DETERMINATION OF ROBOT TRAJECTORIES SATISFYING
JOINT LIMIT AND INTERFERENCE CONSTRAINTS
USING AN OPTIMIZATION METHOD

By
Ralph Oliver Buchal

B.A.Sc., The University of British Columbia, 1980
M.A.Sc., The University of British Columbia, 1984

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

in
THE FACULTY OF GRADUATE STUDIES
Department of Mechanical Engineering

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA
July 1987
© Ralph Oliver Buchal, 1987
In presenting this thesis in partial fulfilment of the requirements for an advanced
degree at the University of British Columbia, I agree that the Library shall make it
freely available for reference and study. I further agree that permission for extensive
copying of this thesis for scholarly purposes may be granted by the head of my
department or by his or her representatives. It is understood that copying or
publication of this thesis for financial gain shall not be allowed without my written
permission.

Department of Mechanical Engineering
The University of British Columbia
1956 Main Mall
Vancouver, Canada
V6T 1Y3

Date July 22, 1987
ABSTRACT

An important problem in robotics research is the automatic off-line planning of optimal robot trajectories to perform specified tasks while satisfying physical constraints.

This thesis proposes a method for finding an optimal geometric robot trajectory subject to the constraints of joint displacement limits and interference avoidance. A geometric method for calculating the distance between convex polyhedra is presented, and the method is implemented in two dimensions for the calculation of interference. Point-to-point trajectory planning is posed as a two-point boundary value problem in the calculus of variations. The kinematic constraints are formulated as exterior penalty functions and are combined with other optimization criteria to form a cost functional. The problem is solved by discretizing the problem and numerically minimizing the cost functional by using a steepest-descent approach to iteratively modify the trajectory. Any starting trajectory which satisfies the boundary conditions is acceptable, but different starting trajectories may converge to different locally optimal final trajectories.

The method has been implemented for the two-dimensional case by an interactive FORTRAN program running on a VAX 11/750 computer. Successful results were obtained for a number of test cases, and further work has been identified to allow application of the method to a wide range of problems.
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td></td>
<td>ii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td></td>
<td>ix</td>
</tr>
<tr>
<td>NOMENCLATURE</td>
<td></td>
<td>xiii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENT</td>
<td></td>
<td>xx</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Background and Motivation</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1.2 A Summary of Previous Work in Trajectory Planning</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>1.2.1 Configuration Space Methods</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>1.2.2 Local Optimization Methods</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>1.2.3 Potential Field Methods</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>1.2.4 Optimal Control Methods</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>1.3 Contribution of this Thesis</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>2. KINEMATICS</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>2.1 The Gripper Position as a Function of the Joint Variables</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>2.2 The Denavit-Hartenberg Transformation</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>2.3 The Forward Kinematic Equation</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>2.4 Specification of the Task Coordinate Vector</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>2.4.1 A Linearized Representation of Rotational Error</td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>2.5 Inverse Kinematics</td>
<td></td>
<td>17</td>
</tr>
<tr>
<td>2.5.1 Explicit Solutions</td>
<td></td>
<td>17</td>
</tr>
<tr>
<td>2.5.2 A Linearized General Solution</td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>2.6 Inverse Kinematic Solutions for Redundant Systems</td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>2.6.1 Finding an Optimal Inverse Solution</td>
<td></td>
<td>20</td>
</tr>
</tbody>
</table>
2.7 An Iterative Numerical Solution ....................... 20
  2.7.1 Linearization Error ......................... 25
  2.7.2 An Iterative Algorithm ...................... 26
  2.7.3 Numerical Results ......................... 27

3. ANALYSIS OF INTERFERENCE .......................... 45
  3.1 A Mathematical Definition of Interference Avoidance . 45
  3.2 Approximation of the Swept Volume by Convex Hulls .... 46
  3.3 Calculation of the Distance between Stationary
      Objects ....................................... 47
  3.4 The Distance Between Convex Polyhedra ............... 50
      3.4.1 Calculating the Separating Distance for
            an Arbitrary Normal .................... 50
      3.4.2 Finding the Optimal Face Plane ............ 52
      3.4.3 Calculation of the Separating Distance
            for Disjoint Polyhedra ................. 53
            3.4.3.1 Steepest Descent Graph Search ....... 55
            3.4.3.2 Calculating the Optimal Tangent
                    Plane and Separating Distance ....... 56
            3.4.3.3 Distance Calculations for Disjoint
                    Convex Polygons in 2-D ................ 65
      3.4.4 Calculation of Penetration for
            Intersecting Polyhedra ................. 66
            3.4.4.1 Calculation of the Penetration Vector . 67
            3.4.4.2 Calculation of the Net Translation
                    of the $k^{th}$ Position to Reduce
                    Penetration .......................... 70
            3.4.4.3 Best Fit (Minimum Norm) Joint Space
Displacement to Reduce a Set of Penetration Vectors .................72

4. TRAJECTORY OPTIMIZATION ........................................80
   4.1 Optimization by Minimizing a Functional .....................80
      4.1.1 Local and Global Optima .................................80
      4.1.2 Optimality Conditions for the Continuous Case ..........81
      4.1.3 Optimality Conditions for the Discrete Case .............83
      4.1.4 The Gradient of a Discrete Functional ..................86
      4.1.5 A Modified Newton-Raphson Method for Numerical Minimization of a Functional ........88
      4.1.6 Constraints as Penalty Functions .........................90
      4.1.7 Dynamic and Geometric Trajectory Optimization ..........92
   4.2 A Robot Trajectory Planning Problem .........................93
      4.2.1 Geometric Constraints .................................95
      4.2.2 Reduction of Penetration ................................96
      4.2.3 Optimization Criteria ..................................97
      4.2.4 Proposed Quadratic Cost Functional .....................97
      4.2.5 The Discrete Form of the Problem .......................99
      4.2.6 The Discrete Gradient .................................100
      4.2.7 Iterative Reduction of the Cost Functional 102
      4.2.8 Boundary Conditions .................................104
      4.2.9 Feasibility of Local Optima ............................105

5. NUMERICAL OPTIMIZATION .........................................106
5.1 A Discrete Approximation of the Trajectory ..........106
  5.1.1 Specification of a Starting Trajectory ....106
  5.1.2 Generation of the Discrete Trajectory by
        Interpolation ..........................107

5.2 An Algorithm for the Iterative Reduction of the
        Cost Functional ..........................108
  5.2.1 Linearization Error ........................111
  5.2.2 Calculation of the Iteration Step Size ....112
  5.2.3 Optimal Spacing of the Discrete
        Manipulator Configurations ...............113

5.3 Satisfaction of Constraints ........................114

5.4 Finding a Globally Optimal Trajectory ..........114

6. NUMERICAL RESULTS ..............................116
  6.1 Test Cases ...................................119
    6.1.1 A Benchmark Case with Multiple Obstacles
           - Case 1 ..................................120
    6.1.2 Varying the Joint Limits - Cases 2 and 3 ..126
    6.1.3 Varying the Maximum Trajectory
           Linearization Error - Cases 4 and 5 .......126
    6.1.4 Varying the Maximum Iteration
           Linearization Error ........................132
    6.1.5 A Different Starting Trajectory - Case 6 ..132
    6.1.6 A Different Benchmark Case with a Single
           Obstacle - Case 7 ..........................137
    6.1.7 Varying the Iteration Step Size - Cases 8
           and 9 .....................................137
    6.1.8 A Case with Multiple Links - Case 10 ......148
IV.1.2 Running OPT ........................................... 173
IV.1.3 Input Files ........................................... 175
IV.1.4 Output Files .......................................... 181

IV.2 Functional Description of OPT ......................... 182
IV.2.1 Nesting of Subroutines ............................... 182
IV.2.2 Functional Descriptions of Subroutines .......... 183
IV.2.3 Utility Routines ..................................... 187
IV.2.4 Documentation for UBC Singular Value
          Decomposition Routines ........................... 188

IV.3 Common Variables ...................................... 197

IV.4 Generation of a Starting Trajectory - STRAJ ...... 201
IV.4.1 Users Guide to STRAJ ............................... 201
IV.4.2 Functional Description of Subroutines .......... 205

IV.5 Graphical Output Routines ............................. 206

IV.6 FORTRAN Source Listings ............................... 207
LIST OF FIGURES

Figure 2.1 Diagram of manipulator and obstacles. ...........11

2.2 Link geometry and frames:
   a) revolute joint  b) prismatic joint. ....13

2.3 Base and gripper reference frames. .................15

2.4 Position error versus iteration for test 1. .......29

2.5 Linearization error $\epsilon_i$ versus iteration $i$ for
test 1. ................................31

2.6 Iteration factor $\alpha_i$ and deviation weight $\lambda_i$
versus iteration $i$ for test 1. ......................32

2.7 Components of the search vector versus
iteration for test 1. .................................33

2.8 Plot of iterated solution trajectory for test 1. 34

2.9 Trajectories for different deviation factors $\gamma$:
   a) $\gamma = 0.0$  b) $\gamma = 0.6$  c) $\gamma = 1.0$ . ....35

2.10 Trajectories for different trajectory
linearization errors $\epsilon_{\text{max}}$:
   a) $\epsilon_{\text{max}} = 0.5$  b) $\epsilon_{\text{max}} = 0.2$  c) $\epsilon_{\text{max}} = 0.1$ .37

2.11 Plot of iterated solution trajectory for
test 6. ............................................39

2.12 Plot of iterated solution trajectory for
test 7. ............................................40

2.13 Position error $\Delta x_i$ versus iteration $i$ for
test 8. ............................................42

2.14 Components of the search vector versus

iteration for test 8. .................................43

2.15 Plot of iterated solution trajectory for test 8. .................................44

3.1 A discrete approximation of the swept volume. ...48

3.2 Distance between convex hulls. .........................51

3.3 Two intersecting polygons with multiple local maxima. .................................54

3.4 Possible cases for the distance vector d* ......57

3.5 Geometric test for inclusion of a point in a convex polygonal region. .........................60

3.6 The shortest vector connecting two edges. ......63

3.7 Graphical proof that the penetration vector is normal to a face. .........................69

3.8 Translations of link positions due to penetration of the swept volume. .........................73

3.9 Minimum norm joint change reducing two penetrations. .................................75

3.10 Minimum norm translation for two penetration fractions σ: a) initial position  b) σ = 0
c) σ = 1. .................................77

3.11 Oscillation between translation directions. ......78

3.12 Large minimum norm joint change reducing two small penetrations. .................................79

6.1 Starting trajectory and obstacles for case 1. ..121

6.2 Evolution of the discrete trajectory for the first few iterations for case 1: a) starting trajectory  b) 1 iteration  c) 2 iterations
d) 3 iterations. ........................................122

6.3 Plot of the number of discrete trajectory positions at each iteration for case 1. ..........123

6.4 The optimized trajectory for case 1. ..........124

6.5 Plot of the cost functional versus the number of iterations for case 1. .................125

6.6 Plot of the joint trajectories for case 1. .....127

6.7 Plot of the maximum penetration versus the number of iterations for case 1. ..............128

6.8 Plot of the joint trajectories for case 2. .....129

6.9 Plot of the joint trajectories for case 3. .....130

6.10 Comparison of the cost functionals for case 1 and case 3. ........131

6.11 Optimized trajectories with different trajectory linearization errors:
   a) $\varepsilon = 0.01$ b) $\varepsilon = 0.1$ c) $\varepsilon = 0.2$ .......133

6.12 Comparison of the cost functionals for cases 1, 5 and 6 for different maximum trajectory linearization errors. ......................134

6.13 Starting trajectory for case 6. .................135

6.14 Optimized trajectory for case 6. ...............136

6.15 Starting trajectory and obstacles for Case 7. ..138

6.16 Optimized trajectory for case 7. ...............139

6.17 Plot of the cost functional for case 7. ........140

6.18 Plot of the joint trajectories for case 7. .....141

6.19 Plot of the maximum penetration versus the

xi
6.20 Plot of the maximum joint limit violation versus the number of iterations for case 7. ..............143

6.21 Optimized trajectories with different iteration step sizes:
a) $\alpha = 0.1$  b) $\alpha = 0.6$  c) $\alpha = 1.0$ ............144

6.22 Comparison of the cost functionals of cases 7, 8 and 9 for three different iteration step sizes. ..................................................145

6.23 Comparison of the maximum joint limit violation versus the number of iterations for three iteration step sizes. .........................146

6.24 Comparison of the maximum penetration versus the number of iterations for three iteration step sizes. .............................147

6.25 Optimized trajectory for case 10. .........................149

III.1 PUMA 560 link geometry. .........................171

III.2 Sketches of test workpieces. .........................172
NOMENCLATURE

Symbols

\( a_i \)  
manipulator link \( i \) length

\( a \)  
a vector parallel to the common normal of two lines

\( A \)  
set of all optimal trajectories

\( A \)  
weighting matrix of joint limit penalty quadratic

\( b() \)  
vector state equation, \( \dot{q} = b(q(t),u(t),t) \)

\( B \)  
weighting matrix of interference penalty quadratic

\( C \)  
weighting matrix of acceleration quadratic, coriolis and centripetal tensor in Section 4.1.7.

\( d(n) \)  
minimum distance between two objects in the direction \( n \)

\( d^* \)  
minimum distance between two objects, \( d^* = d(n^*) \)

\( d^*_f \)  
minimum distance along a face normal

\( d_{i} \)  
manipulator link \( i \) offset

\( d_i \)  
\( i^{th} \) translation vector

\( d_{i,j} \)  
translation vector applied to link \( L_i \) due to penetration of link \( L_j \)

\( d_{pen} \)  
penetration vector

\( d^* \)  
minimum distance vector

\( \hat{d} \)  
combined translation vector, \( \hat{d} = [d_1^T, d_2^T, \ldots, d_h^T]^T \)

\( D_i \)  
iteration matrix

\( e \)  
optimal edge of a polygon, equivalent to an optimal
face $f$ for a polyhedron

$\hat{f}_i$ optimal face of convex polyhedron $K_i$. The face normal $n_{\hat{f}_i}$ minimizes $d(n)$.

$f$ actuator force vector

$f_{\text{lim}}$ actuator force limit vector

$F_i$ coordinate frame attached to manipulator link $i$

$g$ the number of penetration vectors $d_i$ such that $\|d_i\| > \sigma \|d_i\|

$g()$ cost function

$g_1()$ joint limit penalty function

$g_2()$ interference penalty function

$g_a()$ augmented cost function

$g_f()$ actuator force limit penalty function

$g_p()$ combined penalty function

$g$ gravity vector

$G$ homogeneous projection matrix, $G = I - J^T J$

$h$ total number of penetration vectors acting on a link

$h_i()$ constraint relationships

$H_{i-1,i}$ homogeneous transformation relating frames $F_{i-1}$, $F_i$

$H_k$ convex set representing the convex hull containing $L_{k-1}$ and $L_k$

$H$ Hessian matrix

$I$ identity matrix
J Jacobian matrix
\hat{J} combined Jacobian \hat{J} = [J_1, J_2, \ldots, J_h]
k state index
k_0 initial state index
k_f final state index
k equivalent axis of rotation
K_i convex set representing object i
K deviation weighting matrix
L_k convex set representing a link at position k
m number of task degrees of freedom, number of spatial dimensions
M positive definite weighting matrix, inertia matrix in Section 4.1.7.
n number of manipulator degrees of freedom, number of objects
n unit normal
n,o,a gripper frame basis vectors
N the set of face normals of two polyhedra K_i, K_j
p normal distance from a plane to the origin
p translation of gripper frame relative to base frame
p_{i,j} point on L_i to be translated by d_{i,j}
p_{\text{pen}} penetration point defined by equation (3.17)
p_{\text{tan}} a tangent point
P set of tangent points
\( q_i \) \( \text{ith component of } q \)

\( q \) \( \text{manipulator joint state vector} \)

\( \dot{q} \) \( \text{an inverse kinematic solution satisfying } \dot{x} = f(\dot{q}) \)

\( \ddot{q} \) \( \text{discrete trajectory vector, } \ddot{q} = [q_{k-1}^T, q_k^T, q_{k+1}^T, \ldots, q_{k_f}^T, q_{k_{f+1}}^T, r]^T \)

\( q^0 \) \( \text{mid-point of joint range} \)

\( q_c \) \( \text{linear estimate of nearest collision-free joint state} \)

\( q_{\text{min}}, q_{\text{max}} \) \( \text{minimum and maximum joint limits} \)

\( q^*(t) \) \( \text{an optimal trajectory minimizing } \phi(q(t)) \)

\( q^*_g(t) \) \( \text{the globally optimal trajectory} \)

\( r \) \( \text{a vector representing a point in space} \)

\( r_c \) \( \text{the tangent vertex of a polyhedron which minimizes } d(n) \)

\( r_p \) \( \text{gripper position vector} \)

\( r_{\text{proj}} \) \( \text{projection of } r_c \text{ onto a face plane} \)

\( r_o \) \( \text{gripper orientation vector} \)

\( r' \) \( \text{projection of a point onto a line} \)

\( R_c \) \( \text{actual rotation matrix relative to a reference frame} \)

\( R_d \) \( \text{desired rotation matrix relative to a reference frame} \)

\( s_i \) \( \text{graph node } i \)

\( s_i, t_i \) \( \text{endpoints of an edge } i \)

\( S \) \( \text{set of graph nodes } w_i \)
\( t_0 \) initial time
\( t_f \) final time
\( T \) homogeneous transformation matrix relating the gripper frame to the base frame
\( u \) a control vector
\( u_i \) a vector parallel to an edge
\( v \) shortest connecting vector between an edge of face \( f \) and an edge containing \( \tau_e \)
\( V \) set of vertices defining a convex polyhedron or polygon,
set of possible shortest connecting vectors \( v \)
\( x \) task state vector specifying the gripper position
\( \hat{x} \) a desired gripper position (task state)
\( z \) an arbitrary vector
\( \alpha \) iteration step size
\( \alpha_i \) link \( i \) twist angle, iteration \( i \) step size
\( \gamma \) deviation factor
\( \delta_1 \) position error tolerance
\( \delta_2 \) optimization tolerance
\( \epsilon \) linearization error
\( \theta \) angle of rotation
\( \lambda \) deviation weighting factor
\( \mu \) scalar parameter
\( \Pi(n,p) \) a plane defined as \( \Pi(n,p) = \{ r : r \cdot n - p = 0 \} \)
\( \tau \) discrete time interval

\( \sigma \) penetration fraction

\( \phi \) cost function

\( \phi_i \) a cost function evaluated at graph node \( w_i \)

\( \Phi \) cost functional

\( \Phi_a \) augmented cost functional

\( \Phi_p \) penalty functional

\( \omega \) angular velocity

\( \Omega_i(t) \) a set representing object \( i \) at time \( t \)

**Scalar and Vector Notation**

\( E, e \) symbols in standard type refer to scalar quantities

\( e \) lower case symbols in bold face refer to vectors

\( E \) upper case symbols in bold face refer to matrices

\( E^+ \) pseudoinverse of \( E \)

\( E^T \) matrix transpose of \( E \)

\( E^{-1} \) matrix inverse of \( E \)

\( \| e \| \) Euclidian norm of \( e \)

\( | E | \) determinant of \( E \)
Subscripts and Superscripts

\( i \) subscripts
- iteration index, object index

\( k \) subscripts
- discrete state index

\( * \) superscript
- an optimal value of the variable

\( \max \) superscript
- a maximum bound on a variable

\( \min \) superscript
- a minimum bound on a variable
ACKNOWLEDGEMENT

I would like to thank my supervisor, Professor Dale Cherchas, for his advice and encouragement throughout my Ph.D. work. Some funding was provided for this work through the Canadian Natural Sciences and Engineering Research Council (NSERC) grant number A-4682.
1. INTRODUCTION

1.1 Background and Motivation

In a competitive world economy, it is essential that industry employ the most effective methods available in order to survive and flourish. In every industry, modern technology provides opportunities for increased capabilities and improved efficiency.

In the manufacturing sector, many diverse technologies are being successfully applied. Computer-Aided-Design (CAD) and Computer-Aided-Manufacturing (CAM) are being increasingly utilized. Flexible, computer-controlled workstations allow automation of tasks not suitable for single purpose, "hard" automation.

Industrial robots play an important role in modern manufacturing. Robots can be programmed to perform coordinated motions repetitively and accurately, and can be applied to many tasks previously done by humans.

Because of their flexibility, robots can be used on a stand-alone basis, performing some well-defined, repetitive task. The robot is programmed by physically leading it through its desired actions, which are stored and then replayed to perform the required task. Applications of this type account for most of the robots currently in use.

It is now possible and desirable to integrate all aspects of the manufacturing process into a unified, computer-controlled system. Such a system combines design, engineering analysis,
process planning, and execution and control of the manufacturing process. Most of the required technology exists, but some problems still require satisfactory solutions.

An important problem is the off-line programming of robots. Manual programming is satisfactory for many tasks, but suffers a number of drawbacks. First, the robot and its workstation must be withdrawn from production each time reprogramming is required. The programming procedure may be lengthy, so robots are not effectively used for tasks which are changed frequently. There is also danger of damage and injury if errors are made during programming. Finally, manually programming does not utilize the robots ability to accurately follow a pre-specified trajectory, essential for the full integration of manufacturing under computer control.

Off-line programming and simulation of robot workstations provides many distinct advantages. First, information about the workstation and task can be drawn directly from existing CAD data bases, and used for generation of appropriate motions. Powerful mathematical techniques can be applied to optimize the resulting program. The program can be simulated and assessed graphically without danger. Finally, the robot can be programmed and controlled in concert with the rest of the integrated system, not as a stand-alone unit.

One aspect of off-line programming is the problem of path or trajectory planning. A robot is a complicated kinematic linkage governed by highly non-linear kinematic and dynamic equations of motion. In a realistic workstation, its motion is restricted by a number of constraints including joint limits, actuator force
limits and physical interference between the robot and stationary obstacles. The problem can be stated as follows:

Given a specified task, generate a robot trajectory which performs the task without violating physical constraints. In addition, other performance criteria may be optimized.

1.2 A Summary of Previous Work in Trajectory Planning

Robot trajectory planning is receiving increasing attention from researchers as its practical importance becomes clear. The approaches taken to date can be broadly categorized as follows:

a) Configuration space methods
b) Local optimization methods
c) Potential field methods
d) Optimal control methods.

1.2.1 Configuration Space Methods

The configuration space approach was developed as a solution to the problem of finding the shortest collision-free path to move an object through a field of obstacles.

With this approach, a global map of collision-free robot configurations (the free-space) is generated. The problem is then
reduced to finding a path through this resultant free-space. Many permutations of this method have been developed, but they all share some common difficulties. Arbitrary orientations of an object are not easily handled since the resulting free-space changes as a function of the orientation. Each additional degree of freedom in the motion of the object adds another dimension to the free space. Also, a different free-space exists for each distinct object. Thus, the effort required for explicit calculation of the complete free-space for a general robot and workstation becomes prohibitive, and methods for simplifying the problem must be sought.

Lozano-Perez [1983] considers the problem of moving a polyhedral object among polyhedral obstacles. The configuration-space obstacles (or forbidden regions) are generated by shrinking the object to a point and correspondingly growing the obstacles. Higher dimensional configuration space obstacles are represented approximately by lower dimensional projections. Rotations are discretized and the extra dimension is approximated by a small number of projections. The intermediate positions are represented by an approximation of the swept volume between specified positions. The shortest collision-free path is found by connecting vertices of the configuration-space obstacles.

Brooks [1983] uses a variation of the configuration space method to solve a limited class of pick-and-place problems for a revolute manipulator. Brooks simplifies the problem by specifying motions as pure translations followed by pure rotations, and he asserts that a single rotational degree of freedom is sufficient for many tasks. The robot arm is decomposed into its constituent
links, and the interference constraints are developed geometrically for each link.

Luh and Campbell [1984] consider the problem of a robot with a prismatic joint, and propose the inclusion of "pseudo-obstacles" to account for possible interference with the long boom of the robot.

Park [1984] proposes to generate the free-space by exhaustive enumeration of all collision-free configurations. Joint space is discretised and each discrete configuration is tested for collision. The number of discrete points increases exponentially with the number of joints, so an enormous number of collision tests must be performed for realistic situations. The resulting map of collision-free configurations requires a great deal of computer memory, even when data compression techniques are employed. The advantages of this approach are its simplicity and generality, and the ease of working with the resulting joint-space representation of the collision map.

Configuration-space methods are purely geometrical and do not account for the kinematics and dynamics of robots. Thus, these methods cannot be naturally extended to deal with constraints such as joint displacement limits and actuator force limits.

1.2.2 Local Optimization Methods

Another approach to path planning has evolved out of the study of robot kinematics. The general inverse kinematic solutions for multi-link manipulators have been developed
extensively [Fournier, Khalil 1977], [Konstantinov, Markov, Nechev 1981], [Konstantinov et al. 1982]. Manipulators with redundant degrees of freedom allow the joints to be reconfigured to avoid constraints while tracking a specified trajectory. The constraints can be joint limits, velocities, accelerations or interference with obstacles. The trajectory in joint space evolves by calculating the constraints instantaneously at each point along the trajectory, and moving the joints away from the constraints while following the specified end-effector path. The method does not generate a globally optimal solution since it is based on instantaneous optimization. The resulting trajectory may be trapped in a cul-de-sac, or may be very circuitous.

Konstantinov et al. [1981] have formulated an inverse kinematic solution which maximizes joint availability for redundant robots. Vukobratovic and Kircanski [1982] consider the dynamics of the robot and propose a solution to minimize the energy consumption of the actuators over a specified trajectory. Yoshikawa [1984] defines a measure of robot manipulability, and formulates a solution to maximize this quantity. Interference avoidance is also incorporated by specifying an interference-free manipulator position. The redundancy in the manipulator is used to minimize the difference between the actual and desired position. This approach does not guarantee that a collision-free trajectory will be found even if one exists.

Kircanski and Vukobratovic [1984] have considered the obstacle avoidance problem by calculating distances between links and obstacles. A solution can be formulated to maximize the distance from obstacles while following a specified path.
Maciejewski and Klein [1985] present a more detailed treatment of the same approach.

1.2.3 Potential Field Methods

The potential field method associates a force or potential with each obstacle to be avoided. This "artificial potential" is included in the equations of motion, and has the effect of forcing the joints away from potential collisions. The approach proposed by Khatib and Le Matre [1978] is summarized in their paper:

The manipulator moves in a field of forces. The final position to be reached is an attractive pole for the terminal device, and the obstacles are repulsive surfaces for all the manipulator parts.

Ozaki and Mohri [1986] have refined this basic method substantially. They define artificial potentials associated with the deviation from the desired path, violation of joint limit constraints, and interference. The deviation of the actual position from the specified position is defined as an objective function to be minimized. Linear inequality constraints are formulated for actuator limits, joint displacement limits, joint velocity limits and interferences. The problem is then solved by linear programming at discrete intervals along the proposed trajectory, and a feasible trajectory is thus obtained. If there is no feasible trajectory which satisfies all of the constraints
while following the specified trajectory, the nearest feasible trajectory is obtained. This method does not find an optimal feasible trajectory.

1.2.4 Optimal Control Methods

Gilbert and Johnson [1985] have proposed a method of path planning based on optimal control. A distance function is developed to represent the distance between convex objects, and its derivative is defined. The distance is incorporated into the cost functional as an interior penalty function, and the problem is solved using numerical methods for optimal control. Because interior penalty functions are used, the starting trajectory must be feasible, which means it must be collision-free. Also, no algorithms are presented for actually calculating the distance between objects.

1.3 Contributions of this Thesis

This thesis proposes a method for finding an optimal geometric robot trajectory subject to the constraints of joint displacement limits and interference avoidance. A geometric method for calculating the distance between objects modelled as convex polyhedra is presented. Point-to-point trajectory planning is posed as a two-point boundary value problem in the calculus of variations. The kinematic constraints are formulated as exterior
penalty functions and are combined with other optimization criteria to form a cost functional. The problem is solved by discretizing the problem and numerically minimizing the cost functional by using a steepest-descent approach to iteratively improve the trajectory. Any starting trajectory is acceptable since the use of exterior penalty functions means that all configurations are defined. However, different starting trajectories may converge to different locally optimal final trajectories. In general, several starting trajectories are tried until a resulting final trajectory is found which satisfies the constraints.
2. KINETICS

2.1 The Gripper Position as a Function of the Joint Variables

Let us define a manipulator consisting of \( n \) links connected by \( n \) joints. We say that the manipulator has \( n \) degrees of freedom. Let us also define a task defined by \( m \) coordinates which specify the position and orientation of the end effector or gripper of the robot. We will say that there are \( m \) task degrees of freedom.

We will represent the joint state by the vector

\[
q = [q_1, q_2, \ldots, q_n]^T, \tag{2.1}
\]

and the task state by the vector

\[
x = [x_1, x_2, \ldots, x_m]^T. \tag{2.2}
\]

Given the joint state \( q \) we can always find the task state \( x \) using the forward kinematic equation,

\[
x = f(q), \tag{2.3}
\]

where \( f(q) \), usually a highly non-linear function, can be found for any rigid link manipulator.

Figure 2.1 shows a simple manipulator with four joint degrees of freedom and two task degrees of freedom.
Figure 2.1    Diagram of manipulator and obstacles.
2.2 The Denavit-Hartenberg Transformation

A transformation relating the coordinate frames $F_{i-1}, F_i$ attached to adjacent links $i-1, i$ of a linkage can be specified in a standard form known as the Denavit-Hartenberg transformation [Paul, 1981, pp.50-55],

\[
H_{i-1,i} = \begin{bmatrix}
\cos q_i & -\sin q_i \cos \alpha_i & \sin q_i \sin \alpha_i & a_i \cos q_i \\
\sin q_i & \cos q_i \cos \alpha_i & -\cos q_i \sin \alpha_i & a_i \sin q_i \\
0 & \sin \alpha_i & \cos \alpha_i & d_i \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

(2.4)

where $q_i$ = $i^{th}$ joint variable,

$\alpha$ = link twist,

$a$ = link length,

$d$ = link offset.

The link geometry parameters and link coordinate frames are illustrated in Figure 2.2, with $q_i$ represented by $\theta_i$.

2.3 The Forward Kinematic Equation

The net transformation relating the last frame $F_n$ to the
Figure 2.2 Link geometry and frames:
a) revolute joint b) prismatic joint.
(Figure 2.3) can be found as a product of the Denavit-Hartenberg transformations for each link pair [Buchal, 1984, pp. 17-22], [Buchal et al., 1986, pp. 16-20], [Paul, 1981, pp. 41-42],

\[ H_{1, n} = H_{1, 2} H_{2, 3} \ldots H_{n-1, n}. \]  

(2.5)

The net transformation can also be expressed as a homogeneous transformation,

\[ T = \begin{bmatrix} n & o & a & p \\ n & o & a & p \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 1 \end{bmatrix}, \]  

(2.6)

where the vectors \( n, o, a \) are the basis vectors of the gripper frame, and the vector \( p \) is the location of the gripper frame relative to the base reference frame. Thus, \( T \) specifies the task state of the robot. Equivalently, the vectors \( n, o, a \) compose the columns of a rotation matrix, and \( p \) represents a translation.

We can now express the forward kinematic equation as

\[ T = H_{1, n}(q). \]  

(2.7)

2.4 Specification of the Task Coordinate Vector

The robot task vector in 3-D task space can be defined as

\[ x = \left[ r_p^T, r_o^T \right]^T. \]  

(2.8)
Figure 2.3  Base and gripper reference frames.
where \( \mathbf{r}_p \) is a 3 \( \times \) 1 position vector and \( \mathbf{r}_o \) is a 3 \( \times \) 1 orientation vector.

The position vector is given directly by the translational part of the homogeneous transformation. The orientation vector must be derived from the 3 \( \times \) 3 rotation component of the transformation matrix. The rotation is commonly represented by Euler angles, roll-pitch-yaw angles [Paul, 1981, pp.43-47], or an equivalent axis of rotation [Paul, 1981, pp.29-32].

### 2.4.1 A Linearized Representation of Rotational Error

Suppose the desired orientation of the robot gripper is expressed as a 3 \( \times \) 3 rotation matrix \( R_d \) relative to a reference frame, and the current orientation is given by a matrix \( R_c \). We can define a rotational transformation from the current orientation to the desired orientation as

\[
R_d = \text{rot}(k, \theta) R_c ,
\]

(2.9)

where \( \text{rot}(k, \theta) \) is a rotation of angle \( \theta \) about an axis \( k \).

Rearranging, we get

\[
\text{rot}(k, \theta) = R_d R_c^{-1} .
\]

(2.10)

We can solve for \( k, \theta \) by the method described in [Paul, 1981, pp.25-32]. The orientation error vector we will use is \( \theta k \). Note
that the time derivative of this orientation vector is the angular velocity, \( \omega = \dot{k} \). Thus, this choice of orientation vector allows the use of a Jacobian matrix derived for linear and angular velocities.

2.5 **Inverse Kinematics**

The inverse kinematics problem consists of finding the joint state corresponding to a given task state,

\[
q = g(x).
\]  

(2.11)

There are three possible cases:

1. The system is overconstrained and no solutions exist.

2. There are a finite number of solutions.

3. The robot has redundant degrees of freedom and an infinite number of solutions exist.

2.5.1 **Explicit solutions**

It has been shown [Paul, 1981, pp.65-83] that explicit inverse solutions can be found for many particular non-redundant
(n - m) manipulators of practical interest. These methods do not apply to general non-redundant or redundant manipulators.

2.5.2 A Linearized General Solution

No closed-form inverse solution can be found for the general case. For incremental changes, however, a linearized general solution can be defined as

$$\delta x = J \delta q$$  \hspace{1cm} (2.12)

where J is the Jacobian matrix. Equivalently,

$$\dot{x} = J \dot{q}.$$  \hspace{1cm} (2.13)

The three possible cases can now be defined as follows:

1. No inverse solutions exist if rank J ≠ rank \([J; \delta x]\).
   This is usually the case for overconstrained systems with \(m > n\), or at singular configurations of non-redundant systems.

2. A finite number of inverse solutions exist if \(m - n = \text{rank} J\). This is the usual case for non-redundant systems.

3. An infinite number of inverse solutions exist if \(m < n\)
and \( \text{rank } J = m \) or \( \text{rank } J = \text{rank } [J \delta x] < m \). This is the case for redundant systems.

2.6 Inverse Kinematic Solutions for Redundant Systems

It is well known [Albert, 1972, pp. 30-31] that a general inverse solution can be formulated as

\[
\delta q = J^+ \delta x + (I-J^+J)z
\]  

(2.14)

where \( J^+ \) is the pseudo-inverse of \( J \), \( (I-J^+J) \) is the projection onto the null-space of \( J \) and \( z \) is an arbitrary vector (see Appendix I.1.2). The forward solution \( \delta x = J \delta q \) is invariant with respect to \( z \). The term \( J^+ \delta x \) is the minimum-norm particular solution, and the term \( (I-J^+J)z \) is the homogeneous solution. Two specific cases are worth noting:

1. For over-constrained systems, the solution

\[
\delta q = J^+ \delta x
\]

(2.15)

gives the best-fit (least-squares) solution minimizing

\[
\| \delta x - J \delta q \|
\]

(2.16)

2. For non-redundant systems \( (m = n) \), if \( J \) is nonsingular then \( J^+ = J^{-1} \).
2.6.1 Finding an Optimal Inverse Solution

For systems with redundancy, it is possible to optimize the incremental solution $\delta q$ with respect to selected criteria by choosing an appropriate $z$. The vector $z$ can be chosen to maximize joint availability [Konstantinov, Markov and Nechev, 1981], maximize joint manipulability [Yoshikawa, 1984], and avoid collisions [Kircanski and Vukobratovic, 1984], [Maciejewski and Klein, 1985]. This optimization is strictly instantaneous, and global criteria such as elapsed time or total energy cannot be optimized in this way. The instantaneously optimal trajectory depends on the starting configuration, and the trajectory which is globally optimal may not be instantaneously optimal.

An absolute inverse solution $q$ can be obtained by iteratively applying the incremental solution until a desired task state $x$ is reached. One such iterative algorithm is presented in [Goldenberg, Benhabib, and Fenton, 1985].

2.7 An Iterative Numerical Solution

The absolute inverse kinematics problem can be stated as follows:

For a desired task vector $\hat{x}$, find a corresponding joint vector $\hat{q}$ satisfying the forward kinematic equation,
Note that for redundant systems, there may be infinitely many solutions $\hat{q}$.

Let $q$ specify a manipulator configuration and $x = f(q)$. Define the position error

$$\Delta x = \hat{x} - x ,$$  \hspace{1cm} (2.18)

and the joint error

$$\Delta q = \hat{q} - q .$$  \hspace{1cm} (2.19)

The inverse kinematics problem can be posed as a minimization problem, where the function $\phi$ to be minimized is a quadratic of the position error,

$$\phi = (\hat{x} - x)^T(\hat{x} - x) .$$  \hspace{1cm} (2.20)

Equivalently, we can minimize a quadratic of the joint error,

$$\phi = (\hat{q} - q)^T(\hat{q} - q) .$$  \hspace{1cm} (2.21)

This function can be minimized by the method of steepest descent using an iterative sequence of the form

$$q_{i+1} = q_i + \alpha_i \Delta q_i ,$$  \hspace{1cm} (2.22)
where $\alpha_1 \Delta q_1$ is known as the search vector. Let $\Delta q_1$ be the solution of the redundant problem,

$$\Delta q_1 = J_1^+ \Delta x_1 + (I-J_1^+J_1)z_1,$$

(2.23)

where $J_1^+$ is $J^+$ evaluated at $q_1$, $\Delta x_1 = \hat{x}_1 - x_1$, and $z_1$ is chosen to optimize some set of criteria. For a sufficiently small step size $\alpha_1$, the end effector will move incrementally along a straight line, in the direction $\Delta x$, from its initial position to the goal position. There is no physical reason to constrain end effector motion to a straight line. A less constrained solution is

$$\Delta q_1 = J_1^+ \Delta x_1 + (I-J_1^+J_1)z_1 + K_1 z_1,$$

(2.24)

where $K_1$ is some positive definite weighting matrix. Using this solution, the end effector moves locally in the direction $\Delta x_1 + J_1 K_1 z_1$, and any end effector path is possible, depending on the choice of $z_1$.

The expanded iterative sequence is

$$q_{i+1} = q_i + \alpha_i (J_1^+ \Delta x_1 + (I-J_1^+J_1)z_1 + K_1 z_1).$$

(2.25)

The iteration factor $\alpha_i \in [0,1]$ can be chosen to optimize convergence. When the error $\Delta q_1$ is large, a small $\alpha_i$ produces a linear, steepest descent convergence. When the error becomes small, $\alpha_i$ can approach 1 to give a quadratic convergence. In general, the step size is adjusted so that the linearization error
stays within some predetermined bounds. We can insure that the sequence converges to an inverse solution \( \hat{q} \) by choosing \( K_1 \) so that

\[
\| \hat{x} - x_{i+1} \| < \lambda \| \hat{x} - x_i \|, \quad \lambda < 1,
\] (2.26)

where \( x_{i+1} = f(q_{i+1}) \). We can show that

\[
x_{i+1} = x_i + \Delta x_i + J K_1 z_i,
\] (2.27)

and

\[
\hat{x} - x_{i+1} = \hat{x} - (x_i + \Delta x_i + J K_1 z_i).
\] (2.28)

Combining (2.27) and (2.28) and cancelling terms, we get

\[
\hat{x} - x_{i+1} = - J K_1 z_i.
\] (2.29)

Clearly, we can combine (2.18) and (2.29) to show that

\[
\| \hat{x} - x_{i+1} \| < \lambda \| \hat{x} - x_i \|, \quad \lambda < 1,
\] (2.30)

if

\[
\| J K_1 z_i \| < \lambda \| \Delta x_i \|, \quad \lambda < 1.
\] (2.31)

The redundancy of the system can be exploited to find a solution \( \hat{q} \) which optimizes some set of criteria. We will choose
to maximize the range of available motion at each joint so that
the solution will be as far from the joint limits as possible. If
we define the midpoint of the joint range as \( q^0 \), then we wish to
minimize

\[
\phi_1 = (q - q^0)^T M (q - q^0)
\]  

(2.32)

where \( M \) is some positive-definite weighting matrix. The criteria
of minimum joint error (2.21) and maximum joint availability
(2.32) can be combined to get a new cost function \( \phi_2 \),

\[
\phi_2 = (\hat{q} - q)^T (\hat{q} - q) + (q - q^0)^T M (q - q^0).
\]  

(2.33)

An optimal inverse solution can be found by minimizing (2.33).
The joint error term is reduced by \( \Delta q = J^+ \Delta x \) and the availability
term is reduced along the gradient, \( z = -M(q - q^0) \). Combining
these terms, an iterative sequence which minimizes (2.33) is

\[
q_{i+1} = q_i + \alpha_i \left( J_i^+ \Delta x_i - (I-J_i^+J_i)M(q_i - q^0) - K_i M(q_i - q^0) \right).
\]  

(2.34)

As the error \( \Delta x_i \) is reduced, the minimum norm component \( J_i^+ \Delta x_i \)
decreases. The deviation component \( K_i M(q_i - q^0) \) also decreases as
\( \| K \| \) is reduced (2.31). As the solution is optimized, the
homogeneous component \( (I-J_i^+J_i)M(q_i - q^0) \) decreases.
2.7.1 Linearization Error

Linearization error arises due to the use of a linear approximation for a nonlinear function. The linearization error $\epsilon$ can be defined as the difference between the actual and predicted value of the next task state,

$$\epsilon_i = \left\| x_{i+1} - (x_i + \alpha_i (\hat{x}_i - x_i)) \right\|,$$  \hspace{1cm} (2.35)

where $x_{i+1} = f(q_{i+1})$. This can be rewritten as

$$\epsilon_i = \left\| (1-\alpha_i) \Delta x_i - \Delta x_{i+1} \right\|,$$  \hspace{1cm} (2.36)

where $\Delta x_i = \hat{x}_i - x_i$. Recall that

$$q_{i+1} = q_i + \alpha_i \Delta q_i$$  \hspace{1cm} (2.22)

so as $\alpha_i \to 0$,

$$q_{i+1} \to q_i, \hspace{0.5cm} x_{i+1} \to x_i \hspace{0.5cm} \text{and} \hspace{0.5cm} \epsilon \to 0.$$

Thus we see that the linearization error can be reduced by reducing the step size factor $\alpha_i$. In practice, we will choose $\alpha_i$ to keep the linearization error below a specified limit, $\epsilon_{\max}$. 

25
2.7.2 An Iterative Algorithm

We can simplify the iteration sequence (2.34) with the following restrictions:

\[ M = I \]

\[ K \] is a diagonal matrix with all elements equal to \( \lambda \).

The joint center vector \( q^0 \) is the zero vector.

The simplified iteration sequence is

\[
q_{i+1} = q_i + \alpha_i (J_i^T \Delta x_i - (I-J_i^T J_i) q_i - \lambda_i q_i).
\] (2.37)

This iterative inverse solution can be implemented using the following algorithm:

Specify the goal position \( \hat{\mathbf{x}} \) and the initial configuration \( q_0 \).

