] = Jman->p[0][3]*c10-Jman->p[1][3]*s10;
Jacob->p[0][4] = Jman->p[0][4]*c10-Jman->p[1][4]*s10;

if (M > 5) {
  Jacob->p[0][5] = 1.0e0;
  Jacob->p[0][6] = ZERO;
  Jacob->p[0][7] = -(X_ARM+X_OFFSET)*s10-(Y_ARM+Y_OFFSET)*c10;
}
/****************************New Row 2****************************/
Jacob->p[1][0] = Jman->p[0][0]*s10+Jman->p[1][0]*c10;
Jacob->p[1][1] = Jman->p[0][1]*s10+Jman->p[1][1]*c10;
Jacob->p[1][2] = Jman->p[0][2]*s10+Jman->p[1][2]*c10;
Jacob->p[1][3] = Jman->p[0][3]*s10+Jman->p[1][3]*c10;
Jacob->p[1][4] = Jman->p[0][4]*s10+Jman->p[1][4]*c10;

if (M > 5) {
  Jacob->p[1][5] = ZERO;
  Jacob->p[1][6] = 1.0e0;
  Jacob->p[1][7] = (X_ARM+X_OFFSET)*c10-(Y_ARM+Y_OFFSET)*s10;
}
/****************************ROW 3*********************************/
Jacob->p[2][0] = -LL[0] * sin(Qarray->p[0][0]) -
                 link1 * sin(Qarray->p[0][0] + Qarray->p[1][0]);
Jacob->p[2][1] = -link1 * sin(Qarray->p[0][0] + Qarray->p[1][0]);
Jacob->p[2][2] =          cos(Qarray->p[0][0] + Qarray->p[1][0]);
Jacob->p[2][3] = 0;
Jacob->p[2][4] = 0;

if (M > 5) {
  Jacob->p[2][5] = ZERO;
  Jacob->p[2][6] = ZERO;
  Jacob->p[2][7] = ZERO;
}

/* If orientation control requested pad with ZEROS. Math not yet
developed. */
if (N==6) {
  if (M > 5)
    for (i=3; i <6; i++)
      for (j=0; j<8; j++)
        Jacob->p[i][j] = ZERO;
  else
    for (i=3; i <6; i++)
      for (j=0; j<5; j++)
        Jacob->p[i][j] = ZERO;
}

mat_free(Jman);
} /* End of function JACOBCALC */

/*
  name:     GET_ACTUAL_X
  description:  Forward Kinematics to get X given Joint
     Angles (in Qarray) it returns in x_of_link the
     X,Y,Z and the Z,Y',X'' Euler Angles of the end
     effector.

     CheezyIGRIP also reads this file to configure its
     graphical representation of the manipulator, thus
     all the global variables.  IKOR does not yet require
     all of this information, but in the future will.

  inputs:  Initial Qarray and LL[0]-LL[4], & many globals
  outputs: Position of every link in 3space (x_of_link)

  Authors:      Date:      Modifications:
   c hacker      12/94     Created
                  3/95     Platform Included
*/
void GET_ACTUAL_X(MATRIX *Qarray, MATRIX *x_of_link)
{
  static int not_first_time=0;
  int i,j;
  MATRIX *CurrentFrame, *TempFrame,
   *TOC  ,   /* transform from origin to center of platform */
   *TCR  ,   /* transform rotation about center of platform */
   *TRX  ,   /* transform from center to base of manip */
   *TXB  ,
   *TOPO ,   /* transform from manip base to first joint */
   *TAOP ,   /* transform from first to second joint */
```

```
   *TBA  ,    /* transform from second to third joint */
   *TBPB ,    /* transform from third to fourth joint */
   *TCBP ,    /* transform from fourth to fifth joint */
   *TCPC ,    /* transform from fifth to sixth joint */
   *TDCP ,    /* transform from sixth to seventh joint */
   *TDTT ;    /* transform to conventional EE reference */

  Qarray->p[3][0] = (Qarray->p[1][0]+Qarray->p[0][0])- PI/2;
  /* links 1-4 are drawn on local y axes */
  A->p[1][0]=Robot->LINKS[0]  ; A->p[3][0]=1.0;
  B->p[1][0]=Robot->LINKS[1]  ; B->p[3][0]=1.0;
  C->p[1][0]=Qarray->p[2][0]  ; C->p[3][0]=1.0;
  D->p[1][0]=Robot->LINKS[3]  ; D->p[3][0]=1.0;
  E->p[0][0]=Robot->LINKS[4]  ; E->p[3][0]=1.0;