Specify the position tolerance \( \delta_1 \), the optimization tolerance \( \delta_2 \), the allowable linearization error \( \epsilon_{\text{max}} \), and the deviation factor \( \gamma \in [0,1] \).

Calculate the initial parameters,

\[ x_0 = f(q_0). \]
\[ \Delta x_0 = \hat{x} - x_0, \]
\[ \epsilon_0 = \| \Delta x_0 \|, \]
\[ \alpha_0 = \min( \epsilon_{\text{max}} / \epsilon_0, 1 ), \]
\[ \lambda_0 = \gamma \frac{\| \Delta x_0 \|}{\| J q_0 \|}, \]

\[ i = 0, \]

UNTIL (\[ \| \hat{x} - x_i \| < \delta_i \] and \[ \alpha_i \| (I-J_i^+ J_i) q_i \| < \delta_2 \]) DO

\[ q_{i+1} = q_i + \alpha_i (J_i^+ (\Delta x_i) - (I-J_i^+ J_i) q_i - \lambda_i q_i), \]
\[ \lambda_{i+1} = \gamma \frac{\| \Delta x_i \|}{\| J_i q_i \|}, \]
\[ \epsilon_{i+1} = \| (1-\alpha_i) \Delta x_i - \Delta x_{i+1} \|, \]
\[ \alpha_{i+1} = \frac{1}{2} (\alpha_i + \min( \epsilon_{\text{max}} / \epsilon_{i+1}, 1 )), \]

\[ i = i+1, \]

END UNTIL

This algorithm is implemented in the FORTRAN subroutine INVERS. A source listing is contained in Appendix IV.6.
2.7.3 Numerical Results

Some numerical results were obtained for several initial configurations $q_0$ and goal positions $\hat{x}$. A 2-D, four link manipulator with revolute joints and links of unit length was modelled.

a) Test 1

For the first test, the specifications were the following:

\[ q_0 = [90, 10, 10, 10]^T \text{ degrees} \]

\[ \hat{x} = [-1, -1]^T \text{ (these are the x and y translational components of the 2 degrees of freedom case)} \]

Maximum linearization error $\varepsilon_{\text{max}} = 0.2$

Deviation factor $\gamma = 0.5$

Position tolerance $\delta_1 = 0.01$

Optimization tolerance $\delta_2 = 0.01$

In this case, the algorithm converges to a solution in 25 iterations. Figure 2.4 shows the reduction of the position error versus the number of iterations. The convergence is initially
Figure 2.4  Position error versus iteration for test 1.
linear, becoming quadratic as the error is reduced.

Figure 2.5 shows the linearization error versus iteration number. Note that the linearization error remains below the specified maximum of $e_{\text{max}} = 0.2$.

Figure 2.6 shows the evolution of the iteration factor $\alpha$ and the deviation weight $\lambda$ as the solution converges. The iteration factor $\alpha$ is initially small to give a linear steepest descent convergence, but increases toward 1 to give a more rapid quadratic convergence near the goal position. The deviation weight $\lambda$ is initially large to allow the gripper trajectory to deviate from the minimum norm straight line. The deviation weight $\lambda$ is reduced to zero as the gripper nears its goal position.

Figure 2.7 shows the relative magnitudes of the components of the search vector. The minimum norm component $J^T \Delta x$ and the deviation component $\lambda q$ contribute most to the search vector, with relatively little contribution from the homogeneous component, $(I-J^T J) q$.

Figure 2.8 shows the iterated trajectory.

b) **Test 2**

Same as Test 1, with $\gamma = 0$.

With the deviation factor $\gamma$ set to zero, the solution iterates along the minimum norm path without deviation. The gripper moves in roughly a straight line from its initial position to the goal position, as shown in Figure 2.9 a). The path is not
Figure 2.5  Linearization error $\epsilon_i$ versus iteration $i$ for test 1.
Figure 2.6: Iteration factor $\alpha_i$ and deviation weight $\lambda_i$ versus iteration $i$ for test 1.
Figure 2.7  Components of the search vector versus iteration for test 1.
Figure 2.8  Plot of iterated solution trajectory for test 1.
Figure 2.9 Trajectories for different deviation factors $\gamma$:

a) $\gamma = 0.0$  
b) $\gamma = 0.6$  
c) $\gamma = 1.0$.
exactly straight due to the linearity error.

c) **Test 3**

Same as test 1, with $\gamma = 1.0$

In this case, the deviation factor is made as large as possible while still ensuring convergence. Note that the gripper path is now quite curved, and that the joints follow a more optimal trajectory (Figure 2.9 c)).

d) **Test 4**

Same as test 1, with $\epsilon_{\text{max}} = 0.5$. The resulting trajectory is shown in Figure 2.10 a).

e) **Test 5**

Same as test 1, with $\epsilon_{\text{max}} = 0.1$. The resulting trajectory is shown in Figure 2.10 c).

Figure 2.10 shows the reduction in the iteration step size as the maximum linearization error is reduced. A smaller linearization error results in a greater number of smaller, more accurate iteration steps being taken. The maximum linearization
Figure 2.10  Trajectories for different trajectory linearization errors $\epsilon_{\text{max}}$:  
\[ a) \quad \epsilon_{\text{max}} = 0.5 \quad b) \quad \epsilon_{\text{max}} = 0.2 \quad c) \quad \epsilon_{\text{max}} = 0.1 \]
error should be chosen to optimize the trade off between the number of iterations and the accuracy of the linearization.

f) Test 6

Same as Test 1, with 3 degrees of freedom,

\[ \Delta x = [ -1, -1, 90 ] \]

In this case, the final link orientation is 90 degrees from horizontal. This solution was found after 22 iterations. The iterated trajectory is shown in Figure 2.11.

g) Test 7

Same as test 1, with a different starting configuration,

\[ q_0 = [ 10, 10, 10, 10 ]^T \text{ degrees} \]

In this case, a different locally optimal inverse solution was found after 27 iterations, as shown in Figure 2.12.

h) Test 8

Same as test 1, with a different starting configuration and
Figure 2.11  Plot of iterated solution trajectory for test 6.
Figure 2.12 Plot of iterated solution trajectory for test 7.
goal position,

\[ q_0 = [0, 0, 180, 180]^T \text{ degrees} \]

\[ \hat{x} = [2, 0]^T \quad (2 \text{ translational degrees of freedom}) \]

This example illustrates the behavior of the algorithm when the initial gripper position is at or near the goal position, but the joint configuration is sub-optimal. The initial configuration has the manipulator links folded upon themselves, resulting in large joint angles. The robot joint angles are iteratively modified using the homogeneous part of the inverse solution to reduce the norm of the joint angles without changing the gripper position. Figure 2.13 shows the position error as a function of the number of iterations. Figure 2.14 shows that the homogeneous component, \((I - J^+J)q\), is the major contributor to the search vector. The homogeneous component is reduced as the joint configuration is optimized. The resulting trajectory is shown in Figure 2.15.
Figure 2.13  Position error $\Delta x_1$ versus iteration $i$ for test 8.
Figure 2.14 Components of the search vector versus iteration for test 8.
Figure 2.15 Plot of iterated solution trajectory for test 8.
3. **ANALYSIS OF INTERFERENCE**

3.1 **A Mathematical Definition of Interference Avoidance**

A robot workstation is composed of a number of physical objects which move along various trajectories through space as a function of time. Interference avoidance requires that no objects collide or intersect at any time. The problem can be stated mathematically as follows.

Let us represent \( n \) physical objects by sets of points \( \Omega_i \), \( \Omega_i \subset \mathbb{R}^n \), which are functions of time,

\[
\Omega_i(t), \; i=1,...,n, \; t \in [t_0, t_f]. \tag{3.1}
\]

The condition for interference avoidance is

\[
\Omega_i(t) \cap \Omega_j(t) = \emptyset, \; i \neq j, \; i=1,...,n, \; j=1,...,n, \; t \in [t_0, t_f]. \tag{3.2}
\]

Define the volume swept out by an object \( \Omega_i \) as it moves along a trajectory from time \( t_1 \) to \( t_2 \) as

\[
\Omega_i(t_1, t_2). \tag{3.3}
\]

If we discretize time as \( t_k = k\Delta t \) and replace \( t_k \) by \( k \), we get a discrete condition for interference avoidance,
\[ \Omega_i(k,k+1) \cap \Omega_j(k,k+1) = \emptyset, \]

\[ i \neq j, \quad i=1,\ldots,n, \quad j=1,\ldots,n, \quad k=k_0,\ldots,k_{l-1}. \]

3.2 Approximation of the Swept Volume by Convex Hulls

In the following analysis, we will impose the restriction that all obstacles in the workstation are fixed, and we will assume that the manipulator cannot interfere with itself. The method to be described can be easily extended to the more general case.

Collisions between the moving manipulator and stationary obstacles are avoided if the volume of space swept out by the manipulator does not intersect any of the obstacles. In general, the manipulator traces a highly nonlinear path in joint space, and the resulting swept volume is very complex. The path can be approximated by short, linear joint displacements connecting discrete joint positions. For closely spaced discrete positions, acceptable results can be obtained by testing each discrete position for interference. As the spacing between the positions increases, however, the union of the static manipulator volumes is no longer a good approximation of the swept volume. In particular, gaps may appear in the approximated swept volume. The swept volume can be represented more accurately by approximating the volume swept by a link moving between adjacent positions by the convex hull containing the two positions. This is a valid
assumption if the trajectory between two positions is approximately linear. This requirement is satisfied for any reasonable discrete trajectory approximation.

Interference detection is then performed by testing for intersection between the set of connecting convex hulls and the stationary obstacles. The problem is thus reduced to a set of stationary interference problems.

The discrete approximation of the swept volume is illustrated in Figure 3.1 for two adjacent discrete positions \( k, k+1 \).

### 3.3 Calculation of the Distance Between Stationary Objects

Algorithms for collision avoidance require some quantitative determination of interference and/or clearance between objects. Simple collision detection requires only a true or false result, while collision avoidance requires magnitude and direction information.

Gilbert and Johnson [1985] present a general mathematical treatment of distance functions using set theory, but do not elaborate on algorithms for calculating the actual quantities. We propose an alternative and functionally equivalent distance function based on intuitively clear geometrical considerations. The method presented here is an expansion of work described in [Buchal et al, 1986].

Define a plane

\[
\Pi(n,p) = \{ r : (r \cdot n) - p = 0 , \quad r \in \mathbb{R}^m \},
\]  

(3.5)
Figure 3.1  A discrete approximation of the swept volume.
where \( n \) is the unit plane normal, and \( \| p \| \) is the distance of the plane from the origin. For two dimensional space, \( m=2 \), and for three dimensional space, \( m=3 \). Let \( K_i \subseteq \mathbb{R}^m, i=1,\ldots,n \) be convex sets representing objects. For any \( n \in \mathbb{R}^m, \| n \| = 1 \), there exists a unique \( p_i \in \mathbb{R} \) such that the plane \( \Pi_i(n,p_i) \) is tangent to \( K_i \) and \( n \) points out of \( K_i \) at the point of tangency. There also exists a unique \( p_j \in \mathbb{R} \) such that the plane \( \Pi_j(-n,p_j) \) is tangent to \( K_j \) and \(-n\) points out of \( K_j \) at the point of tangency.

The distance between the two planes thus specified is

\[
d(n) = (r_j - r_i) \cdot n, \quad r_i \in \Pi_i, \quad r_j \in \Pi_j.
\]  

(3.6)

The minimum distance \( d^* \) between the two objects is found by maximizing \( d(n) \),

\[
d^* = \max \left\{ d(n) : n \in \mathbb{R}^m, \| n \| = 1 \right\}.
\]  

(3.7)

Let \( n^* \) be a unit vector for which

\[
d^* = d(n).
\]  

(3.8)

Note that a positive distance results if the objects are disjoint, and a negative distance results if the objects intersect.

Physically, the distance is the magnitude of the smallest pure translation which will bring the two objects into contact. We can define a distance vector \( d^* = d^* n^* \), which is the shortest translation vector to move \( K_i \) into contact with \( K_j \). The above
definitions are clarified in Figure 3.2.

Maximizing $d(n)$ is a standard problem of optimizing a function. The distance $d^* = d(n^*)$ is a local minimum or maximum if

$$\left. \frac{\partial d(n)}{\partial n} \right|_{n=n^*} = 0. \quad (3.9)$$

3.4 The Distance between Convex Polyhedra

The problem is considerably simplified if we consider only convex polyhedral objects. Convex polyhedra can be specified concisely and unambiguously by simple boundary representations, and the minimization problem can be discretized and simplified. Let us represent objects $K_i$, $K_j$, by their respective sets of vertices, $V_i$, $V_j$.

3.4.1 Calculating the Separating Distance for an Arbitrary Normal

Assume we are given a normal $n$ and a corresponding tangent point $r_i \in K_i$. The separating distance along $n$ is then

$$d(n) = \min \left( (r - r_i) \cdot n : r \in K_j \right). \quad (3.10)$$

In general, $d(n)$ can be found numerically using gradient methods, since only one local minimum exists for two convex objects.
Figure 3.2 Distance between convex hulls.
If $K_j$ is a convex polyhedron, then every tangent plane contains at least one vertex of the polyhedron. We can now restrict $r$ to the set of vertices, and

$$d(n) = \min \left\{ (r - r_i) \cdot n : r \in V_j \right\}.$$  \hspace{1cm} (3.11)

The minimization can be accomplished by a steepest descent graph search. The vertex of $K_j$ which minimizes $d(n)$ is $r_e$.

3.4.2 **Finding the Optimal Face Plane**

For convex polyhedra, the problem can be simplified if we consider only the tangent planes which contain the faces of the polyhedra $K_i$, $K_j$. If we define the set $N$ of all face normals of $K_i$, and if we specify that $n \in N$, then the solution which maximizes $d(n)$, $n \in N$ is

$$d^*_f = \max \left\{ (r_j - r_i) \cdot n : r_i \in \Pi_i, r_j \in \Pi_j, n \in N, \|n\| = 1 \right\}.$$  \hspace{1cm} (3.12)

The optimal face plane of $K_i$ has the surface normal $n^*_f \in N$, and $d^*_f \leq d^*$. Combining equations (3.11) and (3.12), we get

$$d^*_f = \max \left\{ \min \left\{ (r - r_i) \cdot n : r \in V_j \right\} : r_i \in \Pi_i, n \in N, \|n\| = 1 \right\},$$  \hspace{1cm} (3.13)
or, equivalently,

\[ d^*_f = \max \left\{ d(n) : n \in N, \|n\| = 1 \right\}. \] (3.14)

If the polyhedra \( K_i \) and \( K_j \) are interchanged, then another value of \( d^*_f \) can be calculated. Because of the asymmetry of the definitions, the interchanged case must be flagged to allow the correct order to be restored later. The optimal face for the pair of polyhedra belongs to the polyhedron which gives the largest \( d^*_f \). The closest vertex of the opposing polyhedron, \( r_e \), is found as a result of the face evaluations. Once the optimal face is found, the separating vector \( d^* \) can be easily determined.

If the two polyhedra are disjoint, then the distance function \( d(n) \) has only one local maximum, and the optimal face plane can be found by performing a steepest descent graph search of the faces of each polyhedron. If the two polyhedra intersect, then the distance function may have several local maxima, and an exhaustive search is necessary. Figure 3.3 shows two intersecting polygons with more than one local maximum.

3.4.3 Calculation of the Separating Distance for Disjoint Polyhedra

For convex polyhedra, we know that the separating distance \( d(n) \) increases monotonically along the gradient, with a single maximum. Polyhedra can be represented by a graph of vertices, connected by edges. An alternative representation is a graph of
Figure 3.3 Two intersecting polygons with multiple local maxima.
faces, connected by edges. We can use a steepest descent graph search to locate the maximum.

3.4.3.1 **Steepest Descent Graph Search**

Define a graph as a network of connected nodes, which can be vertices, faces or other elements. Let $S$ be the set of all nodes $s$, and let $S_i$ be the set of nodes connected to node $s_i$. Let $\phi_i$ be some function to be minimized, evaluated for node $s_i$.

Open a starting node $s_i$ and evaluate $\phi_i$.

UNTIL there are no adjacent nodes or $\phi_i$ is minimized DO

Evaluate $\phi_j$ for all adjacent nodes $s_j \in S_i$, saving the node $s_k : \phi_k = \min \{ \phi_j \}$.

Remove the nodes $s_j$ from the graph.

IF $\phi_k < \phi_i$ THEN

$\phi_i = \phi_k$

$s_i = s_k$

ELSE

$\phi_i$ is a minimum

ENDIF

END UNTIL
This algorithm returns the optimal node, and its evaluation.

3.4.3.2 Calculating the Optimal Tangent Plane and the Separating Distance

Let the optimal face of polyhedron $K_i$ be $f$ and the closest vertex of polyhedron $K_j$ be $r_c$. We can show that the optimal tangent plane $\Pi_i(n,p_i)$ contains at least one vertex of $f$, and $\Pi_j(-n,p_j)$ contains $r_c$.

Once the optimal face and closest vertex of a pair of polyhedra are known, the optimal tangent plane $\Pi$ and the minimum distance vector $d^*$ can be easily found. The points, edges or faces of tangency are also found. Based on geometrical considerations, it is clear that there are a finite number of possible cases:

a) $d^*$ is normal to $f$. The tangent plane contains face $f$ (Figure 3.4.a).

b) $d^*$ is the common normal of an edge of $f$ and an edge containing $r_c$. The tangent plane contains an edge of face $f$ (Figure 3.4.b).

c) $d^*$ is normal to an edge of $f$ and contains $r_c$. The tangent plane contains an edge of face $f$ (Figure 3.4.c).
Figure 3.4  Possible cases for the distance vector $d^*$. 

57
d) \( d^* \) is normal to an edge containing \( r_c \) and contains a vertex of face \( f \). The tangent plane contains one vertex of face \( f \) (Figure 3.4.d).

e) \( d^* \) connects a vertex of \( f \) to \( r_c \). The tangent plane contains one vertex of face \( f \) (Figure 3.4.e).

If the projection of \( r_c \) onto the plane of \( f \) lies within the boundaries of \( f \), then \( n^* \) is the face normal and the projection vector is \( d^* \). Otherwise, the shortest connecting vector \( v_c \) must be found between each edge of \( f \) and each edge containing \( r_c \). If the set of shortest connecting vectors is \( V_c \), then the distance vector is

\[
    d^*: \|d^*\| = \min \left\{ \|v_c\| : v_c \in V_c \right\}, \quad d^* \in V_c
\]  

(3.15)

and the distance is

\[
    d^* = \|d^*\|. \quad (3.16)
\]

The set of tangent points \( p_i \) of object \( K_i \) is \( P_i \) and the set of tangent points \( p_j \) of object \( K_j \) is \( P_j \). For convex polyhedra, the sets \( P \) may contain a single vertex, a line segment or a planar polygonal region. If we translate \( K_i \) by \( d^* \), a point, edge or surface of \( K_i \) will contact \( K_j \) without penetrating. The set of contact points on \( K_i \) will be called penetration points, \( p_{pen} \). These penetration points satisfy
Projection of the Closest Vertex onto the Optimal Face Plane

The projection of a point $r_c$ onto a plane is the intersection of the plane and the line which is normal to the plane and contains $r_c$. The equation of the plane is

$$ (r \cdot n) - p = 0 $$

(3.18)

and the equation of the projection vector is

$$ r = r_c + \mu n. $$

(3.19)

We can find $\mu$ from (3.18), (3.19),

$$ \mu = \frac{p - r_c \cdot n}{n \cdot n} $$

(3.20)

Knowing that $n \cdot n = 1$, we can substitute (3.20) into (3.19) to get the projected point $r_{proj}$,

$$ r_{proj} = r_c + (p - r_c \cdot n) n. $$

(3.21)

If the projected point $r_{proj}$ lies within the boundaries of the face $f$, then any other plane $\pi$ containing $r_{proj}$ and not containing $f$ intersects the polygonal boundary of $f$ at exactly two points, $p_1$, $p_2$. The two points define the endpoints of a line segment which contains $r_{proj}$. This geometric test (Figure 3.5)
Figure 3.5  Geometric test for inclusion of a point in a convex polygonal region.
can be used to determine if $\mathbf{r}_{\text{proj}}$ is on the face $f$. The points $\mathbf{r}_{\text{proj}}$ and $\mathbf{r}_c$ are points of tangency.

Edge to Edge Calculation

If the projection of $\mathbf{r}_c$ does not lie within the boundaries of $f$, then the shortest connecting vectors for each pair of edges must be found.

The shortest vector connecting two lines in space is their common normal. Given two edges defined by endpoints $s^1$, $t^1$, let

$$\mathbf{u}_1 = t^1 - s^1$$

(3.22)

and

$$\mathbf{u}_2 = t^2 - s^2.$$  

(3.23)

We know that for some $\mu_1, \mu_2$,

$$s^1 + \mu_1 \mathbf{u}_1 = s^2 + \mu_2 \mathbf{u}_2 + \mathbf{a},$$

(3.24)

where $\mathbf{a}$ is parallel to the common normal. From standard vector geometry, we have

$$\mathbf{a} = \frac{(s^1 - s^2) \cdot (\mathbf{u}_1 \times \mathbf{u}_2)}{\| \mathbf{u}_1 \times \mathbf{u}_2 \|^2} (\mathbf{u}_1 \times \mathbf{u}_2)$$

(3.25)

A singularity occurs if the two lines are parallel. In that case, $\mathbf{u}_1 \times \mathbf{u}_2 = 0$, and another method must be used.

We can rearrange equation (3.24),

$$\mu_1 \mathbf{u}_1 - \mu_2 \mathbf{u}_2 = (s_2 - s_1) + \mathbf{a}.$$ 

(3.26)
We can now easily solve for $\mu_1$ and $\mu_2$. There are four possible cases:

1. IF $(0 \leq \mu_1 \leq 1)$ and $(0 \leq \mu_2 \leq 1)$ THEN

   The common normal intersects both edges, and is the shortest connecting vector (Figure 3.6.a).

2. ELSE

   The shortest connecting vector STARTS at an endpoint of edge 1 and is normal to edge 2 (Figure 3.6.b).

   Project each endpoint of edge 1 onto the line of
Figure 3.6 The shortest vector connecting two edges.
edge 2. If the projected point lies between the endpoints of edge 2, it represents a connecting vector.

3. ELSE

The shortest connecting vector ENDS at an endpoint of edge 2 and is normal to edge 1 (Figure 3.6.c).

Do the same as above, with the edges reversed.

4. ELSE

The shortest connecting vector joins an endpoint of edge 1 to an endpoint of edge 2 (Figure 3.6.d).

Connect all pairs of endpoints and select the shortest resulting vector.

Projection of a Point onto a Line
Suppose we have a line segment with endpoints $s, t$, and a point $r$ not on the line. Let $r'$ be the projection of $r$ onto the line. Let

$$ u = t - s, \quad (3.27) $$

$$ v = r - s, \quad (3.28) $$
and

\[ r' = s + \mu \cdot u \]  \hspace{1cm} (3.29)

We can show that

\[ \mu = \frac{u \cdot v}{u \cdot u} \]  \hspace{1cm} (3.30)

If \( 0 \leq \mu \leq 1 \), then \( r' \) lies on the line segment.

The algorithms and techniques described above for disjoint polyhedra have been successfully implemented for 3-D interference detection [Buchal et al, 1986]. A PUMA 560 robot was modelled kinematically, and the links were approximated by convex polyhedra. Continuous path tool trajectories were specified for welding of a variety of workpieces, which were modelled geometrically as clusters of polyhedra. The robot and workstation geometry is shown in Appendix III. Separating distances between the links of the robot and the obstacles in the workstation were calculated and used to determine interference. In this implementation, discrete positions were used for interference calculations rather than swept volumes.

3.4.3.3 Distance Calculations for Disjoint Convex Polygons in 2-D

Calculation of the separating distance for two convex polygons in two dimensions is simpler than the case for polyhedra.
although the procedure is the same. Define the two polygonal objects as $K_i, K_j \in \mathbb{R}^2$. We can summarize the approach as follows:

1. Using an ordered search, find the optimal edge line of each polygon which maximizes the distance between the polygons. Choose the best of the two optimal edges as the optimal edge $e$ for the pair, and save the closest vertex $r_c$ of the opposing polygon.

2. If the projection of $r_c$ lies on edge $e$, the projection is the distance $d^*$. Otherwise, the two edges of $K_j$ containing $r_c$ must be tested against the optimal edge $e$ of $K_i$.

The distance vector is one of the following:

a) The normal vector connecting two parallel edges.

b) The projection of the closest vertex onto the optimal edge.

c) A vector connecting the closest vertex to an endpoint of the optimal edge.

3.4.4 Calculation of Penetration for Intersecting Polyhedra

The steepest descent graph search used for disjoint polyhedra
no longer applies if two polyhedra intersect, because multiple maxima may exist. In this case, an exhaustive search of all tangent plane normals must be performed.

3.4.4.1 Calculation of the Penetration Vector

Define the penetration vector $d_{pen}$ as

$$d_{pen} = 0, \quad d^* > 0,$$

(3.31)

$$d_{pen} = d^*, \quad d^* < 0.$$  

(3.32)

If two polyhedra intersect, then the distance function $d(n)$ may have multiple local maxima. The penetration vector $d_{pen}$ can be found by first using an exhaustive search to find the optimal face plane which maximizes $d(n)$. Note that for $d < 0$, this corresponds to minimizing $|d(n)|$. Once the optimal face is known, $d_{pen}$ can be found by evaluating the limited set of possible alternatives, as before.

The case can be simplified in 2-D. We can show that

$$d_{pen} = \mu \ n, \quad n \in N,$$

(3.33)

where $\mu$ is some scalar, and $N$ is the set of edge normals. In other words, the penetration vector is always parallel to an edge normal. Thus, once the optimal edge is found, $d_{pen}$ is known.

This assertion can be proven geometrically as follows.
Suppose we have found the optimal edge $f$ and the closest vertex $r_c$ for two intersecting polygons, $K_i$ and $K_j$. The tangent line $\Pi_i(n^*)$ contains edge $f$, the tangent line $\Pi_j(-n^*)$ contains $r_c$ and the penetration vector is $d_{pen}$. We can see by simple trigonometry that for any other pair of tangent lines $\Pi_i(n), \Pi_j(-n)$, $\|d\| > \|d_{pen}\|$. Figure 3.7 illustrates the above argument.

The following algorithm is used to determine if two polygons intersect, and to calculate the penetration vector. The search terminates immediately if the two polygons are determined to be disjoint. Otherwise, the separating distance $d(n)$ is calculated for all edge normals of both polygons. The maximum $d(n)$ is the penetration distance. Since $d(n)$ is negative for intersection, this corresponds to the minimum absolute value $|d(n)|$.

Let $N_1, N_2$ be the sets of edge normals $n$ for the two polygons.

\[ d_{pen} = 0. \]

FOR $i=1,2$

FOR all $n \in N_i$

calculate the separating distance $d(n)$.

IF $d(n) > 0$,

the polygons are disjoint and $d_{pen} = 0$.

ELSE IF $|d(n)| < \|d_{pen}\|$ THEN
Figure 3.7  Graphical proof that the penetration vector is normal to a face.
3.4.4.2 Calculation of the Net Translation of the $k^{th}$ Position to Reduce Penetration

Penetration can be calculated between the convex hulls containing sequential discrete link positions and fixed obstacles in the workstation. In order to reduce the penetration, we wish to translate the penetration point $p_{\text{pen}}$ on the convex hull in the direction of the penetration vector $d_{\text{pen}}$. The convex hull is determined by the contained link positions, so we need to transform the penetration vector into equivalent translations applied to the contained link positions.

A manipulator link at position $k$ is contained in two connecting convex hulls, the hull containing positions $k-1$, $k$ and the hull containing positions $k$, $k+1$. Also, each convex hull contains two adjacent positions. A penetration vector $d_{\text{pen}}$ and penetration point $p_{\text{pen}}$ can be found for each hull.

Let the convex set representing a link at position $k$ be $L_k$ and the convex hull containing $L_{k-1}$, $L_k$ be the convex set $H_k$. Let the vector $d_{i,j}$ be the translation vector applied to $L_i$ due to
penetration of hull $H_j$, and let $p_{i,j}$ be the corresponding translation point on the boundary of $L_i$. Thus, each position $L_k$ has two translation vectors, $d_{k,k}$ and $d_{k,k+1}$. The translation vectors are determined by the following:

If $p_{pen_k}$ lies on the boundary of $L_k$, then $d_{k,k} = d_{pen_k}$ and

$$p_{k,k} = p_{pen_k}.$$ (3.34)

If $p_{pen_k}$ lies on the boundary of $L_{k-1}$, then $d_{k-1,k} = d_{pen_k}$ and

$$p_{k-1,k} = p_{pen_k}.$$ (3.35)

If $p_{pen_k}$ lies on an edge of $H_k$ connecting $L_{k-1}$, $L_k$, then

$$d_{k-1,k} = (1-\mu) d_{pen_k}.$$ (3.36)

$$d_{k,k} = \mu d_{pen_k},$$ (3.37)

for some scalar $\mu \in [0,1]$. If a vertex $v_{k-1}$ of $L_{k-1}$ and a vertex $v_k$ of $L_k$ are adjacent vertices specifying the connecting edge of the hull $H_k$, then

$$p_{k-1,k} = v_{k-1},$$ (3.38)

$$p_{k,k} = v_{k,k},$$ (3.39)

and the edge of $H_k$ containing $v_{k-1}$, $v_k$ is defined by
\[ r = v_{k-1} + \mu (v_k - v_{k-1}), \quad 0 \leq \mu \leq 1 \] (3.38)

For the point \( p_{\text{pen}}_k \), we can find \( \mu \),

\[ \mu = \frac{p_{\text{pen}}_k - v_{k-1}}{v_k - v_{k-1}}. \] (3.39)

Note that two penetration vectors are applied to \( L_k \), \( d_{k,k} \) and \( d_{k,k+1} \), due to the two hulls \( H_k, H_{k+1} \) containing \( L_k \).

3.4.4.3. **Best-fit (Minimum Norm) Joint Space Displacement to Reduce a Set of Penetration Vectors**

Suppose several translation vectors act on a manipulator link \( L_k \) due to penetration of hulls \( H_k, H_{k+1} \) by one or more obstacles. The problem is now to find the joint displacement which will give the desired set of translations. In general, the problem is overconstrained, so a best-fit solution is sought.

Suppose we have a set of \( h \) translation vectors \( d_i \in \mathbb{R}^n \) and a corresponding set of translation points \( p_i \in \mathbb{R}^n, i = 1, \ldots, h \), as shown in Figure 3.8. The Jacobian matrices evaluated at the points \( p_i \) are \( J_i \in \mathbb{R}^{m \times n} \). If \( hm > n \), the problem is overconstrained. We can now define the problem as follows:
Figure 3.8  Translations of link positions due to penetration of the swept volume.
Define a combined translation vector

\[ \hat{d} = \begin{bmatrix} d_1^T, d_2^T, \ldots, d_h^T \end{bmatrix}^T, \quad \hat{d} \in \mathbb{R}^{hn}, \]  \hspace{1cm} (3.40)

and a corresponding combined Jacobian,

\[ \hat{J} = \begin{bmatrix} J_1, J_2, \ldots, J_h \end{bmatrix}, \quad \hat{J} \in \mathbb{R}^{hn}. \]  \hspace{1cm} (3.41)

We want to find an incremental joint change

\[ \hat{J} \delta q = \hat{d}^* \]  \hspace{1cm} (3.42)

such that \( \| \hat{d} - \hat{d}^* \| \) is minimized. The minimum norm solution which best fits the desired displacement is

\[ \delta q = \hat{J}^+ \hat{d} \]  \hspace{1cm} (3.43)

where \( \hat{J}^+ \) is the pseudoinverse of the combined Jacobian. Figure 3.9 shows a minimum norm joint change reducing two penetrations.

Note that summing the joint changes corresponding to each penetration vector,

\[ \delta q = \sum_{i=1}^{h} \hat{J}_i^+ d_i \]  \hspace{1cm} (3.44)

is not equivalent, and does not give a minimum norm solution.

Suppose the total set of \( h \) non-zero translation vectors are ordered such that
Figure 3.9 Minimum norm joint change reducing two penetrations.
\|d_1\| > \|d_2\| > \ldots > \|d_h\|. \quad (3.45)

We can choose a penetration fraction \( \sigma \in [0,1] \) to define a subset of \( g < h \) translation vectors such that

\[ \|d_1\| > \|d_2\| > \ldots > \|d_g\| > \sigma \|d_h\|. \quad (3.46) \]

If \( \sigma \) is small, then all of the translations will be applied with equal weight. The small translations will tend to constrain the problem more than necessary, and will result in a poor minimum norm solution for reducing the largest penetrations (Figure 3.10).

If \( \sigma \) is close to one, then only the largest penetration will be reduced. If the second largest penetration has a similar magnitude and a nearly opposite direction, then oscillation between the two translation directions can occur during successive iterations (Figure 3.11). This problem can be solved by reducing the iteration step size.

A large minimum norm solution may result from two small translations which are closely spaced and opposite in direction (Figure 3.12).
Figure 3.10 Minimum norm translation for two penetration fractions $\sigma$:  

- a) initial position
- b) $\sigma = 0.0$
- c) $\sigma = 1.0$
Figure 3.11  Oscillation between translation directions.
Figure 3.12 Large minimum norm joint change reducing two small penetrations.
4. TRAJECTORY OPTIMIZATION

4.1 Optimization by Minimizing a Functional

We wish to develop a method for generating a robot trajectory which satisfies collision and joint limit constraints while optimizing some specified global criteria. We can formulate a general functional of the trajectory which specifies the cost or performance as a function of the geometric trajectory as well as the velocity and the acceleration profiles:

$$\Phi(q(t)) = \int_{t_0}^{t_f} g(q(t), \dot{q}(t), \ddot{q}(t), t) \, dt \quad (4.1)$$

where

$$q(t), \ t \in [t_0, t_f], \quad (4.2)$$

is the robot joint trajectory.

Finding an optimal trajectory $q^*(t)$ which satisfies specified boundary conditions and which minimizes $\Phi$ is a non-linear two-point boundary value problem.

4.1.1 Local and Global Optima

An optimal trajectory $q^*(t)$ minimizes $\Phi$ if

$$\Phi(q^*) < \Phi(q^*+\delta q), \quad (4.3)$$
where \( \delta q \) is an arbitrary variation of the trajectory. There may be more than one optimal trajectory, corresponding to different local minima. Let us define the set \( A \) of all optimal trajectories as

\[
A = \{ q_1^*, q_2^*, \ldots, q_n^* \}.
\] (4.4)

We can now define the globally optimal trajectory \( q^*_g \) as

\[
\Phi(q^*_g) = \min \{ \Phi(q^*) : q^* \in A \}.
\] (4.5)

4.1.2 Optimality Conditions for the Continuous Case

The variation of \( \Phi \) can be expressed as

\[
\delta \Phi(q, \delta q, \delta t) = \int_{t_0}^{t_f} \left\{ \delta q^T \frac{\partial g(q, \dot{q}, \ddot{q}, t)}{\partial q} + \delta \dot{q}^T \frac{\partial g(q, \dot{q}, \ddot{q}, t)}{\partial \dot{q}} + \delta \ddot{q}^T \frac{\partial g(q, \dot{q}, \ddot{q}, t)}{\partial \ddot{q}} \right\} \, dt + \int_{t_f}^{t_f + \delta t} g(q, \dot{q}, \ddot{q}, t) \, dt \] (4.6)

To get this expression entirely in terms of \( \delta q \) and \( \delta t \), we can integrate by parts the terms containing \( \delta \dot{q} \) and \( \delta \ddot{q} \) as follows. Let

\[
u = \frac{\partial g}{\partial \dot{q}}
\] (4.7)

and

\[
dv = \delta \dot{q} \, dt.
\] (4.8)
Integrating by parts, we have

\[ \int_{t_0}^{t} dv^T u = v^T u - \int_{t_0}^{t} v^T du. \quad (4.9) \]

From (4.7) and (4.8) we can obtain

\[ v = \delta q \quad (4.10) \]

and

\[ du = \frac{d}{dt} \left( \frac{\partial g}{\partial q} \right) dt. \quad (4.11) \]

Combining the above equations, we get

\[ \int_{t_0}^{t} \delta q^T \frac{\partial g}{\partial q} dt - \delta q^T \frac{\partial g}{\partial q} \bigg|_{t_0}^{t} - \int_{t_0}^{t} \delta q^T \frac{d}{dt} \left( \frac{\partial g}{\partial q} \right) dt. \quad (4.12) \]

Integrate by parts twice to get

\[ \int_{t_0}^{t} \delta q^T \frac{\partial g}{\partial q} dt - \delta q^T \frac{\partial g}{\partial q} \bigg|_{t_0}^{t} - \delta q^T \frac{d}{dt} \left( \frac{\partial g}{\partial q} \right) \bigg|_{t_0}^{t} + \int_{t_0}^{t} \delta q^T \frac{d^2}{dt^2} \left( \frac{\partial g}{\partial q} \right) dt. \quad (4.13) \]

The second integral of (4.6) can be approximated by

\[ \int_{t^*_f}^{t_f} g(q, \dot{q}, \ddot{q}, t) dt \approx g(q, \dot{q}, \ddot{q}, t^*_f) \delta t_f. \quad (4.14) \]
Combining (4.6), (4.12), 4.13 and (4.14), we get

\[
\delta \Phi(q, \delta q) = \int_{t_0}^{t_f} \delta q^\top \left\{ \frac{\partial g}{\partial q} - \frac{d}{dt} \left[ \frac{\partial g}{\partial q} \right] + \frac{d^2}{dt^2} \left[ \frac{\partial g}{\partial q} \right] \right\} dt
\]

\[
+ \delta q^\top \frac{\partial g}{\partial q} \bigg|_{t_0}^{t_f} + \delta q^\top \frac{\partial g}{\partial q} \bigg|_{t_0}^{t_f} - \delta q^\top \frac{d}{dt} \left[ \frac{\partial g}{\partial q} \right] \bigg|_{t_0}^{t_f} + g(q, \dot{q}, q, t_f) \delta t_f.
\]

(4.15)

An optimal trajectory \( q^*(t) \) minimizes \( \Phi \), so the variation \( \delta \Phi(q^*, \delta q, \delta t_f) \) must be zero for any \( \delta q, \delta t_f \). The following condition must hold for \( g = g(q^*, \dot{q}^*, q^*, t) \):

\[
\frac{\partial g}{\partial q^*} - \frac{d}{dt} \left[ \frac{\partial g}{\partial q^*} \right] + \frac{d^2}{dt^2} \left[ \frac{\partial g}{\partial q^*} \right] = 0
\]

(4.16)

for all \( t \in [t_0, t_f] \).

This is the Euler-Lagrange equation. In addition, the terms outside the integral must be zero. These are the boundary conditions.

4.1.3 Optimality Conditions for the Discrete Case

In general, an analytical trajectory \( q^*(t) \) satisfying the Euler-Lagrange equation cannot be found directly, but if the problem is stated in discrete form, then numerical methods can be
applied. The Euler-Lagrange equation is a statement that the
gradient of the cost functional is zero at an optimum. Clearly,
an approximate solution can be found by iteratively improving the
trajectory until the gradient is reduced to a small value.

Let us divide time into equal intervals of length $r$, and let
$q_k$ be the discrete value of $q(t)$ at $t=kr$. We can now specify $q(t)$
as a sequence of discrete states,

$$q_k, \quad k = k_0, \ldots, k_f.$$  \hspace{1cm} (4.17)

Using the forward difference approximation for the first and
second derivative,

$$q_k = \frac{q_{k+1} - q_k}{r},$$ \hspace{1cm} (4.18)

$$\dot{q}_k = \frac{q_{k+1} - 2q_k + q_{k-1}}{r^2}$$ \hspace{1cm} (4.19)

for small $r$.

We can now specify a discrete cost functional as

$$\Phi(q, r) = \sum_{k=k_0}^{k_f} g(q_{k-1}, q_k, q_{k+1}, r, k) r,$$ \hspace{1cm} (4.20)

and we can write the variation as

$$\delta \Phi = \sum_{k=k_0}^{k_f} \left\{ \delta q_{k-1}^T \frac{\partial g}{\partial q} + \delta q_k^T \frac{\partial g}{\partial q} + \delta q_{k+1}^T \frac{\partial g}{\partial q} + \frac{\partial g}{\partial r} \right\} r + \sum_{k=k_0}^{k_f} g \ dr.$$ \hspace{1cm} (4.21)
If we replace $k$ by $m+1$, we can rewrite the first term of the summation,

\[
\sum_{k=k_0}^{k_f} \delta q_{k-1}^{T} \frac{\partial g}{\partial q_{k-1}} = \sum_{m=k_0+1}^{k_f+1} \delta q_{m}^{T} \frac{\partial g(q_{m}, q_{m+1}, q_{m+2}, m+1)}{\partial q_{m}} \quad (4.22)
\]

Now if we let $k=m$, and we let the summation go from $k=k_0$ to $k=k_f$, we get

\[
\sum_{k=k_0}^{k_f} \delta q_{k-1}^{T} \frac{\partial g}{\partial q_{k-1}} = \sum_{k=k_0}^{k_f} \delta q_{k}^{T} \frac{\partial g(q_{k}, q_{k+1}, q_{k+2}, k+1)}{\partial q_{k}}
\]

\[
+ \delta q_{k}^{T} \frac{\partial g(q_{k}, q_{k+1}, q_{k+2}, k+1)}{\partial q_{k}} \quad \left|_{k=k_{0}^{-1}}^{k=k_{f}^{+1}} \right. \quad (4.23)
\]

Similarly, the third term can be rewritten as

\[
\sum_{k=k_0}^{k_f} \delta q_{k+1}^{T} \frac{\partial g}{\partial q_{k+1}} = \sum_{k=k_0}^{k_f} \delta q_{k}^{T} \frac{\partial g(q_{k-2}, q_{k-1}, q_{k}, k-1)}{\partial q_{k}}
\]

\[
+ \delta q_{k}^{T} \frac{\partial g(q_{k-2}, q_{k-1}, q_{k}, k-1)}{\partial q_{k}} \quad \left|_{k=k_{0}^{-1}}^{k=k_{f}^{+1}} \right. \quad (4.24)
\]

Thus, the variation can be rewritten as
\[ \delta \Phi = \sum_{k=k_0}^{k_f} \{ \delta q_k^T \left[ \begin{array}{c} \frac{\partial g (q_k, q_{k+1}, q_{k+2}, k+1)}{\partial q_k} \\ \frac{\partial g (q_{k-1}, q_k, q_{k+1}, k)}{\partial q_k} \\ + \frac{\partial g (q_{k-2}, q_{k-1}, q_k, k-1)}{\partial q_k} \end{array} \right] \} r + \sum_{k=k_0}^{k_f} \left[ g + \frac{\partial g}{\partial r} \right] \, dr \]

\[ \begin{align*}
+ \delta q_k^T \frac{\partial g (q_k, q_{k+1}, q_{k+2}, k+1)}{\partial q_k} \\
+ \delta q_k^T \frac{\partial g (q_{k-2}, q_{k-1}, q_k, k-1)}{\partial q_k}
\end{align*} \]

\[4.25\]

4.1.4 \textbf{The Gradient of a Discrete Functional}

For the discrete problem, define a trajectory vector \( \dot{q} \),

\[ \dot{q} = [ q_{k_0-1}^T, q_{k_0}^T, q_{k_0+1}^T, \ldots, q_{k_f}^T, q_{k_f+1}^T, r ]^T. \quad (4.26) \]

and define a gradient \( \nabla \Phi \),

\[ \nabla \Phi = \left[ \begin{array}{cccccc}
\frac{\partial \Phi}{\partial q_{k_0-1}} & \frac{\partial \Phi}{\partial q_k} & \frac{\partial \Phi}{\partial q_{k+1}} & \ldots & \frac{\partial \Phi}{\partial q_{k_f}} & \frac{\partial \Phi}{\partial q_{k_f+1}} & \frac{\partial \Phi}{\partial r}
\end{array} \right]^T. \quad (4.27) \]
We can now write the variation as

\[ \delta \Phi (\tilde{q}, \delta \tilde{q}) = \delta \tilde{q}^T \nabla \Phi. \]  

(4.28)

The components of the gradient \( \nabla \Phi \) can be found from (4.25).