  R->p[2][0]=0.0; R->p[3][0]=1.0;  /* manip base offset drwn on local
y axis */
  X->p[2][0]=0.0; X->p[3][0]=1.0;

  if (not_first_time)
  {
    mat_free(XW);
    mat_free(RW);
    mat_free(AW);
    mat_free(BW);
    mat_free(CW);
    mat_free(DW);
    mat_free(EW);
  }

  /* shift 2 x,y plt cntr */
  TOC          = getT2(0,0,0,Qarray->p[5][0],Qarray->p[6][0],0);
  /* rotate about plt cntr*/
  TCR          = getT2(Qarray->p[7][0]+PI/2,0,0,0,0,0);
  CurrentFrame = mat_mul2(TOC,TCR);

  /* locate the four corners of the platform */
  PW1=mat_mul2(CurrentFrame,P1);
  PW2=mat_mul2(CurrentFrame,P2);
  PW3=mat_mul2(CurrentFrame,P3);
  PW4=mat_mul2(CurrentFrame,P4);

  /* rot about platform center to dir of base of manip */
  TRX    = getT2(0,0,0,-Robot->PLAT.L_OFF*sin(Robot -> PLAT.ANG_OFF),

Robot->PLAT.L_OFF*cos(Robot->PLAT.ANG_OFF),0);
  TempFrame    = mat_mul2(CurrentFrame,TRX); mat_free(CurrentFrame);
  XW           = mat_mul2(TempFrame,X);

  TXB          = getT2(0,0,0,0,0,Robot->PLAT.Z_OFF);
  CurrentFrame = mat_mul2(TempFrame,TXB); mat_free(TempFrame);
  RW           = mat_mul2(CurrentFrame,R);/*locate the base of manip
*/


  /* shift to joint 1 frame */
  TOPO         = getT2(0,0,PI/2+Qarray->p[0][0],0,0,0);
  TempFrame    = mat_mul2(CurrentFrame,TOPO);  mat_free(CurrentFrame);
  AW           = mat_mul2(TempFrame,A);        /* locate end of link 1
*/
```

```
  /* shift to jnt 2 frame */
  TAOP         = getT2(0,PI,-Qarray->p[1][0],0,Robot->LINKS[0],0);
  CurrentFrame = mat_mul2(TempFrame,TAOP);   mat_free(TempFrame);
  BW           = mat_mul2(CurrentFrame,B);        /* locate end of
link 2 */

  /* shift to joint 3 frame */
  TBA          = getT2(0,0,0,0,Robot->LINKS[1]+Qarray->p[2][0],0);

  TempFrame    = mat_mul2(CurrentFrame,TBA); mat_free(CurrentFrame);
  CW           = mat_mul2(TempFrame,C);     /* locate end of link 4 */

  /* shift to joint 4 frame */
  TBPB         = getT2(0,0,Qarray->p[3][0],0,Qarray->p[2][0],0);
  CurrentFrame = mat_mul2(TempFrame,TBPB);   mat_free(TempFrame);
  DW           = mat_mul2(CurrentFrame,D);

  /* shift to joint 5 frame */
  TCPC   = getT2(PI/2,-PI/2,Qarray->p[4][0]-PI/2,0,Robot->LINKS[3],0);
  TempFrame    = mat_mul2(CurrentFrame,TCPC);
mat_free(CurrentFrame);

  /* shift to EE frame */
  TDCP         = getT2(-PI/2,-PI,0,0,Robot->LINKS[4],0);
  CurrentFrame = mat_mul2(TempFrame,TDCP);     mat_free(TempFrame);

  /* locate the four drawing points of the end effector */
  EW1 = mat_mul2(CurrentFrame,E1);
  EW2 = mat_mul2(CurrentFrame,E2);
  EW3 = mat_mul2(CurrentFrame,E3);
  EW4 = mat_mul2(CurrentFrame,E4);

  TDTT         = getT2(0,PI/2,0,0,0,0);
  TempFrame    = mat_mul2(CurrentFrame,TDTT);
  ExtractRPY2(TempFrame, x_of_link);