For \( k = k_0 - 1 \),

\[ \frac{\partial \Phi}{\partial q_k} = \tau \frac{\partial g (q_k, q_{k+1}, q_{k+2}, k+1, \tau)}{\partial q_k}. \]  

(4.29)

For \( k = k_0 \),

\[ \frac{\partial \Phi}{\partial q_k} = \tau \left\{ \frac{\partial g (q_k, q_{k+1}, q_{k+2}, k+1, \tau)}{\partial q_k} + \frac{\partial g (q_{k-1}, q_k, q_{k+1}, k, \tau)}{\partial q_k} \right\}. \]  

(4.30)

For \( k = k_0 + 1, k_0 + 2, \ldots, k_f - 1 \),

\[ \frac{\partial \Phi}{\partial q_k} = \tau \left\{ \frac{\partial g (q_k, q_{k+1}, q_{k+2}, k+1, \tau)}{\partial q_k} + \frac{\partial g (q_{k-1}, q_k, q_{k+1}, k, \tau)}{\partial q_k} \right\} \]  

\[ + \frac{\partial g (q_{k-2}, q_{k-1}, q_k, k-1, \tau)}{\partial q_k} \}. \]  

(4.31)

For \( k = k_f \),

\[ \frac{\partial \Phi}{\partial q_k} = \tau \left\{ \frac{\partial g (q_{k-1}, q_k, q_{k+1}, k, \tau)}{\partial q_k} + \frac{\partial g (q_{k-2}, q_{k-1}, q_k, k-1, \tau)}{\partial q_k} \right\}. \]  

(4.32)
For \( k = k + 1 \),

\[
\frac{\partial \Phi}{\partial q_k} = \tau \left\{ \frac{\partial g(q_{k-2}, q_{k-1}, q_k, k, \tau)}{\partial q_k} \right\}. \tag{4.33}
\]

Finally,

\[
\frac{\partial \Phi}{\partial \tau} = \sum_{k=k_0}^{k_f} \left[ g(q_{k-1}, q_k, q_{k+1}, k, \tau) + \frac{\partial g(q_{k-1}, q_k, q_{k+1}, k, \tau)}{\partial \tau} \right]. \tag{4.34}
\]

4.1.5 **A Modified Newton-Raphson Method for Numerical Minimization of a Functional**

Using the usual method of steepest descent, we can minimize the functional \( \Phi \) by iteratively improving the trajectory vector \( q \). A steepest descent iterative sequence has the form

\[
q_{i+1} = q_i - \alpha_i \nabla \Phi(q_i), \tag{4.35}
\]

where \( \alpha_i \) is some scalar factor.

Methods with an iterative form

\[
q_{i+1} = q_i - \alpha_i D \nabla \Phi(q_i) \tag{4.36}
\]

are known as quasi-Newton methods [Gottfried and Weisman, 1973, p.109].
Let

$$D_1 = H_1^{-1},$$  \hspace{1cm} (4.37)

where $H_1$ is the Hessian matrix evaluated at $q_1$. The Hessian matrix is the matrix of second partial derivatives of $\Phi$ evaluated at $q_1$,

$$H_1 = \begin{bmatrix}
\frac{\partial^2 \Phi}{\partial q_1^2} & \frac{\partial^2 \Phi}{\partial q_1 \partial q_2} & \ldots & \frac{\partial^2 \Phi}{\partial q_1 \partial q_n} \\
\frac{\partial^2 \Phi}{\partial q_2 \partial q_1} & \frac{\partial^2 \Phi}{\partial q_2^2} & \ldots & \frac{\partial^2 \Phi}{\partial q_2 \partial q_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial^2 \Phi}{\partial q_n \partial q_1} & \frac{\partial^2 \Phi}{\partial q_n \partial q_2} & \ldots & \frac{\partial^2 \Phi}{\partial q_n^2}
\end{bmatrix}.$$  \hspace{1cm} (4.38)

In general, the Hessian matrix is computationally expensive to calculate. However, if $\Phi$ is a quadratic of the form

$$\Phi = q^T X q + y^T q + z,$$  \hspace{1cm} (4.39)

then

$$\frac{\partial \Phi}{\partial q} = X q + y,$$  \hspace{1cm} (4.40)

and

$$\frac{\partial^2 \Phi}{\partial q^2} = X.$$  \hspace{1cm} (4.41)

The result is a modified Newton-Raphson method,

$$q_{i+1} = q_i - \alpha X_q^{-1} \nabla \Phi(q_i).$$  \hspace{1cm} (4.43)
If \( \alpha = 1 \), (4.43) reduces to the standard Newton-Raphson form.

### 4.1.6 Constraints as Penalty Functions

The problem discussed so far is unconstrained. In robot trajectory planning, certain constraints must be satisfied by the trajectory. A more general problem statement is:

Minimize \( \Phi(q(t)) = \int_{t_0}^{t_f} g(q(t), \dot{q}(t), \ddot{q}(t), t) \, dt \) (4.44)

subject to some generally nonlinear constraints

\[
f_i(q(t), \dot{q}(t), \ddot{q}(t), t) = 0, \quad i = 1, m, \tag{4.45}
\]

\[
f_i(q(t), \dot{q}(t), \ddot{q}(t), t) \geq 0, \quad i = m+1, n. \tag{4.46}
\]

One approach is to transform this general constrained problem into a simpler unconstrained problem.

Let us define a penalty functional \( \Phi_p \), which is a measure of violation of the constraints. Penalty functions which have a value of zero when constraints are satisfied and have a rapidly increasing positive value when the constraints are violated are known as exterior penalty functions.

This penalty functional can be added to the cost functional to create an augmented cost functional \( \Phi_a \). Minimization of the augmented functional implies minimization of the constraint
violations. By appropriate weighting of the penalty functions, the solution can be forced arbitrarily close to the constraint boundaries.

Let

\[ \Phi_p(q(t)) = \int_{t_0}^{t_f} g_p(q(t), \dot{q}(t), \ddot{q}(t), t) \, dt \] (4.47)

where \( g_p(q(t), \dot{q}(t), \ddot{q}(t), t) \) is a penalty function reflecting the violation of the constraints at time \( t \). Combining the cost functional \( \Phi \) (4.44) and the penalty functional \( \Phi_p \) (4.47), we get an augmented cost functional \( \Phi_a \),

\[ \Phi_a = \int_{t_0}^{t_f} \left\{ g(q(t), \dot{q}(t), \ddot{q}(t), t) + g_p(q(t), \dot{q}(t), \ddot{q}(t), t) \right\} \, dt. \] (4.48)

We can combine functions \( g \) and \( g_p \) to get an augmented cost function,

\[ g_a = g + g_p. \] (4.49)

Substituting (4.49) into (4.48), we get

\[ \Phi_a = \int_{t_0}^{t_f} g_a(q(t), \dot{q}(t), \ddot{q}(t), t) \, dt. \] (4.50)

The problem is now reduced to the unconstrained minimization of \( \Phi_a \).
If we solve the general problem with $t_f$ free ($r$ free in the discrete case) we can find the optimal dynamic trajectory as a function of time. If we fix $t_f$ (or fix $r$ in the discrete case) then we can find an optimal geometric trajectory which is independent of time.

If we consider the dynamics of the system, we can optimize criteria such as minimum time, minimum energy or minimum actuator work subject to dynamic and geometric constraints. The dynamic constraints include joint actuator force limits, joint velocity limits and interference avoidance when the obstacles are moving. Geometric constraints include joint displacement limits, and avoidance of interference with stationary obstacles.

From the dynamics equations of the manipulator, we can show that the joint actuator generalized force vector $f$ is

$$f(q,t) = M(q) \ddot{q} + \dot{q}^T C(q) \dot{q} + g(q).$$

(4.51)

where $M(q)$ is the inertia matrix, $C(q)$ is a coriolis and centipetal tensor and $g(q)$ is a vector of gravity terms. We can formulate a joint actuator force limit penalty function $g_f$ of the form

$$g_f = (f(q,t) - f_{lim})^T W (f(q,t) - f_{lim})$$

(4.52)

where $f_{lim}$ is the force limit vector and $W$ is a weighting matrix.
The unconstrained two-point boundary value problem can now be solved to get the optimal trajectory \( q(t) \). The optimal control which will produce the desired trajectory is given by (4.51).

The dynamic optimization has not yet been taken beyond the establishment of this mathematical framework. The calculation of the gradient requires the calculation of

\[
\frac{\partial f}{\partial q} - \frac{\partial M(q)}{\partial q} \dot{q} + \dot{q}^T \frac{\partial C(q)}{\partial q} \ddot{q} + \frac{\partial g(q)}{\partial q} \tag{4.53}
\]

which may be computationally expensive for realistic manipulators.

A simpler problem is geometric optimization without consideration of the dynamics. This corresponds to the case with final time \( t_f \) fixed. Dynamic constraints can be satisfied by time scaling the optimal geometric trajectory. The dynamic trajectory resulting from this two step procedure is not generally optimal. For example, the large accelerations resulting from a high rate of curvature of the geometric trajectory can be reduced by reducing the speed. True dynamic optimization requires reducing the curvature of the trajectory.

4.2 A Robot Trajectory Planning Problem

We can now develop a detailed description of the trajectory planning problem we wish to solve. We will consider point-to-point trajectories only, although the theory can be extended to continuous path problems as well. We will also restrict ourselves to finding a geometric trajectory.
Let us define a task trajectory \( \mathbf{x}(t) \) specifying the position and orientation of the gripper. We can characterize a point-to-point trajectory as follows:

The initial robot joint configuration, \( q(t_0) \), is specified.

The final gripper position and orientation \( x(t_f) \) is specified. Admissible joint configurations \( q(t_f) \) must satisfy the kinematic equation,

\[
x(t_f) = f(q(t_f)).
\] (4.54)

For redundant manipulators, there may be infinitely many solutions \( q(t_f) \).

Any trajectory satisfying the above boundary conditions is a solution. If a geometric trajectory is sought, then the variable \( t \) is simply a parameter varying between fixed limits \( t_0, t_f \).

We have defined an unconstrained two-point boundary value problem with the following conditions:

\begin{enumerate}
  \item a) \( q(t_0), t_0 \) are fixed.
  \item b) \( t_f \) is fixed.
  \item c) \( q(t_f) \) must satisfy \( x(t_f) = f(q(t_f)) \).
\end{enumerate}
From the kinematics, we can show that an admissible variation $\delta q(t_f)$ which does not change $x(t_f)$ must satisfy

$$\delta q(t_f) = G(q(t_f)) \delta z,$$  \hspace{1cm} (4.55)

where

$$G(q(t_f)) = (I - J^+ J),$$ \hspace{1cm} (4.56)

$J = J(q(t_f))$, the manipulator Jacobian,

and $\delta z$ is an arbitrary variation.

4.2.1 Geometric Constraints

We can identify the following geometric constraints which must be satisfied:

Joint limits- The range of joint motion is constrained by physical limits. We can specify the range of allowable joint limits as

$$q(\tau) \in [q_{\min}, q_{\max}], \quad \tau \in [\tau_0, \tau_f].$$ \hspace{1cm} (4.57)

Interference- No interference between objects may occur at any point along the trajectory. If we define the minimum distance between objects at time $t$ as $d^*(t)$, then we require that
4.2.2 Reduction of Penetration

As described in Chapter 3, we can define a penetration vector \( d_{\text{pen}} \) such that

\[
d_{\text{pen}} = d^*, \quad \text{for } d^* < 0, 
\]

\[
d_{\text{pen}} = 0, \quad \text{for } d^* > 0. 
\]

Thus, penetration occurs when the separating distance is negative.

Define a point \( p_{\text{pen}} \) as the point of maximum penetration on the robot link. We wish to move the robot such that \( p_{\text{pen}} \) is translated in the direction \( d_{\text{pen}} \). For a small penetration, we can specify an incremental joint change which will reduce the penetration as

\[
\delta q = J^+(q, p_{\text{pen}}) d_{\text{pen}}. 
\]

where \( J^+(q, p_{\text{pen}}) \) is the inverse Jacobian specified at the point \( p_{\text{pen}} \). We can rewrite this as

\[
\delta q = q_c - q 
\]

where \( q_c \) is a linear estimate of the nearest joint configuration for which the penetration is zero.
A more complete analysis of interference can be found in Chapter 3.

4.2.3 Optimization Criteria

We may wish to optimize the trajectory with respect to certain criteria, subject to the given constraints. An intuitively desirable goal is to find the "smoothest" trajectory which satisfies the constraints. This is based on an appreciation that large accelerations and correspondingly large actuator forces are associated with a "jagged" trajectory. We can achieve this result by minimizing the following cost functional:

$$
\Phi = \frac{1}{2} \int_{t_0}^{t_f} \ddot{q}(t)^T C \ddot{q}(t) \, dt \quad (4.63)
$$

where $C$ is a positive definite weighting matrix.

4.2.4 Proposed Quadratic Cost Functional

We propose to use a quadratic cost functional composed of terms representing the joint limit and interference penalty functions combined with a term representing an optimization criterion.

For the joint limit penalty function, we will use a simple quadratic of the form
\[ g_1 = \frac{1}{2} (q - q_{lim})^T A (q - q_{lim}) \]  \hspace{1cm} (4.64)

where \( A \) is a positive diagonal weighting matrix with

\[ a_{i,i} = \begin{cases} 
0, & q_{i,\text{min}} \leq q_i \leq q_{i,\text{max}} \\
\lambda_i, & \text{otherwise}, \lambda_i \text{ some scalar factor.}
\end{cases} \]  \hspace{1cm} (4.65)

If a joint limit is exceeded, then the active limit vector \( q_{lim} \) is defined as follows.

If \( q_i \leq q_{i,\text{min}} \) then \( q_i = q_{i,\text{min}} \).

or if \( q_i \geq q_{i,\text{max}} \) then \( q_i = q_{i,\text{max}} \).

The interference penalty function is represented by a sum of quadratics, representing the net penalty for penetrations between a number of object pairs,

\[ g_2 = \frac{1}{2} \sum_{i=1}^{m} (q_{c_i} - q_{c_i})^T B_i (q_{c_i} - q_{c_i}), \]  \hspace{1cm} (4.66)

\( m = \) number of object pairs.

The penalty functions (4.65), (4.66) can be combined with the optimization function (4.63) to get an augmented cost functional \( \Phi \).
\[
\Phi_e = \frac{1}{2} \int_{t_0}^{T_f} \left\{ (q(t) - q_{11m})^T A (q(t) - q_{11m}) \\
+ \sum_{i=1}^{m} (q(t) - q_c(t))^T B_i (q(t) - q_c(t)) + \ddot{q}(t)^T C \ddot{q}(t) \right\} \, dt .
\]

\[\text{(4.67)}\]

4.2.5 **The Discrete Form of the Problem**

The discrete boundary conditions of the problem are the following:

a) The initial configuration \( q_{0} \) is fixed.

b) The number of discrete steps is fixed, so \( k_0 \) and \( k_f \) are fixed.

c) The time increment \( \tau \) is fixed. This corresponds to the final time \( t_f \) fixed in the continuous case.

d) The final gripper position is fixed. The final joint configuration \( q_{k_f} \) must satisfy

\[
f(q_{k_f}) = x_{k_f} .
\]

\[\text{(4.68)}\]

e) The manipulator starts from rest and returns to rest.

Thus,
If we arbitrarily let $\tau = 1$, then the cost functional (4.67) can be expressed in discrete form as

$$
\Phi_a = \frac{1}{2} \sum_{k=k_0}^k \left\{ (q_k-q_{1m})^T A (q_k-q_{1m}) + \sum_{i=1}^m (q_{1i}-q_{c_i})^T B (q_{1i}-q_{c_i}) \\
+ (q_{k-1} - 2q_k + q_{k+1})^T C (q_{k-1} - 2q_k + q_{k+1}) \right\}.
$$

(4.71)

Henceforth, we will drop the subscript from $\Phi_a$, and $\Phi$ will be understood to represent the augmented cost functional.

### 4.2.6 The Discrete Gradient

Recall (4.28) that the variation of the cost functional can be expressed as

$$
\delta \Phi = \delta q^T \nabla \Phi,
$$

(4.72)

where $\delta q$ is the discrete trajectory vector, and $\nabla \Phi$ is the gradient of the discrete cost functional.
Let us define the following:

$$
\delta \Phi_k = \delta q_k^T \nabla \Phi_k ,
$$

(4.73)

where

$$
\nabla \Phi_k = \frac{\partial \Phi}{\partial q_k} .
$$

(4.74)

The discrete gradient can now be stated as

$$
\nabla \Phi_k = A(q_k - q_{lim}) + \sum_{i=1}^{m} B_i (q_{c} - q_k) + C(q_{k-2} - 4q_{k-1} + 6q_k - 4q_{k+1} + q_{k+2}),
$$

$$
k \in [k_0 + 1, k_f - 1].
$$

(4.75)

The functional is minimized when the variation vanishes,

$$
\delta \Phi = 0.
$$

(4.76)

This condition for optimality can be expressed equivalently as

$$
\delta q^T \nabla \Phi = 0.
$$

(4.77)

For this condition to hold for an arbitrary variation \( \delta \bar{q} \) about an optimal trajectory \( q^* \), the gradient \( \nabla \Phi \) must be zero. This is directly analogous to the minimization of a function of the form \( f(x) \). If desired, the discrete Euler-Lagrange equations and boundary conditions can be obtained by expanding this equation.
4.2.7 Iterative Reduction of the Cost Functional

We can now specify a modified Newton-Raphson updating equation,

\[ (q_k)_{i+1} = (q_k)_i - \alpha_i H^{-1} \nabla \Phi \quad k \in [k_0+1,k_f]. \quad (4.78) \]

The Hessian matrix \( H \) for the quadratic cost functional is

\[ H = A + \sum_{i=1}^{m} B + 6 C \quad \text{for } k \in [k_0+1,k_f-1], \quad (4.79) \]

\[ H = A + \sum_{i=1}^{m} B + 2 C \quad \text{for } k = k_f. \quad (4.80) \]

For \( k = k_0 \),

\[ \delta q_k = 0. \quad (4.81) \]

For \( k = k_0 + 1 \), \( q_{k-2} = q_{k-1} \) and

\[ \nabla \Phi = A(q_k - q_{lim}) + \sum_{i=1}^{m} B(q_i - q_{c,i}) + C(-3q_{k-1} + 6q_k - 4q_{k+1} + q_{k+2}). \quad (4.82) \]

For \( k \in [k_0+2,k_f-2] \),

\[ \nabla \Phi = A(q_k - q_{lim}) + \sum_{i=1}^{m} B(q_i - q_{c,i}) + C(q_{k-2} - 4q_{k-1} + 6q_k - 4q_{k+1} + q_{k+2}). \quad (4.83) \]
For $k - k_f - 1$, \( q_{k+2} = q_{k+1} \) and

\[
\nabla \Phi_k = A(q_k - q_{1_{im}}) + \sum_{i=1}^{m} B_i (q_{c_i} - q_{c_i}) + C(q_{k-2} - 4q_{k-1} + 6q_k - 3q_{k+1}).
\]

(4.84)

For $k = k_f$, $q_{k+2} = q_{k+1} = q_k$ and

\[
\nabla \Phi'_k = A(q_k - q_{1_{im}}) + \sum_{i=1}^{m} B_i (q_{c_i} - q_{c_i}) + C(q_{k-2} - 4q_{k-1} + 3q_k).
\]

(4.85)

For $k = k_f + 1$, $q_{k+2} = q_{k+1} = q_k = q_{k-1}$ and

\[
\nabla \Phi'_k = C(q_{k-2} - q_{k-1}).
\]

(4.86)

Since $q_{k+1} = q_k$, $\delta q_{k+1} = \delta q_k$ and

\[
\delta \Phi = \delta q^T \nabla \Phi' + \delta q^T \nabla \Phi'.
\]

(4.87)

Equivalently,

\[
\delta \Phi = \delta q^T \left[ \nabla \Phi'_k + \nabla \Phi'_{k+1} \right].
\]

(4.88)

If we redefine the gradient as

\[
\nabla \Phi_k = \nabla \Phi'_k + \nabla \Phi'_{k+1}
\]

(4.89)
then for $k=k_f$,

$$
\nabla \Phi_k = A(q_k - q_{\text{lim}}) + \sum_{i=1}^{m} B(q - q_i) + C(q_{k-2} - 3q_{k-1} + 2q_k).
$$

(4.90)

4.2.8 Boundary Conditions

In order to satisfy the boundary condition that the gripper position $x_{k_f}$ is fixed, $q_{k_f}$ must satisfy the forward kinematic equation,

$$
x_{k_f} = f(q_{k_f}).
$$

(4.91)

From the general inverse kinematics solution, we see that

$$
\delta x_{k_f} = J_{k_f} \delta q_{k_f} = 0
$$

(4.92)

for

$$
\delta q_{k_f} = (I - J_{k_f}^+ J_{k_f}) \delta z,
$$

(4.93)

where $J_{k_f}$ is the Jacobian evaluated at $q_{k_f}$,

$J_{k_f}^+$ is the pseudoinverse,

and $\delta z$ is an arbitrary variation.

The steepest descent variation $\delta z$ is in the direction of the gradient $\nabla \Phi_{k_f}$. The closest admissible variation $\delta q_{k_f}$ is the projection of $\delta z$ onto the null space of $J_{k_f}$.
\[ \delta q_k = \mu \sum_{i=0}^{f} G_k \nabla \phi_k, \]

for some scalar \( \mu \), where

\[ G_k = I - J_k^+ J_k. \]

We see that the boundary condition is satisfied by an iterative sequence of the form

\[ (q_k)_{i+1} = (q_k)_i - \alpha H^{-1} G_k \nabla \phi_k, \quad k=k_f. \]

4.2.9 Feasibility of Local Optima

Optimal trajectories may exist which locally minimize the cost functional, but which do not satisfy the constraints. In other words, some points along the trajectory remain well within the boundaries of the constraint penalty functions. We will call these non-feasible optimal trajectories. Since optimality does not guarantee feasibility, each optimal trajectory must be tested to determine if the constraints are satisfied.
5. NUMERICAL OPTIMIZATION

5.1 A Discrete Approximation of the Trajectory

In order to find an optimal trajectory numerically, the trajectory must be represented in discrete form. We will represent the trajectory by a sequence of discrete geometric joint states,

\[ q_k, k=1,n. \]  

(5.1)

5.1.1 Specification of a Starting Trajectory

Numerical optimization requires the specification of a starting trajectory. Any trajectory which satisfies the boundary conditions can be used, but rapid convergence to a geometrically feasible and possibly globally optimal trajectory requires a good choice for the starting trajectory. A good starting trajectory can be generated using interactive graphics, allowing the users intuition and judgement to be applied.

A gross trajectory can be defined by specifying a small number of intermediate positions through which the starting trajectory should pass. Several alternative trajectories can be specified in this way.

In the current 2-D implementation, a simple but somewhat limited scheme is used to generate a starting trajectory. First, the user interactively specifies the joint configuration of the
manipulator at its initial position. Next, the desired final gripper position is specified in task coordinates. If necessary, several intermediate gripper positions can be specified along the trajectory. These intermediate positions are ordered from the initial position to the goal position. The joint angles for each position are calculated by the iterative inverse kinematics method described in Chapter 2, using the joint configuration of the immediately preceding configuration as a starting point.

The main drawback of this method is that the user has no direct control over the joint configurations at each intermediate position. Using a predetermined algorithm for calculating the joint configurations can lead to unpredictable joint trajectories. A more successful approach may be to allow the user to specify intermediate gripper positions, and to interactively vary and select the corresponding joint configurations.

5.1.2 Generation of the Discrete Trajectory by Interpolation

The starting trajectory specification is very coarse. Before the numerical optimization can begin, the step size between adjacent positions must be reduced. This can be done by successively subdividing each interval and inserting interpolated positions until the step size is small and the linearization error is within a specified tolerance.

Clearly the number of discrete states is not fixed if this approach is used. The numerical method is not affected, however, because the number of states is arbitrary, and the cost functional
is reduced for a fixed number of states at each iteration.

5.2 **An Algorithm for the Iterative Reduction of the Cost Functional**

As shown previously, the cost functional can be reduced by iteratively updating the trajectory. The trajectory is modified by taking small steps in the direction of the modified Newton-Raphson search vector of the functional. The step size is chosen so that the linearization error remains within a specified tolerance.

An understanding of the algorithm used in the software implementation can be obtained from the following qualitative description.

UNTIL the number of iterations reaches a limit, or the penetration and joint limit constraints are satisfied,

FOR each position along the trajectory,

Calculate the convex hull enclosing the current and previous positions.

Calculate the penetrations of the hull and the obstacles.

IF penetrations are detected,
Calculate the best-fit joint change to reduce the penetrations.

END IF

Calculate the modified Newton-Raphson search vector of the cost functional.

IF the current position is the last position of the trajectory,

Multiply the search vector by the projection $(I-J^+J)$ to get a joint change which will not change the gripper position.

Multiply the result by the iteration factor to get the joint change.

Calculate the linearity error, and reduce the iteration factor if necessary.

Update the final joint configuration.

Calculate the resulting change in the gripper position due to linearity error. If necessary, calculate the minimum norm joint change which will remove the position error,
and modify the final configuration.

ELSE

Calculate the desired joint change and estimate the linearization error.

Reduce the iteration factor if necessary.

Update the manipulator configuration.

Calculate the linearization error in the discrete approximation of the trajectory, and save it.

ENDIF

END FOR

FOR every position along the trajectory,

Check the local linearization error.

Insert or delete intermediate positions if necessary.

END FOR
The details of the algorithm are well documented in the software source listing in Appendix IV.

5.2.1 Linearization Error

Our linear approximation states that

$$ \delta x = J(q) \delta q $$  \hspace{1cm} (5.2)

for sufficiently small $\delta q$. Given $q_k$, $\delta q_k$, and $x_k$, we can calculate a prediction $\hat{x}_{k+1}$ for $x_{k+1}$,

$$ \hat{x}_{k+1} = x_k + J(q_k) \delta q_k $$  \hspace{1cm} (5.3)

The exact value is

$$ x_{k+1} = f(q_k + \delta q) $$  \hspace{1cm} (5.4)

and the linearization error is

$$ \epsilon = \| x_{k+1} - \hat{x}_{k+1} \| $$  \hspace{1cm} (5.5)
5.2.2 Calculation of the Iteration Step Size

We can now determine the largest iteration step size which does not exceed some maximum linearization error $\varepsilon_{\text{max}}$ as follows:

1. Calculate the modified Newton-Raphson joint step $\Delta q_i$ from

$$
\Delta q_i = -\alpha_i H^{-1}_i \nabla \Phi_i ,
$$

(5.6)

where $\alpha_i \in [0,1]$.

2. For $\Delta q_i$, calculate $\hat{x}_{k+1}' , \hat{x}_{k+1}$.

3. Calculate the iteration linearization error $\varepsilon$.

4. If $\varepsilon > \varepsilon_{\text{max}}$, then scale the step size,

$$
q_{i+1} = q_i + \frac{\varepsilon_{\text{max}}}{\varepsilon} \Delta q_i .
$$

(5.7)

The maximum allowable iteration linearization error $\varepsilon_{\text{max}}$ can be selected by the user.
5.2.3 **Optimal Spacing of the Discrete Manipulator Configurations**

Exactly the same method can be used to determine the optimal spacing of the discrete manipulator configurations along the trajectory as follows.

1. Calculate the joint change $\Delta q_k$ between adjacent configurations,

$$\Delta q_k = q_{k+1} - q_k .$$  \hspace{1cm} (5.8)

2. For $\Delta q_k$, calculate $\dot{x}_{k+1}$, $\ddot{x}_{k+1}$.

3. Calculate the trajectory linearization error $\epsilon$.

4. If $\epsilon > \epsilon_{\text{max}}$, then subdivide the interval by inserting a new position,

$$q_{k+2} = q_{k+1}$$  \hspace{1cm} (5.9)

$$q_{k+1} = \frac{q_{k+2} + q_k}{2} .$$  \hspace{1cm} (5.10)

The maximum allowable trajectory linearization error can be selected by the user.
5.3 Satisfaction of Constraints

Since external penalty functions are used, the constraints will generally be violated, although the solution can be forced arbitrarily close to the constraint boundary. This problem can be surmounted by enlarging the constraints by some tolerance, and reducing the constraint violation to below that tolerance. This approach is summarized as follows:

Decrease the joint limits by an amount $\delta_{\text{joint}}$, and increase the size of the obstacles by an amount $\delta_{\text{pen}}$.

Iterate until the joint limits are exceeded by less than $\delta_{\text{joint}}$ and the maximum penetration is less than $\delta_{\text{pen}}$.

5.4 Finding a Globally Optimal Trajectory

In general, a large number of locally optimal trajectories exist. Some of these solutions may not be feasible because they violate the constraints, and a feasible solution may be difficult to find if one exists. If a number of feasible solutions exist, how can the best one be found? There is no general analytical method for finding a global optimum, but an acceptable solution can often be found using search techniques.

The most appropriate and effective method for finding optimal solutions with a minimum of searching is to utilize the judgement and intuition of a human operator to postulate several possible
starting trajectories. Each starting trajectory is then optimized and assessed. Depending on the goodness of the choice of the starting trajectories, and the severity of the constraints, one or more feasible final trajectories may be found. If no feasible trajectories are found, further starting trajectories can be tried until the user is convinced that no feasible trajectory exists. If several feasible trajectories are found, the one with the minimum cost is selected as the optimum.
6. **NUMERICAL RESULTS**

The trajectory optimization method has been implemented in a FORTRAN program called OPT, running on a VAX 11/750 host computer. The software is documented fully in Appendix IV. Numerical trajectory optimization for simple cases like the ones described in this chapter typically takes about 100 seconds of CPU time.

OPT models a simple planar manipulator in 2-D to test and evaluate the trajectory optimization method. The user can specify the workstation geometry, the starting trajectory, and the set of numerical parameters used in the optimization.

A. **Workstation Geometry.** The following parameters are specified in a data file which can be edited by the user:

a) the number of degrees of freedom and link geometry of the manipulator.

b) the number of task degrees of freedom.

c) the location and shape of convex polygonal obstacles.

d) specification of convex polygons representing the manipulator links.

e) specification of manipulator joint limits. This parameter is entered interactively when OPT is run.
B. **Starting trajectory.** The following specifications of the starting trajectory are generated interactively by the user, as described in Chapter 5.

a) initial manipulator joint configuration.

b) goal gripper position.

c) intermediate gripper positions.

C. **Numerical parameters.** The parameters used in the numerical optimization are stored in a file. The user can interactively change any parameter when OPT is run, and the new value will be stored. For simplicity, the weighting matrices are specified to be diagonal, with equal elements. Thus, the weighting matrices A, B, and C can be replaced by scalar weighting factors. The following parameters can be specified:

a) Joint limit penalty function weighting factor.
   
   This scalar factor is equivalent to the weighting matrix A in equation (4.75).

b) Interference penalty function weighting factor.
   
   This scalar factor is equivalent to the weighting
matrices $B$ in equation (4.75). The factor is the same for all obstacles.

c) Penetration fraction. All penetration vectors with a magnitude larger than this fraction of the largest penetration will be reduced by a best-fit procedure (see sect. 3.4.4.3).

d) Joint acceleration weighting factor. This scalar factor is equivalent to the weighting matrix $C$ in equation (4.75).

e) Maximum number of iterations. This is the maximum number of iterations which will be executed before the program terminates.

f) Iteration factor. This is the factor $\alpha$ in the iteration equation (4.78).

g) Maximum allowable iteration linearization error. This is an upper bound on the linearization error incurred by an iteration step, as described in Section 5.2.2.

h) Maximum allowable trajectory linearization error. This is an upper bound on the linearization error due to the step size between discrete trajectory positions, as described in Section 5.2.3.
1) Penetration and joint limit constraint tolerances. These tolerances are used to 'grow' the constraints, allowing satisfaction of the constraints as described in Section 5.3.

6.1 Test Cases

OPT was executed for a number of test cases to evaluate and demonstrate the optimization method. The following parameter settings were used except where noted otherwise:

Joint limit penalty function weighting factor = 100.
Interference penalty function weighting factor = 1000.
Penetration fraction = 0.5.
Joint acceleration weighting factor = 10.
Maximum number of iterations = 50.
Iteration factor $\alpha$ = 0.6.
Maximum allowable iteration linearization error = 0.1.
Maximum allowable trajectory linearization error = 0.05.
Penetration tolerance = 0.01.
Joint limit constraint tolerance = 5 degrees.
Maximum joint limits = 90 degrees for all joints.
Minimum joint limits = -90 degrees for all joints.
A Benchmark Case with Multiple Obstacles - Case 1

Figure 6.1 shows a set of obstacles, an initial manipulator configuration, a gripper goal position and two intermediate manipulator positions defining a starting trajectory. The obstacles are predefined in a data file. The manipulator initial configuration is input by the user as a set of joint angles. The goal position is specified in task coordinates. Intermediate positions are also specified in task coordinates. An iterative inverse kinematic solution (Chapter 2) is used to generate the corresponding joint angles for each position.

Additional positions are inserted into the trajectory by interpolation after each iteration until the linearization error is below some bound. Figure 6.2 shows the evolution of the trajectory after one, two and three iterations. Figure 6.3 shows the change in the number of discrete positions as a function of the number of iterations. Note that the number of positions increases rapidly for the first few iterations and then stabilizes, although further adjustment can occur.

The penetration and joint violation are reduced below the specified tolerance after 26 iterations for this case. Figure 6.4 shows the resulting trajectory.

The cost functional is reduced as shown in Figure 6.5. The anomaly in the cost after the seventh iteration is due to a large minimum norm joint change \( \delta q \) being specified to reduce small penetrations in a case such as shown in Figure 3.12. The norm of \( \delta q \) is assumed to be linearly related to the penetration magnitude, which is inaccurate in this case. Thus, calculation of the
Figure 6.1 Starting trajectory and obstacles for case 1.
Figure 6.2  Evolution of the discrete trajectory for the first few iterations for case 1: a) starting trajectory b) 1 iteration c) 2 iterations d) 3 iterations.
Figure 6.3  Plot of the number of discrete trajectory positions at each iteration for case 1.
Figure 6.4   The optimized trajectory for case 1.
Figure 6.5  Plot of the cost functional versus the number of iterations for case 1.
penetration magnitude based on $\delta q$ gives a large error. The anomaly is strictly local, and has no effect on later iterations.

The joint trajectories after 26 iterations are shown in Figure 6.6. Note that the joint limit constraints of ±90 degrees are not active at any point along the trajectory.

Figure 6.7 shows the reduction of the maximum penetration as a function of the number of iterations. The maximum penetration is reduced to zero after 26 iterations.

6.1.2 Varying the Joint Limits - Cases 2 and 3

Cases 2 and 3 are the same as case 1, with joint limits specified as ±75° and ±65°, respectively. Figures 6.8 and 6.9 show the resulting joint trajectories. With limits of ±75° the upper joint limit is active for joint 3. With limits of ±65° the upper limit is active for joint 3 and the lower limit is active for joint 1. The imposition of more severe joint constraints results in a larger cost for the resultant trajectory as shown in Figure 6.10. In addition, the constraints are not satisfied in the more constrained case even after 50 iterations, and in fact there may be no feasible solution.

6.1.3 Varying the Maximum Trajectory Linearization Error - Cases 4 and 5

Cases 4 and 5 are the same as case 1, with the maximum
Figure 6.6 Plot of the joint trajectories for case 1.
Figure 6.7  Plot of the maximum penetration versus the number of iterations for case 1.
Figure 6.8  Plot of the joint trajectories for case 2.
Figure 6.9 Plot of the joint trajectories for case 3.
Figure 6.10  Comparison of the cost functionals for case 1 and case 3.
trajectory linearization error $\epsilon_{\text{max}_{\text{traj}}}$ specified as 0.01 and 0.2, respectively. The resulting trajectories for $\epsilon_{\text{max}_{\text{traj}}} = 0.01, 0.1, 0.2$ are compared in Figure 6.11. Note that a smaller $\epsilon_{\text{max}_{\text{traj}}}$ results in a finer spacing of adjacent discrete joint positions. A trade-off must be established between the accuracy of the discretization and the number of discrete positions which must be processed. The cost functionals for the three cases are shown in Figure 6.12.

6.1.4 Varying the Maximum Iteration Linearization Error

Several different maximum allowable iteration linearization errors $\epsilon_{\text{max}_{\text{iter}}}$ were specified for the benchmark case, without significant effect on the results. The algorithm appears to be relatively insensitive to the choice of $\epsilon_{\text{max}_{\text{iter}}}$.

6.1.5 A Different Starting Trajectory - Case 6

Case 6 is the same as case 1, with a different starting trajectory as shown in Figure 6.13. The resulting trajectory is shown in Figure 6.14. This different starting trajectory leads to a different locally optimal trajectory than in case 1.
Figure 6.11 Optimized trajectories with different trajectory linearization errors:
a) $\epsilon = 0.01$  
b) $\epsilon = 0.1$  
c) $\epsilon = 0.2$.
Figure 6.12 Comparison of the cost functionals for cases 1, 5 and 6 for different maximum trajectory linearization errors.
Figure 6.13 Starting trajectory for case 6.
Figure 6.14  Optimized trajectory for case 6.
6.1.6 *A Different Benchmark Case with a Single Obstacle - Case 7*

The obstacles were redefined, and a starting trajectory shown in Figure 6.15 was specified for case 7. The joint limit weighting factor was set to 1000. The resulting trajectory after 50 iterations is shown in Figure 6.16. The cost functional is reduced as shown in Figure 6.17. Figure 6.18 shows the final joint trajectories. In this case, the joint limit of 90° is active for joint 3 and the limit of -90° is active for joint 1. The maximum penetration is reduced as shown in Figure 6.19 and the maximum joint limit violation is reduced as shown in Figure 6.20.

6.1.7 *Varying the Iteration Step Size - Cases 8 and 9*

Cases 8 and 9 are the same as case 7, with an iteration step sizes of $\alpha = 0.1$ and $\alpha = 1.0$. The resulting trajectories are compared in Figure 6.21. The costs for the three different step sizes are shown in Figure 6.22. The convergence is slow for a small step size, and is rapid but unstable for a large step size of $\alpha = 1.0$. A step size of $\alpha = 0.6$ gives a good compromise between rapid convergence and stability.

The joint violations are rapidly reduced for all three cases as shown in Figure 6.23. Figure 6.24 shows that the maximum penetration is also reduced for all cases, but with $\alpha = 0.1$ the reduction is slow due to the small step size.
Figure 6.15  Starting trajectory and obstacles for case 7.
Figure 6.16  Optimized trajectory for case 7.
Figure 6.17 Plot of the cost functional for case 7.
Figure 6.18  Plot of the joint trajectories for case 7.
Figure 6.19  Plot of the maximum penetration versus the number of iterations for case 7.
Figure 6.20  Plot of the maximum joint limit violation versus the number of iterations for case 7.
Figure 6.21 Optimized trajectories with different iteration step sizes:

a) $\alpha = 0.1$  
b) $\alpha = 0.6$  
c) $\alpha = 1.0$
Figure 6.22 Comparison of the cost functionals for cases 7, 8, and 9 for three different iteration step sizes.
Figure 6.23  Comparison of the maximum joint limit violation versus the number of iterations for three iteration step sizes.
Figure 6.24 Comparison of the maximum penetration versus the number of iterations for three iteration step sizes.
6.1.8 A Case with Multiple Links - Case 10

The test cases so far have modelled a single link of the robot as a rectangular area (equivalent to a solid in 3-D). OPT allows any number of moving polygonal areas to be specified. Figure 6.25 shows the results of running case 10, which is the same as case 7 with the last two links of the robot modelled by rectangular areas.
Figure 6.25  Optimized trajectory for case 10.
7. **FURTHER WORK AND POTENTIAL APPLICATIONS**

The mathematical techniques and algorithms proposed in this thesis provide a strong foundation for a range of useful applications. Some fruitful areas for further work can now be identified.

7.1 **Extension to 3-D**

The numerical optimization algorithm can be extended to handle realistic 3-D manipulators by straightforward extension. Calculation of separating distances for 3-D polyhedra has already been implemented [Buchal et al, 1986]. Further effort is required to develop and implement an efficient algorithm for calculating convex hulls in 3-D. The proposed penetration calculation algorithm for convex polyhedra must be implemented. More sophisticated data structures, such as developed in [Buchal et al, 1986] are required. Simulation and verification of trajectories in 3-D requires 3-D graphics capabilities. To be useful, the software implementation should incorporate or interface with CAD and workstation simulation software. No fundamental difficulties are foreseen in the development of these capabilities.

7.2 **Optimization of Continuous Path Problems**

The method demonstrated in this thesis solves a
point-to-point trajectory planning problem. Such trajectories are encountered in many applications such as assembly and part transfer. Many other applications, such as continuous arc welding, require that the tool position and orientation be specified continuously along the trajectory. This problem can be solved as an optimal control problem. Consider a problem of the general form

$$\begin{align*}
\min_{u} & \quad \Phi(u) = h(q(t), t) + \int_{t_0}^{t_f} g(q(t), u(t), t) \, dt \\
\text{subject to} & \quad q(t) = b(q(t), u(t), t),
\end{align*}$$

(7.1)

with the following state equation:

$$\dot{q}(t) = b(q(t), u(t), t).$$

(7.2)

The following special case can be solved numerically:

$$\Phi(u) = \int_{t_0}^{t_f} (q^T \Lambda q + \dot{q}^T \mathbf{B} \dot{q} + (q - q_*)^T \mathbf{C} (q - q_*)) \, dt$$

(7.3)

$$\dot{q}(t) = J^+(q) \dot{x}(t) + (I - J^+ J) u,$$

(7.4)

where the tool trajectory $\dot{x}(t)$, $t \in [t_0, t_f]$ is specified. The redundancy of the system is represented by the control $u$, which is
7.3 Optimization of System Dynamics

It is possible to formulate and solve a general problem to find an optimal trajectory as an explicit function of time such that both the geometric and dynamic constraints are satisfied. For example, the minimum-time trajectory can be found which satisfies joint limit and interference constraints, as well as joint velocity and actuator force limits. Further study of this approach is needed to assess its feasibility.

7.4 Workcell Layout Optimization

For many applications, a robot workcell must be designed to perform some specified task. Current workcell design methods are based largely on trial and error, and the resulting layout is usually sub-optimal.

Let us pose a hypothetical but typical problem. Suppose we wish to find an optimal fixed position for a workpiece such that a continuous path manipulator tool trajectory specified relative to the workpiece is feasible and some cost functional is minimized. This problem can be solved numerically by making the cost functional a function of the trajectory and the workpiece position.
7.5 *Mobile Robot Trajectory Planning*

The methods described in this thesis can be easily applied to mobile robot path planning. In general, a mobile robot moves on a 2-D plane among a field of obstacles. The problem is exactly analogous to the 2-D point-to-point problem. The problem is simplified because the robot can be controlled directly in cartesian task coordinates.
This thesis presents an innovative approach to the problem of off-line robot trajectory planning which goes beyond any previous work. A general mathematical framework has been established to allow a better understanding of the fundamental problems involved in trajectory planning. Based on this mathematical foundation, a numerical method has been developed for optimization of robot trajectories subject to constraints due to joint limits and interference. The method models objects as convex polyhedra, and allows any starting trajectory to be used for the optimization. A discrete cost functional is proposed, and constraints are represented as penalty functions. A steepest descent numerical iteration scheme is used to modify the trajectory so that the cost functional is reduced.

A detailed analysis of interference between convex polyhedra is presented, and methods for calculating both separating vectors for disjoint polyhedra and penetration vectors for intersecting polyhedra are developed. A convex hull approximation is used for the volume swept by a polyhedron moving between discrete positions. This convex hull is used for the calculation of collisions with obstacles. Penetration is reduced iteratively by translating the polyhedra in a direction which reduces the penetration vector. This is a discrete approximation of reducing a continuous distance function by a steepest descent move along the gradient.

An alternative approach to solving the inverse kinematic problem is also presented. This method modifies the general
linearized inverse solution to allow the iterated gripper path to deviate from the straight line path which results if the minimum norm solution alone is used to reduce the position error. With this approach, the joint motion is less constrained when the gripper is far from the goal, and the optimization of the joint configuration can be more heavily weighted. The result is often a better inverse solution.

The above techniques have been implemented in a FORTRAN program to find an optimal geometric trajectory for a simple two dimensional point-to-point trajectory optimization problem. The program has successfully found trajectories for a number of test cases. Implementation of the method in 3-D is straightforward.

The approach can also be applied to related problems such as optimization of continuous robot trajectories with a specified gripper path, optimization of the dynamic trajectory, and workstation layout optimization.
REFERENCES


KONSTANTINOV, M., MARKOV, M., NENCHEV, D. "Kinematic Control of Redundant Manipulators", *Proceedings of the 11th


I.1 **Definition of the Pseudoinverse**

The pseudoinverse of a matrix can be defined as follows [Albert, 1972, pp. 15-42]. For any $n \times m$ matrix $H$, a pseudoinverse matrix $H^+$ always exists. For any $x \in \mathbb{R}^n$,

$$q^* - H^+ x, \quad q^* \in \mathbb{R}^m$$

(I.1)

is the vector of minimum norm among all $q$ which minimize

$$\| x - Hq \|.$$  

(I.2)

If $H$ is square and nonsingular,

$$H^+ = H^{-1}.$$  

(I.3)

If the rows of $H$ are linearly independent,

$$H^+ = H^T (HH^T)^{-1}.$$  

(I.4)

If the columns of $H$ are linearly independent,

$$H^+ = (H^T H)^{-1} H^T.$$  

(I.5)
1.2 Solution of a Linear System of Equations

Suppose we have a set of linear equations,

\[ x - Hq, \quad x \in \mathbb{R}^n, \quad q \in \mathbb{R}^m, \quad H \in \mathbb{R}^{mxn}. \]  

(I.6)

The general solution is

\[ q = H^+x + (I-H^+H)z, \quad z \in \mathbb{R}^n, \quad z \text{ arbitrary} \]  

(I.7)

where \( H^+x \) is the minimum norm particular solution and \( (I-H^+H)z \) is the homogeneous solution. The matrix \( (I-H^+H) \) is a projection onto the null-space of \( H \).

A better understanding is possible if we now rewrite equation (A.6) as

\[ x = H (H^+x) + H (I-H^+H) z \]  

(I.8)

From the definition of a pseudoinverse we know that \( H = HH^+H \), so the second term is zero for all \( z \) and \( x \) is invariant with respect to \( z \).

1.3 The Singular Value Decomposition

The singular value decomposition provides a stable and general method for determining the pseudoinverse of a matrix. A more detailed discussion of the singular value decomposition can
be found in [Klema and Laub, 1980] and [Golub and Van Loan, 1983].

Define any matrix $H \in \mathbb{R}^{m \times n}$. There exist orthogonal matrices $U \in \mathbb{R}^{m \times m}$, $V \in \mathbb{R}^{n \times n}$ such that

$$H = U \Sigma V^T$$  \hspace{1cm} (I.9)

where

$$\Sigma = \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix},$$  \hspace{1cm} (I.10)

$$S = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r),$$  \hspace{1cm} (I.11)

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0.$$  \hspace{1cm} (I.12)

The $\sigma_i$ are called the singular values of $H$. These are the positive square roots of the eigenvalues of $H^T H$. The columns of $U$ are the eigenvectors of $HH^T$. The columns of $V$ are the eigenvectors of $H^T H$. The number $r$ of non-zero singular values is the rank of $H$,

$$r = \text{rank}(H).$$  \hspace{1cm} (I.13)

The pseudo-inverse $H^+$ can be found from the singular value decomposition as

$$H^+ = V \Sigma^+ U^T$$  \hspace{1cm} (I.14)

where
\[ \Sigma^+ = \text{diag}(\sigma_1^+, \sigma_2^+, \ldots, \sigma_r^+) \]  
(I.15)

and

\[ \sigma_i^+ = \begin{cases} 
\frac{1}{\sigma_i} & \text{for } \sigma_i > 0 \\
0 & \text{for } \sigma_i = 0.
\end{cases} \]  
(I.16)

The University of British Columbia Computing Center FORTRAN routine UBC SVD was used to calculate the singular value decomposition.

I.4 \hspace{1cm} \textbf{Singularities and Rank Deficiency}

If a matrix \( H \) of rank \( r \) is perturbed until two rows (or columns) become dependent, then \( \sigma_r \) becomes zero and the rank is reduced to \( r - 1 \). Matrix \( H \) is then said to be rank deficient. If \( H \) is the Jacobian of a manipulator, then rank deficiency corresponds to kinematic singularity. Numerical difficulties can arise during the transition when two rows (or columns) are nearly dependent, and \( \sigma_r \) becomes very small. Physically, this can lead to very large joint velocities near manipulator singularities. A common solution is to reduce the rank of \( H \) when the smallest singular value becomes smaller than some threshold, i.e., if \( \sigma_r \leq \epsilon \), then rank \( H = r - 1 \), and \( \sigma_r = 0 \).

A more reliable measure of the nearness of rank deficiency is the condition number of \( H \),

\[ k(H) = \frac{\sigma_1}{\sigma_r}. \]  
(I.17)
Wampler [1986] has proposed an improved method for dealing with near rank deficiency using a damped least squares approach.

I.5 Calculation of the Jacobian

The Jacobian $J$ is a matrix of partial derivatives which relates incremental changes in the joint coordinates, $\delta q$, to incremental changes in task coordinates (end effector position and orientation), $\delta x$.

$$\delta x = J \delta q,$$  \hspace{1cm} (I.18)

where

$$\delta x \in \mathbb{R}^m, \quad \delta q \in \mathbb{R}^n, \quad J = \frac{\partial \delta x}{\partial q}.$$  \hspace{1cm} (I.19)

The task coordinate vector is composed of a position vector $r_p$ and an orientation vector $r_o$, where $r_o$ is a vector of roll, pitch and yaw angles.

$$x = \begin{bmatrix} r_p \\ r_o \end{bmatrix}.$$  \hspace{1cm} (I.20)

For a 2-D manipulator moving in a plane,

$$r_p \in \mathbb{R}^2, \quad r_o \in \mathbb{R}.$$  \hspace{1cm} (I.21)
For a 3-D manipulator,

\[ \mathbf{r}_p \in \mathbb{R}^3, \mathbf{r}_o \in \mathbb{R}^3. \]  

(I.22)

The Jacobian is

\[ J = \frac{\partial \mathbf{x}}{\partial \mathbf{q}} \]  

(I.23)

which can be decomposed into

\[
J = \begin{bmatrix}
    \frac{\partial \mathbf{r}_p}{\partial q_1} & \frac{\partial \mathbf{r}_p}{\partial q_2} & \cdots & \frac{\partial \mathbf{r}_p}{\partial q_n} \\
    \frac{\partial \mathbf{r}_o}{\partial q_1} & \frac{\partial \mathbf{r}_o}{\partial q_2} & \cdots & \frac{\partial \mathbf{r}_o}{\partial q_n}
\end{bmatrix}
\]  

(I.24)

Letting \( \frac{\partial \mathbf{r}_p}{\partial q_1} = \beta_1, \frac{\partial \mathbf{r}_o}{\partial q_1} = \gamma_1 \), then

\[
J = \begin{bmatrix}
    \beta_1 & \beta_2 & \cdots & \beta_n \\
    \gamma_1 & \gamma_2 & \cdots & \gamma_n
\end{bmatrix}.
\]  

(I.25)

Several methods exist for calculating the elements of the Jacobian [Orin and Schrader, 1984], [Paul, 1981, pp.101-108]. We will use the method of Olson and Ribble as described by Orin and Schrader. The method is summarized below.

The transformation relating two adjacent links \( i-1, i \) is given by the Denavit-Hartenberg matrix,
where \( s_{q_i}, c_{q_i} \) signify the sine and cosine of the joint angles \( q_i \), and \( \alpha_i, a_i \) and \( d_i \) are link parameters. The upper-left sub-matrix is a \( 3 \times 3 \) rotation matrix, \( U_{i-1,i} \), and the last column is the translation vector \( p_{i-1,i} \).

Define \( T_{0,i} \) as the transformation from the robot base frame to the \( i^{th} \) link frame.

\[
T_{0,0} = I \tag{I.27}
\]

\[
T_{0,i} = T_{0,i-1} T_{i-1,i}, \quad i = 1, \ldots, n+1 \tag{I.28}
\]

For a revolute joint:

\[
\gamma_i = U_{0,i-1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad i = 1, \ldots, n, \tag{I.29}
\]

\[
r_{i-1} = p_{i-1} - p_{n+1}' \tag{I.30}
\]

\[
\beta_i = \gamma_i \times (-r_{i-1}), \quad i = 1, \ldots, n. \tag{I.31}
\]

For a prismatic joint:

\[
\gamma_i = 0, \tag{I.32}
\]
APPENDIX II. CALCULATION OF THE CONVEX HULL ENCLOSING TWO POLYGONS

To find the minimum convex hull enclosing two polygons, we can use a standard "gift-wrapping" algorithm.

Define the set of vertices

\[ v_{i,j}, \quad i=1,2, \quad j=1, nverts(i) \tag{II.1} \]

where \( nverts(i) \) is the number of vertices of polygon \( i \). The vertices of each polygon are ordered in a clockwise direction.

Define the edges as

\[ e_{i,j} = v_{i,j+1} - v_{i,j} \tag{II.2} \]

and define the outward normal of \( e_{i,j} \) as

\[ n_{i,j} = \begin{bmatrix} -e_2 \\ e_1 \end{bmatrix}_{i,j'} \tag{II.3} \]

where \( e_1, e_2 \) are the components of the edge vector \( e \). The equation of the line containing \( e_{i,j} \) is
An edge $e_{i,j}$ is an edge of the convex hull if

$$v_{k,m} \cdot n_{i,j} > d_{i,j}$$  \hspace{1cm} (II.5)

for $k=1$ if $i=2$, else $k=2$ if $i=1$,

$m = 1, \text{verts}(k)$.

All vertices lie on one side of the line containing an edge of the convex hull. Given an edge of the convex hull $e_1 = v_{2} - v_{1}$ and a set of vertices $v_j \in V$, the next edge $e_{i+1}$ of the convex hull is found by minimizing the dot product of edge $e_i$ and candidate succeeding edges $e_j$:

$$e_{i+1} - e_j : \min \{ e_{i+1} - e_j : e_{i+1} - v_j - v_{j-1}, v_j \in V \}.$$  \hspace{1cm} (II.6)

The preceding mathematical definitions are incorporated into an algorithm which is described below.

Test all edges of both polygons until an edge is found such that all vertices of both polygons lie on the same side of the edge. This edge will be taken as the first edge of the convex hull.

If no edges of either polygon is an edge of the hull, then test the line segments created by joining the first vertex of the first polygon to each vertex of the second polygon until
one satisfies the above condition, and make that segment the first edge of the hull.

The current edge is the first hull edge found above. The current polygon is the polygon containing the edge, or the first polygon.

UNTIL the starting vertex is the second vertex of the current edge,

Test if the next edge of the current polygon is an edge of the convex hull.

If so

make it the current edge, and add it to the hull.

ELSE,

join the second vertex of the current edge to every vertex of the other polygon. The line segment which makes the smallest angle (the smallest dot product) to the current edge becomes the current edge, and is added to the hull. The other polygon becomes the current polygon.

ENDIF

END UNTIL
After a convex hull is generated, it is processed to remove redundant vertices which are within some threshold distance of adjacent vertices. This corrects for the accumulation of very closely spaced vertices due to computer round-off errors, and allows more efficient and reliable processing.

II.1 Implementation of the Algorithm

The convex hull algorithm is implemented in FORTRAN subroutine HULL2D, a listing of which is found in Appendix IV.6. Great care is required in identifying and dealing with degenerate situations. The following degenerate cases can be identified:

1. Two or more vertices are coincident.

2. An edge of one polygon is collinear with an edge of the second polygon.

3. A vertex of one polygon lies on the line containing an edge of the other polygon.

Some particular cases are:

1. A pair of edges are coincident.

2. The polygons are identical and coincident.
2. The polygons are identical and coincident.

Numerical problems can be encountered for situations which are close to any of the singular conditions described above.

Because the polygons are convex and their vertices are ordered, it is possible to reduce the search required to find edges of the convex hull. An improved search algorithm has not yet been implemented.
APPENDIX III. GEOMETRY OF 3-D IMPLEMENTATION

The kinematic parameters and geometry of a PUMA 560 robot are shown in Figure III.1. The values of the Denavit-Hartenberg parameters are given in the following table:

<table>
<thead>
<tr>
<th>Link i</th>
<th>Variable</th>
<th>$\alpha_i$</th>
<th>$a_i$</th>
<th>$d_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$q_1$</td>
<td>-90</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>$q_2$</td>
<td>0</td>
<td>432</td>
<td>149.5</td>
</tr>
<tr>
<td>3</td>
<td>$q_3$</td>
<td>90</td>
<td>-20.5</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>$q_4$</td>
<td>-90</td>
<td>0</td>
<td>432</td>
</tr>
<tr>
<td>5</td>
<td>$q_5$</td>
<td>90</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>$q_6$</td>
<td>0</td>
<td>0</td>
<td>56.5</td>
</tr>
</tbody>
</table>

where the dimensions are in millimeters. The set of test workpieces is shown in Figure III.2.
Figure III.1  PUMA 560 link geometry.
Figure III.2  Sketches of test workpieces.
IV.1  Users Guide to OPT

IV.1.1  Compiling and Linking OPT

The executable program OPT is generated by compiling and linking the following FORTRAN source files:

OPT, CORECT, COSTT, CPUTIME, CVECFT, FINPAT, GMULT, GROW, GVERTS, HULL2D, INPUT, INSERT, INTERF, INT2D, JACOB, RDFILE, SOLVE1, SOLVE2, SORT, SVDWS, TRANDB

IV.1.2  Running OPT

The FORTRAN program for numerical optimization of a manipulator trajectory can be run by following these steps:

1. Set up the appropriate kinematic specifications for the manipulator by editing the file PARAM1.DAT.

2. Set up the geometry of the objects in the workstation by editing the file BOX.DAT.

3. Generate a starting trajectory by running the interactive program STRAJ.
4. Run the program OPT by entering

RUN OPT

OPT will respond with the following:

Do you wish to enter new parameters? (T/F)

If you want to change one or more parameters, respond by pressing T. OPT will then display the current value of each parameter from PARAM.DAT, and will ask if you wish to change the value by prompting

Change? (T/F)

If you wish to change the parameter, respond by pressing T. OPT will then prompt for a new value. After a new value is entered, OPT will continue to the next parameter in the list. The parameter file PARAM.DAT is automatically updated with the new values.

When the parameters have been selected, OPT begins the iterative optimization. After each iteration, the following information will appear on the screen:

IND1, LAST, COST, DMAX, FMAX

where IND1 is the iteration index, LAST is the number of discrete trajectory positions, COST is the value of the
cost functional for the current iteration, DMAX is the largest penetration magnitude for the current iteration, and FMAX is the largest violation of the joint limits. In addition, a message will appear if trajectory positions are added or removed after an iteration.

When the stopping conditions are satisfied, the message

**** Interference and joint limits satisfied ****

will appear on the screen, and the program will terminate. Otherwise, OPT will terminate after the specified maximum number of iterations.

5. If desired, the output files can be plotted using a package such as Tektronix EZGRAF. If the Tektronix IGL graphics library is available, then the manipulator trajectory can be shown graphically by running the program DRLINK.

IV.1.3 Input Files

Filename: box.dat

Description: contains the geometry of the objects in the robot workstation
The file is read as follows:

```
read(l,*) NLINKS
do 1 i=1,NLINKS
    read(l,*) NVERTS(i),IDOF(i)
    do 2 j=1,NVERTS(i)
        read(l,*) (SVERTS(k,j,i),k=1,2)
    2 continue
1 continue

read(l,*) NOBST
do 3 i=1,NOBST
    read(l,*) NVERTS(i)
    do 4 j=1,NVERTS(i)
        read(l,*) (SVERTS(k,j,i),k=1,2)
    4 continue
3 continue
```

Variables:

- **NLINKS**: the number of manipulator links modelled
- **NOBST**: the number of obstacles modelled
- **NVERTS(i)**: the number of vertices of link i (or obstacle i)
IDOF(i): the number of degrees of freedom of link i

SVERTS: the numerical coordinates of the vertices'

Filename: paraml.dat

Description: contains the kinematic specifications of the manipulator

Contents:

*ndof,nxdoe
NDOF,NXDOF
*workstation size
SIZE
*jtype
JTYPE(I),I=1,NDOF
*alpha
ALPHA(I),I=1,NDOF
*len
LEN(I),I=1,NDOF
*dist
DIST(I),I=1,NDOF
The lines starting with * indicate text entries for guidance during file editing.

Variables:

NDOF: the number of link degrees of freedom
NXDOF: the number of task degrees of freedom
SIZE: a scaling factor indicating the order of magnitude of the workstation size in the chosen units
JTYPE: an array indicating the type of each joint,
  0- revolute
  1- prismatic
ALPHA: an array of the link twist angles
LEN: an array of the link lengths
DIST: an array of the link joint offsets

Filename: param.dat

Description: contains the parameters needed for the numerical reduction of the cost functional

Contents:

JFACT
Variables:

\( j_{\text{fact}} \): the joint limit penalty function weighting factor \( A \) (Equation 4.75)

\( p_{\text{fact}} \): the interference penalty function weighting factor \( B_1 \) (Equation 4.75)

\( f_{\text{acti}} \): the penetration fraction \( \sigma \) (Section 3.4.4.3)

\( a_{\text{fact}} \): the acceleration cost function weighting factor \( C \) (Equation 4.75)

\( n_{\text{max}} \): the maximum number of iterations

\( t_{\text{au}} \): the iteration step size factor \( \alpha \) (Equation 4.78)

\( s_{\text{tepsi}} \): the maximum allowable trajectory iteration error (Section 5.2.2)

\( c_{\text{max}} \): the maximum allowable trajectory linearization error (Section 5.2.3)

\( m_{\text{axpen}} \): the penetration tolerance \( \delta_{_{\text{pen}}} \) (Section 5.3)
jtoller: the joint limit violation tolerance \( \delta_{\text{Joint}} \)  

(Section 5.3)

vmin: the minimum joint limits \( q_{\text{min}} \) (Section 4.2.4)

vmax: the maximum joint limits \( q_{\text{max}} \) (Section 4.2.4)

Filename: traj.dat

Description: contains the set of discrete joint positions specifying a starting trajectory

Contents: the file is read by the following routine,

\[
\begin{align*}
\text{read(1,*)(XGOAL(I),I=1,3), ANGOAL} \\
\text{read(1,*)(LAST)} \\
\text{do I=1,LAST} \\
\quad \text{read(1,*)(VALUES(J,I),J=1,NDOF}} \\
1 \quad \text{continue}
\end{align*}
\]

Variables:

XGOAL: coordinates of desired gripper position

ANGOAL: desired gripper orientation angle

LAST: the number of positions in the starting trajectory

VALUES: the set of joint angles defining each discrete position
IV.1.4 Output Files

A number of output files are generated by OPT, primarily for debugging and verification. The files contain one record per line to allow the use of the Tektronix EZGRAF plotting package. The following files are generated:

- **ang1.dat**: the final trajectory of joint 1 angles
- **ang2.dat**: the final trajectory of joint 2 angles
- **ang3.dat**: the final trajectory of joint 3 angles
- **ang4.dat**: the final trajectory of joint 4 angles
- **fmax.dat**: the maximum joint limit violation for each iteration
- **dmax.dat**: the maximum penetration for each iteration
- **last.dat**: the number of discrete trajectory positions for each iteration
- **dist.dat**: penetration versus position for the final iteration
- **cos1.dat**: the total cost for each iteration
- **cos2.dat**: the interference penalty cost for each iteration
- **cos3.dat**: the joint limit penalty cost for each iteration
- **cos4.dat**: the acceleration cost for each iteration
IV.2 Functional Description of OPT

OPT is a FORTRAN program which finds an optimal robot trajectory subject to joint limit and interference constraints. A cost functional is defined for the trajectory, using penalty functions to represent the constraints. A quasi-Newton iterative procedure is used to reduce a cost functional.

The current implementation applies to a 2-D planar manipulator moving among obstacles represented by convex polygons. The convex hull enclosing two adjacent discrete link positions is used for the interference calculations. The initial manipulator position is given, and the final configuration must satisfy the gripper goal position.

OPT has been written for evaluation and demonstration of the methods developed for trajectory optimization, and is not intended to be a complete or final software product.

IV.2.1 Nesting of Subroutines

The nesting of subroutine calls is specified as follows. Each level of indentation corresponds to a level of nesting.
IV.2.2 Functional Description of Subroutines

All subroutines are stored in separate FORTRAN files with the same name as the subroutine, except as noted. For example, subroutine CORECT is stored in file CORECT.FOR. The subroutines
used by OPT can be described functionally as follows.

**CORECT**
Calculates a joint space correction vector which will reduce interference between the swept volume of a moving object and a stationary obstacle.

**COSTT**
Evaluates the cost functional by summing the components of each discrete position along the trajectory.

**CVECT**
Accesses the database containing the correction vectors for the current and previous link positions.

**FINPAT**
This subroutine performs the iterative optimization of the robot trajectory.

**GROW**
Grows the obstacles by some amount DELTA to compensate for the use of exterior penalty functions. If the penetration of the grown obstacles is reduced below DELTA, there is no penetration of the true obstacles.

**GVERTS**
Accesses the geometry database for a given object.
HULL2D

Finds the convex hull enclosing two convex polygons in 2-D.

INPUT

Initializes the parameters for the iterative trajectory optimization algorithm. The parameters are read from the file PARAM.DAT, and can be interactively modified by the user.

INSERT

Inserts and removes discrete positions along the trajectory to maintain a desired linearization error in the discrete approximation of the trajectory.

INTERF

Tests for interference between a swept volume and an obstacle, and finds the necessary correction vectors to reduce interference.

INT2D

Tests two 2-D polygons for intersection, and finds the penetration vector and penetration point relative to the first polygon.

JACOB

Calculates the Jacobian matrix for a given manipulator configuration.
RDFILE

Reads the contents of the setup file PARAM1.DAT and initializes the variables.

SOLVE1

Finds the projection onto the nullspace of a matrix using the singular value decomposition.

SOLVE2

Finds the minimum norm inverse kinematic solution.

SORT

Tests all the penetration vectors and flags those which are larger than some fraction of the largest penetration.

SVD

Performs a singular value decomposition to find a minimum norm solution. This is a University of British Columbia routine, UBC SVD. The source code is contained in file SVDWS.FOR.

TRANDB

Transforms the geometric database specifying kinematic chains of rigid links by performing a chain of transformations on each link.
IV.2.3 Utility Routines

Several utility subroutines are called by OPT, and are listed here for completeness.

GMULT:

Multiplies two matrices.

Function GET_CPU_TIME:

Returns elapsed CPU time. The source code is stored in file CPUTIME.FOR.
IV.2.4 Documentation for UBC Singular Value Decomposition Routines
SINGULAR VALUE DECOMPOSITION OF A MATRIX

Carolyn Moore

Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Note</td>
<td>1</td>
</tr>
<tr>
<td>2. Acknowledgement</td>
<td>1</td>
</tr>
<tr>
<td>3. Preliminary Discussion</td>
<td>1</td>
</tr>
<tr>
<td>b. Least Squares Solutions of Overdetermined or Underdetermined Systems of Equations</td>
<td>2</td>
</tr>
<tr>
<td>c. Solution of Homogeneous Equations</td>
<td>2</td>
</tr>
<tr>
<td>4. Purpose</td>
<td>2</td>
</tr>
<tr>
<td>5. Type of Routine</td>
<td>3</td>
</tr>
<tr>
<td>6. Availability</td>
<td>3</td>
</tr>
<tr>
<td>7. How To Use</td>
<td>3</td>
</tr>
<tr>
<td>a. Double Precision Routines</td>
<td>3</td>
</tr>
<tr>
<td>b. Single Precision Routines</td>
<td>4</td>
</tr>
<tr>
<td>8. Restrictions</td>
<td>4</td>
</tr>
<tr>
<td>9. Accuracy</td>
<td>4</td>
</tr>
<tr>
<td>10. Sample Program</td>
<td>5</td>
</tr>
<tr>
<td>11. Timing</td>
<td>7</td>
</tr>
<tr>
<td>12. Method</td>
<td>7</td>
</tr>
<tr>
<td>13. Comments</td>
<td>7</td>
</tr>
<tr>
<td>14. References</td>
<td>7</td>
</tr>
</tbody>
</table>
1. NOTE

The following changes have been made to the writeup UBC SVD since the version published July 1976.

— The restriction of \( N \geq M \) is removed. The matrix \( A \) must be dimensioned at least \( \max(M,N) \times M \).

— The routines work when \( A \) is dimensioned \( 1 \times 1 \) and for certain special cases when one or more singular values are calculated to be exactly zero.

— The timings are updated.

2. ACKNOWLEDGEMENT

This routine is adapted from the ALGOL routines given in section c.

3. PRELIMINARY DISCUSSION

Consider an \( N \times M \) matrix \( A \) of rank \( r \leq \min(M,N) \). It has been shown that there exists a decomposition of the matrix

\[
A = U \Sigma V^T
\]

where \( U^T U = V^T V = I \) and \( \Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_M) \). \( U \) is a matrix consisting of the orthonormalized eigenvectors of \( A A^T \), and \( V \) is a matrix consisting of the orthonormalized eigenvectors of \( A^T A \). The \( \sigma_i \) are non-negative square roots of the eigenvalues of \( A^T A \); they are called singular values. Assuming \( \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_M \) and \( \text{rank}(A) = r \) then

\[
\sigma_{r+1} = \sigma_{r+2} = \ldots = \sigma_M = 0.
\]

The decomposition can be used for a variety of applications. Some examples are given below.

a. Pseudoinverse

Let \( A \) be an \( N \times M \) real matrix. An \( M \times N \) matrix \( A^* \) is the pseudoinverse of \( A \) if it has the following four properties

\[
AA^* A = A
\]

\[
A^* AA^* = A^*
\]

\[
(AA^*)^T = AA^*
\]

\[
(A^* A)^T = A^* A
\]
A* is unique and can be calculated from a singular value decomposition by the formula:

\[ A^* = V \Sigma^* U^T \]

where \( \Sigma^* = \text{diag}(\Sigma_{ij}^*) \) and

\[ \sigma_{ij}^* = \begin{cases} 1/\sigma_i & \text{for } \sigma_i > 0 \\ 0 & \text{for } \sigma_i = 0 \end{cases} \]

b. Least Squares Solutions of Overdetermined or Underdetermined Systems of Equations

Let \( \mathbf{b} \) be a vector such that the minimization of

\[ \sum_{i=1}^{N} (b_i - A\mathbf{x}_i)^2 \]

is required. If the rank of \( A \) is less than \( M \) (this is always the case for underdetermined systems of equations) then there is no unique solution. However, if the minimum of

\[ \sum_{i=1}^{M} x_i^2 \]

is also required, then this is a unique solution. It can be shown that the unique solution \( \mathbf{x} \) is given by

\[ \mathbf{x} = A^* \mathbf{b} = V \Sigma^* U^T \mathbf{b} = V \Sigma^* \mathbf{c} \]

where \( \Sigma^* \) is defined above and \( \mathbf{c} = U^T \mathbf{b} \). The vector \( \mathbf{c} \) is returned by DSLSVD or DSVD if requested. In fact \( \mathbf{b} \) may be a matrix with \( p \) columns. In this case a matrix \( \mathbf{c} \) of \( p \) columns is returned by the routines.

c. Solution of Homogeneous Equations

If \( A \) has rank \( r \) and \( \mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_M] \) where \( \mathbf{v}_i \) is a column of \( \mathbf{V} \), then \( \mathbf{A} \mathbf{v}_i = 0 \) for \( i = r+1, \ldots, M \). The columns \( \mathbf{v}_{r+1}, \ldots, \mathbf{v}_M \) provide an orthonormal basis for all solutions of the homogeneous equations \( \mathbf{A} \mathbf{x} = 0 \).

4. PURPOSE

This routine will find the singular values \( \sigma_i, i = 1, \ldots, M \) of a real matrix \( A \) dimensioned \( N \times M \). If requested, the matrices \( U \) and/or \( V \) from the singular value decomposition are returned.
If the solution $x$ of a system $Ax=b$ is required, then the matrix $U^Tb$ is returned.

5. TYPE OF ROUTINE

FORTRAN IV SUBROUTINE subprogram.

6. AVAILABILITY

*LIBRARY and *WATLIB.

7. HOW TO USE

The routines have three entry points.

a. Double Precision Routines

- To find the singular values only
  
  \[
  \text{CALL DSING}(A,S,NDIMAUX,N,M,&n)
  \]

- To find the matrix $V$ and/or the matrix $U^Tb$
  
  \[
  \text{CALL DLSVD}(A,S,V,NDIMAUX,NDIMV,N,M,NP,&n)
  \]

- To find $U$ and/or $V$ and/or the matrix $U^Tb$
  
  \[
  \text{CALL DSVDD}(A,S,U,V,NDIMAUX,NDIMV,N,M,NP,NU,NV,&n)
  \]

where

$A$ is a REAL*8 two-dimensional array dimensioned at least $\max(M,N) \times (M + NP)$. On entry, the first $M$ columns of $A$ must contain the matrix whose decomposition is required. If $NP \neq 0$, then the $M+1,...,M+NP$ columns of $A$ contain the right hand sides $b$ of the system $Ax=b$ to be solved. On exit, the first $M$ columns of $A$ are destroyed and the $M+1,...,M+NP$ columns contain $U^Tb$.

$S$ is a REAL*8 one-dimensional array of dimension at least $M$. On exit, $S$ contains the singular values of $A$ such that $S(1) \geq S(2) \geq \ldots \geq S(M)$.

$U$ is a REAL*8 two-dimensional array dimensioned at least $N \times NU$. On exit, $U$ contains the matrix $U$.

$V$ is a REAL*8 two-dimensional array dimensioned at least $M \times NV$. On exit, $V$ contains the matrix $V$.

$NDIMAUX$ is an INTEGER variable or constant. On entry, $NDIMAUX$ contains the first dimension of the arrays $A$ and $U$. $NDIMAUX \geq \max(M,N)$.

$NDIMV$ is an INTEGER variable or constant. On entry, $NDIMV$ contains the first dimension of the array $V$. $NDIMV \geq M$. 

192
**N** is an INTEGER variable or constant. On entry, \( N \) contains the number of rows in the array \( A \) to be decomposed.

**M** is an INTEGER variable or constant. On entry, \( M \) contains the number of columns in the array \( A \) to be decomposed. For the WATFIV versions, \( M \leq 100 \).

**NP** is an INTEGER variable or constant. On entry, \( NP \) is the number of right hand sides \( b \), if the solution of \( Ax = b \) is required. \( NP = 0 \) otherwise.

**NU** is an INTEGER variable or constant. On entry, \( NU = \min(M, N) \) if the \( NxNU \) matrix \( U \) is required. \( NU = 0 \) otherwise.

**NV** is an INTEGER variable or constant. On entry, \( NV = \min(M, N) \) if the \( MxNV \) matrix \( V \) is required. \( NV = 0 \) otherwise.

**n** is a statement number to which control is returned if any of the singular values are not found within 40 iterations. \( n \) is omitted on the WATFIV versions. Instead an error message is printed. When an error condition exists, the singular values are not sorted in decreasing order.

### b. Single Precision Routines

```fortran
CALL SING(A,S,NDIMAU,N,M,&n)
CALL SOLSVD(A,S,V,NDIMAU,NDIMV,N,M,NP,&n)
CALL SVD(A,S,U,V,NDIMAU,NDIMV,N,M,NP,NU,NV,&n)
```

Where \( A, S, U, \) and \( V \) are declared REAL*4.

### 8. RESTRICTIONS

For the WATFIV version, \( M \leq 100 \).

### 9. ACCURACY

For the cases tested, using the single precision routines, singular values were found accurate to five significant figures; and \( \max_{ij}(U^TU-I), \max_{ij}(V^TV-I) \), and \( \max_{ij}(A-U\Sigma V^T) \) lay within the interval \((10^{-6}, 10^{-5})\). When finding the solution of the overdetermined system of equations \( Ax = b \), a singular value \( (\sigma_k \text{say}) \) was considered negligible if \( \sigma_k / \max_i \sigma_i \leq S \), where \( S \) was an input tolerance usually around \( 5E-7 \). \( \sigma_k \) was then set equal to zero. The library routine GMULT was used to find the matrix product \( \Sigma \cdot U^Tb \) (=solution \( x \)). Results were generally accurate to four or five significant figures.
10. SAMPLE PROGRAM

The following program finds the singular value decomposition of a 8*6 matrix A given below, and solves the system Ax=0, and the system Ax=b for 3 right hand side vectors b.

\[
A = \begin{bmatrix}
22 & 10 & 2 & 3 & 7 \\
14 & 7 & 10 & 0 & 8 \\
-1 & 13 & -1 & -11 & 3 \\
-3 & -2 & 13 & -2 & 4 \\
9 & 8 & 1 & -2 & 4 \\
9 & 1 & -7 & 5 & -1 \\
2 & -6 & 6 & 5 & 1 \\
4 & 5 & 0 & -2 & 2
\end{bmatrix}
\]

\[
b_1, b_2, b_3 = \begin{bmatrix}
-1 \\
2 \\
-1 \\
4 \\
2 \\
0 \\
1 \\
0
\end{bmatrix}
\]

\[
\sigma_1 = 35.327038, \quad \sigma_2 = 20, \quad \sigma_3 = 19.595916, \quad \sigma_4 = \sigma_5 = 0.
\]

True solutions:

\[
x_1, x_2, x_3 = \begin{bmatrix}
-1/12 & 0 & -1/12 \\
0 & 0 & 0 \\
1/4 & 0 & 1/4 \\
-1/12 & 0 & -1/12 \\
1/12 & 0 & 1/12
\end{bmatrix}
\]

```
IMPLICIT REAL*8(A-H,O-Z)
DIMENSION A(8,8),U(8,8),V(8,8),S(8),X(8,3)

C READ THE MATRIX A AND PLACE 3 RIGHT HAND SIDES IN COLUMNS M+1,M+2,M+3 OF A
C
READ (5,10) N,M,NP
10 FORMAT(3I2)
READ(5,20) ((A(I,J),I=1,N),J=M1,MNP)
20 FORMAT(4F10.0)
M1=M+1
MNP=M+NP
READ (5,20) ((A(I,J),I=1,N),J=M1,MNP)

CALL DSVD TO FIND U,V, AND U-TRANSPOSE*B

NDIMAU=8
NDIMV=8
CALL TIME(0)
CALL DSVD(A,S,U,V,NDIMAU,NDIMV,N,M,NP,M,M,&140)
```
CALL TIME(15,1)

WRITE SINGULAR VALUES

WRITE (6,30) (S(I),I=1,M)
30 FORMAT('OSINGULAR VALUES ARE',/,1X,5G12.5)

THE FOLLOWING DETERMINES ZERO SINGULAR
VALUES. IF S(J)/S(1).LT.EPS THEN S(J) IS
CONSIDERED TO BE ZERO

EPS=.5D-6
SS=S(1)*EPS
WRITE (6,40)
40 FORMAT('OCOLUMNS OF V CORRESPONDING TO NEGLIGIBLE S(J)')
DO 60 J=1,M
IF (S(J).LT.SS) GO TO 70

V*DIAG(S(1),...,S(M)) IS PLACED IN V

DO 50 I=1,M
50 V(I,J)=V(I,J)/S(J)
60 CONTINUE
J=M+1
GO TO 90

COLUMNS J,...,M OF V ARE A BASIS FOR ALL
SOLUTIONS OF AX=0

WRITE (6,80) ((V(I,K),I=1,M),K=J,M)
80 FORMAT (1X,5G13.5)

COLUMNS OF V FOR 0 SINGULAR VALUES ARE ZEROED

IF (J.GT.M) GO TO 120
DO 110 K=J,M
DO 110 I=1,M
110 V(I,K)=0.D0

FORM X=V*DIAG(S(1),...,S(M))*(U-TRANSPOSE*B)

MP1=M+1
CALL DGMULT(V,A(1,MP1),X,M,M,NP,NDIMV,NDIMAU,8)
WRITE (6,130) ((X(I,J),J=1,NP),I=1,M)
130 FORMAT('OSOLUTION VECTORS',5(/,3G15.7))
STOP

140 WRITE (6,150)
150 FORMAT('ERROR RETURN FROM DSVD')
STOP
END

The output from the above program follows:
11. TIMING

The sample program takes .006 seconds of CPU time on the Amdahl 470 V/6-II computer using the DSVD in *LIBRARY. Using DSVD in the WATFIV library requires .039 seconds of CPU time. The single precision SVD requires .0045 and .034 CPU seconds for the *LIBRARY and *WATLIB versions respectively.

12. METHOD

The matrix A is reduced to a bidiagonal matrix J (whose singular values are the same as those of A) by Householder transformations. The singular values of J are found by an iterative technique based on the QR algorithm for finding the eigenvalues of a matrix.

13. COMMENTS

The method used to find the singular value decomposition of a matrix is numerically stable, although in certain circumstances there may be a slow down in the rate of convergence.

The method used to determine whether a singular value is negligible or not can be stated in the following general terms. If a singular value is 'small' compared to other singular values, then consider that singular value to be zero. What constitutes 'small' should be determined by the user. It depends on several factors, including the condition of the matrix and the amount of accuracy required; it can often only be determined by inspection of the singular values.

14. REFERENCES


IV.3 Common Variables

The following is a list of the variables passed in common blocks.

/cmax/cmax: maximum allowable trajectory linearization error

/d2rad/d2rad: conversion factor from degrees to radians

/fracti/fracti: penetration fraction

/goal/xgoal(3),angoal
xgoal: coordinates of gripper goal position
angoal: gripper goal orientation (2-D)

/ident/ident(4,4): 4×4 identity matrix

/last/last: the number of discrete trajectory positions

/intdis/vectl(2,2,100),vect2(2,100),pointl(2,2,100),point2(2,100)
vectl(klast,i,ipoint)- translation vector applied to position k-1 of link ilink due to penetration of hull k-1 by obstacle iobst, where

\[ ipoint = (ilink-1) \times nobst + iobst \]

vectl(kcurr,i,ipoint)- translation vector applied to position k of link ilink due to penetration of
hull $k$ by obstacle $iobst$. If $klast=1$, then $kcurr=2$, else $klast=2$, $kcurr=1$.

vect2($i,i\text{point}$) - translation vector applied to position $k-1$ of link $ilink$ due to penetration of hull `$k$ by obstacle $iobst$

point1,point2 - the penetration points corresponding to the above translation vectors.

/limits/vmin(4),vmax(4)

vmin: array of minimum joint limits
vmax: array of maximum joint limits

/limtol/maxpen,jtoler

maxpen: the penetration tolerance used to grow the obstacles
jtoler: the joint limit violation tolerance

/lindof/lindof(2,20)

lindof(1,idof) - id of first link with $idof$ degrees of freedom
lindof(2,idof) - number of links with $idof$ degrees of freedom

/nmax/nmax: the maximum number of iterations

/nobjs/nlinks,nobst

nlinks: the number of manipulator links
nobst: the number of obstacles

/ndof/ndof, nxdof

ndof: the number of link or joint degrees of freedom
nxdof: the number of task degrees of freedom

/objs/boxes(2,10), sverts(2,100), tverts(2,100)

boxes: array of pointers into vertex arrays
boxes(1,i) - starting vertex of link i
boxes(2,i) - number of vertices of link i
boxes(1,nlinks+i) - starting vertex of obstacle i
boxes(2,nlinks+i) - number of vertices of obstacle i

sverts: array of fixed vertex coordinates
tverts: array of transformed vertex coordinates

/paraml/jfact, pfact

jfact: joint limit penalty function weighting factor
pfact: penetration penalty function weighting factor

/param3/afact: acceleration cost weighting factor

/robot/link(4), jtype(4), alpha(4), len(4), theta(4), dist(4)

This block contains the kinematic parameters of the manipulator.
link: not used
jtype: array of joint types
0-revolute, 1-prismatic
alpha: array of link twist angles
len: array of link lengths
theta: array of joint displacements
dist: array of joint offsets

/size/size: the scale of the workstation in the chosen units, order of magnitude only

/stepsi/stepsi: maximum allowable iteration linearization error

/tau/tau: iteration step size factor
IV. 4. Generation of a Starting Trajectory - STRAJ

A starting manipulator trajectory can be interactively generated by the user by running the program STRAJ. An initial link configuration and a gripper goal position and orientation are specified by the user. Intermediate gripper positions are then specified to define a coarse starting trajectory. The corresponding joint angles are determined by an iterative inverse kinematics algorithm.