  EW         = mat_mul2(CurrentFrame,E);

  mat_free(CurrentFrame);
  mat_free(TempFrame);
  not_first_time++;

  for (i=0; i<3; i++)
  {
    x_of_link->p[0][i] = RW->p[i][0];
    x_of_link->p[1][i] = AW->p[i][0];
    x_of_link->p[2][i] = BW->p[i][0];
    x_of_link->p[3][i] = CW->p[i][0];
    x_of_link->p[4][i] = DW->p[i][0];
    x_of_link->p[5][i] = EW->p[i][0];
  }

}/*END GET_WORLD_COORDS*/
```

```
ROBOT.DAT
ROBOT_HERMIES.DAT
/***************************** links ***************************
 *                                                             *
 *  Number of Links and thier respective lengths.  Lengths must *
 *  be in meters and in order from base to end effector.       *
 *                                                             *
 ***************************************************************/
       !   Number of Links
       5

       !   link lengths
       0.356
       0.635
       0.508
       0.343
       0.029

/***************************** Angles **************************
 *                                                             *
 *  Number of Angles and thier respective limits (in degrees), *
 *  whether they are translational (Prismatic) or rotational.  *
 *  'N' means not prismatic while 'Y' means is prismatic.      *
 *  Weight is how active the joint will be.  Weight is a value *
 *  greater than zero and less than or equal to one (one is the *
 *  normal weight).  Finally, home is the initial location of  *
 *  the joint or angle.                                        *
 *                                                             *
 ***************************************************************/

       !   Number of Angles
       10

       !   Max Limit,      Min Limit,     Prismatic,   Weight   Home
             360             -360           N          1.0      0
             360             -360           N          1.0    180
             360             -360           N          1.0      0
             360             -360           N          1.0      0
             360             -360           N          1.0      0
             360             -360           N          1.0     90
             360             -360           N          1.0      0
             300             -300           N          1.0      0
             360             -360           N          1.0      0
             300             -300           N          1.0      0

/************************** PLATFORM ***************************
 *                                                             *
 *  'Y' if the manipulator has a platform, 'N' if it does not  *
 *  or will not be used.  It is best to have the platform as   *
 *  the last angles on the kinematic chain so that the user can *
 *  determine real-time whether to use the platform or not.    *
 *  Length, Width, Thickness, offset in Z of arm from platform, *
 *  offset in length of arm from center of platform, and the   *
 *  offset angle, in degrees of the first link are necessary.  *
 *                                                             *
 ***************************************************************/

       !  Use a Platform?
       Y

       !  Plat_Length,      Plat_Width,     Plat_Thick
             4                 3               2

       !  Z_offset,         L_offset,       ANG_offset
             .3                .61             0.0

/************************ INV Kinematic File ******************/
 *                                                             *
 *  This file must contain compatible versions of GET_JACOB and *
 *  GET_ACTUAL_X (Jacobian, and Forward Kinematic procedures   *
 *  respectively.  See the User's manual on how to create com- *
 *  patible versions of this file.                             *
 *                                                             *
 ***************************************************************/

       ! Filename and Path
       Jacob_HERMIES.c
```

```
ROBOT_AIRARM.DAT
/***************************** links ****************************
 *                                                              *
 *  Number of Links and thier respective lengths.  Lengths must *
 *  be in meters and in order from base to end effector.        *
 *                                                              *
 ****************************************************************/
       !   Number of Links
       5

       !   link lengths
       0.6096
       1.3716
       0.6096
       0.1524
       0.2048

/***************************** Angles ***************************
 *                                                              *
 *  Number of Angles and thier respective limits (in degrees),  *
 *  whether they are translational (Prismatic) or rotational.   *
 *  'N' means not prismatic while 'Y' means is prismatic.       *
 *  Weight is how active the joint will be.  Weight is a value  *
 *  greater than zero and less than or equal to one (one is the *
 *  normal weight).  Finally, home is the initial location of   *
 *  the joint or angle.                                         *
 *                                                              *
 ****************************************************************/

       !   Number of Angles
       8

       !   Max Limit,      Min Limit,      Prismatic,   Weight,
Home
            45              -45             N          1.0      -30
            160             +10             N          1.0      135
            24              -0              Y          1.0       12
            360             -360            N          1.0       30
            90              -90             N          1.0        0
            250             -250            N          1.0        0
            250             -250            N          1.0        0
            20000           -20000          N          1.0        0