STRAJ runs on a Retrographics DQ-650 terminal which can emulate a Tektronix graphics terminal. This permits graphical depiction of the trajectory while allowing easy user text interaction through the text window at the bottom of the screen.

STRAJ is intended to be a simple tool to be used with OPT for testing and verification, and as such is somewhat limited in its capabilities. For practical use, a more sophisticated software package is needed.

An executable file can be generated by compiling and linking the following source files: STRAJ.FOR, GMULT.FOR, INVERS.FOR, JACOB.FOR, RDFILE.FOR, ROTMAT.FOR, SVDWS.FOR.

IV. 4.1 Users Guide to STRAJ

1. Before running STRAJ, the kinematic parameters file PARAM.DAT and the geometry file BOX.DAT must exist with the correct contents as specified for OPT.

2. Put the terminal in 4027 graphics mode.
3. To run STRAJ, enter 'RUN STRAJ'.

4. STRAJ will ask for the number of task coordinates: If only the two gripper position coordinates are to be specified, enter '2'. If the gripper orientation angle is also to be specified, enter '3'.

5. STRAJ will ask for the initial angles of the manipulator joints. This is the initial or starting position, which is known. Enter the angles, in degrees.

6. The screen will go blank, and the obstacles and initial manipulator configuration will be drawn.

7. STRAJ will ask

   Save position? (T/F).

   Enter 'F'.

   STRAJ will ask

   Quit? (T/F).

   Enter 'F'.

8. STRAJ will ask for the goal position and orientation.
Enter the x and y position coordinates and a counterclockwise orientation angle in degrees, with the x axis being zero. If two task degrees of freedom were specified, the orientation angle will be ignored when the inverse solution is calculated.

9. STRAJ will ask

Erase screen? (T/F).

If you enter 'T', the screen will be erased and redrawn. If you enter 'F', all graphics will be superimposed on top of the existing screen image.

10. After the goal position has been entered and shown on the screen, STRAJ will ask

Save position? (T/F)

If you wish to save the goal position, enter 'T', otherwise enter 'F'.

11. STRAJ will ask

Quit? (T/F).

If you don't wish to specify any intermediate positions,
enter 'T' and an iterative inverse kinematic solution will be found for the goal position using the initial configuration as the starting point. Each time the iterative inverse kinematics routine is run, STRAJ will ask for the values of the maximum linearization error ERMAX and the deviation factor GAMMA. Values of 0.2 and 0.5, respectively, are good choices. If three task degrees of freedom were specified, then the gripper orientation will be preserved.

12. When a satisfactory goal position has been specified, intermediate gripper positions can be inserted, ordered from the initial configuration to the goal position. After a desired gripper position and orientation is entered, an iterative inverse solution is found for the joint angles, using the previous configuration as a starting point.

13. When a suitable number of intermediate positions have been specified, enter 'T' when STRAJ asks

Quit? (T/F).

An inverse solution will be found for the joint angles of the final configuration, using the last intermediate configuration as a starting point. All of the joint configurations from the initial to the final
configuration will be stored in file TRAJ.DAT, and STRAJ will then terminate.

IV.4.2 **Functional Description of Subroutines**

The interactive starting trajectory generation program STRAJ calls the following subroutines:

**INVERS:**

Implements an iterative algorithm to find an inverse kinematic solution.

**JACOB:**

See Appendix II.2.2.

**RDFILE:**

See Appendix II.2.2.

**ROTMAT:**

Transforms a generalized rotation axis and angle into the equivalent rotation matrix.

**SVD:**

See Appendix II.2.2.
Several utility programs have been written to generate graphical output. The programs are quite short, and the source code is sufficiently clear that no further documentation is required beyond a functional description. These programs require the availability of the Tektronix IGL graphics library, and have been run on a Retro-Graphics terminal which emulates Tektronix 4006 and 4027 graphics terminals. However, the programs can be adapted to any similar graphics environment.

Program name: DRLINK

Description: This program draws a graphical depiction of the final optimized discrete trajectory as a superposition of the discrete link positions composing the trajectory. The obstacles are also shown. The angles for each joint are read from separate files which are specified by the user. Typically, the files are the output files of OPT, ANG1.DAT, ANG2.DAT, ANG3.DAT and ANG4.DAT.

Program name: DRTRAJ

Description: This program shows the starting trajectory by reading the manipulator positions from a file, typically TRAJ.DAT, and superimposing them on the screen.

Program name: DRINV

Description: This program shows the trajectory of joint positions generated by the iterative inverse kinematic solution.
The user enters the names of the files containing the joint angles. The inverse kinematic routine INVERS generates joint files A1.DAT, A2.DAT, A3.DAT and A4.DAT.
The following files are listed, in alphabetical order:

- CORECT
- COSTT
- CVECT
- DRINV
- DRLINK
- FINPAT
- CPURTIME
- GMULT
- GVERTS
- HULL2D
- INPUT
- INSERT
- INT2D
- INVERS
- JACOB
- MINVER
- OPT
- RDFILE
- ROTMAT
- SOLVE1
- SOLVE2
- SORT
- STRAJ
- SVDWS
- TRANDB
subroutine corect(thedot,thdotl,vmaxl)

Calculate a joint space correction vector which will reduce interference between the swept volume of a moving object and a stationary obstacle.

Parameters:
- thedot - the joint change vector (in deg) which is the minimum norm solution minimizing the set of penetration vectors.
- thdotl - the joint change vector minimizing the largest penetration.
- vmaxl - the magnitude of the largest penetration vector.

Variables:
- vlist(i,j) - array pointing to the two largest penetrations
  - vlist(1,j) - link id
  - vlist(2,j) - obstacle id
  - vlist(3,j) - current or previous penetration vector
  - vlist(4,j) - number of degrees of freedom of link
- ftable(i,j) - table of penetration vectors larger than the threshold
  - i=link id, j=obst id
  - values:
    - 1 - current penetration only
    - 2 - previous penetration only
    - 3 - both penetrations

real thetot(4)
real obvel(2),vpoint(2)
real tnet(4,4),jac(6,5),hmult(4,4)
real ident(4,4),tinv(2,2),pinv(2),thetadot(4)
integer lindof(2,10)
integer ftable(10,10)
real mat(20,5)
real point(2,2),tpoint(2)
real len(4),dist(4),alpha(4),theta(4)
integer jtype(4),link(4)
real vlens(2)
integer vlist(4,2)
real thdotl(4),matl(4,5)
common /robot/link,jtype,alpha,len,theta,dist
common /ident/ident
common /rdof/ndof,nxdo
common /lindof/lindof
common /nobjs/nlinks,nobst
common /frac1/frac1

= 0.0174533
icount=0
d1 l i=1,4
  thetot(1)=0.
  thdotl(1)=0.
continue
Find all penetration vectors larger than some fraction of the largest penetration. Also find the largest and second largest penetrations.

frac=1.-fract
call sort(frac,vmax1,vmax2,vlist,ftable)

Calculate correction vectors for each link of the robot.

do 150 ind=1,ndof
  vlen=0.
  istart=indof(1,ind)
  inumb=indof(2,ind)
  if (istart.ne.0) then
    Calculate the net transformation matrix relating the final link frame to the world frame, 'tnet'.
    call jacob(ident,ident,len,alpha,dist,theta,jtype,ind,tnet,jac)
    Calculate the inverse transformation, transforming world coordinates to final link coordinates.
    do B3 i=1,2
      continue
    do B6 j=1,2
      tinv(i,j)=tnet(j,i)
      continue
    do 200 ilink=istart,istart+inumb-1
      do 250 iobst=1,nobst
        if (ftable(ilink,iobst).ne.0) then
          Get the penetration vectors for the current pair of objects.
          call cvect(1,ilink,iobst,vlsns,vect.point)
          Multiply the closest point in world coordinates by tinv to find the penetration point on the last link.
          do 133 ind4=1,2
            if (ftable(ilink,iobst).eq.ind4.or.
              ftable(ilink,iobst).eq.3) then
              do 156 i=1,2
                obvel(i)=vect(i,ind4)
                tpoint(i)=point(i,ind4)
              continue
            call gmult(tinv,tpoint,vpoint,2,2,1,2,2,2)
            vpoint(1)=vpoint(1)+pinv(1)
continue

do 266 i=1,4
   do 267 j=1,4
      hmult(i,j)=0.
      continue
   hmult(i,i)=1.
   continue
266
   hmult(1,4)=vpoint(1)
267
   hmult(2,4)=vpoint(2)
268
   Calculate the jacobian of the penetration point.

Calculation of Hmult

The desired joint angle change is a linear combination of the minimum norm solution for the largest penetration and the minimum norm solution for the set of penetration vectors. Thus the solution is weight to favor the largest penetration while minimizing the set of penetration vectors.
0177  if (icount.gt.2) then
0178      call solve2(2,mat,20,4,icount,ndof,homog,thedot)
0179      do 900 i=1,ndof
0180         thedot(i)=amult*thdot1(i)+(1.-amult)*thedot(i)
0179      900 continue
0181    else
0182      do 901 i=1,ndof
0183         thedot(i)=thdot1(i)
0184      901 continue
0185    endif
0186  endif

c Convert to degrees and update the net change.
0187  do 312 i=1,ndof
0188      thedot(i)=thedot(i)/d2rad
0189      thdot1(i)=thdot1(i)/d2rad
0190  312 continue
0191 return
0192 end

PROGRAM SECTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>#CODE</td>
<td>891</td>
<td>PIC CON REL LCL SHR EXE RD NONRT LONG</td>
</tr>
<tr>
<td>#DATA</td>
<td>16</td>
<td>PIC CON REL LCL SHR NOEXE RD NONRT LONG</td>
</tr>
<tr>
<td>#LOCAL</td>
<td>1608</td>
<td>PIC CON REL LCL NOEXE NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>ROBOT</td>
<td>96</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>IDENT</td>
<td>64</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>NDOF</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>LDOF</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>NOBJ</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>FRACTI</td>
<td>4</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
</tbody>
</table>

Total Space Allocated 2775

ENTRY POINTS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000</td>
<td>CORECT</td>
<td></td>
</tr>
</tbody>
</table>

penetrations.
### VARIABLES

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>RA4</td>
<td>AMULT</td>
<td>AA</td>
<td>RA4</td>
<td>D2RAD</td>
</tr>
<tr>
<td>2-000000514</td>
<td>RA4</td>
<td>HOMOG</td>
<td>AA</td>
<td>IA4</td>
<td>I</td>
</tr>
<tr>
<td>2-0000004FC</td>
<td>IA4</td>
<td>IND</td>
<td>AA</td>
<td>IA4</td>
<td>J</td>
</tr>
<tr>
<td>7-000000004</td>
<td>IA4</td>
<td>NOBST</td>
<td>AA</td>
<td>IA4</td>
<td>N</td>
</tr>
<tr>
<td>2-0000004FB</td>
<td>RA4</td>
<td>VMAX2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### ARRAYS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-000000020</td>
<td>RA4</td>
<td>ALPHA</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>3-000000050</td>
<td>RA4</td>
<td>DIST</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-000000130</td>
<td>IA4</td>
<td>FTABLE</td>
<td>400</td>
<td>(10, 10)</td>
</tr>
<tr>
<td>2-0000000DB</td>
<td>RA4</td>
<td>HMULT</td>
<td>64</td>
<td>(4, 4)</td>
</tr>
<tr>
<td>4-000000000</td>
<td>RA4</td>
<td>IDENT</td>
<td>64</td>
<td>(4, 4)</td>
</tr>
<tr>
<td>2-000000060</td>
<td>RA4</td>
<td>JAC</td>
<td>120</td>
<td>(6, 5)</td>
</tr>
<tr>
<td>3-000000010</td>
<td>IA4</td>
<td>JTYPE</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>3-000000030</td>
<td>RA4</td>
<td>LHE</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>6-000000000</td>
<td>IA4</td>
<td>LINDDF</td>
<td>80</td>
<td>(2, 10)</td>
</tr>
<tr>
<td>3-000000000</td>
<td>IA4</td>
<td>LINK</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-0000002C0</td>
<td>RA4</td>
<td>MAT</td>
<td>400</td>
<td>(20, 5)</td>
</tr>
<tr>
<td>2-0000004A0</td>
<td>RA4</td>
<td>MAT1</td>
<td>80</td>
<td>(4, 5)</td>
</tr>
<tr>
<td>2-000000010</td>
<td>RA4</td>
<td>ORVEL</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>2-000000128</td>
<td>RA4</td>
<td>PINV</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>2-000000450</td>
<td>RA4</td>
<td>POINT</td>
<td>16</td>
<td>(2, 2)</td>
</tr>
<tr>
<td>AP-00000008B</td>
<td>RA4</td>
<td>THDUT1</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>AP-00000000B</td>
<td>RA4</td>
<td>THDUT2</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>3-000000040</td>
<td>RA4</td>
<td>THETTA</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-000000000</td>
<td>RA4</td>
<td>THETOT</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-000000118</td>
<td>RA4</td>
<td>TINV</td>
<td>16</td>
<td>(2, 2)</td>
</tr>
<tr>
<td>2-000000020</td>
<td>RA4</td>
<td>TWET</td>
<td>64</td>
<td>(4, 4)</td>
</tr>
<tr>
<td>2-000000470</td>
<td>RA4</td>
<td>TPOINT</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>2-000000460</td>
<td>RA4</td>
<td>VECT</td>
<td>16</td>
<td>(2, 2)</td>
</tr>
<tr>
<td>2-000000478</td>
<td>RA4</td>
<td>VLIST</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>2-000000480</td>
<td>IA4</td>
<td>VLIST</td>
<td>32</td>
<td>(4, 2)</td>
</tr>
<tr>
<td>2-0000001B8</td>
<td>RA4</td>
<td>VPOINT</td>
<td>8</td>
<td>(2)</td>
</tr>
</tbody>
</table>

### LABELS

<table>
<thead>
<tr>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>1</td>
<td>AA</td>
<td>85</td>
<td>AA</td>
<td>86</td>
<td>AA</td>
<td>88</td>
<td>AA</td>
<td>150</td>
</tr>
<tr>
<td>AA</td>
<td>156</td>
<td>AA</td>
<td>200</td>
<td>AA</td>
<td>250</td>
<td>AA</td>
<td>266</td>
<td>AA</td>
<td>267</td>
</tr>
<tr>
<td>AA</td>
<td>280</td>
<td>AA</td>
<td>312</td>
<td>AA</td>
<td>375</td>
<td>AA</td>
<td>380</td>
<td>AA</td>
<td>900</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>155</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>275</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>901</td>
</tr>
</tbody>
</table>
FUNCTIONS AND SUBROUTINES REFERENCED

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CVECT</td>
<td></td>
<td>GMULT</td>
<td></td>
<td>JACOB</td>
<td></td>
<td>SOLVE2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
subroutine cost(values, vmin, vmax, thesav, last, 
lul, lu2, lu3, lu4, cost)

Evaluate the cost functional by summing the cost components at each 
discrete position along the trajectory.

Parameters:
values - the joint values along the trajectory
vmin, vmax - the minimum and maximum joint limits
thesav - the joint change corresponding to the maximum
penetration
last - number of joint positions
lul, ..- logic unit numbers for the output files
cost - the value of the cost functional

real values(4,*), vmin(4), vmax(4)
real cost
real jmult(4)
real vlim(4)
real part(4)
real tnorm(4)
real tnorm(4)
common /param/jfact, pfact
common /param3/ afact

cost=0.
do 300 i=1,4
  part(i)=0.
continue
300

do 302 j=1, last
  do 333 k=1,4
    tnorm(k)=0.
  continue
333

do 301 j=1,4

Calculate joint limit factors.
if (values(j,i).gt.vmax(j)) then
  jmult(j)=jfact
  vlim(j)=vmax(j)
elsif (values(j,i).lt.vmin(j)) then
  jmult(j)=jfact
  vlim(j)=vmin(j)
else
  jmult(j)=0.
endif

Calculate joint acceleration factors.
if (i.eq.1) then
  accel=values(j,i+1)-values(j,i)
  accel=3.*values(j,i)-4.*values(j,i+1)+values(j,i+2)
else
  accel=0.
elself (i.eq.2) then
  
  accell=3.*values(j,i-1)+6.*values(j,i)
  accel=values(j,i+1)+values(j,i+2)
  endif

endif

a

asult(j)=afact

temp(1)=.5*pfact*abs(thesav(j,i)**2)
temp(2)=.5*asult(j)**values(j,i)-vlim(j)**2

temp(3)=.5*asult(j)**accel**2

continue

301

continue

302

continue

303

continue

304

continue

305

continue

306

continue

307

continue

308

continue

309

continue

310

continue

311

continue

312

continue

313

continue

314

continue

315

continue

316

continue

317

continue

318

continue

319

continue

320

continue

321

continue

322

continue

323

continue

324

continue

325

continue

326

continue

327

continue

328

continue

329

continue

330

continue

331

continue

332

continue

333

continue

334

continue

335

continue

336

continue

337

continue

338

continue

339

continue

340

continue

341

continue

342

continue

343

continue

344

continue

345

continue

346

continue

347

continue

348

continue

349

continue

350

continue

351

continue

352

continue

353

continue

354

continue

355

continue

356

continue

357

continue

358

continue

359

continue

360

continue

361

continue

362

continue

363

continue

364

continue

365

continue

366

continue

367

continue

368

continue

369

continue

370

continue

371

continue

372

continue

373

continue

374

continue

375

continue

376

continue

377

continue

378

continue

379

continue

380

continue

381

continue

382

continue

383

continue

384

continue

385

continue

386

continue

387

continue

388

continue

389

continue

390

continue

391

continue

392

continue

393

continue

394

continue

395

continue

396

continue

397

continue

398

continue

399

continue

400

continue

401

continue

402

continue

403

continue

404

continue

405

continue

406

return

407

end
### PROGRAM SECTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ACODE</td>
<td>607</td>
<td>PIC CON REL LCL SHR EXE RD NOMRT LONG</td>
</tr>
<tr>
<td>2 @LOCAL</td>
<td>196</td>
<td>PIC CON REL LCL NSWHR NOEXEC RD WRT LONG</td>
</tr>
<tr>
<td>3 PARAM1</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOEXEC RD WRT LONG</td>
</tr>
<tr>
<td>4 PARAM3</td>
<td>4</td>
<td>PIC OVR REL GBL SHR NOEXEC RD WRT LONG</td>
</tr>
</tbody>
</table>

Total Space Allocated: 815

### Entry Points

<table>
<thead>
<tr>
<th>Address Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000</td>
<td>COSTT</td>
</tr>
</tbody>
</table>

### Variables

<table>
<thead>
<tr>
<th>Address Type</th>
<th>Name</th>
<th>Address Type</th>
<th>Name</th>
<th>Address Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>R4</td>
<td>ACCEL</td>
<td>AA</td>
<td>R4</td>
<td>ACCEL</td>
</tr>
<tr>
<td>2-00000070</td>
<td>I4</td>
<td></td>
<td>AA</td>
<td>I4</td>
<td>IND</td>
</tr>
<tr>
<td>AA</td>
<td>I4</td>
<td>K</td>
<td>AA</td>
<td>I4</td>
<td>J</td>
</tr>
<tr>
<td>AP-000000200</td>
<td>I4</td>
<td>LU3</td>
<td>AP-000000240</td>
<td>I4</td>
<td>LU4</td>
</tr>
</tbody>
</table>

### Arrays

<table>
<thead>
<tr>
<th>Address Type</th>
<th>Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-00000030</td>
<td>R4</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00000010</td>
<td>R4</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00000040</td>
<td>R4</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00000050</td>
<td>R4</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-000000000</td>
<td>R4</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>AP-000000100</td>
<td>R4</td>
<td>THESAV</td>
<td>AA</td>
</tr>
<tr>
<td>2-00000060</td>
<td>R4</td>
<td>TNORM</td>
<td>16</td>
</tr>
<tr>
<td>AP-000000040</td>
<td>R4</td>
<td>VALUES</td>
<td>AA</td>
</tr>
<tr>
<td>2-00000020</td>
<td>R4</td>
<td>ULIM</td>
<td>16</td>
</tr>
<tr>
<td>AP-0000000C0</td>
<td>R4</td>
<td>VMAX</td>
<td>16</td>
</tr>
<tr>
<td>AP-000000080</td>
<td>R4</td>
<td>VMIN</td>
<td>16</td>
</tr>
</tbody>
</table>

### Labels

<table>
<thead>
<tr>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>300</td>
<td>AA</td>
<td>301</td>
<td>AA</td>
<td>302</td>
<td>AA</td>
<td>303</td>
<td>AA</td>
<td>333</td>
<td>AA</td>
<td>334</td>
</tr>
<tr>
<td>Type</td>
<td>Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>MTHSQRT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
subroutine cvect(imode,llink,iobst,vlens,vects,points)

Access the database containing the correction vectors for the current and previous link positions.

Parameters:

- imode - 1 - return vectl(k-1),pointl(k-1),vect2(k),point2(k)
- 2 - store ' ' 
- llink - the moving link id
- iobst - the stationary obstacle id
- vlens - lengths of the correction vectors
- vects - correction vectors
- points - correction points

Variables:

- vectl(klast,i,ipoint) - translation vector applied to position k-1 of link 'llink' due to penetration of hull(k-1) by obstacle 'iobst'
- vlens(2),vect2(2),point2(2)
- vl(2),v2(2),pl(2),p2(2)
- vlens(2),vects(2,2),points(2,2)
- kcurr,klast - pointers to the structures for positions (k),(k-1)
- kstart - start pointer for the current position

Initialise.

If (kcurr.eq.0) then
  kcurr=1
  klast=2
endif

For object pair llink,iobst,

return vectl(k-1),pointl(k-1),vect2(k),point2(k).

and the vector lengths.

if (imode.eq.1) then
  ipoint=(llink-1)*nobe+io
  vlens(1)=0.
  vlens(2)=0.
do 20 i=1,2

CVECT

0058     vects(i,1)=vectl(klast,i,ipoint)
0059     points(i,1)=pointl(klast,i,ipoint)
0060     vects(i,2)=vect2(i,ipoint)
0061     points(i,2)=point2(i,ipoint)
0062     vlens(1)=vlens(1)+vects(i,1)**2
0063     vlens(2)=vlens(2)+vects(i,2)**2
0064     continue
0065     vlens(1)=sqrt(vlens(1))
0066     vlens(2)=sqrt(vlens(2))
0067
elseif (imode.eq.2) then
0068     c For link 'number', add vect1,point1,vect2,point2 to database.
0069     c Increment the position.
0070     K>
0071     if (ilink.eq.Land.iobst.aq.l) then
0072     ksave=klast
0073     klast=kcurr
0074     kcurr=ksave
0075     endif
0076     K>
0080     ipoint=ilink-1 nobst+iobst
0081     do 50 i=1,2
0082     vect1(kcurr,i,ipoint)=vects(i,1)
0083     pointl(kcurr,i,ipoint)=points(i,1)
0084     vect2(i,ipoint)=vects(i,2)
0085     point2(i,ipoint)=points(i,2)
0086     continue
0087     endif
0088
0089     return
0090     end
### PROGRAM SECTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 CODE</td>
<td>346</td>
<td>PIC CON REL LCL SHR EXE RD N0MRT LONG</td>
</tr>
<tr>
<td>2 LOCAL</td>
<td>100</td>
<td>PIC CON REL LCL N0SHR NOE0E RD WRT LONG</td>
</tr>
<tr>
<td>3 INTDIS</td>
<td>4800</td>
<td>PIC OVR REL GBL SHR NOE0E RD WRT LONG</td>
</tr>
<tr>
<td>4 NOBJS</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOE0E RD WRT LONG</td>
</tr>
</tbody>
</table>

**Total Space Allocated**: 5254

### ENTRY POINTS

- **Address**: 0-00000000
- **Name**: CVECT

### VARIABLES

**Address** | **Type** | **Name** | **Address** | **Type** | **Name** | **Address** | **Type** | **Name** | **Address** | **Type** | **Name** |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>I4</td>
<td>I</td>
<td>AA</td>
<td>I4</td>
<td>ILINK</td>
<td>AA</td>
<td>I4</td>
<td>IMODE</td>
<td>AA</td>
<td>I4</td>
<td>INODE</td>
</tr>
<tr>
<td>AA</td>
<td>I4</td>
<td>IPOIN T</td>
<td>AA</td>
<td>I4</td>
<td>KCURR</td>
<td>AA</td>
<td>I4</td>
<td>IOBST</td>
<td>AA</td>
<td>I4</td>
<td>KSAVE</td>
</tr>
<tr>
<td>4-00000000</td>
<td>I4</td>
<td>NLINKS</td>
<td>4-00000000</td>
<td>I4</td>
<td>NOBST</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### ARRAYS

**Address** | **Type** | **Name** | **Bytes** | **Dimensions** |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2-00000010</td>
<td>R4</td>
<td>P1</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>2-00000018</td>
<td>R4</td>
<td>P2</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>3-00000960</td>
<td>R4</td>
<td>POINT1</td>
<td>1600</td>
<td>(2, 2, 100)</td>
</tr>
<tr>
<td>3-00000968</td>
<td>R4</td>
<td>POINT2</td>
<td>800</td>
<td>(3, 100)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>POINTS</td>
<td>16</td>
<td>(2, 2)</td>
</tr>
<tr>
<td>2-00000000</td>
<td>R4</td>
<td>V1</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>2-00000008</td>
<td>R4</td>
<td>V2</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>3-00000060</td>
<td>R4</td>
<td>VECT1</td>
<td>1600</td>
<td>(2, 2, 100)</td>
</tr>
<tr>
<td>3-00000640</td>
<td>R4</td>
<td>VECT2</td>
<td>800</td>
<td>(2, 100)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VECTS</td>
<td>16</td>
<td>(2, 2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VLENS</td>
<td>8</td>
<td>(2)</td>
</tr>
</tbody>
</table>

### LABELS

**Address** | **Label** | **Address** | **Label** |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>20</td>
<td>AA</td>
<td>50</td>
</tr>
</tbody>
</table>
FUNCTIONS AND SUBROUTINES REFERENCED

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>R4</td>
<td>MTH$SQR</td>
</tr>
</tbody>
</table>

COMMAND QUALIFIERS

FOR/LIST OPT.FOR, RDFILE.FOR, INPUT.FOR, GROW.FOR, FNPAT.FOR, SOLVE1.FOR, INT2D.FOR, HULL2D.FOR, GMULT.FOR, JACOB.FOR, COSTT.FOR,

/ CHECK = (NOBOUNDS, OVERFLOW, NOUNDERFLOW)
/ DEBUG = (NOSYMBOLS, TRACEBACK)
/ STANDARD = (NOSYNTAX, NOSOURCE FORM)
/ SHOW = (NOPREPROCESSOR, NOINCLUDE, MAP, NODICTIONARY, SINGLE)
/ WARNINGS = (GENERAL, NODECLARATIONS)
/ CONTINUATIONS = 19 / NOCROSS_REFERENCE / MOD_LINES / NOEXTEND_SOURCE / F77
/ NOG_FLOATING / I4 / NOMACHINE_CODE / OPTIMIZE

COMPILATION STATISTICS

Run Time: 3.30 seconds
Elapsed Time: 9.92 seconds
Page Faults: 589
Dynamic Memory: 845 pages
program drinv

Draw the trajectory of joint positions generated by the iterative
inverse kinematic solution.

character*20 file(4)
real obst(2,4)
real boxl(2,10,10)
integer links(4)
integer jtype(4),link(4)
real alpha(4),len(4),theta(4),dist(4)
common /robot/link,jtype,alpha,len,theta,dist
common /nrdof/ndof,nxdof

Read the kinematic parameters.
call rdfile
c Input the joint angle file names.
do 111 i=1,ndof
   write(*,*) 'Enter joint file name'
   read(*,*) filed)
continue

c Initialize the workstation geometry.
open(1, file='box.dat', status='old')
read(1,*) nlinks
do 29 k=1,nlinks
   read(1,*) nvert,links(k)
do 13  i=1,nvert
      read(1,*) (boxl(j,i,k),j=1,2)
continue

close(1)
do 112 i=1,ndof
   open(10+i, file=file(i), status='old')
continue

call qrstrt(4006,1)
call newpag
call window(-4.,4.,-4.,4.)
call vviewport(0.,100.,0.,100.)
do 5 ind=1,1000
   do 113 i=1,ndof
      read(10+i,*,end=100) theta(i)
continu

call tridnt(.false.)
link=1
do 10 ind=1,ndof
angle=theta(ind)
call rotate(angle,angle)
call transl(len(ind),0.)
call move(0.,0.)
if (ind.gt.l) call dashpt(1)
call draw(-len(ind),0.)
if (ind.eq.l) call arc(.05,0.,360.)
call dashpt(0)
if (ind.eq.ndof) then
call move(.2,.1)
call draw(0.,.1)
call draw(0.,-.1)
call draw(.2,-.1)
endif
continue
5 continue
100 continue
call tridnt(.false.)
do 11 ind=1,ndof
angle=theta(ind)
call rotate(angle,angle)
call transl(len(ind),0.)
call move(0.,0.)
call draw(-len(ind),0.)
call arc(.05,0.,360.)
call dashpt(0)
11 continue
continue
call grstop
stop
end
PROGRAM SECTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 @CODE</td>
<td>729</td>
<td>PIC CON REL LCL SHR EXE RD NONRT LONG</td>
</tr>
<tr>
<td>1 @DATA</td>
<td>81</td>
<td>PIC CON REL LCL SHR NOEXE RD NONRT LONG</td>
</tr>
<tr>
<td>2 @LOCAL</td>
<td>1248</td>
<td>PIC CON REL LCL NOSH SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>3 ROBOT</td>
<td>96</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>4 NRDOF</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
</tbody>
</table>

Total Space Allocated 2162

ENTRY POINTS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000</td>
<td>DRINV</td>
<td></td>
</tr>
</tbody>
</table>

VARIABLES

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-000003B4</td>
<td>R4</td>
<td>ANGLE</td>
<td>2-000003AO</td>
<td>I4</td>
<td>NLINKS</td>
<td>2-000003A4</td>
<td>I4</td>
<td>MVERT</td>
</tr>
<tr>
<td>2-000003AO</td>
<td>I4</td>
<td>IND1</td>
<td>2-000003AO</td>
<td>I4</td>
<td>NLINKS</td>
<td>2-000003A4</td>
<td>I4</td>
<td>MVERT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2-000003A4</td>
<td>I4</td>
<td>NVERT</td>
<td>4-00000004</td>
<td>I4</td>
<td>NXDOF</td>
</tr>
</tbody>
</table>

ARRAYS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-00000020</td>
<td>R4</td>
<td>ALPHA</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00000020</td>
<td>R4</td>
<td>BOX1</td>
<td>800</td>
<td>(2, 10, 10)</td>
</tr>
<tr>
<td>3-00000050</td>
<td>R4</td>
<td>DIST</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-000000350</td>
<td>CHAR FILE</td>
<td>80</td>
<td>(4)</td>
<td></td>
</tr>
<tr>
<td>3-00000010</td>
<td>I4</td>
<td>JTYPE</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>3-00000030</td>
<td>R4</td>
<td>LEN</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>3-00000000</td>
<td>I4</td>
<td>LINK</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-000000340</td>
<td>I4</td>
<td>LINKS</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00000000</td>
<td>R4</td>
<td>OBST</td>
<td>32</td>
<td>(2, 4)</td>
</tr>
<tr>
<td>3-00000040</td>
<td>R4</td>
<td>THETA</td>
<td>16</td>
<td>(4)</td>
</tr>
</tbody>
</table>

LABELS

<table>
<thead>
<tr>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td></td>
<td>10</td>
<td></td>
<td>11</td>
<td></td>
<td>13</td>
<td></td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>111</td>
<td></td>
<td>112</td>
<td></td>
<td>113</td>
<td></td>
<td></td>
<td>0-0000025F</td>
<td>100</td>
</tr>
<tr>
<td>Type Name</td>
<td>Type Name</td>
<td>Type Name</td>
<td>Type Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARC</td>
<td>DASHPT</td>
<td>DRAW</td>
<td>GRSTOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRIDNT</td>
<td>VWPORT</td>
<td>WINDOW</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FORCLOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FOROPEN</td>
<td>ROTATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHSTOP</td>
<td>TRANSPI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
program drllnk

      c Draw the discrete joint trajectory generated by OPT.
      character*20 file(4)
      real boxx(2,10,10),boxy(2,10,10)
      integer links(4)
      integer jtype(4),link(4)
      real alpha(4),len(4),theta(4),dist(4).
      common /robot/link,jtype,alpha,len,theta,dist
      common /nrdof/ndof,nxdo

      c Read the kinematic parameters from a file.
      call rdfile

      c Enter the names of the joint angle files.
      do 111 i=1,ndof
         write(*,'(a,a)') 'Enter joint',i,' file name'
         read(*,*) file(i)
      111 continue

      c Initialize the geometry of the workstation.
      open(1,file='box.dat',status='old')
      read(1,*),nlinks
      do 29 k=1,nlinks
         read(1,*),nvert,links(k)
         do 13 i=1,nvert
            read(1,*), (boxx(j,i,k),j=1,2)
         13 continue
      29 continue
      read(1,*),nobst
      do 19 k=1,nobst
         read(1,*),nvert
         do 14 i=1,nvert
            read(1,*), (boxy(j,i,k),j=1,2)
         14 continue
      19 continue
      close(1)
      do 112 i=1,ndof
         open(10+i,file=file(i),status='old')
      112 continue
      call vstart(4006,1)
      call newpag
      call window(-4.,4.,-4.,4.,)
      call vviewport(0.,100.,0.,100.,)
      do 22 k=1,nobst
         call move(boxx(1,1,k),boxy(2,1,k))
         call draw(boxx(1,2,k),boxy(2,2,k))
         call draw(boxx(1,3,k),boxy(2,3,k))
         call draw(boxx(1,4,k),boxy(2,4,k))
call draw(box2(1,1,k),box2(2,1,k))
continue

do 5 indl=1,1000
    do 113 i=1,ndof
        read(10+i,*,end=100) theta(i)
    continue
    call tridnt(.false.)
    ilink=1
    do 10 ind=1,ndof
        angle=theta(ind)
        call rotate(angle,angle)
        call transl(len(ind),0.)
        call move(0.,0.)
        call dashpt(1)
        if (ind.eq.links(ilink)) then
            call move(-len(ind),0.)
        else
            call draw(-len(ind),0.)
        endif
        call dashpt(0)
    endif
    call dashpt(0)
    if (ind.eq.links(ilink)) then
        call move(box1(1,1,ilink),box1(2,1,ilink))
        call draw(box1(1,2,ilink),box1(2,2,ilink))
        call draw(box1(1,3,ilink),box1(2,3,ilink))
        call draw(box1(1,4,ilink),box1(2,4,ilink))
        call draw(box1(1,1,ilink),box1(2,1,ilink))
        ilink=ilink+1
    endif
5 continue
10 continue

if (ind.eq.links(ilink)) then
    call move(box1(1,1,ilink),box1(2,1,ilink))
    call draw(box1(1,2,ilink),box1(2,2,ilink))
    call draw(box1(1,3,ilink),box1(2,3,ilink))
    call draw(box1(1,4,ilink),box1(2,4,ilink))
    call draw(box1(1,1,ilink),box1(2,1,ilink))
else
    call draw(-len(ind),0.)
endif

!end

continue
### PROGRAM SECTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CODE</td>
<td>1001</td>
<td>PIC CON REL LCL SHR EXE RD NOWRT LONG</td>
</tr>
<tr>
<td>$PDRTA</td>
<td>61</td>
<td>PIC CON REL LCL SHR NOEXE RD NOWRT LONG</td>
</tr>
<tr>
<td>$LOCAL</td>
<td>1964</td>
<td>PIC CON REL LCL NO SHR NO EXE RD WRT LONG</td>
</tr>
<tr>
<td>ROBOT</td>
<td>96</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>NRDOF</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
</tbody>
</table>

Total Space Allocated: 3130

### ENTRY POINTS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000</td>
<td></td>
<td>DRLINK</td>
</tr>
</tbody>
</table>

### VARIABLES

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-000006B8</td>
<td>R4</td>
<td>ANGLE</td>
</tr>
<tr>
<td>2-000006A0</td>
<td>I4</td>
<td>NLINKS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000</td>
<td></td>
<td>DRLINK</td>
</tr>
</tbody>
</table>

### ARRAYS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-00000020</td>
<td>R4</td>
<td>ALPHA</td>
</tr>
<tr>
<td>2-000000320</td>
<td>R4</td>
<td>BOX1</td>
</tr>
<tr>
<td>3-000000050</td>
<td>R4</td>
<td>DIST</td>
</tr>
<tr>
<td>2-00000650</td>
<td>CHAR</td>
<td>FILE</td>
</tr>
<tr>
<td>3-00000030</td>
<td>R4</td>
<td>LEN</td>
</tr>
<tr>
<td>3-00000000</td>
<td>I4</td>
<td>LINK</td>
</tr>
<tr>
<td>3-00000040</td>
<td>R4</td>
<td>THETA</td>
</tr>
</tbody>
</table>

### LABELS

<table>
<thead>
<tr>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>5</td>
<td>**</td>
<td>10</td>
<td>**</td>
<td>13</td>
<td>**</td>
<td>14</td>
</tr>
<tr>
<td>**</td>
<td>29</td>
<td>0-0000003DB</td>
<td>100</td>
<td>**</td>
<td>111</td>
<td>**</td>
<td>112</td>
</tr>
</tbody>
</table>

Note: The ** symbol indicates a label that is not directly referenced in the code but is part of the label table.
<table>
<thead>
<tr>
<th>Type Name</th>
<th>Type Name</th>
<th>Type Name</th>
<th>Type Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DASHPT</td>
<td>DRAW</td>
<td>FOR$CLOSE</td>
<td>GRSTOP</td>
</tr>
<tr>
<td>MOVE</td>
<td>NEHPAC</td>
<td>FOR$OPEN</td>
<td>GRSTRT</td>
</tr>
<tr>
<td>VHPORT</td>
<td>WINDOW</td>
<td>ROTATE</td>
<td>TRIDNT</td>
</tr>
</tbody>
</table>
subroutine finpat(values)
    c This subroutine numerically optimizes a manipulator trajectory.
    c The initial configuration is given, and the final configuration
    c (the goal position) must satisfy the final position constraint.
    c The final position is moved over the
    c constraint surface to optimize the functional.
    c The cost functional is the integral of several cost terms over the path.
    c The cost terms are of quadratic form and include:
    c  1. joint accelerations
    c  2. joint limit violations
    c  3. interference (penetration) approximated by joint changes
    c The cost functional is minimized by iteratively modifying the
    c trajectory along the gradient of the cost functional using a
    c quasi-Newton iteration series.
    c The convex hull enclosing two adjacent link positions is used
    c for the interference calculations.
    Parameters:
    values- an array of joint values defining the trajectory.
    On entry, contains the starting trajectory.
    On exit, contains the optimized trajectory.
    The following output test files are generated:
    fmax.dat- largest joint violations for each iteration
    dmax.dat- largest penetrations for each iteration
    last.dat- number of positions in trajectory for each iteration
    dist.dat- largest penetrations along trajectory
    cost1.dat- total value of cost functional
    cost2.dat- value of interference component
    cost3.dat- value of joint limit component
    cost4.dat- value of joint acceleration component
    angl.dat- file of Ith joint angles
    real len(4),dist(4),alpha(4),theta(4),thetal(4),theta2(4)
    integer jtype(4),link(4)
    real xgoal(3)
    real mat(4,5),matl(4,3)
    real tnet(4,4)
    real values(4,100)
    real nvalue(4,100)
    real dell(4),dlast(4)
    real cost(1000)
    real thedot(4)
    real thetot(4)
    real delx(2)
    real temp(4)
    real homog(4,4)
    real vmin(4),vmax(4)
    real vlin(4)
    real jmult(4)
c Initialize parameters.

d2rad=.0174533

toler=.01

toler1=.0001

cmin=4.

cstop=.false.

c Copy the initial and final configuration joint values.

do 222 i=1,ndof

    theta1(1)=values(i,1)
    theta2(1)=values(i,1)

222 continue

Open some files for storing results.

Iteratively improve the trajectory until the cost functional is minimized. Initially a very coarse trajectory with large steps between adjacent positions is used. The trajectory is progressively subdivided with each iteration until the step size linearization error is within a preselected tolerance.

```
0128 c Iteratively improve the trajectory until the cost functional is minimized. Initially a very coarse trajectory with large steps between adjacent positions is used. The trajectory is progressively subdivided with each iteration until the step size linearization error is within a preselected tolerance.
0129 c
0130 c between adjacent positions is used. The trajectory is progressively subdivided with each iteration until the step size linearization error is within a preselected tolerance.
0131 c subdivided with each iteration until the step size linearization error is within a preselected tolerance.
0132 c error is within a preselected tolerance.
0133 0134 indl=0
0135 115 continue
0136 0137 if (indl.ge.nmax.or.stop) then
0138         goto 100
0139 else
0140         indl=indl+1
0141 0142 0143 0144 0145 c Iterate along the trajectory, updating the joint values at each position using a quasi-Newton procedure to reduce the cost functional.
0146 c
0147 0148 0149 0150 c Transform the initial manipulator link positions. Transform the data base, then load the array of vertices.
0151 c
0152 0153 0154 0155 0156 0157 0158 0159 0160 0161 0162 0163 0164 0165 0166 0167 0168 0169 0170 0171
0172 modify the intermediate trajectory points.
0173 nadd=0
0174 nsub=0
0175 dmax=0.
0176 fmax=0.
0177 istart=2
0178```

Iteratively improve the trajectory until the cost functional is minimized. Initially a very coarse trajectory with large steps between adjacent positions is used. The trajectory is progressively subdivided with each iteration until the step size linearization error is within a preselected tolerance.
FIST  AT 13-May-1987 12:53:32  VAX FORTRAN V4.4-177

iend=last+1

do 200 ind2=istart,iend
   inflag(ind2)=.false.
   iprev=ind2-1
   indist(iprev)=0.
   do 201 i=1,ndof
      theta(i)=values(i,ind2)
      continue
201
   c
   Transform the link vertices in world coordinates.

   c
   Specify the link positions to be tested against the obstacles.

   c
   Calculate the convex hulls representing the swept volume
   between sequential positions.

   c
   If the two positions are identical, then the hull is just
   box1.

   if (ind2.eq.last+1) then
      nvert=nvlink
      do 155 j=1,nvert
         do 156 i=1,2
            box1(j,i)=box(j,i,ilink)
            box2(j,i,ilink)=lbox(j,i)
            continue
561
      continue
560
   else
      call hull2d(box1,lbox,nvlink,hull,nvlink,hull,nvert,iverts)
   endif
   Perform interference tests between the hulls and the
   obstacles.

   do 181 iobst=1,nobst
      call gverts(1,2,iobst,obox,nvobst)
call interf(hull, obx, nvert, nvoxt, ilink, ibst, 
  iverts, intf1g)

inflag(ind2)=inflag(ind2).or.intflg

continue

If a trajectory position results in interference with the 
swepjt volume, modify the trajectory to avoid the interference.

if (inflag(ind2).or.inflag(iprev)) then
  do 209 i=1,nvof
      theta(i)=values(i,iprev)
  continue
  call corectlthedot,thdotl,indi»t(lprev))
  if (indist(lprev).gt.dmax) then 
     dmax=indist(lprev)
  endif
  do 312 i=1,nvof
      thesav(i,lprev)=thedot(i)
      pmult(i)=pfact
  continue
  else
  do 313 i=1,nvof
      thesav(i,lprev)=0.
      pmult(i)=0.
  continue
  endif

Calculate the joint limit penalty function coefficients.

d0 119 i=1,nvof
  if (values(i,iprev).gt.vmax(i)) then
      ftemp=abs(values(i,iprev)-vmax(i))
      if (ftemp.gt.fmax) fmax=ftemp 
      jmult(i)=jfact
      vlim(i)=vmax(i)
  elseif (values(i,iprev).lt.vmin(i)) then
      ftemp=abs(values(i,iprev)-vmin(i))
      if (ftemp.gt.fmax) fmax=ftemp 
      jmult(i)=jfact
      vlim(i)=vmin(i)
  else
      jmult(i)=0.
  endif
  continue

Calculate the acceleration minimization weighting matrix.
Update the joint variables, using a quasi-Newton search direction. The step size is variable.

c Update the joint variables, using a quasi-Newton search direction. The step size is variable.

c Calculate the elements of the Hessian matrix. In this case, the Hessian matrix is a linear combination of the diagonal weighting matrices, and its inverse is found by inverting the diagonal elements.

denom=amult(1)+pmult(1)+amult(i)*6.

denoml=amult(1)+pmult(1)+amult(i)*3.

c Calculate the acceleration term for different cases.

c Calculate and save the norm of the acceleration. This is necessary only if the accelerations are to be saved in a file.

c Calculate and save the norm of the acceleration. This is necessary only if the accelerations are to be saved in a file.

Calculate the quasi-Newton search direction of the cost functional.

if (iprev.gt.1.and.iprev.lt.last) then
  if (denom.eq.0.) then
    dlast(i)=0.
  else
    if (iprev.eq.2) then
      dlast(i)=(values(i,iprev)-values(i,iprev-1))-
      (values(i,iprev)+values(i,iprev+1))/denom
    elseif (iprev.eq.last-1) then
      dlast(i)=(values(i,iprev)-values(i,iprev-2))-
      (values(i,iprev-1)+values(i,iprev+1))/denom
    else
      dlast(i)=(values(i,iprev)-values(i,iprev-1))-
      (values(i,iprev-2)+values(i,iprev+1))/denom
    endif
  endif
endif
elseif (iprev.eq.last) then
  if (denom.eq.0.) then
    dlast(i)=0.
  else
    dlast(i)=(values(i,iprev)-values(i,iprev-1))-
    (values(i,iprev-2)+values(i,iprev+1))/denom
  endif
endif
endif
elseif (iprev.eq.last) then
  if (denom.eq.0.) then
    dlast(i)=0.
  else
    dlast(i)=(values(i,iprev)-values(i,iprev-1))-
    (values(i,iprev-2)+values(i,iprev+1))/denom
  endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
dmult=tau

For the last position, project the desired joint change onto
the null space of J so that the tool coordinates are not
changed.

if (iprev.eq.last) then
  
calculate the jacobian for the end configuration.
  
call jacob(ident,ident,len,alpha,dist,theta2,jtype,
           ndof,tnet,jac)

  do 25 i=1,ndof
      mat(1,i)=jac(4,i)
      mat(2,i)=jac(5,i)
  25 continue

do 411 i=1,2
  xlast(i)=tnet(i,4)
  xact(i,1,1)=xlast(i)
  411 continue

  find the homogeneous part of the inverse solution.
  
call solvel(mat,ndof,ndof,2,4,homog)
  
call gmult(homog,dlast,dell,ndof,ndof,1,4,4,4)

  do 419 i=1,ndof
      dangle(i)=dmult*dell(i)*ad2rad
  419 continue

  calculate the linearised estimate of the updated gripper position.
  
call gmult(mat1,dangle,delx1,2,ndof,1,4,4,2)

  do 415 i=1,2
      xlast(i)=xlast(i)+delx1(i)
  415 continue

  calculate the updated joint values and the corresponding gripper
  position.
  
do 412 i=1,ndof
      theta3(i)=theta2(i)+dmult*dell(i)
  412 continue

  call jacob(ident,ident,len,alpha,dist,theta3,jtype,ndof,tnet,jac)

  calculate the linearised formula error, and scale the joint
  change.
  
do 413 i=1,2
      delx1(i)=tnet(i,4)-xlast(i)

```
0457  413       continue
0458  0459       dtempl=sqrt(delxl(1)**2+delxl(2)**2)
0460  0461       if (dtempl.gt.stepsi) then
0462  0463       endif
0464  0465       c       Update the final configuration.
0466  0467       do 160 i=1,ndof
0468  0469       ndif=nvalue(i,last)+dmult*dell(i)
0470  0471       theta2(i)=nvalue(i,last)
0472  0473       continue
0474       continue
0475  0476       Calculate the change in the tool position due to linearization error.
0477  0478       call jacob(ident,ident,len,alpha,dist,theta2,jtype,
0479       nread,ntab,jac)
0480       delen=0.
0481       do 111 i=1,2
0482  0483       delx(i)=ntab(i,4)-zgoal(i)
0484  0485       delen=delen+delx(i)**2
0486  0487       continue
0488       delen=sqrt(delen)
0489  0490       if (delen.lt.toler) then
0491  0492       goto 105
0493  0494       else
0495       c       Remove the error.
0496       do 12 i=1,ndof
0497  0498       mat(i,1)=jac(4,i)
0499  0500       mat(2,1)=jac(5,1)
0501       continue
0502       do 11 i=1,2
0503       mat(i,ndof+1)=delx(i)
0504  0505       continue
0506       call solv2(2,mat,4,4,nread,ndof,homog,thedot)
0507       nvalue(i,last)=nvalue(i,last)-thedot(i)/d2rad
0508       theta2(i)=nvalue(i,last)
0509       continue
0510       continue
0511       endif
0512       goto 105
0513  0514       continue
```
else
  
  c Estimate the local linearity of the Jacobian, and determine a suitable joint step size.
  
  do 310 i=1,ndof
    theta3(i)=values(i,iprev)
    theta4(i)=theta3(i)+dmult*dlast(1)
    dangle(i)=dmult*dlast(1)*d2rad
  continue
  
  c Calculate the Jacobian at the current configuration.
  
  call jacob(ident,ident,len,alpha,dist,theta3,jtype,ndof,tnet,jac)
  
  do 315 i=1,2
    xlast(1)=tnet(1,4)
    xact(i,iprev)=xlast(1)
  continue
  
  do 311 1=1,2
    do 317 j=1,ndof
      mat(1,j)=jac(i+3,j)
    continue
    continue
  
  c Calculate the change in the gripper position based on a linear Jacobian and a desired joint change.
  
  call gmult(mat,dangle,delz1,2,ndof,1,4,4,2)
  
  c Calculate the trajectory step size error.
  
  do 331 i=1,ndof
    dangled(i)=(values(i,iprev+1)-values(i,iprev))*d2rad
  continue
  
  do 332 i=1,2
    xest(i,iprev+1)=xlast(i)+delz2(i)
  continue
  
  c Calculate the Jacobian for the updated joint configuration, and find the actual new gripper position.
  
  a measure of the non-linearity of the Jacobian, or the formula error of the linear approximation, is given by the ratio of the difference between the linearized and exact gripper position, and the magnitude of the linearized position change. The step size is chosen to keep this ratio below some specified tolerance.
  
  call jacob(ident,ident,len,alpha,dist,theta4,jtype,ndof,tnet,jac)
  
  do 316 i=1,2
    xlast(i)=xlast(i)+delt1(i)
    xcurr(i)=tnet(i,4)
  
  call jacob(ident,ident,len,alpha,dist,theta4,jtype,ndof,tnet,jac)
delx2(1) = xcurr(1) - xlast(1)
continue

dtemp2 = sqrt(delx2(1)**2 + delx2(2)**2)

If the formula error exceeds the threshold, scale the step size.
if (dtemp2.gt.steps!) then
dmult = dmult * steps1 / dtemp2
endif

c
Update the joint angles.
if (iprev.ne.1) then
do 318 1 = 1, ndof
   nvalue!(i, iprev) = values!(i, iprev) + dmult * dlast(i)
318 continue
endif
endif

c
Update the trajectory.
do 444 i = 2, last
   do 446 j = 1, ndof
      values!(j, i) = nvalue!(j, i)
446 continue
444 continue

c
Calculate the linearization error for the discrete approximation to
the trajectory.
do 101 ind4 = 2, last
   error = 0.
   do 271 i = 1, 2
      error = error + (xest(i, ind4) - xact(i, ind4))**2
271 continue
   error = sqrt(error)
101 continue

c
This section is necessary only if vnorm is to be saved.
vnorm = 0.
   do 277 i = 1, ndof
      veloc = (values(i, ind4) - values(i, ind4-1))
      vnorm = vnorm + veloc**2
277 continue
   vnorm = sqrt(vnorm)
101 continue

c
If the linearization error is greater than the tolerance,
insert an interpolated position to reduce the step size.
Conversely, if the linearization error is very small, a
position can be removed to increase the step size.
if (error.gt.cmax) then
nadd=nadd+1
adlist(nadd)=ind4-1
elseif (ind4.ne.1.and.(error+errorprev).lt.cmax/cmmin) then
nsub=nsub+1
sublist(nsub)=ind4
endif

eprev=error
101 continue

c Integrate cost.
call cost(values,vain,vmax,thesav,last,
11,12,13,14,cost(indl))
write(5,
A) last

c Insert or remove discrete positions as previously determined.
if (nadd.gt.0.or.nsub.gt.0) then
write(A,
A) 'new positions added at iteration ',indl
write(A,
A) 'adlist ',(adlist(i),i-1,nadd)
write(A,
A) 'sublist ',(sublist(i),i-1,nsub)
callinsert(nadd,nsub,values)
endif

Stop if the penetration and joint violation tolerances are met and
if the position step size error is within tolerance.
if (maxmaxlt.maxpen.and.fmax.lt.jtoler.and.nadd.eq.0) then
stop .true.
write(A,
A) 'AAAA Interference and joint limits satisfied AAAAA'
endif
write(3,
A) maxmax-maxpen
write(1,
A) fmax-jtoler
write(A,
A) 'last,cost,dmax,fmax ',indl,last,cost(indl),maxmax,fmax
write(A,
A)
goto 115
100 continue
do 500 i=1,last
write(20,
A) values(i,1)
write(21,
A) values(i,2)
write(22,
A) values(i,3)
write(23,
A) values(i,4)
write(7,
A) indist(i)-maxpen
500 continue

endif

endf

100 continue

do 500 i=1,last
write(20,
A) values(i,1)
write(21,
A) values(i,2)
write(22,
A) values(i,3)
write(23,
A) values(i,4)
write(7,
A) indist(i)-maxpen
500 continue

endf

endf

endf
0685 close(2)
0686 close(3)
0687 close(5)
0688 close(6)
0689 close(7)
0690 close(8)
0691 close(9)
0692 close(10)
0693 close(11)
0694 close(12)
0695 close(13)
0696 close(14)
0697 close(20)
0698 close(21)
0699 close(22)
0700 close(23)
0701 return
0702 end

PROGRAM SECTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $CODE</td>
<td>3768</td>
<td>PIC CON REL LCL SHR EXE RD NOMRT LONG</td>
</tr>
<tr>
<td>1 $DATA</td>
<td>252</td>
<td>PIC CON REL LCL SHR NOEXE RD NOMRT LONG</td>
</tr>
<tr>
<td>2 $LOCAL</td>
<td>16140</td>
<td>PIC CON REL LCL NOMSR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>3 ROBOT</td>
<td>96</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>4 PARM1</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>5 PARM3</td>
<td>4</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>6 NMAX</td>
<td>4</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>7 TAU</td>
<td>4</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>8 GOAL</td>
<td>16</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>9 IDENT</td>
<td>64</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>10 LAST</td>
<td>4</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>11 LIMITS</td>
<td>32</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>12 NOBS</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>13 CMAX</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>14 STEPSI</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>15 LIMTOL</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>16 NDOF</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
</tbody>
</table>

Total Space Allocated 20424

ENTRY POINTS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000</td>
<td>FINPAT</td>
<td></td>
</tr>
</tbody>
</table>
### VARIABLES

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-000000000</td>
<td>R4</td>
<td>APACT</td>
<td>8-00000000C</td>
<td>R4</td>
<td>ANGOAL</td>
<td>AA</td>
<td>R4</td>
<td>ATEMP</td>
</tr>
<tr>
<td>AA</td>
<td>R4</td>
<td>CHIN</td>
<td>AA</td>
<td>R4</td>
<td>D2RAD</td>
<td>AA</td>
<td>R4</td>
<td>DELEN</td>
</tr>
<tr>
<td>AA</td>
<td>R4</td>
<td>DENOM1</td>
<td>AA</td>
<td>R4</td>
<td>DMAX</td>
<td>AA</td>
<td>R4</td>
<td>DMULT</td>
</tr>
<tr>
<td>AA</td>
<td>R4</td>
<td>DTEMF1</td>
<td>AA</td>
<td>R4</td>
<td>EREV</td>
<td>AA</td>
<td>R4</td>
<td>ERROR</td>
</tr>
<tr>
<td>AA</td>
<td>R4</td>
<td>DTEMP</td>
<td>AA</td>
<td>R4</td>
<td>JEND</td>
<td>AA</td>
<td>R4</td>
<td>FMAX</td>
</tr>
<tr>
<td>2-00003A2C</td>
<td>R4</td>
<td>IND1</td>
<td>AA</td>
<td>R4</td>
<td>IND2</td>
<td>AA</td>
<td>R4</td>
<td>IND4</td>
</tr>
<tr>
<td>2-00003A60</td>
<td>R4</td>
<td>IOBST</td>
<td>AA</td>
<td>R4</td>
<td>IPREV</td>
<td>AA</td>
<td>R4</td>
<td>ISTART</td>
</tr>
<tr>
<td>4-00000000</td>
<td>R4</td>
<td>JFACT</td>
<td>AA</td>
<td>R4</td>
<td>JTEMP</td>
<td>AA</td>
<td>R4</td>
<td>J</td>
</tr>
<tr>
<td>2-00003A10</td>
<td>R4</td>
<td>NADD</td>
<td>AA</td>
<td>R4</td>
<td>NDOF</td>
<td>AA</td>
<td>R4</td>
<td>NLINKS</td>
</tr>
<tr>
<td>12-00003A04</td>
<td>R4</td>
<td>NOBST</td>
<td>AA</td>
<td>R4</td>
<td>NSUB</td>
<td>AA</td>
<td>R4</td>
<td>NVERT</td>
</tr>
<tr>
<td>2-00003A6B</td>
<td>R4</td>
<td>NVORST</td>
<td>16-00000004</td>
<td>R4</td>
<td>NXDOF</td>
<td>4-00000004</td>
<td>R4</td>
<td>PFACT</td>
</tr>
<tr>
<td>2-00003A24</td>
<td>R4</td>
<td>STOP</td>
<td>7-000000000</td>
<td>R4</td>
<td>TAU</td>
<td>AA</td>
<td>R4</td>
<td>TOLER</td>
</tr>
<tr>
<td>AA</td>
<td>R4</td>
<td>VELOC</td>
<td>AA</td>
<td>R4</td>
<td>VNORM</td>
<td>AA</td>
<td>R4</td>
<td>TOLER1</td>
</tr>
</tbody>
</table>

### ARRAYS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-00002E60</td>
<td>R4</td>
<td>ACC</td>
<td>400</td>
<td>(100)</td>
</tr>
<tr>
<td>2-00002B10</td>
<td>R4</td>
<td>ADLIST</td>
<td>400</td>
<td>(100)</td>
</tr>
<tr>
<td>2-00002B18</td>
<td>R4</td>
<td>AMULT</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00003A10</td>
<td>R4</td>
<td>ATERM</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-0000242B</td>
<td>R4</td>
<td>BOX1</td>
<td>80</td>
<td>(2, 10)</td>
</tr>
<tr>
<td>2-0000247B</td>
<td>R4</td>
<td>BOX2</td>
<td>800</td>
<td>(2, 10, 10)</td>
</tr>
<tr>
<td>2-00001DEB</td>
<td>R4</td>
<td>BOXES</td>
<td>800</td>
<td>(2, 10, 10)</td>
</tr>
<tr>
<td>2-00000760</td>
<td>R4</td>
<td>COST</td>
<td>4000</td>
<td>(1000)</td>
</tr>
<tr>
<td>2-000033C0</td>
<td>R4</td>
<td>DANGLE</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00000740</td>
<td>R4</td>
<td>DELI</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00001720</td>
<td>R4</td>
<td>DELX</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>2-00003180</td>
<td>R4</td>
<td>DELX1</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>2-000033B8</td>
<td>R4</td>
<td>DELX2</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>3-00000050</td>
<td>R4</td>
<td>DTEXT</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00000750</td>
<td>R4</td>
<td>DLAST</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-0000173B</td>
<td>R4</td>
<td>HOMOG</td>
<td>64</td>
<td>(4, 4)</td>
</tr>
<tr>
<td>2-000027AB</td>
<td>R4</td>
<td>HULL</td>
<td>160</td>
<td>(2, 20)</td>
</tr>
<tr>
<td>9-00000000</td>
<td>R4</td>
<td>IDENT</td>
<td>64</td>
<td>(4, 4)</td>
</tr>
<tr>
<td>2-000031E0</td>
<td>R4</td>
<td>INDIST</td>
<td>400</td>
<td>(100)</td>
</tr>
<tr>
<td>2-0000284B</td>
<td>L4</td>
<td>INFLAG</td>
<td>400</td>
<td>(100)</td>
</tr>
<tr>
<td>2-000029DB</td>
<td>R4</td>
<td>IVERTS</td>
<td>160</td>
<td>(2, 20)</td>
</tr>
<tr>
<td>2-00002B2B</td>
<td>R4</td>
<td>JAC</td>
<td>120</td>
<td>(6, 5)</td>
</tr>
<tr>
<td>2-00001788</td>
<td>R4</td>
<td>JMULT</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>3-00000010</td>
<td>R4</td>
<td>JJTYPE</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00002A78</td>
<td>R4</td>
<td>LBOX</td>
<td>80</td>
<td>(2, 10)</td>
</tr>
<tr>
<td>3-00000030</td>
<td>R4</td>
<td>LEN</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>3-00000000</td>
<td>R4</td>
<td>LINK</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00000200</td>
<td>R4</td>
<td>MAT</td>
<td>80</td>
<td>(4, 5)</td>
</tr>
<tr>
<td>2-00000070</td>
<td>R4</td>
<td>MAT1</td>
<td>80</td>
<td>(4, 5)</td>
</tr>
<tr>
<td>2-00000000</td>
<td>R4</td>
<td>NVALUE</td>
<td>160</td>
<td>(4, 5)</td>
</tr>
<tr>
<td>2-00002A8C</td>
<td>R4</td>
<td>OBOX</td>
<td>80</td>
<td>(2, 10)</td>
</tr>
<tr>
<td>2-00001788</td>
<td>R4</td>
<td>FMULT</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-000027A0</td>
<td>R4</td>
<td>POINT</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>Address</td>
<td>Label</td>
<td>Address</td>
<td>Label</td>
<td>Address</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>---------</td>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>2-00002D30</td>
<td>IA4</td>
<td>SUBLST</td>
<td>400 (100)</td>
<td>2-00001728</td>
</tr>
<tr>
<td>2-0000210B</td>
<td>RA4</td>
<td>TRXES</td>
<td>800 (2, 10, 10)</td>
<td>2-00000177</td>
</tr>
<tr>
<td>2-00003370</td>
<td>RA4</td>
<td>TRADE1</td>
<td>16 (4)</td>
<td>2-00000000</td>
</tr>
<tr>
<td>2-00001700</td>
<td>RA4</td>
<td>THEOT</td>
<td>16 (4)</td>
<td>2-00000010</td>
</tr>
<tr>
<td>2-000017A8</td>
<td>RA4</td>
<td>THEAV</td>
<td>1600 (4, 100)</td>
<td>2-00000010</td>
</tr>
<tr>
<td>2-00000040</td>
<td>RA4</td>
<td>THETA</td>
<td>16 (4)</td>
<td>2-00000010</td>
</tr>
<tr>
<td>2-00000000</td>
<td>RA4</td>
<td>THEA1</td>
<td>16 (4)</td>
<td>2-00000010</td>
</tr>
<tr>
<td>2-000000A0</td>
<td>RA4</td>
<td>THEA2</td>
<td>16 (4)</td>
<td>2-00000010</td>
</tr>
<tr>
<td>2-00001710</td>
<td>RA4</td>
<td>THEtot</td>
<td>16 (4)</td>
<td>2-00000010</td>
</tr>
<tr>
<td>2-000000CD</td>
<td>RA4</td>
<td>THEX</td>
<td>64 (4, 4)</td>
<td>2-0000000B</td>
</tr>
<tr>
<td>AP-00000004B</td>
<td>RA4</td>
<td>VALUES</td>
<td>1600 (4, 100)</td>
<td>2-0000001B</td>
</tr>
</tbody>
</table>

**FUNCTIONS AND SUBROUTINES REFERENCED**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORECT</td>
<td>COSTT</td>
<td>FORCLOSE</td>
<td>FOROPEN</td>
<td>GMULT</td>
<td>GUERTS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HULL2D</td>
<td>INSERT</td>
<td>INTERF</td>
<td>JACOB</td>
<td>MTH$SQRT</td>
<td>SOLVE1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOLVE2</td>
<td>TRANDB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### GET_CPU_TIME

A real function that returns the elapsed CPU time since the initial call to this routine. The CPU time is returned as a real*4 value in seconds. The resolution of the time is 0.01 seconds.

Before the CPU time may be returned, the system timer must be initialized. This is done by calling GET_CPU_TIME with an argument equal to 0. On return, CPU_TIME will contain the address of the storage block initialized by the system routine LIB$INIT_TIMER.

On subsequent calls to CPU_TIME, the cpu time elapsed since the initialization call is returned by passing the address of the storage block as the argument. The cpu time will be contained in the variable CPU_TIME.

**Input:**
- **Base_time**: A real*4 variable which has the value:
  1. 0 if the timer is to be initialized.
  2. The address of the timer storage block returned by a previous call to CPU_TIME.

**Output:**
- **Cpu_time**: A real*4 variable containing:
  1. The address of the timer storage block if CPU_TIME was called with a parameter of 0.
  2. The elapsed cpu time since the timer was initialized if CPU_TIME was called with a timer storage block as the parameter.

```fortran
real function get_cpu_time ( base_time )

Programmer: Gary Lepp Nov. 5, 1984

Description:
This routine returns Elapsed CPU time since the initial call to this routine. The CPU time is returned as a real*4 value in seconds. The resolution of the time is 0.01 seconds.

Before the CPU time may be returned, the system timer must be initialized. This is done by calling GET_CPU_TIME with an argument equal to 0. On return, CPU_TIME will contain the address of the storage block initialized by the system routine LIB$INIT_TIMER.

On subsequent calls to CPU_TIME, the cpu time elapsed since the initialization call is returned by passing the address of the storage block as the argument. The cpu time will be contained in the variable CPU_TIME.

Input:
- **Base_time**: A real*4 variable which has the value:
  1. 0 if the timer is to be initialized.
  2. The address of the timer storage block returned by a previous call to CPU_TIME.

Output:
- **Cpu_time**: A real*4 variable containing:
  1. The address of the timer storage block if CPU_TIME was called with a parameter of 0.
  2. The elapsed cpu time since the timer was initialized if CPU_TIME was called with a timer storage block as the parameter.

equivalence ( itime_storage, time_storage )

igt_cpu_time = 2

Initialize the timer - return the timer storage block

if ( base_time .eq. 0. ) then
  itime_storage = 0
  call lib$init_timer ( itime_storage )
  get_cpu_time = time_storage
else
  Return the elapsed CPU time in units of seconds
```
GET_CPU_TIME

```
time_storage = base_time
call lib$stat_timer(get_cpu_time, icpu_time, itime_storage)
get_cpu_time = 0.01 * icpu_time
end if
return
end
```

**PROGRAM SECTIONS**

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
<td>67</td>
<td>PIC CON REL LCL</td>
</tr>
<tr>
<td>LOCAL</td>
<td>40</td>
<td>PIC CON REL LCL</td>
</tr>
</tbody>
</table>

Total Space Allocated: 107

**ENTRY POINTS**

<table>
<thead>
<tr>
<th>Address Type Name</th>
<th>Address Type Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000 R*4 GET_CPU_TIME</td>
<td></td>
</tr>
</tbody>
</table>

**VARIABLES**

```
<table>
<thead>
<tr>
<th>Address Type Name</th>
<th>Address Type Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE_TIME 2-0000000C I*4 ICPU_TIME</td>
<td></td>
</tr>
<tr>
<td>TIME_STORAGE 2-00000008 I*4 IGET_CPU_TIME</td>
<td></td>
</tr>
<tr>
<td>ITIME_STORAGE 2-00000000 I*4</td>
<td></td>
</tr>
</tbody>
</table>
```

**FUNCTIONS AND SUBROUTINES REFERENCED**

```
<table>
<thead>
<tr>
<th>Type Name</th>
<th>Type Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIB$INIT_TIMER</td>
<td>LIB$STAT_TIMER</td>
</tr>
</tbody>
</table>
```

**COMMAND QUALIFIERS**

```
FOR/LIST OPT.FOR, RDFILE.FOR, INPUT.FOR, GROW.FOR, FINPUT.FOR, SOLVE1.FOR, INT2D.FOR, HULL2D.FOR, GMULT.FOR, JACOB.FOR, COSTT.FOR,
/CHECK= (NOSBOUNDS, OVERFLOW, NOUNDERFLOW)
/DEBUG= (NOSYMBOLS, TRACEBACK)
/STANDARD= (NOSYNTAX, NOSOURCE_FORM)
/SHOW= (NOPREPROCESSOR, NOINCLUDE, MAP, NODictionary, SINGLE)
/WARNINGS= (GENERAL, NODECLARATIONS)
/CONTINUATIONS= 19 /NOCROSS_REFERENCE /NOextend_SOURCE /F77
/NOG_FLOATING /I4 /NOMACHINE_CODE /OPTIMIZE
```
subroutine gmult(a,b,c,m,n,ndima,ndimb,ndimc)

Multiply two matrices a(m x n), b(n x l) to get the product c(m x l).

Parameters:
- a, b - the matrices to be multiplied.
- c - the product
- m, n, l - the dimensions of the matrices.
- ndima - first dimension of fortran array 'a'
- ndimb - first dimension of fortran array 'b'
- ndimc - first dimension of fortran array 'c'

real a(ndima,*) , b(ndimb,*) , c(ndimc,*)

do 10 ind1=1,1
  do 20 ind2=1,m
      temp=0.
      do 30 ind3=1,n
          temp=temp+a(ind2,ind3)*b(ind3,ind1)
    30 continue
    c(ind2,ind1)=temp
  20 continue
 10 continue
10 continue
return
end
**PROGRAM SECTIONS**

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
<td>192</td>
<td>PIC CON REL LCL SHR EXE RD NONRT LONG</td>
</tr>
<tr>
<td>LOCAL</td>
<td>168</td>
<td>PIC CON REL LCL NOSHR NOEXEC RD WRT LONG</td>
</tr>
</tbody>
</table>

**Total Space Allocated**: 360

**ENTRY POINTS**

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000</td>
<td></td>
<td>GMULT</td>
</tr>
</tbody>
</table>

**VARIABLES**

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-00000000</td>
<td>I×4</td>
<td>IND1</td>
<td>AA</td>
<td>I×4</td>
<td>IND2</td>
<td>AA</td>
<td>I×4</td>
<td>IND3</td>
<td>AP-00000018</td>
<td>I×4</td>
<td>L</td>
</tr>
<tr>
<td>AP-00000010</td>
<td>I×4</td>
<td>M</td>
<td>I×4</td>
<td>N</td>
<td>AP-00000001</td>
<td>I×4</td>
<td>NDIMA</td>
<td>AP-00000002</td>
<td>I×4</td>
<td>NDIMB</td>
<td></td>
</tr>
<tr>
<td>AP-00000024</td>
<td>I×4</td>
<td>NDIMC</td>
<td>AA</td>
<td>R×4</td>
<td>TEMP</td>
<td>AA</td>
<td>R×4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AP-00000004</td>
<td>R×4</td>
<td>A</td>
<td>AA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AP-00000008</td>
<td>R×4</td>
<td>B</td>
<td>AA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AP-0000000C</td>
<td>R×4</td>
<td>C</td>
<td>AA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ARRAYS**

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP-00000004</td>
<td>R×4</td>
<td>A</td>
<td>AA</td>
<td>(A, A)</td>
</tr>
<tr>
<td>AP-00000008</td>
<td>R×4</td>
<td>B</td>
<td>AA</td>
<td>(A, A)</td>
</tr>
<tr>
<td>AP-0000000C</td>
<td>R×4</td>
<td>C</td>
<td>AA</td>
<td>(A, A)</td>
</tr>
</tbody>
</table>

**LABELS**

<table>
<thead>
<tr>
<th>Address</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>10</td>
</tr>
<tr>
<td>AA</td>
<td>20</td>
</tr>
<tr>
<td>AA</td>
<td>30</td>
</tr>
</tbody>
</table>

**COMMAND QUALIFIERS**

FOR/LIST OPT FOR,RDFILE FOR,INPUT FOR,GROM FOR,FINPAT FOR,SOLVE1 FOR,INT2D FOR,HULL2D FOR,CMULT FOR,JACOB FOR,COSTT FOR,

/CHECK= (NO BOUNDS, OVERFLOW, UNDERFLOW)
/DEBUG= (NOSYMBOLS, TRACEBACK)
/STANDARD= (NOSYNTAX, NOSOURCE FORM)
/SHOW= (NOPREPROCESSOR, NOINCLUDE, MAP, NODICTIONARY, SINGLE)
/WARNINGS= (GENERAL, NODECLARATIONS)
/CONTINUATIONS= 19 /NOCROSS REFERENCE /NOD_LINES /NODEXTEND_SOURCE /F77
/NOC FLOATING /14 /NOMACHINE_CODE /OPTIMIZE
subroutine grow(delta)

Grow the obstacles by an amount 'delta'.
This compensates for the penetrations resulting from using an
exterior penalty function for interference.
There is no interference if the largest penetration is smaller
than 'delta'.

real sverts(2,100),tverts(2,100)
integer boxes(2,10)
real temp(2,100),nave(2),norm(2),nlast(2),nfirst(2),edge(2)
common /obje/boxes,sverts,tverts
common /nobjs/nllnks,nobst
istart=nllnks+1
iend=nllnks+nobst

do 10 ind=istart,iend
  ifirst=boxes(1,ind)
  ilast=ifirst+boxes(2,ind)-1
  do 20 ind2=ifirst,ilast
    if (ind2.eq.ilast) then
      inext=ifirst
    else
      inext=ind2+1
    endif
    elen=0.
    do 30 i=1,2
      edge(i)=sverts(i,inext)-sverts(i,ind2)
      elen=elen+edge(i)**2
    30 continue
    elen=sqrt(elen)
    if (ind2.ne.ifirst) then
      do 31 i=1,2
        nlast(i)=norm(i)
      31 continue
      continue
    endif
    norm(1)=edge(2)/elen
    norm(2)=edge(1)/elen
    if (ind2.eq.ifirst) then
      do 32 i=1,2
        nfirst(i)=norm(i)
      32 continue
      continue
    endif
    if (ind2.ne.ifirst) then
      rlen=0.
      do 33 i=1,2
        nave(i)=nlast(i)+nfirst(i)**2
      33 continue
      continue
    endif
  20 continue
continue
rlen=sqrtrlen)
do 34 i=1,2
nave(i)=nave(i)/rlen
continue
Do 34 i=1,2
cos=nave(1)norm(1)+nave(2)norm(2)
dist=delta/cos
continue
Do 35 i=1,2
temp(i,ind2)=avert(i,ind2)+nave(i)dist
continue
diff
if (ind2.eq.ind1) then
rlen=0.
do 40 i=1,2
nave(i)=norm(1)+nf(i)
rlen=rlen+nave(i)**2
continue
Do 40 i=1,2
nave(i)=nave(i)/rlen
continue
Do 41 i=1,2
cos=nave(1)norm(1)+nave(2)norm(2)
dist=delta/cos
continue
Do 45 i=1,2
temp(i,ind1)=avert(i,ind1)+nave(i)dist
continue
diff
ifirst=boxes(1,istart)
ilast=boxes(1,iend)+boxes(2,iend)-1
do 50 ind=ifirst,ilast
do 51 i=1,2
avert(i,ind)=temp(i,ind)
continue
Do 51 i=1,2
continue
return
end
PROGRAM SECTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 CODE</td>
<td>544</td>
<td>PIC CON REL LCL  SHR EXE RD NONRT LONG</td>
</tr>
<tr>
<td>2 LOCAL</td>
<td>868</td>
<td>PIC CON REL LCL NOSHR NOEXE RD NRT LONG</td>
</tr>
<tr>
<td>3 OBJS</td>
<td>1680</td>
<td>PIC OVR REL GBL SHR NOEXE RD NRT LONG</td>
</tr>
<tr>
<td>4 NOBJS</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOEXE RD NRT LONG</td>
</tr>
</tbody>
</table>

Total Space Allocated: 3100

ENTRY POINTS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000</td>
<td></td>
<td>GROW</td>
</tr>
</tbody>
</table>

VARIABLES

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>** AA RA4 I**</td>
<td>R4</td>
<td>COS</td>
</tr>
<tr>
<td>2-0000034C</td>
<td>I4</td>
<td>IEND</td>
</tr>
<tr>
<td>2-00000348</td>
<td>I4</td>
<td>IND2</td>
</tr>
<tr>
<td>4-00000000</td>
<td>I4</td>
<td>NLINKS</td>
</tr>
<tr>
<td>AA RA4 I**</td>
<td>R4</td>
<td>DELTA</td>
</tr>
<tr>
<td>2-0000034C</td>
<td>I4</td>
<td>IEND</td>
</tr>
<tr>
<td>2-00000348</td>
<td>I4</td>
<td>IND2</td>
</tr>
<tr>
<td>4-00000000</td>
<td>I4</td>
<td>NLINKS</td>
</tr>
<tr>
<td>AA RA4 I**</td>
<td>R4</td>
<td>DIST</td>
</tr>
<tr>
<td>AA RA4 I**</td>
<td>R4</td>
<td>I1ST</td>
</tr>
<tr>
<td>2-00000350</td>
<td>I4</td>
<td>I2ST</td>
</tr>
<tr>
<td>AA RA4 I**</td>
<td>R4</td>
<td>I3ST</td>
</tr>
<tr>
<td>2-00000000</td>
<td>I4</td>
<td>ELEN</td>
</tr>
<tr>
<td>AA RA4 I**</td>
<td>R4</td>
<td>I4ST</td>
</tr>
<tr>
<td>2-00000350</td>
<td>I4</td>
<td>I5ST</td>
</tr>
<tr>
<td>AA RA4 I**</td>
<td>R4</td>
<td>I6ST</td>
</tr>
<tr>
<td>2-00000350</td>
<td>I4</td>
<td>I7ST</td>
</tr>
</tbody>
</table>

ARRAYS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-00000000</td>
<td>I4</td>
<td>BOXES</td>
<td>80</td>
<td>(2, 10)</td>
</tr>
<tr>
<td>2-000000140</td>
<td>R4</td>
<td>EIGE</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>2-000000320</td>
<td>R4</td>
<td>NAVE</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>2-000000338</td>
<td>R4</td>
<td>NFIRST</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>2-000000330</td>
<td>R4</td>
<td>NLAST</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>2-000000328</td>
<td>R4</td>
<td>NORM</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>3-00000050</td>
<td>R4</td>
<td>SVERTS</td>
<td>800</td>
<td>(2, 100)</td>
</tr>
<tr>
<td>2-000000000</td>
<td>I4</td>
<td>TEMP</td>
<td>800</td>
<td>(2, 100)</td>
</tr>
<tr>
<td>3-00000370</td>
<td>R4</td>
<td>TVERTS</td>
<td>800</td>
<td>(2, 100)</td>
</tr>
</tbody>
</table>

LABELS

<table>
<thead>
<tr>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>** AA</td>
<td>10</td>
<td>AA</td>
<td>20</td>
<td>AA</td>
<td>30</td>
<td>AA</td>
<td>31</td>
<td>AA</td>
<td>32</td>
</tr>
<tr>
<td>** AA</td>
<td>34</td>
<td>AA</td>
<td>35</td>
<td>AA</td>
<td>40</td>
<td>AA</td>
<td>41</td>
<td>AA</td>
<td>45</td>
</tr>
<tr>
<td>** AA</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

** GROW **
FUNCTIONS AND SUBROUTINES REFERENCED

Type     Name

R4        MTLSQRT

COMMAND QUALIFIERS

FOR/List OPT.FOR, RDFILE.FOR, INPUT.FOR, GROW.FOR, MPAT.FOR, SOLVE1.FOR, INT2D.FOR, HULL2D.FOR, GMULT.FOR, JACOB.FOR, COSTT.FOR,

/Check*(NOBOUNDS, OVERFLOW, ROUNDERFLOW)
/Debug*(NOSYMBOLS, TRACEBACK)
/Standard*(NOSYNTAX, NOSOURCEFORM)
/Show*(NOPREPROCESSOR, NOINCLUDE, MAP, MODICTIONARY, SINGLE)
/Warnings*(GENERAL, NODECLARATIONS)
/Continuations=19 /NoCrossReference /Nod_LINES /NoExtendSource /F77
/Noc_Floating /I4 /NOMACHINE_CODE /Optimize

COMPILATION STATISTICS

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Time:</td>
<td>5.01 seconds</td>
</tr>
<tr>
<td>Elapsed Time:</td>
<td>15.25 seconds</td>
</tr>
<tr>
<td>Page Faults:</td>
<td>608</td>
</tr>
<tr>
<td>Dynamic Memory:</td>
<td>462 pages</td>
</tr>
</tbody>
</table>
subroutine gvertal (imode, ltype, number, verts, nverts)

Access the geometry database for a given object.

Parameters:

- **imode** - 1 - get static vertex coordinates for an object
- 2 - get transformed vertex coordinates for an object
- 3 - store new transformed vertex coordinates for an object

- **ltype** - 1 - the object is a moving link
- 2 - the object is a stationary obstacle

- **number** - the object id number
- **verts** - the array of vertex coordinates.
- **nverts** - the number of vertices

Geometry database structure:

- **boxes** - contains pointers to the vertex arrays
- **boxes(1,i)** - starting vertex of link i
- **boxes(2,i)** - number of vertices of link i
- **boxes(1,links+i)** - starting vertex of obstacle i
- **boxes(2,links+i)** - number of vertices of obstacle i
- **sverts**(1,i) - array of fixed vertex coordinates
- **tverts**(1,i) - array of transformed vertex coordinates

```fortran
integer ltype, number
real verts(2,10)
integer boxes(2,10)
real sverts(2,100), tverts(2,100)
common /objs/boxes, sverts, tverts
common /nobjs/nlinks, nobst

Calculate the object pointer.

if (ltype.eq.1) then
  ipoint=number
elseif (ltype.eq.2) then
  ipoint=nlinks+number
endif

Find the starting vertex and the number of vertices.

istart=boxes(1,ipoint)
inumb=boxes(2,ipoint)
if (imode.eq.1) then
  icount=0
  do 10 ind=istart, istart+inumb-1
    icount=icount+1
    do 20 i=1,2
      verts(i,icount)=sverts(i,ind)
      continue
```