/************************** PLATFORM ****************************
 *                                                              *
 *  'Y' if the manipulator has a platform, 'N' if it does not   *
 *  or will not be used.  It is best to have the platform as    *
 *  the last angles on the kinematic chain so that the user can *
 *  determine real-time whether to use the platform or not.     *
 *  Length, Width, Thickness, offset in Z of arm from platform, *
 *  offset in length of arm from center of platform, and the    *
 *  offset angle, in degrees of the first link are necessary.   *
 *                                                              *
 ****************************************************************/

       !  Use a Platform?
       Y

       !  Plat_Length,     Plat_Width,     Plat_Thick
            4               3               2

       !  Z_offset,       L_offset,       ANG_offset
            .0              .1              0.0

/************************ INV Kinematic File ********************/
 *                                                              *
 *  This file must contain compatible versions of GET_JACOB and *
 *  GET_ACTUAL_X (Jacobian, and Forward Kinematic procedures    *
 *  respectively.  See the User's manual on how to create com-  *
 *  patible versions of this file.                              *
 *                                                              *
 ****************************************************************/

       ! Filename and Path
       Jacob_AIRARM.c
```

JACOB_UTILS.c

```
/*
    Program Name:   Jacob_UTILS.c

    description:  Driver for GET_JACOB as well as code for
        the transformation matrixes and the extraction of
        Euler Angles.

    Procedures:
        GET_JACOBIAN            :
        GET_JACOBIAN_ALTERED    :
        get_T2                  :
        ExtractRPY2             :
                                                          */
#include "general.h"        /* Generic Constants         */
/*
    name:     GET_JACOBIAN /GET_JACOBIAN_ALTERED
    description:  if N SPACE == 6 or 3, then the jacobian
       is a (6x7), or (6x10), or a (3x7), or (3x10) matrix.

       For Obstacle Avoidance, the Jacobian must be comput-
       ed at the point of intersection.  Functions in
       Jacob_UTILS.c accomplish this assuming the follow-
       ing hold:
         1. The links should be labeled as LL[0] - LL[4]
         2. Change constants such as Link Lengths ect..
            into a data file (see ROBOT_AIRARM.dat and
            the Users Guide)

    inputs:  Initial Qarray and LL[0]-LL[4]
    outputs: Returns the Jacobian

    Authors:      Date:     Modifications:
     c hacker      10/94    Created
                                                          */
void GET_JACOBIAN (MATRIX *Jacob, MATRIX *Qarray)
{
  int i;
  for (i=0; i<Robot->NL; i++) LL[i] = Robot->LINKS[i];
  GET_JACOB (Jacob, Qarray); }

void GET_JACOBIAN_ALTERED (MATRIX *Jacob, MATRIX *Qarray)
{ GET_JACOB (Jacob, Qarray); }

/*
    name:                    getT2
    description:   Calculate transformation matrix for a
        given three angular rotations and three transla-
        tions.  Parameters entered in the order: z axis
        rotation, y axis rotation, x axis rotation, x axis
        translation, y axis translation, z axis transla-
        tion. The following equations come from the mult-
        iplication of three separate rotation matrices and
        one translation matrix (all are 4x4).

    inputs:  x,y,z rot and x,y,z tran
    outputs: Returns the Transformation Matrix

    Authors:      Date:     Modifications:
    unkown        ?/??      Created
                                                          */
MATRIX *getT2(double ZA,double YB,double XG,
              double tx,double ty,double tz)
{
double sa, sb, sg, ca, cb, cg;
MATRIX *T;

T=mat_malloc(4,4);

sa = sin(ZA); sb = sin(YB); sg = sin(XG);
ca = cos(ZA); cb = cos(YB); cg = cos(XG);