```fortran
10 continue
20 continue
```
continue
nverts=inumb

elseif (imode.eq.2) then
Load the transformed vertex array for the requested object.

icount=0
do 11 ind=istart,istart+inumb-1
   icount=icount+1
   do 21 i=1,2
      verts(i,icount)=tverts(i,ind)
   continue
11
continue

endif
return
end
### Program Sections

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $CODE</td>
<td>292</td>
<td>PIC CON REL LCL SHR EXE RD NONRT LONG</td>
</tr>
<tr>
<td>2 $LOCAL</td>
<td>28</td>
<td>PIC CON REL LCL NOSHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>3 OBJ5</td>
<td>1680</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>4 NOBS</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
</tbody>
</table>

Total Space Allocated: 2008

### Entry Points

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000</td>
<td></td>
<td>GVERTS</td>
</tr>
</tbody>
</table>

### Variables

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000 I*4</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>0-00000000 I*4</td>
<td>INUMB</td>
<td>INUMB</td>
</tr>
<tr>
<td>4-00000000 I*4</td>
<td>NLINKS</td>
<td>NLINKS</td>
</tr>
<tr>
<td>AP-000000000 I*4</td>
<td>IMODE</td>
<td>IMODE</td>
</tr>
<tr>
<td>AP-000000004 I*4</td>
<td>IPOINT</td>
<td>IPOINT</td>
</tr>
<tr>
<td>AP-000000004 I*4</td>
<td>NOBST</td>
<td>NOBST</td>
</tr>
<tr>
<td>AP-000000000 I*4</td>
<td>NUMBER</td>
<td>NUMBER</td>
</tr>
<tr>
<td>AP-000000014 I*4</td>
<td>NVERTS</td>
<td>NVERTS</td>
</tr>
</tbody>
</table>

### Arrays

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-000000000 I*4</td>
<td>BOXES</td>
<td>80</td>
<td>(2, 10)</td>
<td></td>
</tr>
<tr>
<td>3-00000000550 R*4</td>
<td>SVERTS</td>
<td>800</td>
<td>(2, 100)</td>
<td></td>
</tr>
<tr>
<td>3-0000000370 R*4</td>
<td>TVERTS</td>
<td>800</td>
<td>(2, 100)</td>
<td></td>
</tr>
<tr>
<td>AP-0000000100 R*4</td>
<td>VERTS</td>
<td>80</td>
<td>(2, 10)</td>
<td></td>
</tr>
</tbody>
</table>

### Labels

<table>
<thead>
<tr>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-000000000</td>
<td>10</td>
<td>0-000000001</td>
<td>11</td>
<td>0-000000002</td>
<td>12</td>
<td>0-000000003</td>
<td>20</td>
<td>0-000000004</td>
<td>21</td>
<td>0-000000005</td>
<td>22</td>
</tr>
</tbody>
</table>
subroutine hull2d(box1,box2,nvl,nv2,hull,nverts,iverts)

Find the convex hull enclosing two convex polygons in 2-D.

Parameters:

- box1, box2 - arrays of vertices defining two polygons.
- nvl, nv2 - the number of vertices in box1, box2
- hull - an array of vertices defining the minimum enclosing convex hull
- nverts - the number of vertices in hull
- iverts - an array linking the vertices of hull to the vertices of box1, box2
- iverts(1,i) - the box containing hull vertex i
- iverts(2,i) - the index of the corresponding vertex in box

Variables:

- more - .true. if there are more edges to be found
- search - .true. if the next edge of the hull links the two polygons
- iflag - .true. until a stopping condition is reached
- more, search, iflag, stop - logical
- more = .true.
- search = .true.
- iflag = .true.
- stop = .false.
- ntemp = 0

common /size/size

Initialize flags.

more = .true.
iflag = .true.
search = .false.
stop = .false.
ntemp = 0

Scale the tolerance according to the size of the world.

toler = size/1000.

Load vertex array.

do 1 i = 1, nvl
  do 2 j = 1, 2
    boxes(j, 1, i) = box1(j, i)
  continue
  continue
1
2
do 3 i=1,nv2
  do 4 j=1,2
    boxes(j,i,2)=box2(j,i)
  continue
  3 continue
  c Find a starting edge of the hull.
  0064  c First, test all the edges of box 1.
  0065  c If no hull edge is found, then test all the edges of box 2.
  0066  c
  0067  do 200 ibox=1,2
    if (iflag) then
      if (ibox.eq.1) then
        iother=2
        inumbl=nv1
        inumb2=nv2
      else
        iother=1
        inumbl=nv2
        inumb2=nv1
      endif
    endif
    do 10 ind=1,inumbl
      if (iflag) then
        if (ind.eq.inumbl) then
          inext=1
        else
          inext=ind+1
        endif
      endif
      do 120 i=1,2
        vl(1)=boxes(1,ind,ibox)-boxes(1,ind,ibox)
      120 continue
  0090 120  continue
  0091  c Calculate the outward normal for a clockwise ordering of vertices.
  0092  vlen=sqrt(vl(1)**2+vl(2)**2)
  0093  vnor1(vl(1))/vlen
  0094  vnor2(vl(2))/vlen
  0095  c Calculate the constant of the line equation.
  0096  const=boxes(1,ind,ibox)*vnor1+boxes(2,ind,ibox)*vnor2
  0097  c Test if all the vertices of box2 are on the same side.
  0098  dain2=-10000.
  0099  do 25 ind2=1,nv2
    test2=boxes(1,ind2,iother)*vnor1+boxes(2,ind2,iother)*vnor2
    if (test2.gt.dain2) then
      dain2=test2
      endif
  0100  25 continue
  0101  c
If the test edge is an edge of the convex hull,
if (ijflag-const.1.e.toler) then
   iflag=.false.
   do 30 i=1,2
      thull(1,1)=boxes(i,ind,ibox)
      thull(1,2)=boxes(i,inext,ibox)
      ivtemp(1,1)=ibox
      ivtemp(2,1)=ind
      ivtemp(1,2)=ibox
      ivtemp(2,2)=inext
      ifirst=ind
      ilast=inext
      ntemp=2
      iboxl=ibox
      do 30 continue
   endif
   endif
   continue
   endif
30 continue

If no edge of either box is an edge of the hull, then find an
edge which connects the first vertex of box1 to a vertex of box2.
if (iflag) then
   iboxl=1
   do 10 ind=1,nv2
      if (iflag) then
         c Generate the edge vector.
         do 121 i=1,2
            vl(i)=boxes(i,1,1)-boxes(i,ind,2)
         121 continue
         c Calculate the outward normal.
         vlen=sqrt(vl(1)**2+vl(2)**2)
         if (vlen.gt.toler) then
            vnorn(1)=-vl(2)/vlen
            vnorn(2)=vl(1)/vlen
            c Calculate the constant of the line equation.
            const=boxes(1,1,1)*vnorn(1)+boxes(2,1,1)*vnorn(2)
            c Test if all the vertices of box2 are on the same side.
            dmax2=-10000.
            itemp=ind
            do 26 ind=1,nv2-1
               itemp=itemp+1
               if (itemp.gt.nv2) itemp=1
            26 continue
         endif
      endif
   10 continue
test2 = boxes(1, itemp, 2) * vnorm(1) + boxes(2, itemp, 2) * vnorm(2)
if (test2 > dmn2) then
  dmn2 = test2
endif
continue
if (dmn2 > const > toler) then
  iflag = false.
  do 31 i = 1, 2
    thull(1, i) = boxes(1, i, 1)
    thull(2, i) = boxes(1, ind, 2)
    ivtemp(1, i) = 1
    ivtemp(2, i) = 1
  endif
  ifirst = 1
  ilast = ind
  ntemp = 2
  continue
endif
endif
continue
31
endif
endif
11
continue
197
continue
198
else
iother = 1
inumbl = nv2
inumb2 = nv1
endif
endif
continue
100
continue
if (ibox .eq. 1) then
  iother = 2
  inumbl = nv1
  inumb2 = nv2
else
  iother = 1
  inumbl = nv2
  inumb2 = nv1
endif
Test the next vertex.

To start, the current box is box1.

Starting from the first edge of the hull, find the edges of the enclosing hull in a clockwise direction.

If the next edge of the current box is not on the hull, test all possible edges connecting the last vertex to vertices of the other box.

Select the one with the smallest angle to the previous edge.

Search for an edge connecting the current box to the other box.
iflag=.true.

c Calculate and normalize the previous edge for comparison.
do 71 i=1,2
    vlast(i)=thull(i,ntemp)-thull(i,ntemp-1)
    continue
vlen1=sqrt(vlast(1)**2+vlast(2)**2)
vlast(1)=vlast(1)/vlen1
vlast(2)=vlast(2)/vlen1

Test all connecting edges and select the best one.
dtest=-1.
ntemp=ntemp+1
do 60 ind3=1,inumb2
    if (iflag) then
        do 70 i=1,2
            vl(i)=boxes(1,ind3,iother)-thull(1,ntemp-1)
            continue
        vlen2=sqrt(vl(1)**2+vl(2)**2)
        if (vlen2.gt.toler) then
            vl(1)=vl(1)/vlen2
            vl(2)=vl(2)/vlen2
            test2=vl(1)*vlast(1)+vl(2)*vlast(2)
        else
            test2=10.
        endif
        if (test2.gt.dtest) then
            dtest=test2
            do 130 i=1,2
                thull(i,ntemp)=boxes(1,ind3,iother)
                continue
            ivtemp(i,ntemp)=iother
            ivtemp(i,ntemp)=ind3
            ilast=ind3
            ibox=iother
            endif
        endif
    endif
    continue
60 continue

If the next edge is nearly collinear to the current edge, merge the
two edges into one by removing the intermediate vertex from the
hull.
error=(vlen1+vlen2)*sqrt(1.-testA*2)/4.
if (error.lt.toler) then
    do 135 i=1,2
        thull(i,ntemp-1)=thull(i,ntemp)
        ivtemp(i,ntemp-1)=ivtemp(i,ntemp)
    continue
    ntemp=ntemp-1
    endif
135 continue

c If the last vertex is the same as the first, or if the last
vertex is very close to the first, remove it from the hull list and quit.

search=.false.

if (ilast.eq.ifirst.and.ibox.eq.ibox1) then
  more=.false.
  ntemp=ntemp-1
else
  vlen=0.
do 137 i=1,2
    vdiff(i)=thull(1,ilast)-thull(1,i)
vlen=vlen+vdiff(i)**2
  137 continue
  vlen=sqrt(vlen)
  if (vlen.lt.toler) then
    more=.false.
    ntemp=ntemp-1
    endif
  endif
else
  Test the next edge of the current box.
  inext=ilast+1
endif

Test if the next vertex is the first vertex, or if it is very close to the first vertex. If so, stop.

if (inext.eq.ifirst.and.ibox.eq.ibox1) then
  more=.false.
else
  vlen=0.
do 147 i=1,2
    vdiff(i)=boxes(1,inext,ibox)-thull(1,i)
vlen=vlen+vdiff(i)**2
  147 continue
  if (vlen.lt.toler) then
    more=.false.
  else
    do 20 i=1,2
      vl(i)=boxes(1,inext,ibox)-boxes(1,ilast,ibox)
    20 continue
    c Calculate the outward normal.
    vlen=sqrt(vl(1)**2+vl(2)**2)
    vnorm(1)=-vl(2)/vlen
    vnorm(2)=vl(1)/vlen
0343 c Calculate the constant of the line equation.
0344 const=boxes(1,inext,ibox)*vnorm(1)+boxes(2,inext,ibox)*vnorm(2)
0346 c Test if all the vertices of the other box are on the same side.
0348 dmin2=-10000.
0349 do 145 ind2=1,inumb2
0351 test2=boxes(1,ind2,iother)*vnorm(1)+boxes(2,ind2,iother)*vnorm(2)
0354 if (test2.gt.dmin2) then
0355 dmin2=test2
0356 endif
0357 145 continue
0359 if (dmin2-const.le.toler) then
0361 iflag=.false.
0362 ntemp=ntemp+1
0363 ilast=inext
0364 do 230 i=1,2
0365 thull(i,ntemp)=boxes(i,inext,ibox)
0366 continue
0367 ivtemp(1,ntemp)=inext
0368 ivtemp(2,ntemp)=ntemp
0370 else
0371 search=.true.
0372 endif
0373 endif
0375 endif
0377 if (more) then
0378 goto 100
0379 endif
0380 c Remove redundant vertices which are close to other vertices.
0382 nverts=0
0384 do 400 ind=1,ntemp
0385 if (ind.eq.1) then
0386 ilast=ntemp
0387 else
0388 ilast=ind-1
0389 endif
0390 vlen=0.
0391 do 401 i=1,2
0392 vlen=vlen+vi(i)**2
0393 401 continue
0394 vlen=sqrt(vlen)
0395 if (vlen.gt.toler) then
0397 nverts=nverts+1
0399 do 402 i=1,2
0400 hull(i,nverts)=thull(i,ind)
HULL2D

```
0400  iverts(i,nverts)=ivtemp(i,ind)
0401  continue
0402  endif
0403  continue
0404  return
0405  end
```

**PROGRAM SECTIONS**

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 CODE</td>
<td>1806</td>
<td>PIC CON REL LCL</td>
</tr>
<tr>
<td>2 LOCAL</td>
<td>652</td>
<td>PIC CON REL LCL WOSHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>3 SIZE</td>
<td>4</td>
<td>PIC OVR REL GBL</td>
</tr>
</tbody>
</table>

**ENTRY POINTS**

```
0-00000000  HULL2D
```

**VARIABLES**

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-0000020B</td>
<td>R*4</td>
<td>IFLAG</td>
</tr>
<tr>
<td>2-00000218</td>
<td>I*4</td>
<td>NOTHER</td>
</tr>
<tr>
<td>2-0000021B</td>
<td>R*4</td>
<td>TEST2</td>
</tr>
<tr>
<td>2-00000234</td>
<td>R*4</td>
<td>VLEN2</td>
</tr>
<tr>
<td>2-00000000</td>
<td>R*4</td>
<td>BOX1</td>
</tr>
<tr>
<td>2-00000000</td>
<td>R*4</td>
<td>BOX2</td>
</tr>
<tr>
<td>2-00000000</td>
<td>R*4</td>
<td>BOXES</td>
</tr>
<tr>
<td>2-0000001B</td>
<td>I*4</td>
<td>IVENTS</td>
</tr>
<tr>
<td>2-0000015B</td>
<td>R*4</td>
<td>IVTEMP</td>
</tr>
<tr>
<td>2-00000000</td>
<td>R*4</td>
<td>TMULL</td>
</tr>
<tr>
<td>2-00000000</td>
<td>R*4</td>
<td>V1</td>
</tr>
<tr>
<td>2-000001FB</td>
<td>R*4</td>
<td>VDIFF</td>
</tr>
</tbody>
</table>

**ARRAYS**

```
AP-00000000   R*4  BOX1  80  (2, 10)
AP-00000000   R*4  BOX2  80  (2, 10)
AP-00000000   R*4  BOXES 160  (2, 10, 2)
AP-00000001   R*4  HULL  160  (2, 20)
AP-00000001   R*4  IVENTS 160  (2, 20)
AP-00000001   R*4  IVTEMP 160  (2, 20)
AP-00000000   R*4  TMULL 160  (2, 20)
AP-00000000   R*4   V1    8   (2)
AP-00000000   R*4   VDIFF  8   (2)
```

**Total Space Allocated**

2462
### LABELS

<table>
<thead>
<tr>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>1</td>
<td>AA</td>
<td>2</td>
<td>AA</td>
<td>3</td>
<td>AA</td>
<td>4</td>
<td>AA</td>
<td>10</td>
</tr>
<tr>
<td>AA</td>
<td>20</td>
<td>AA</td>
<td>25</td>
<td>AA</td>
<td>26</td>
<td>AA</td>
<td>30</td>
<td>AA</td>
<td>31</td>
</tr>
<tr>
<td>AA</td>
<td>70</td>
<td>AA</td>
<td>71</td>
<td>0-00000350</td>
<td>100</td>
<td>AA</td>
<td>120</td>
<td>AA</td>
<td>121</td>
</tr>
<tr>
<td>AA</td>
<td>135</td>
<td>AA</td>
<td>137</td>
<td>AA</td>
<td>145</td>
<td>AA</td>
<td>147</td>
<td>AA</td>
<td>200</td>
</tr>
<tr>
<td>AA</td>
<td>400</td>
<td>AA</td>
<td>401</td>
<td>AA</td>
<td>402</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### FUNCTIONS AND SUBROUTINES REFERENCED

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>R4</td>
<td>MTH$SQRT</td>
</tr>
</tbody>
</table>

### COMMAND QUALIFIERS

- FOR/LIST
- OPT.FOR, COLFILE.FOR, INPUT.FOR, CROM.FOR, PINFAT.FOR, SOLVE1.FOR, INT2D.FOR, HULL2D.FOR, GMULT.FOR, JACOB.FOR, COSTT.FOR,
- /CHECK= (NOSOURCES, OVERFLOW, ROUNDERFLOW)
- /DEBUG= (NOSYMBOLS, TRACEBACK)
- /STANDARD= (NOSYNTAX, NOSOURCE_FORM)
- /SHOW= (NOCROSSREFERENCE, NOINCLUDE, MAP, NODICTIONARY, SINGLE)
- /WARNINGS= (GENERAL, NODECLARATIONS)
- /CONTINUATIONS=19 /NUCROSSREFERENCE /NOD_LINES /NOEXTEND_SOURCE /F77
- /NOG_FLOATING /I4 /NOMACHINE_CODE /OPTIMIZE

### COMPILATION STATISTICS

- Run Time: 14.07 seconds
- Elapsed Time: 44.56 seconds
- Page Faults: 882
- Dynamic Memory: 845 pages
0001 subroutine Input
0002
0003 c Input the parameters for the path-planning algorithm.
0004 c The parameters can be read from a file, or they can be input
0005 c interactively by the user.
0006 c The parameters are passed through common blocks.
0007
0008 real theta(4)
0009 real jfact, tau, pfact
0010 integer nmax
0011 integer link(4), jtype(4)
0012 real len(4), dist(4), alpha(4)
0013 real vmin(4), vmax(4)
0014 real maxpen, jtoler
0015 logical ans
0016 common /robot/link, jtype, alpha, len, theta, dist
0017 common /param1/jfact, pfact
0018 common /param3/alafct
0019 common /tau/tau
0020 common /cmax/cmax
0021 common /nmax/nmax
0022 common /limits/vmin, vmax
0023 common /stepsi/stepsi
0024 common /listol/maxpen, jtoler
0025 common /ndof/ndof, ndof
0026 common /fracti/fracti
0027 common /liBitB
0028 c User can interactively enter parameters, or they can be read from a file.
0029 c
0030 write(*,*) 'Do you wish to enter new parameters? (T/F)'
0031 read(*,*) ans
0032 if (ans) then
0033 open(1, file='param.dat', status='old')
0034 read(1,*) jfact
0035 write(*,*) 'Joint limit weighting factor', jfact
0036 write(*,*) 'Change? (t/f)'
0037 read(*,*) ans
0038 if (ans) then
0039 write(*,*) 'Enter new value'
0040 read(*,a) jfact
0041 endif
0042 read(1,*) pfact
0043 write(*,*) 'Interference weighting factor', pfact
0044 write(*,*) 'Change? (t/f)'
0045 read(*,*) ans
0046 if (ans) then
0047 write(*,*) 'Enter new value'
0048 read(*,a) pfact
0049 endif
read(1,A) fracti
write(A,A) 'Penetration fraction ', fracti
write(A,A) 'Change? (t/f)'
read(A,A) ans
if (ans) then
    write(A,A) 'Enter new value'
    read(A,A) fracti
endif
read(1,A) afact
write(A,A) 'Joint acceleration weighting factor',
afact
write(A,A) 'Change? (t/f)'
read(A,A) ans
if (ans) then
    write(A,A) 'Enter new value'
    read(A,A) afact
endif
read(1,A) nmax
write(A,A) 'Maximum number of iterations', nmax
write(A,A) 'Change? (t/f)'
read(A,A) ans
if (ans) then
    write(A,A) 'Enter new value'
    read(A,A) nmax
endif
read(1,A) tau
write(A,A) 'Iteration factor (between 0 and 1)', tau
write(A,A) 'Change? (t/f)'
read(A,A) ans
if (ans) then
    write(A,A) 'Enter new value'
    read(A,A) tau
endif
read(1,A) stepsi
write(A,A) 'Maximum iteration linearization error', stepsi
write(A,A) 'Change? (t/f)'
read(A,A) ans
if (ans) then
    write(A,A) 'Enter new value'
    read(A,A) stepsi
endif
read(1,A) cmax
write(A,A) 'Maximum trajectory linearization error', cmax
write(A,A) 'Change? (t/f)'
read(A,A) ans
if (ans) then
    write(A,A) 'Enter new value'
    read(A,A) cmax
endif
read(1,A) maxpen, jtoler
0115 write(A,A) 'Maximum penetration and joint limit tolerance',
0116 maxpen,jtoler
0117 write(A,A) 'Change? (t/f)' 
0118 read(A,A) ans
0119 if (ans) then
0120 write(A,A) 'Enter new value'
0121 read(A,A) maxpen,jtoler
0122 endif
0123
0124 read(l,*), (vnin(i),i=1,ndof)
0125 write(A,A) 'Minimum joint angles',(vnin(i),i=1,ndof)
0126 write(A,A) 'Change? (t/f)' 
0127 read(A,A) ans
0128 if (ans) then
0129 write(A,A) 'Enter new values'
0130 read(A,A) (vnin(i),i=1,ndof)
0131 endif
0132
0133 read(l,*), (vmax(i),i=1,ndof)
0134 write(A,A) 'Maximum joint angles',(vmax(i),i=1,ndof)
0135 write(A,A) 'Change? (t/f)' 
0136 read(A,A) ans
0137 if (ans) then
0138 write(A,A) 'Enter new values'
0139 read(A,A) (vmax(i),i=1,ndof)
0140 endif
0141 c Write the parameters back into the file.
0142
0143 rewind(l)
0144
0145 write(l,A) jfact
0146 write(l,A) pfact
0147 write(l,A) fracti
0148 write(l,A) afact
0149 write(l,A) nmax
0150 write(l,A) tau
0151 write(l,A) stepsi
0152 write(l,A) cmax
0153 write(l,A) maxpen,jtoler
0154 write(l,A) (vnin(i),i=1,ndof)
0155 write(l,A) (vmax(i),i=1,ndof)
0156 c else
0157 read the parameters from a file.
0158 c
0159 open(1,file='param.dat',status='old')
0160
0161 read(l,A) jfact
0162 read(l,A) pfact
0163 read(l,A) fracti
0164 read(l,A) afact
0165 read(l,A) nmax
0166 read(l,A) tau
0167 read(l,A) stepsi
0168 c read(l,A) cmax
read(1,A) maxpen, jtoler
read(1,A) (vmin(i), i=1, ndof)
read(1,A) (vmax(i), i=1, ndof)
write(*,*) 'joint limit weighting factor', jfact
write(*,*) 'interference weighting factor', pfact
write(*,*) 'penetration fraction', fracti
write(*,*) 'joint acceleration weighting factor', afact
write(*,*) 'maximum number of iterations', nmax
write(*,*) 'iteration factor (between 0 and 1)', tau
write(*,*) 'Maximum iteration linearization error', stepsi
write(*,*) 'maximum trajectory linearization error', cmax
write(*,*) 'maximum penetration and joint limit tolerance',
write(*,*) 'minimum joint angles', (vain(i), i=1, ndof)
write(*,*) 'Maximum joint angles', (vmax(i), i=1, ndof)
endif
end
VARIABLES

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-00000000</td>
<td>R4</td>
<td>AFACT</td>
<td>-00000000</td>
</tr>
<tr>
<td>2-00000000</td>
<td>I4</td>
<td>ANS</td>
<td>-00000000</td>
</tr>
<tr>
<td>7-00000000</td>
<td>R4</td>
<td>CMAX</td>
<td>-00000000</td>
</tr>
<tr>
<td>11-00000000</td>
<td>R4</td>
<td>JTOLER</td>
<td>-00000000</td>
</tr>
<tr>
<td>13-00000000</td>
<td>R4</td>
<td>FRACTI</td>
<td>-00000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>12-00000000</td>
<td>I4</td>
<td>NDOF</td>
<td>-00000000</td>
</tr>
<tr>
<td>6-00000000</td>
<td>R4</td>
<td>STEPSI</td>
<td>-00000000</td>
</tr>
</tbody>
</table>

ARRAYS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-00000020</td>
<td>R4</td>
<td>ALPHA</td>
<td>-00000020</td>
</tr>
<tr>
<td>1-00000050</td>
<td>R4</td>
<td>DIST</td>
<td>-00000050</td>
</tr>
<tr>
<td>1-00000050</td>
<td>R4</td>
<td>LEN</td>
<td>-00000050</td>
</tr>
<tr>
<td>1-00000030</td>
<td>R4</td>
<td>LINK</td>
<td>-00000030</td>
</tr>
<tr>
<td>1-00000040</td>
<td>R4</td>
<td>THETA</td>
<td>-00000040</td>
</tr>
<tr>
<td>1-00000010</td>
<td>R4</td>
<td>VMAX</td>
<td>-00000010</td>
</tr>
<tr>
<td>1-00000000</td>
<td>R4</td>
<td>VMIN</td>
<td>-00000000</td>
</tr>
</tbody>
</table>

FUNCTIONS AND SUBROUTINES REFERENCED

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOR$CLOSE</td>
<td></td>
<td>FOR$OPEN</td>
<td></td>
</tr>
</tbody>
</table>

COMMAND QUALIFIERS

FOR/LIST OPT.FOR, REF1E.FOR, INPUT.FOR, GROM.FOR, FINPAT.FOR, SOLVE1.FOR, INT2D.FOR, HULL2D.FOR, CMUL1T.FOR, JACOB.FOR, COSTT.FOR,

/ CHECK = NOBOUNDS, OVERFLOW, UNDERFLOW
/ DEBUG = NOSYMBOLS, TRACEBACK
/ STANDARD = NOSYNTAX, NOSOURCE FORM
/ SHOW = NOPREPROCESSOR, NOINCLUDE, MAP, NOdictionary, SINGLE
/ WARNINGS = GENERAL, NODECLARATIONS
/ CONTINUATIONS = 19 / NOCROSS_REFERENCE / NO LINES / NO EXTEND SOURCE / F77
/ NOM FLOATING / 14 / NO MACHINE_CODE / OPTIMIZE

COMPILATION STATISTICS

Run Time: 9.69 seconds
Elapsed Time: 29.22 seconds
Page Faults: 727
Dynamic Memory: 462 pages
subroutine insert(nadd, nsub, imode, adlist, sublist, values)

Insert and remove discrete positions along the trajectory to maintain a desired linearization error in the discrete approximation of the trajectory.

Parameters:
- nadd: number of positions to be added
- nsub: number of positions to be removed
- imode: current iteration direction, forward or backward
- adlist: list of positions to be added
- sublist: list of positions to be removed
- values: the updated array of joint values

Common blocks changed:
- last: the number of positions

real values(4, 100)
integer adlist(100), sublist(100)
real temp(4, 100)

common /last/ last

if (imode .eq. 1) then
    istart = last
    iend = 1
    incr = -1
else
    istart = 1
    iend = last
    incr = 1
endif

index = 0
iadd = 1
isub = 1

do 10 ind = istart, iend, incr
    index = index + 1
    do 20 i = 1, 4
        temp(1, index) = values(i, ind)
    continue
20
    do 40 i = 1, 4
        if (isub .le. nsub .and. sublist(isub) .eq. ind) then
            isub = isub + 1
            index = index - 1
        endif
    endif
40
    do 11 i = 1, 4
        temp(1, index) = (values(i, ind + incr) + values(i, ind)) / 2.
11
continue
endif
10 continue
0052
0053
0054
last = index
if (imode .eq. 1) then
  iend = 1
  incr = 1
else
  iend = last
  incr = 1
endif
ind2 = 0

do 50 ind = start, iend, incr
  ind2 = ind2 + 1
  do 55 j = 1, i
    values(j, ind2) = temp(j, ind)
  enddo
  continue
continue

do 60 i = 1, last
  valuea(1, last + 1) = valuea(i, last)
  continue

return
end

PROGRAM SECTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CODE</td>
<td>398</td>
<td>PIC COR REL LCL SHR EXE RD NCOMRT LONG</td>
</tr>
<tr>
<td>LOCAL</td>
<td>1692</td>
<td>PIC COR REL LCL NOSHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>LAST</td>
<td>4</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
</tbody>
</table>

Total Space Allocated: 2094

ENTRY POINTS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000</td>
<td></td>
<td>INSERT</td>
</tr>
</tbody>
</table>

VARIABLES

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>I4</td>
<td>I</td>
<td>AA</td>
<td>I4</td>
<td>IADD</td>
<td>AA</td>
<td>I4</td>
<td>IEND</td>
<td>AA</td>
<td>I4</td>
<td>IMODE</td>
</tr>
<tr>
<td>2-00000640</td>
<td>I4</td>
<td>INCR</td>
<td>2-0000064C</td>
<td>I4</td>
<td>IND</td>
<td>AA</td>
<td>I4</td>
<td>IND2</td>
<td>2-00000644</td>
<td>I4</td>
<td>INDEX</td>
</tr>
<tr>
<td>AA</td>
<td>I4</td>
<td>ISTART</td>
<td>2-00000648</td>
<td>I4</td>
<td>ISUB</td>
<td>AA</td>
<td>I4</td>
<td>J</td>
<td>3-00000000</td>
<td>I4</td>
<td>LAST</td>
</tr>
<tr>
<td>AP-000000049</td>
<td>I4</td>
<td>NADD</td>
<td>AP-000000089</td>
<td>I4</td>
<td>NSUB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
INSERT

13-May-1987 12:57:37  VAX FORTRAN V4.4-177
7-May-1987 17:44:33  _BUCHAU1.OPTJINSERT.FOR;20

ARRAYS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP-00000010B I4</td>
<td>ADLST</td>
<td>400</td>
<td>(100)</td>
<td></td>
</tr>
<tr>
<td>AP-00000014B I4</td>
<td>SUBLST</td>
<td>400</td>
<td>(100)</td>
<td></td>
</tr>
<tr>
<td>2-000000000  R*4</td>
<td>TEMP</td>
<td>1600</td>
<td>(4, 100)</td>
<td></td>
</tr>
<tr>
<td>AP-00000018B R*4</td>
<td>VALUES</td>
<td>1600</td>
<td>(4, 100)</td>
<td></td>
</tr>
</tbody>
</table>

LABELS

<table>
<thead>
<tr>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>10</td>
<td>AA</td>
<td>11</td>
<td>AA</td>
<td>20</td>
<td>AA</td>
<td>50</td>
<td>AA</td>
<td>55</td>
</tr>
</tbody>
</table>

COMMAND QUALIFIERS

FOR/LIST OPT.FOR, RFILE.FOR, INPUT.FOR, CROM.FOR, FINPAT.FOR, SOLVE1.FOR, INT2D.FOR, HULL2D.FOR, GMULT.FOR, JACOB.FOR, COSTT.FOR,

/ CHECK=(NOBOUNDS, OVERFLOW, NOUNDERFLOW)
/ DEBUG=(NOSYMBOLS, TRACEBACK)
/ STANDARD=(NOSYNTAX, NOSOURCE_FORM)
/ SHOW=(NOPREPROCESSOR, NOINCLUDE, MAP, NODictionary, SIMPLE)
/ WARNINGS=(GENERAL, NODECLARATIONS)
/ CONTINUATIONS=19 / NOCROSS_REFERENCE / NO_SOURCE_FORM / F77
/ NOC_FLOATING / I4 / NOMICRO_CODE / OPTIMIZE

COMPILATION STATISTICS

Run Time: 3.51 seconds
Elapsed Time: 8.96 seconds
Page Faults: 581
Dynamic Memory: 845 pages
subroutine interf(hull, obst, nvert1, nvert2, ilink, iobst, iverts, inflag)

Test for interference between a swept volume and an obstacle, and find the necessary correction vectors to reduce the interference.

Parameters:

- hull - array of convex hull vertices
- obst - array of obstacle vertices
- nvert1, nvert2 - number of vertices in hull, obst
- ilink - moving link id
- iobst - stationary obstacle id
- iverts - array linking hull vertices to corresponding polygon vertices
- inflag - .true. if interference is detected

real hull(2,20), obst(2,10)
real vect(2), point(2)
real vlens(2), vects(2,2), points(2,2)
integer iverts(2,20), ipair(2)
logical inflag, ledge, iflag2

do i = 1, 2
   vect1(i) = 0.
   vect2(i) = 0.
   point1(i) = 0.
   point2(i) = 0.
continue

Test for interference between the convex hull representing a swept volume and a convex obstacle.

call int2d(hull, obst, nvert1, nvert2, inflag, distan, 
      vect, point, ivert, ledge)

iflag2 = .false.

If interference is detected...

if (inflag) then
   if (ledge) then
      itest1 = iverts(1, ivert)
      ipair(1) = ivert

      Find the endpoints of the edge, and the objects to which they belong.
      if (ivert.eq.nvert1) then
         itest2 = iverts(1, 1)
         ipair(2) = 1
      else
         itest2 = iverts(1, ivert+1)
If the endpoints lie on different objects, then the edge is a swept edge.

if (itest1.ne.itest2) then
   lflag2=.true.
   if (itest1.eq.2) then
      ipair(1)=ipair(2)
      ipair(2)=i1ert
   endif
endif
endif
endif
endif
endif

If the penetration vector intersects a swept edge, then both the initial and final positions should be modified. The relative motions of the end positions are weighted so that when combined the swept volume will move away from the penetration in the direction of the penetration vector.

if (iflag2) then
   rlen=0.
   rlen2=0.
   do 1001 i=1,2
      rlen=rlen+(hull(i,ipair(1))-hull(i,ipair(2)))**2
      rlen2=rlen2+(point(i)-hull(i,ipair(2)))**2
   1001 continue
   rlam=sqrtf(rlen2/rlen)
   alph=rlam/(1.-rlam*rlam)**2
   beta=1.-rlam*alph
   temp=0.
   do 1003 i=1,2
      vectl(i)=alph*vect(i)
      pointl(i)=hull(i,ipair(2))
   1003 continue
else
   if (i1ert.gt.0.) and..i1ert.lt.9 and.i1erts(1,i1ert).eq.2) then
      beta=0.
      temp=0.
      do 1010 i=1,2
         vectl(i)=vect(i)
         pointl(i)=point(i)
      1010 continue
   else
      beta=1.
      do 1011 i=1,2
   endif
endif
else
   c
   If the penetration vector intersects the final position, then only the final position is modified.
   if (i1ert.gt.0.) and..i1ert.lt.9 and..i1erts(1,i1ert).eq.2) then
      beta=0.
      temp=0.
      do 1010 i=1,2
         vectl(i)=vect(i)
         pointl(i)=point(i)
      1010 continue
   else
      beta=1.
      do 1011 i=1,2
   endif
endif
IF there is no penetration, set the penetration vector and point to zero.

ELSE
   DO 1100 I=1,2
      vect(i)=0.
      point(i)=0.
   CONTINUE
ENDIF

Update the correction to be saved for the next iteration.

DO 920 I=1,2
   vect2(i)=beta*vect(i)
   IF (iflag2) THEN
      point2(i)=hull(i,ipair(1))
   ELSE
      point2(i)=point(i)
   ENDIF
   CONTINUE
ENDDO

Save the correction vectors and points in the database.

DO 99 I=1,2
   vecs(i,1)=vect1(i)
   vecs(i,2)=vect2(i)
   points(i,1)=point1(i)
   points(i,2)=point2(i)
ENDDO

CONTINUE

CALL CVEST(2,link,iobst,viens,vecs,points)

RETURN
END
PROGRAM SECTIONS

Name            Bytes   Attributes
0 $CODE          528     PIC CON REL LCL  SHR  EXE  RD  NOMRT  LONG
1 $DATA          4       PIC CON REL LCL  SHR  NOEXE  RD  NOMRT  LONG
2 $LOCAL         248     PIC CON REL LCL  NOSHR  NOEXE  RD  MRT  LONG

Total Space Allocated

ENTRY POINTS

Address  Type  Name
0-00000000  INTERF

VARIABLES

Address  Type  Name
AA       R4     ALPH
2-00000060 I4     IEDGE
AP-00000010B I4     IOBST
AP-0000000CB I4     NVERT1
AA       R4     RLEN2

Address  Type  Name
AA       R4     BETA
2-00000064 I4     IFLAG2
AP-000000014B I4     ITEST1
AP-00000010B I4     ITEST2
AA       R4     RLAMB

Address  Type  Name
AA       R4     DI STAN
AP-00000014H 1*4     ILINK
AA       R4     RLAMB

ARRAYS

Address  Type  Name  Bytes  Dimensions
AP-00000004B R4     HULL  160  (2, 20)
AP-00000005B I4     IPAIR  8  (2)
AP-00000001CB I4     IVERTS  160  (2, 20)
AP-00000000CB R4     OBST  80  (2, 10)
2-00000028 R4     POINT  8  (2)
2-00000010 R4     POINT1  8  (2)
2-00000018 R4     POINT2  8  (2)
2-0000004B R4     POINTS  16  (2, 2)
2-00000020 R4     VECT  8  (2)
2-00000000 R4     VECT1  8  (2)
2-0000000B R4     VECT2  8  (2)
2-0000003B R4     VECT3  16  (2, 2)
2-00000030 R4     VLENS  8  (2)

LABELS

Address  Label  Address  Label  Address  Label  Address  Label  Address  Label
AA       11     AA       99     AA       920     AA       1001     AA       1003     AA       1010
### FUNCTIONS AND SUBROUTINES REFERENCED

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CVECT</td>
<td></td>
<td>INT2D</td>
<td></td>
<td>R*4</td>
</tr>
<tr>
<td></td>
<td>MTHSQRT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### COMMAND QUALIFIERS

FOR/LIST OPT.FOR, RDFILE.FOR, INPUT.FOR, GROW.FOR, FINPAT.FOR, SOLVE1.FOR, INT2D.FOR, HULL2D.FOR, CMULT.FOR, JACOB.FOR, COSTT.FOR, ...

/CHECK= (NOSYMBOLS, OVERFLOW, MOUNDERFLOW)  
/DEBUG= (NOSYMBOLS, TRACEBACK)  
/STANDARD= (NOSYNTAX, NOSOURCE_FORM)  
/SHOW= (NOREPROCESSOR, NOINCLUDEN, MAP, NODICTIONARY, SINGLE)  
/WARNINGS= (GENERAL, NODECLARATIONS)  
/CONTINUATIONS=19  
/MCROSS_REFERENCE /MO_LINES /MEXTEND_SOURCE /F77  
/NOMACHINE_CODE  
/OPTIMIZE

### COMPILATION STATISTICS

- Run Time: 5.16 seconds
- Elapsed Time: 14.92 seconds
- Page Faults: 623
- Dynamic Memory: 845 pages
subroutine int2d(polyl,poly2,nvert1,nvert2,iflag,dist,vect,point,ivert,ledge)

        c Test two 2-D polygons for intersection, and find the penetration vector and penetration point relative to the first polygon.
        c
        c Parameters:
        c polyl,poly2- arrays of vertices representing two 2-D polygons.
        c nvert1,nvert2- the number of vertices in polyl,poly2
        c iflag- returns .true., if the polygons intersect.
        c dist- returns the interference distance.
        c vect- returns the minimum interference vector, directed from polygon 1 boundary to polygon 2 boundary.
        c point- returns the head of vector, attached to polygon 1.
        c ivert- index of the closest vertex or first vertex of closest edge of polygon 1 if 'ledge'= .true.
        c ledge- .true., if the minimum distance vector intersects an edge of the first polygon

        Data structures:
        c boxes- combines the two polygons in a single array
        c ipair- ipair(1)= index of closest vertex of current polygon
        c ipair(2)= id of other polygon
        c ipair(3)= index of closest vertex of other polygon

        real point(2)
        real poly1(2,nvert1),poly2(2,nvert2)
        real boxes(2,10,2)
        logical iflag
        real vect(2)
        real vnorm(2)
        real vl(2),v2(2)
        integer ipair(3)
        integer nvert1
        logical ledge
        logical .true.
        logical .false.
        real dist=10000.
        ivert=1
        thresh=.00001

        Combine the polygons into a single array.
        do 1 i=1,2
        do 2 j=1,nvert1
                boxes(1,j,1)=poly1(i,j)
        continue
        do 3 j=1,nvert2
                boxes(1,j,2)=poly2(i,j)
        continue
4    1
        continue
4    2
        continue
4    3
        continue
        nvert1=nvert1
        nvert2=nvert2

        Test the first polygon against the second polygon.
do 5 ind3=1,2
if (lflag) then
   ioth=2
   if (ind3.eq.2) ioth=1
   c Find the edge which maximizes the separating distance.
do 10 ind=1,nvert(ind3)
if (lflag) then
   if (ind.eq.nvert(ind3)) then
      do 15 j=1,2
      vl(j)=boxes(j,ind-l,ind3)-boxes(j,nvert(ind3),ind3)
      continue
   else
      do 20 j=1,2
      vl(j)=boxes(j,ind+1,ind3)-boxes(j,ind,ind3)
      continue
   endif
   c Calculate the outward normal.
   vlen=sqrt(vl(l)**2+vl(2)**2)
   vnorm(1)=-vl(2)/vlen
   vnorm(2)=vl(1)/vlen
   c Calculate constant of line equation.
   const=boxes(1,ind,ind3)*vnorm(l)+boxes(2,ind,ind3)*vnorm(2)
   c Plug in an adjacent vertex of box1.
   if (ind.eq.1) then
      itemp=nvert(ind3)
   else
      itemp=ind-1
   endif
   testl=boxes(1,itemp,ind3)*vnorm(l)
   +boxes(2,itemp,ind3)*vnorm(2)-const
   if (testl.gt.dmin2) then
      dmin2=testl
      ipair(1)=ind
      ipair(2)=ioth
      ipair(3)=ind2
   endif
   c Test if all the vertices of box2 are on the opposite side and save
   the closest vertex.
do 25 ind2=1,nvert(ioth)
      if (lflag) then
         test2=boxes(1,ind2,ioth)*vnorm(1)+
            boxes(2,ind2,ioth)*vnorm(2)-const
         if (test2.lt.dmin2) then
            dmin2=test2
            ipair(l)=ind
            ipair(2)=ioth
            ipair(3)=ind2
         endif
c If so, the boxes are disjoint.
endif
continue
c If the boxes are not disjoint, update the interference data.
if (test1Admin2.lt.thresh) then
iflag=.false.
endif

c If the boxes are disjoint, reset the output parameters.
if (iflag.and.dmin2.gt.dist) then
dist=dmin2
if (ind3.eq.1) then
   do 55 i=1,2
      vect(i)=dist*vnorm(i)
      point(i)=boxes(i,ipair(3),ipair(2))-dist*vnorm(i)
   continue
   ivert=ipair(1)
   1edge=.true.
else
   do 60 i=1,2
      vect(i)=-dist*vnorm(i)
      point(i)=boxes(i,ipair(3),ipair(2))
   continue
   ivert=ipair(3)
   1edge=.false.
   endif
endif
endif
continue
c If the boxes are disjoint, reset the output parameters.
if (.not.iflag) then
   1edge=.false.
do 150 i=1,2
      vect(i)=0.
point(i)=0.
continue
150
endif
return
end
**PROGRAM SECTIONS**

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $CODE</td>
<td>785</td>
<td>PIC CON REL LCL SHR EXE RD NONRT LONG</td>
</tr>
<tr>
<td>2 $GLOBAL</td>
<td>364</td>
<td>PIC CON REL LCL MSHR NOEXE RD WRT LONG</td>
</tr>
</tbody>
</table>

Total Space Allocated: 1149

**ENTRY POINTS**

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000</td>
<td>INT2D</td>
<td></td>
</tr>
</tbody>
</table>

**VARIABLES**

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AA</strong></td>
<td>R4</td>
<td>CONST</td>
<td>AP-000000180</td>
<td>R4</td>
<td>DIST</td>
<td><strong>AA</strong></td>
<td>R4</td>
<td>DMIN2</td>
</tr>
<tr>
<td>AP-0000002B0</td>
<td>L4</td>
<td>IEDGE</td>
<td>AP-000000140</td>
<td>L4</td>
<td>IFLAG</td>
<td>2-000000D0</td>
<td>I4</td>
<td>IND</td>
</tr>
<tr>
<td><strong>AA</strong></td>
<td>I4</td>
<td>IND3</td>
<td>AP-000000CC0</td>
<td>I4</td>
<td>IOTHER</td>
<td><strong>AA</strong></td>
<td>I4</td>
<td>IITEM</td>
</tr>
<tr>
<td><strong>AA</strong></td>
<td>I4</td>
<td>J</td>
<td>AP-000000000</td>
<td>I4</td>
<td>IVERT1</td>
<td>AP-000000100</td>
<td>I4</td>
<td>IVERPT</td>
</tr>
<tr>
<td><strong>AA</strong></td>
<td>R4</td>
<td>TEST2</td>
<td>AA</td>
<td>R4</td>
<td>THRESH</td>
<td><strong>AA</strong></td>
<td>R4</td>
<td>VLEN</td>
</tr>
</tbody>
</table>

**ARRAYS**

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-000000000</td>
<td>R4</td>
<td>BOXES</td>
<td>160</td>
<td>(2, 10, 2)</td>
</tr>
<tr>
<td>2-000000B8</td>
<td>I4</td>
<td>IPAIR</td>
<td>12</td>
<td>(3)</td>
</tr>
<tr>
<td>2-000000CC4</td>
<td>I4</td>
<td>NVERT</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>AP-0000002D0</td>
<td>R4</td>
<td>POINT</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>AP-000000049</td>
<td>R4</td>
<td>POLY1</td>
<td>AA</td>
<td>(3, 4)</td>
</tr>
<tr>
<td>AP-0000000B0</td>
<td>R4</td>
<td>POLY2</td>
<td>AA</td>
<td>(2, 4)</td>
</tr>
<tr>
<td>2-000000A8</td>
<td>R4</td>
<td>V1</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>2-000000B0</td>
<td>R4</td>
<td>V2</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>AP-0000000C0</td>
<td>R4</td>
<td>VECT</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>2-000000A0</td>
<td>R4</td>
<td>VNORM</td>
<td>8</td>
<td>(2)</td>
</tr>
</tbody>
</table>

**LABELS**

<table>
<thead>
<tr>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AA</strong></td>
<td>1</td>
<td><strong>AA</strong></td>
<td>2</td>
<td><strong>AA</strong></td>
<td>3</td>
<td><strong>AA</strong></td>
<td>5</td>
<td><strong>AA</strong></td>
<td>10</td>
</tr>
<tr>
<td><strong>AA</strong></td>
<td>20</td>
<td><strong>AA</strong></td>
<td>25</td>
<td><strong>AA</strong></td>
<td>55</td>
<td><strong>AA</strong></td>
<td>60</td>
<td><strong>AA</strong></td>
<td>150</td>
</tr>
</tbody>
</table>
## INT2D

### FUNCTIONS AND SUBROUTINES REFERENCED

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>R4</td>
<td>MTM4SORT</td>
</tr>
</tbody>
</table>
subroutine invers(len, alpha, dist, theta, jtype, ndof, nxdo, xdot,
    rotdes, xgoal, theta2)

  Find an optimal joint configuration for a given tool position.
An optimal inverse solution is found for a redundant manipulator
by iteratively reducing the difference between the desired and
actual gripper positions, while simultaneously using redundancy to
maximize joint availability.

Parameters:

len, alpha, dist- fixed link parameters
theta- joint angles
jtype- array specifying joint types (prismatic or revolute)
ndof, nxdo- number of joint and task coordinates
ixdot- an array mapping a (1 X nxdo) task vector onto a
standard (1 X 6) vector containing three rotations
and three translations.
kgoal, anggoal- the desired generalized rotation vector and angle
rotdes- desired orientation matrix
xgoal- the desired cartesian gripper position
theta2- the inverse joint solution

real len(*), dist(*), alpha(*), theta(*), theta2(*)
integer jtype(4)
real S(10), UD(10,6), VD(10,10)
real xdot(6), xdot2(6)
real mat(10,11)
real dist(10), angle(10)
real thedot(10)
real tsave(10)
real tnet(4,4)
real xgoal(*)
real asave(10)
real angles(10)
real k(3), kgoal(3)
real term(3)
logical ixdot(6)
real temp1(6)
real temp3(6)
real Jac(6,10), ident(4,4)
real rotdes(3,3)
real tmat(3,3), tinv(3,3)
real rot(3)
logical repeat, rotflg
real vdt(10,10), homog(10,10), tbdotl(10), matl(10,10), xdotl(6)
real temp(6,6), jacinv(10,6)
logical stop2

data ident/1., 4*0., 1., 4*0., 1., 4*0., 1./

c Test files.
open(1, file= 'a1.dat', status= 'new')
open(2, file='a2.dat', status= 'new')
open(3, file='a3.dat', status= 'new')
Program to calculate the inverse of a matrix.