T->p[0][0]=  ca*cb;  T->p[0][1]=ca*sb*sg-sa*cg;
T->p[0][2]=ca*sb*cg+sa*sg;
T->p[1][0]=  sa*cb;  T->p[1][1]=sa*sb*sg+ca*cg;
T->p[1][2]=sa*sb*cg-ca*sg;
T->p[2][0]= -sb;     T->p[2][1]=cb*sg;          T->p[2][2]=cb*cg;
T->p[3][0]=  0.0;    T->p[3][1]=0.0;            T->p[3][2]=0.0;
T->p[0][3]= tx;
T->p[1][3]= ty;
T->p[2][3]= tz;
T->p[3][3]=  1;

return (T);
}  /* getT2 */


/*
    name:                    ExtractRPY2
    description: Now to extract Euler Angles from c matrix
        which is in effect the homogenous transform matrix
        from base to end effector.  Purpose- extracts roll,
        pitch, and yaw angles from orthogonal rotation
        matrix using  J.J. Craig's algorithm(2nd ed. 1989)

    inputs:  position and transformation matrix
    outputs: Returns Euler angles

    Authors:      Date:     Modifications:
    unkown        ?/??      Implemented
    c hacker      3/95      icorporated new x_of_link
                                                          */
void ExtractRPY2(MATRIX * T, MATRIX *x_of_link)
{
  double temp1, temp2, temp3,
         yaw = 0, pitch = 0, roll = 0;
  temp1=sqrt(T -> p[0][0] * T -> p[0][0] + T -> p[1][0]*T -> p[1][0]);
  temp2= -T -> p[2][0];
  pitch = atan2(temp2, temp1);       /* pitch or attitude angle  */
  if (fabs(pitch - (PI / 2.0e0)) < SMALL)
  {
    yaw = ZERO;                      /* yaw or heading angle  */
    temp1 = T -> p[1][1];
```

```
    temp2 = T -> p[0][1];
    roll = atan2(temp2, temp1);   /* roll or bank angle    */
  }
  else if (fabs(pitch + (PI / 2.0e0)) < SMALL)
  {
    yaw = ZERO;                   /* yaw or heading angle  */
    temp1 = T -> p[1][1];
    temp2 = T -> p[0][1];
    roll = -atan2(temp2, temp1);  /* roll or bank angle    */
  }
  else
  {
    temp3 = cos(pitch);
    temp2 = T -> p[1][0] / temp3;
    temp1 = T -> p[0][0] / temp3;
    yaw = atan2(temp2, temp1);       /* yaw or heading angle  */
    temp2 = T -> p[2][1] / temp3;
    temp1 = T -> p[2][2] / temp3;
    roll = atan2(temp2, temp1);      /* roll or bank angle    */
  }

  if (x_of_link->cols > 3)  x_of_link -> p[Robot->NL][3] = yaw;
  if (x_of_link->cols > 4)  x_of_link -> p[Robot->NL][4] = pitch;
  if (x_of_link->cols > 5)  x_of_link -> p[Robot->NL][5] = roll;

}                          /* ExtractRPY2 */
```

INTERNAL DISTRIBUTION

1. G. A. Armstrong
2. J. E. Baker
3. E. C. Bradley
4. B. L. Burks
5. J. B. Chesser
6. J. V. Draper
7. V. B. Graves
8. D. C. Haley
9. W. R. Hamel
10. J. H. Hannah
11–15. J. N. Herndon
16. J. F. Jansen
17. S. M. Killough
18. R. L. Kress
19. C. T. Kring
20. M. W. Noakes
21. C. E. Oliver
22. H. S. Payne
23–27. F. G. Pin
28. A. H. Primm
29. B. S. Richardson
30–34. S. L. Schrock
35. K. U. Vandergriff
36. V. K. Varma
37. B. S. Weil
38. H. R. Yook
39–40. Laboratory Records
41. Laboratory Records ORNL-RC
42. RPSD Publications Office
43. ORNL Patent Section
44. Central Research Library
45. Document Reference Section

EXTERNAL DISTRIBUTION

46–50. Gregory A. Fries, 1205 N. Belgrade Rd., Silver Spring, MD 20902
51–55. Charles J. Hacker, 5973 Poplar Pike Extended #5, Memphis, TN 38119
56. S. R. Martin, Jr., Acting Program Manager, Fusion and Nuclear Technology Branch, Energy Programs Division, Department of Energy, X-10 Site, P.O. Box 2008, Oak Ridge, TN 37831-6269
57. Kristi Morgansen, 25 Parker St., Arlington, MA 02174
58. Faithlyn Tulloch, Escondido Village, Apt. 89C, Stanford, CA 94305
59. Office of Assistant Manager for Energy Research and Development, Oak Ridge Operations Office, Department of Energy, P.O. Box 2008, Oak Ridge, TN 37831-6269
60–61. Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, TN 37831