```fortran
0058 open(4, file='a4.dat', status='new')
0059 open(5, file='err.dat', status='new')
0060 open(9, file='fnull.dat', status='new')
0061 open(11, file='t1.dat', status='new')
0062 open(12, file='t2.dat', status='new')
0063 open(13, file='fact.dat', status='new')
0064 open(14, file='erln.dat', status='new')
0065 open(15, file='t3.dat', status='new')
0066 open(16, file='anor.dat', status='new')
0067 c Set maximum linearization error, joint availability factor, maximum
0068 c number of iterations.
0069 write(*, *) 'Enter ermax, gamma'
0070 read(*, *) ermax, gamma
0071 nmax=100
0072 stop2=.false.
0073 pi=3.1415927
0074 toler=.01
0075 toler2=.1
0076 toler4=.0001
0077 d2rad=.0174533
0078 stoler=.0001
0079 thomog=1.
0080 c Set a flag if there are any rotational task coordinates.
0081 rotflg=ixdot(1).or.ixdot(2).or.ixdot(3)
0082 c Convert angles to radians.
0083 do 1 i=1,ndof
0084 angles(i)=d2rad*theta(i)
0085 asave(i)=angles(i)
0086 1 continue
0087 do 251 i=1,ndof
0088 thedot(i)=0.
0089 thdot(i)=0.
0090 251 continue
0091 c Iteratively converge towards an optimal joint configuration.
0092 number=0
0093 do 100 i=1,ndof
0094 if (stop2.or.number.gt.nmax) goto 200
0095 number=number+1
0096 do 2 i=1,ndof
0097 theta(i)=angles(i)/d2rad
0098 2 continue
0099 if (stop2.or.number.gt.nmax) goto 200
0100 number=number+1
0101 do 251 i=1,ndof
0102 thedot(i)=0.
0103 thdot(i)=0.
0104 251 continue
```

Calculate tool Jacobian.

call jacob(ident,ident,nlen,alpha,dist,theta,jtype,ndof,tnet,Jac)

Calculate the equivalent angle and axis of rotation for the current frame orientation. (see Paul, pp.25-33)

if (rotflg) then
  do 222 i=1,3
    do 333 j=1,3
      tinv(i,j)=tnet(j,i)
    333 continue
  222 continue
  call gmult(rotdes,tinv,tmat,3,3,3,3,3)

  term(1)=(tmat(3,2)-tmat(2,3))
  term(2)=(tmat(1,3)-tmat(3,1))
  term(3)=(tmat(2,1)-tmat(1,2))
  term(4)=tmat(1,1)+tmat(2,2)+tmat(3,3)-1

  ang=atan2(sqrt(term(1)**2+term(2)**2+term(3)**2),term4)
  cosang=cos(ang)

  if (ang.gt.toler4) then
    if (ang.lt.1.5) then
      denom=2.*sinang
      do 110 i=1,3
        k(i)=term(i)/denom
      110 continue
      else
        kmax=0.
        do 111 i=1,3
          ttemp=tmat(1,i)-cosang
          if (ttemp.gt.0.) then
            k(i)=sign(1.,term(i))*sqrt(ttemp)/(1.-cosang)
          else
            k(i)=0.
          endif
          if (abs(k(i)).gt.kmax) then
            kmax=abs(k(i))
            isave=1
          endif
        111 continue
    endif
  else
    denom=2.*k(1)*(1.-cosang)
    k(2)=(tnet(2,1)+tnet(1,2))/denom
    k(3)=(tnet(3,1)+tnet(1,3))/denom
  endif
else if (isave.eq.2) then
  denom=2.*k(2)*(1.-cosang)
  k(1)=(tnet(2,1)+tnet(1,2))/denom
  k(3)=(tnet(3,2)+tnet(2,3))/denom
else
  denom=2.*k(3)*(1.-cosang)
It (1) = (\text{tnet}(3,1) + \text{tnet}(1,3)) / \text{denom}
\kern1em k(2) = (\text{tnet}(3,2) + \text{tnet}(2,3)) / \text{denom}
\kern1em \text{endif}
\kern1em \text{endif}
\kern1em \text{else}
\kern1em \text{ang}=0.
\kern1em \text{endif}
\kern1em \text{endif}
\kern1em \text{endif}
\kern1em \text{do} 223 \text{i}=1,3
\kern1em \text{rot}(i)=k(i) \cdot \text{ang}
\kern1em \text{continue}
\kern1em \text{endif}
\kern1em \text{c}
\kern1em \text{The error is the difference between the tool position for the current iteration and the desired tool position.}
\kern1em \text{c}
\kern1em \text{Save the previous position error.}
\kern1em \text{c}
\kern1em \text{do} 300 \text{i}=1,6
\kern1em \text{xdot2}(i)=\text{xdot}(i)
\kern1em \text{continue}
\kern1em \text{c}
\kern1em \text{Calculate the vector difference between the desired and actual orientations. This is not accurate if the orientations differ substantially, but the accuracy improves as the difference is reduced.}
\kern1em \text{c}
\kern1em \text{error}=0.
\kern1em \text{if} (\text{rotflg}) \text{then}
\kern1em \text{do} 120 \text{ind}=1,3
\kern1em \text{if} (\text{ixdot}(\text{ind})) \text{then}
\kern1em \text{xdot}(\text{ind})=\text{rot}(\text{ind})
\kern1em \text{error}=\text{error}+\text{rot}(\text{ind})^2
\kern1em \text{endif}
\kern1em \text{continue}
\kern1em \text{endif}
\kern1em \text{c}
\kern1em \text{Calculate the position error.}
\kern1em \text{c}
\kern1em \text{do} 5 \text{i}=1,3
\kern1em \text{xdot}(i+3)=-\text{tnet}(i,4)+\text{goal}(i)
\kern1em \text{error}=\text{error}+\text{xdot}(i+3)^2
\kern1em \text{continue}
\kern1em \text{error} = \sqrt{\text{error}}
\kern1em \text{c}
\kern1em \text{Stop the iterations when the actual gripper position is within a tolerable distance from the desired position and the configuration is optimized.}
if (error.lt.toler.and.thetomog.lt.toler) then
  stop2=.true.
else
  c Load the matrices for the singular value decomposition routine.
  icount=0
  do 20 i=1,6
     if (ixdot(i)) then
        icount=icount+1
        do 21 j=1,ndof
           mat(icount,j)=jact(i,j)
        enddo 21
        continue
     endif
     mattr(icount,ndof+1)=xdot(i)
  enddo 20
  c Perform a singular value decomposition.
  c Calculate the minimum norm joint change which will reduce the error
  c to a specified tolerance.
  call svd(mat,S,UD,VD,10,10,nxdof,ndof,1,0,nxdof)
  c If the matrix is nearly rank deficient, approximate it by a matrix
  c of reduced rank. Physically this means when two link degrees of
  c freedom are nearly dependent, remove one of them.
  c Otherwise, joint displacements can become very large.
  c The number of non-zero singular values is the rank of the matrix.
  c The magnitude of the smallest singular value is a measure of the
  c 'distance' to a matrix of reduced rank.
  do 130 i=1,nxdof
     if (s(i)/s(1).gt.stoler) then
        imax=i
        endif
  enddo 130
  c Load the singular value matrix.
  do 131 i=1,imax
     do 132 j=1,imax
        if (i.eq.j) then
           temp(j,i)=1./s(j)
        else
           temp(j,i)=0.
        endif
     enddo 132
  enddo 131
  do 140 i=1,imax
     temp3(i)=mat(i,ndof+1)
  enddo 140
Calculate the minimum norm solution.

call gmult(temp,temp3,tempi,imax,imax,1,6,6,6)
call gmult(vd,temp,thedot,ndof,imax,1,10,6,10)

calculate the homogeneous solution.
do 800 i=1,ndof
do 801 j=1,ndof
vd(1,j)=vd(1,i)
continue 801
continue 800

call gmult(vd,vdt,homog,ndof,imax,ndof,10,10,10)
do 39 i=1,ndof
homog(i,i)=homog(i,i)+1.
continue 39

call gmult(homog,angles,thedot,ndof,ndof,1,10,10,10)

Adjust the step size to ensure rapid convergence.
The step size is chosen to give a steepest descent iteration when
the error is large, and a rapidly converging Newton-Raphson iteration
when the error is small.

Optimize the solution to maximize joint availability.

Calculate the linearization error and calculate the new step size
factor.
erlin=0.
if (number.eq.1) then
do 310 i=1,6
erlin=erlin+xdot(i)**2
continue 310
erlin=sqrt(erlin)
fmult=amin1((ermax/erlin),1.)
endif

if (erlin.gt.2.*ermax) then
do 303 i=1,ndof
angles(i)=asave(i)
thedot(i)=tsave(i)
continue 303
repeat=.true.
else
repeat=.false.
write(14,* erlin
write(1,* angles(1)/d2rad
write(2,* angles(2)/d2rad
write(3,* angles(3)/d2rad
write(4,* angles(4)/d2rad
endif

Calculate the displacement resulting from maximizing the joint availability.
call gmult(mat1,angles,xdot1,nxdo1,ndof,1,10,10,10)

Calculate the weight of the joint change to increase availability.
The factor is chosen so that the joint change increasing availability
is some fixed fraction of the minimum norm joint change reducing
the error.

do 305 i=1,6
  anorm=anorm+xdotl(i)**2
continue
anorm=sqrt(anorm)
do 351 i=1,6
  tlen=tlen+xdotl(i)**2
continue
tlen=sqrt(tlen)
if (anorm.ne.0. and tlen.gt.toler2) then
  fact=gamma*tlen/anorm
else
  fact=0.
endif

Update the joint variables.

thomog=0.
tlen1=0.
tlen2=0.
terr=0.
do 41 i=1,ndof
  thomog=thomag+thdotl(i)**2
tlen1=tlen1+thdotl(i)**2
tlen2=tlen2+fact*angles(i)**2
  asavel(i)=angles(i)
tsave(i)=thetad(i)
dlast(i)=dangle(i)
dangle(i)+thetad(i)-thdotl(i)*fact*angles(i)
  angles(i)=angles(i)+fmult*dangle(i)
  tterr=terr+(dangle(i)-dlast(i))**2
  tnorm=tnorm+angles(i)**2
continue

Map the angles into the range -180 deg. to 180 deg.
if (angles(i).gt.pi) then
0400     angles(i)=angles(i)-2.*pi
0401     elseif (angles(i).lt.-pi) then
0402     angles(i)=angles(i)+2.*pi
0403     endif
0404     continue
0405
0406     tnorm=sqrt(tnorm)
0407     terr=sqrt(terr)
0408
0409     tlen1=fault*sqrt(tlen1)
0410     tlen2=fault*sqrt(tlen2)
0411     thomog=fault*sqrt(thomog)
0412     tlen3=tlen1+tlen2
0413
0414     endif
0415
0416
0417     if (.not.repeat) then
0418     write(*,*) error
0419     write(5,*) error
0420     write(9,*) fault
0421     write(12,*) tlen1
0422     write(15,*) thomog
0423     write(16,*) tnorm
0424     write(13,*) fact
0425     endif
0426     goto 100
0427     continue
0428
0429     do 900 i=1,ndof
0430     continue
0431
0432     c Convert the angles to degrees.
0433     do 900 i=1,ndof
0434     theta2(i)=angles(i)/d2rad
0435     continue
0436
0437     if (.not.repeat) then
0438     write(*,*) 'number= ',number
0439     close(1)
0440     close(2)
0441     close(3)
0442     close(4)
0443     close(5)
0444     close(6)
0445     close(7)
0446     close(8)
0447     close(9)
0448     close(10)
0449     close(11)
0450     close(12)
0451     close(13)
0452
0453     return
0454     end
## PROGRAM SECTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $CODE</td>
<td>2800</td>
<td>PIC CON REL LCL SHR EXE RD NONR WRT LONG</td>
</tr>
<tr>
<td>1 $DATA</td>
<td>144</td>
<td>PIC CON REL LCL SHR NOEXE RD NONR WRT LONG</td>
</tr>
<tr>
<td>2 $LOCAL</td>
<td>4708</td>
<td>PIC CON REL LCL NOSHR NOEXE RD WRT LONG</td>
</tr>
</tbody>
</table>

Total Space Allocated: 7652

## ENTRY POINTS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000</td>
<td></td>
<td>INVERS</td>
</tr>
</tbody>
</table>

## VARIABLES

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA R4 ANG</td>
<td>AA R4 ANORM</td>
<td>**</td>
<td>R4 COSANG</td>
<td>**</td>
</tr>
<tr>
<td>AA R4 DENOM</td>
<td>AA R4 ERLIN</td>
<td>2-00000E14 R4 ERMAX</td>
<td>2-00000E30 R4 ERROR</td>
<td>**</td>
</tr>
<tr>
<td>IA4 ICOUNT</td>
<td>IA4 IMAX</td>
<td>2-00000E18 R4 GAMMA</td>
<td>2-00000E2C IA4 ISAVE</td>
<td>**</td>
</tr>
<tr>
<td>IA4 J</td>
<td>IA4 KMAX</td>
<td>AP-0000001800 IA4 NOOF</td>
<td>**</td>
<td>IA4 BMX</td>
</tr>
<tr>
<td>IA4 NUMBER</td>
<td>AP-0000001B00 IA4 NOOFF</td>
<td>**</td>
<td>R4 PI</td>
<td>2-00000E08 LA4 REPEAT</td>
</tr>
<tr>
<td>IA4 ROTFLG</td>
<td>IA4 SINANG</td>
<td>2-00000E09 R4 STOLER</td>
<td>2-00000E10 LA4 STOP2</td>
<td></td>
</tr>
<tr>
<td>IA4 TERN4</td>
<td>2-00000E34 R4 TERN</td>
<td>2-00000E1C IA4 THOMOG</td>
<td>**</td>
<td>RA4 TLEN</td>
</tr>
<tr>
<td>IA4 TLEN</td>
<td>IA4 TLEN2</td>
<td>2-00000E44 R4 TLEN2</td>
<td>**</td>
<td>RA4 TOLER</td>
</tr>
<tr>
<td>AA R4 TOLER2</td>
<td>**</td>
<td>AA R4 TOLER4</td>
<td>**</td>
<td>R4 TTEMP</td>
</tr>
</tbody>
</table>

## ARRAYS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP-0000000B8 R4 ALPHA</td>
<td>** (4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-00000598 R4 ANGLES</td>
<td>40 (10)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-00000570 R4 ASAVE</td>
<td>40 (10)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-0000048B R4 DANGLE</td>
<td>40 (10)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AP-0000000C0 B4 DIST</td>
<td>** (4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-00000490 R4 DLAST</td>
<td>40 (10)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-00000928 R4 HOMOG</td>
<td>400 (10, 10)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-00000704 R4 IDENT</td>
<td>64 (4, 4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AP-00000002B8 IA4 IDOT</td>
<td>24 (6)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-00000614 R4 JAC</td>
<td>240 (6, 10)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-00000D18 R4 JACINV</td>
<td>240 (10, 6)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AP-00000014B IA4 JTYPE</td>
<td>16 (4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-000005C0 R4 K</td>
<td>12 (3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-000005CC R4 KGAL</td>
<td>12 (3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AP-000000048 B4 LEN</td>
<td>** (4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-000002DB R4 MAT</td>
<td>440 (10, 11)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-00000AE0 R4 MAT1</td>
<td>400 (10, 10)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-0000078C R4 ROT</td>
<td>12 (3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AP-00000024B R4 ROUTDES</td>
<td>36 (3, 3)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FUNCTIONS AND SUBROUTINES REFERENCED

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOR$CLOSE</td>
<td>MTH$SIGN</td>
<td>FOR$OPEN</td>
<td>MTH$SIN</td>
<td>GMULT</td>
<td>JACOB</td>
<td>R$4</td>
<td>MTH$ATAN2</td>
</tr>
<tr>
<td>R$4</td>
<td>MTH$SIGN</td>
<td>R$4</td>
<td>MTH$SIN</td>
<td>R$4</td>
<td>MTH$SQR</td>
<td>R$4</td>
<td>MTH$COS</td>
</tr>
</tbody>
</table>

COMMAND QUALIFIERS

FOR/LIST OPT.FOR, RDFILE.FOR, INPUT.FOR, GROW.FOR, FINPAT.FOR, SOLVE1.FOR, INT2D.FOR, HULL2D.FOR, CMULT.FOR, JACOB.FOR, COSTT.FOR,

/ CHECK=(NOUNDEFFLOH, OVERFLOW, NOUNDERFLOH)
/DEBUC=(NOSYMBOLS, TRACEBACK)
/STANDARD=(NOSYNTAX, NOSOURCE FORM)
/SHOW=(NOPREPROCESSOR, NOINCLUDE, MAP, NODECTIONARY, SINGLE)
/WARNINGS=(GENERAL, NODECLARATIONS)
/CONTINUATIONS=19 /MOCROSS_REFERENCE /MO_LINES /NOEXTEND_SOURCE /F77
/NOMULTI_FLOATING /NOMACHINE_CODE /OPTIMIZE
subroutine jacob(tpre, tpost, len, alpha, dth, dtheta, jtype, nrdof, tnet, jac)

Calculates the Jacobian matrix for a given manipulator configuration.

Parameters:
- tpre: world to robot base coordinate transformation
- tpost: transformation from last link frame to a point
- len, alpha, dth, dtheta: Denavit-Hartenberg link parameters
- jtype: array specifying joint types (0-revolute, 1-prismatic)
- nrdof: number of links
- tnet: net transformation matrix relating the frame of interest on the last link to the world frame
- jac: Jacobian matrix

real tpre(4,4), tpost(4,4)
real amat(4,4)
real len(*), alpha(*), dth(*), dtheta(*)
real jac(6,*)
integer jtype(*)
real tnet(4,4), T(4,4), p(3,10), rdist(3)
real temp(4,4)

Load transformation relating robot base coordinates to world coordinates.
do 200 i=1,4
   do 201 j=1,4
      T(i,j)=tpre(j,i)
   continue
200 continue

d2rad=.0174533

Recursively calculate elements of Jacobian matrix.
do 10 ind=1,nrdof
   theta=dtheta(ind)*d2rad
   amat(1,1)=cos(theta)
   amat(2,1)=sin(theta)
   amat(1,2)*=sin(theta)*cos(alpha(ind))
   amat(2,2)*=cos(alpha(ind))
   amat(3,2)*=sin(alpha(ind))
   amat(1,3)*=sin(theta)*sin(alpha(ind))
   amat(2,3)*=-cos(theta)*sin(alpha(ind))
   amat(3,3)*=cos(alpha(ind))
   amat(1,4)*=len(ind)*cos(theta)
   amat(2,4)*=len(ind)*sin(theta)
   amat(3,4)*=dist(ind)
   amat(4,4)=1,
call gmult(T,amat,Temp,4,4,4,4,4,4,4,4,4,4)
do 300 i=1,4
do 301 j=1,4
   t(j,j)=temp(j,1)
   continue
   continue
   if (jtype(ind).eq.0) then
      Revolute joint.
do 20  ind2=1,3
      jac(ind2,ind)=T(ind2,3)
P(ind2,ind)=T(ind2,4)
      continue
   endif
   else
      Prismatic joint.
do 30  ind2=1,3
      jac(ind2,ind)=0.
jac(ind2+3,ind)=T(ind2,3)
      continue
   endif
   continue
   Postmultiply by fixed transformation relating the last link to the tool frame.
call gmult(T,tnet,Tpost,4,4,4,4,4,4,4,4,4,4)
The net transformation 'tnet' relates the tool frame to the world reference frame. In other words, 'tnet' contains the tool position and orientation.
calculate Jacobian elements which depend on the position of the tool frame.
do 40  ind=1,nr dof
   if (jtype(ind).eq.0) then
      if (ind.eq.1) then
         do 50 i=1,3
            rdist(i)=-tnet(i,4)
         continue
         else
            do 55 i=1,3
               rdist(i)=P(i,ind-1)-tnet(i,4)
            continue
         endif
         endif
         continue
      endif
      endif
   endif
   else
      Jac(4,ind)=Jac(2,ind)*rdist(3)+Jac(3,ind)*rdist(2)
```
PROGRAM SECTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0  $CODE</td>
<td>652</td>
<td>PIC CON REL LCL SHR EXE RD NONRT LONG</td>
</tr>
<tr>
<td>1  $DATA</td>
<td>4</td>
<td>PIC CON REL LCL SHR NOEXE RD NONRT LONG</td>
</tr>
<tr>
<td>2  $LOCAL</td>
<td>592</td>
<td>PIC CON REL LCL NOSHR NOEXE RD NRT LONG</td>
</tr>
</tbody>
</table>

Total Space Allocated: 1248

ENTRY POINTS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000</td>
<td></td>
<td>JACOB</td>
</tr>
</tbody>
</table>

VARIABLES

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>R4</td>
<td>D2RAD</td>
<td>**</td>
<td>I4</td>
<td>I</td>
<td>**</td>
<td>I4</td>
<td>IND</td>
</tr>
<tr>
<td>**</td>
<td>I4</td>
<td>J</td>
<td>AP-00000020B</td>
<td>I4</td>
<td>NRDOF</td>
<td>**</td>
<td>R4</td>
<td>THETA</td>
</tr>
</tbody>
</table>

ARRAYS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP-00000010R</td>
<td>R4</td>
<td>ALPHA</td>
<td>**</td>
<td>(A)</td>
</tr>
<tr>
<td>2-00000000R</td>
<td>R4</td>
<td>AMAT</td>
<td>64</td>
<td>(4, 4)</td>
</tr>
<tr>
<td>AP-00000014R</td>
<td>R4</td>
<td>DIST</td>
<td>**</td>
<td>(A)</td>
</tr>
<tr>
<td>AP-00000018R</td>
<td>R4</td>
<td>DTHETA</td>
<td>**</td>
<td>(A)</td>
</tr>
<tr>
<td>AP-00000020R</td>
<td>R4</td>
<td>JAC</td>
<td>**</td>
<td>(6, A)</td>
</tr>
<tr>
<td>AP-0000001CR</td>
<td>I4</td>
<td>JTYPE</td>
<td>**</td>
<td>(A)</td>
</tr>
<tr>
<td>AP-000000CCR</td>
<td>R4</td>
<td>LEN</td>
<td>**</td>
<td>(A)</td>
</tr>
<tr>
<td>2-0000000UR</td>
<td>R4</td>
<td>P</td>
<td>120</td>
<td>(3, 10)</td>
</tr>
<tr>
<td>2-000000FB</td>
<td>R4</td>
<td>RDIST</td>
<td>12</td>
<td>(3)</td>
</tr>
<tr>
<td>2-00000049R</td>
<td>R4</td>
<td>T</td>
<td>64</td>
<td>(4, 4)</td>
</tr>
<tr>
<td>2-00000014R</td>
<td>R4</td>
<td>TEMP</td>
<td>64</td>
<td>(4, 4)</td>
</tr>
<tr>
<td>AP-00000024R</td>
<td>R4</td>
<td>TNET</td>
<td>64</td>
<td>(4, 4)</td>
</tr>
<tr>
<td>AP-00000008R</td>
<td>R4</td>
<td>TPOST</td>
<td>64</td>
<td>(4, 4)</td>
</tr>
<tr>
<td>AP-00000004R</td>
<td>R4</td>
<td>TPRE</td>
<td>64</td>
<td>(4, 4)</td>
</tr>
</tbody>
</table>
```
LABELS

<table>
<thead>
<tr>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>30</td>
<td>30</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>201</td>
<td>201</td>
<td>300</td>
<td>300</td>
<td>301</td>
<td>301</td>
</tr>
</tbody>
</table>

FUNCTIONS AND SUBROUTINES REFERENCED

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GMULT</td>
<td></td>
<td>R*4</td>
</tr>
<tr>
<td></td>
<td>MTH*COS</td>
<td></td>
<td>MTH*SIN</td>
</tr>
</tbody>
</table>

COMMAND QUALIFIERS

FOR/LIST OPT.FOR, REFILE.FOR, INPUT.FOR, GROW.FOR, FINPAT.FOR, SOLVE1.FOR, INT2D.FOR, HULL2D.FOR, GMULT.FOR, JACOB.FOR, COSTT.FOR,

/CHECK=(NOBOUNDS, OVERFLOW, RUNDERFLOW)
/DEBUG=(NOSYMBS, TRACERACK)
/STANDARD=(NOSYNTAX, NOSOURCE_FORM)
/SHOW=(NOPREPROCESSOR, NOINCLUDE, MAP, NODITIONARY, SINGLE)
/CONTINUATIONS=19 /NO CROSSREFERENCE /NO LINES /NO EXTEND SOURCE /F77
/NOC FLOATING /I4 /NOMACHINE CODE /OPTIMIZE

COMPILATION STATISTICS

Run Time: 5.00 seconds
Elapsed Time: 14.14 seconds
Page Faults: 667
Dynamic Memory: 845 pages
program minver
      c Perform iterative inverse kinematic solution and show the results
graphically.
      c
      real theta(4,10)
      real box1(2,10,10), box2(2,10,10)
      integer links(4)
      logical ierase, stop
      real xdot(3), xgoal(3), xcurr(3)
      real alpha(4), len(4), dist(4)
      integer jtype(4), link(4)
      logical idxdot(6)
      real kgoal(3)
      real theta1(4)
      real theta2(4)
      real rotdes(3,3)
      real xpos(3)
      integer jtypel(4), linkf(4)
      logical lxdot(6)
      real kgoal(3)
      c Read the geometry of the links and obstacles from a file.
      open(1, file='box.dat', status='old')
      read(1,*) nlinks
      do 29 k=1, nlinks
         read(1,*) links(k)
      29 continue
      close(1)
      c
      write(*,*) 'Enter initial configuration angles'
      read(*,*) theta(i, npos), i=1, ndof
      c
      do 25 i=1, ndof
         theta1(i)=theta(i, npos)
      25 continue
      c
      d2rad=.0174533
continue

Initialize the graphics environment.
call grstrt(4027,1)
call newpag
call window(-4.,4.,-4.,4.)
call wport(0.,100.,0.,100.)

first=.true.
erase=.true.
final=.true.
continue

if (.not.first) then
   stop=.true.
call clos

write(*,*) 'Enter goal position and orientation'
read(*,*) xpos(1),xpos(2),angle
   angle=angle*2*rad

Find an optimal inverse solution for the desired position.
call rotaat(kgoal,angle,rotdes)
call invers(len,alpha,dist,theta1,jtype,ndof,nxdo,ldof,ldot,rotdes,
xpos,theta2)
write(*,*) 'X,Y pos, angle ',xpos(1),xpos(2),angle
   do 23 i=1,ndof
      thetal(i)=theta2(i)
   continue

Draw the link positions.
do 15 ipos=1,npos
call tridnt(.false.)
   ilink=1
   do 10 ind=1,ndof
      angle=thetad(i)
call rotate(angle,angle)
call transl(len(ind),0.)
call move(0.,0.)
call dashpt(1)
call draw(-len(ind),0.)
call dashpt(0)
if (ind.eq.ndof) then
  call move(0.,0.)
  call draw(.2,.2)
  call move(0.,0.)
  call draw(-.2,-.2)
endif
if (ind.eq.llnks(ilink)) then
  call move(boxl(1,1,ilink),boxl(2,1,ilink))
  call draw(boxl(1,1,ilink),boxl(2,1,ilink))
  call draw(boxl(1,3,ilink),boxl(2,3,ilink))
  call draw(boxl(1,4,ilink),boxl(2,4,ilink))
  call draw(boxl(1,1,ilink),boxl(2,1,ilink))
  illnk=ilink+1
endif
continue
if (.not.stop) goto 100
continue
first=.false.
if (.not.stop) goto 100
continue
call grstop
stop
end
### Program Sections

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 CODE</td>
<td>811</td>
<td>PIC CON REL LCL SHR EXE RD NONRT LONG</td>
</tr>
<tr>
<td>1 $FDATA</td>
<td>177</td>
<td>PIC CON REL LCL SHR NOEXE RD NONRT LONG</td>
</tr>
<tr>
<td>2 LOCAL</td>
<td>2096</td>
<td>PIC CON REL LCL NOSHR NOEXE RD NONRT LONG</td>
</tr>
<tr>
<td>3 ROBOT</td>
<td>240</td>
<td>PIC OVR REL GBL SHR NOEXE RD NONRT LONG</td>
</tr>
<tr>
<td>4 NRDOF</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOEXE RD NONRT LONG</td>
</tr>
</tbody>
</table>

Total Space Allocated: 3332

### Entry Points

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000</td>
<td>MINVER</td>
<td></td>
</tr>
</tbody>
</table>

### Variables

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-000006F4</td>
<td>RA4</td>
<td>ANGLE</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td></td>
<td>AA</td>
<td>I</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td></td>
<td>AA</td>
<td>IND</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-000006EC</td>
<td>RA4</td>
<td>NLINKS</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-000006F0</td>
<td>IA4</td>
<td>STOP</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td></td>
<td>AA</td>
<td>FIRST</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td></td>
<td>AA</td>
<td>ILINK</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td></td>
<td>AA</td>
<td>K</td>
<td>16</td>
<td>(4)</td>
</tr>
</tbody>
</table>

### Arrays

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-00000200</td>
<td>RA4</td>
<td>ALPHA</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00000000</td>
<td>RA4</td>
<td>BOX1</td>
<td>800</td>
<td>(2, 10, 10)</td>
</tr>
<tr>
<td>2-00000320</td>
<td>RA4</td>
<td>BOX2</td>
<td>800</td>
<td>(2, 10, 10)</td>
</tr>
<tr>
<td>3-000000E0</td>
<td>RA4</td>
<td>DIST</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00000674</td>
<td>LA4</td>
<td>IXDOT</td>
<td>24</td>
<td>(6)</td>
</tr>
<tr>
<td>2-00000610</td>
<td>IA4</td>
<td>JTYPE</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-000006BC</td>
<td>RA4</td>
<td>KGOAL</td>
<td>12</td>
<td>(3)</td>
</tr>
<tr>
<td>3-00000030</td>
<td>RA4</td>
<td>LEN</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>3-00000000</td>
<td>IA4</td>
<td>LINK</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00000640</td>
<td>IA4</td>
<td>LINKS</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-000006BB</td>
<td>RA4</td>
<td>RUTDES</td>
<td>36</td>
<td>(3, 3)</td>
</tr>
<tr>
<td>3-00000040</td>
<td>RA4</td>
<td>THETA1</td>
<td>160</td>
<td>(4, 10)</td>
</tr>
<tr>
<td>2-0000069B</td>
<td>RA4</td>
<td>THETA1</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-0000069A</td>
<td>RA4</td>
<td>THETA2</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-0000066B</td>
<td>RA4</td>
<td>XCURR</td>
<td>12</td>
<td>(3)</td>
</tr>
<tr>
<td>2-00000650</td>
<td>RA4</td>
<td>XDOT</td>
<td>12</td>
<td>(3)</td>
</tr>
<tr>
<td>2-0000065C</td>
<td>RA4</td>
<td>XGOAL</td>
<td>12</td>
<td>(3)</td>
</tr>
<tr>
<td>2-000006DC</td>
<td>RA4</td>
<td>XPOS</td>
<td>12</td>
<td>(3)</td>
</tr>
</tbody>
</table>
### LABELS

<table>
<thead>
<tr>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000190</td>
<td><strong>10</strong></td>
<td>00000220</td>
<td><strong>13</strong></td>
</tr>
<tr>
<td><strong>15</strong></td>
<td><strong>23</strong></td>
<td><strong>25</strong></td>
<td><strong>29</strong></td>
</tr>
</tbody>
</table>

### FUNCTIONS AND SUBROUTINES REFERENCED

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>NAME</td>
<td>TYPE</td>
<td>NAME</td>
</tr>
<tr>
<td>CMCLOS</td>
<td>CMOPEN</td>
<td>DASHPT</td>
<td>DRAM</td>
</tr>
<tr>
<td>CMSTOP</td>
<td>CMSTOP</td>
<td>INVERS</td>
<td>MOVE</td>
</tr>
<tr>
<td>NEWPAG</td>
<td>NEWPAG</td>
<td>RDFILE</td>
<td>RDFILE</td>
</tr>
<tr>
<td>POINT</td>
<td>POINT</td>
<td>WINDOW</td>
<td>WINDOW</td>
</tr>
</tbody>
</table>
This program finds an optimal robot trajectory by minimizing a cost functional. Interference and joint limit constraints are represented by penalty functions.

The current implementation applies to 2-D planar manipulators where objects are represented by convex polygons. The convex hull enclosing two adjacent link positions is used for the interference calculations.

The initial configuration is given, and the final configuration (the goal position) must satisfy the final position constraint.

The cost functional is the integral of the joint velocities and accelerations over the path, plus a penalty function for violating joint and penetration limits.

```fortran
program opt

integer jtype(4),link
real xgoal(3)
real values(4,100)
integer boxes(2,10)
real averts(2,100),tverts(2,100)
logical intraj
integer lindof(2,20)
real maxpen,jtoler
real kgoal(3)
real rotdes(3,3)
logical ixdot(6)
common /robot/link,jtype,alpha,len,theta,dist
common /nmax/nmax
common /goal/xgoal,angoal
common /last/last
common /objs/boxes,sverts,tverts
common /nobjs/nlinks,nobst
common /lindof/lindof
common /nrdof/ndof

Start the elapsed CPU timer.
base_time=get_cpu_time(0.)

For the current implementation, 2-D rotation about x-axis only.
kgoal(1)=0.
kgoal(2)=0.
kgoal(3)=1.

Degrees of freedom are rotation about z, translation along x,y.
```
ixdot(3) = .true.
ixdot(4) = .true.
ixdot(5) = .true.

c Read kinematic parameters file.
call rdfile

User selects numerical optimisation parameters.
call input

c Read the goal position and initial trajectory from a file.
onen(1, file='traj.dat', status='old')
read(1, a) (xgoal(i), i=1,3), angoal
read(1, a) last
do 100 i=1, last
read(1, a) (values(j, i), j=1, ndof)
100 continue

c Read the 2-D obstacle geometry data from a file.
onen(1, file='box.dat', status='old')
read(1, a) nlinks
istart=1
do 5 =1, nlinks
read(1, a) nverts, idof
if (lindof(1, idof).eq.0) then
lindof(1, idof) = 1
endif
lindof(2, idof) = lindof(2, idof) + 1
boxes(1, i) = istart
boxes(2, i) = nverts
do 6 j = istart, istart + nverts - 1
read(1, a) (verts(k, j), k=1, 2)
6 continue
istart = istart + nverts
5 continue
read(1, a) nobet

do 7 = 1, nobet
read(1, a) nverts
boxes(1, nlinks + i) = istart
boxes(2, nlinks + i) = nverts
do 8 j = istart, istart + nverts - 1
read(1, a) (verts(k, j), k=1, 2)
8 continue
istart = istart + nverts
7 continue
7 close(1)
8 close(1)

Grow the obstacles by the penetration threshold.
call grow(maxpen)
0115
0116 c Generate a path from the given initial trajectory.
0117
0118 c call finpatt(values)
0119
0120 c Get the total elapsed CPU time.
0121
0122 c time= get_cpu_time(base_time)
0123
0124 write(*,*) 'Elapsed CPU time= ',etime
0125
0126 stop
0127 end

PROGRAM SECTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $CODE</td>
<td>644</td>
<td>PIC CON REL LCL SHR EDX RD NOSPRT LONG</td>
</tr>
<tr>
<td>1 $PDATA</td>
<td>39</td>
<td>PIC CON REL LCL SHR NOEXE RD NOSPRT LONG</td>
</tr>
<tr>
<td>2 $LOCAL</td>
<td>1844</td>
<td>PIC CON REL LCL NOSH SHR NOEXE RD NOSPRT LONG</td>
</tr>
<tr>
<td>3 ROBOT</td>
<td>96</td>
<td>PIC OVR REL GBL SHR NOEXE RD NOSPRT LONG</td>
</tr>
<tr>
<td>4 NMAX</td>
<td>4</td>
<td>PIC OVR REL GBL SHR NOEXE RD NOSPRT LONG</td>
</tr>
<tr>
<td>5 GOAL</td>
<td>16</td>
<td>PIC OVR REL GBL SHR NOEXE RD NOSPRT LONG</td>
</tr>
<tr>
<td>6 LAST</td>
<td>4</td>
<td>PIC OVR REL GBL SHR NOEXE RD NOSPRT LONG</td>
</tr>
<tr>
<td>7 OBJS</td>
<td>1680</td>
<td>PIC OVR REL GBL SHR NOEXE RD NOSPRT LONG</td>
</tr>
<tr>
<td>8 NENTS</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOEXE RD NOSPRT LONG</td>
</tr>
<tr>
<td>9 LINDOF</td>
<td>160</td>
<td>PIC OVR REL GBL SHR NOEXE RD NOSPRT LONG</td>
</tr>
<tr>
<td>10 LIMTOL</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOEXE RD NOSPRT LONG</td>
</tr>
<tr>
<td>11 NDOF</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOEXE RD NOSPRT LONG</td>
</tr>
</tbody>
</table>

Total Space Allocated 4511

ENTRY POINTS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000</td>
<td>OPT</td>
<td></td>
</tr>
</tbody>
</table>

VARIABLES

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-00000000</td>
<td>R4</td>
<td>ANG0AL</td>
</tr>
<tr>
<td>2-00000068</td>
<td>R4</td>
<td>BASE_TIME</td>
</tr>
<tr>
<td>6-00000000</td>
<td>R4</td>
<td>ETIME</td>
</tr>
<tr>
<td>2-00000069</td>
<td>I4</td>
<td>IDOF</td>
</tr>
<tr>
<td>10-00000004</td>
<td>I4</td>
<td>JTOLER</td>
</tr>
<tr>
<td>11-00000000</td>
<td>I4</td>
<td>NDOF</td>
</tr>
<tr>
<td>2-0000006C</td>
<td>I4</td>
<td>NVERTS</td>
</tr>
<tr>
<td>6-00000000</td>
<td>I4</td>
<td>K</td>
</tr>
<tr>
<td>6-00000000</td>
<td>I4</td>
<td>LAST</td>
</tr>
<tr>
<td>8-00000000</td>
<td>I4</td>
<td>NLINKS</td>
</tr>
<tr>
<td>4-00000000</td>
<td>I4</td>
<td>NHAX</td>
</tr>
<tr>
<td>8-00000004</td>
<td>I4</td>
<td>NOBST</td>
</tr>
</tbody>
</table>
### ARRAYS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-00000020</td>
<td>R*4</td>
<td>ALPHA</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>7-00000000</td>
<td>I*4</td>
<td>BOXES</td>
<td>80</td>
<td>(2, 10)</td>
</tr>
<tr>
<td>3-00000050</td>
<td>R*4</td>
<td>DIST</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00000670</td>
<td>L*4</td>
<td>IXDOT</td>
<td>24</td>
<td>(6)</td>
</tr>
<tr>
<td>3-00000100</td>
<td>I*4</td>
<td>JTYPE</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00000640</td>
<td>R*4</td>
<td>KGOAL</td>
<td>12</td>
<td>(3)</td>
</tr>
<tr>
<td>3-00000030</td>
<td>R*4</td>
<td>LEN</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>9-00000000</td>
<td>I*4</td>
<td>LINDOF</td>
<td>160</td>
<td>(2, 20)</td>
</tr>
<tr>
<td>3-00000000</td>
<td>I*4</td>
<td>LINK</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00000640</td>
<td>R*4</td>
<td>ROTDES</td>
<td>36</td>
<td>(3, 3)</td>
</tr>
<tr>
<td>7-00000050</td>
<td>R*4</td>
<td>SVERTS</td>
<td>800</td>
<td>(2, 100)</td>
</tr>
<tr>
<td>3-00000040</td>
<td>R*4</td>
<td>THETA</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>7-00000370</td>
<td>R*4</td>
<td>TVERTS</td>
<td>800</td>
<td>(2, 100)</td>
</tr>
<tr>
<td>2-00000000</td>
<td>R*4</td>
<td>VALUES</td>
<td>1600</td>
<td>(4, 100)</td>
</tr>
<tr>
<td>5-00000000</td>
<td>R*4</td>
<td>XGOAL</td>
<td>12</td>
<td>(3)</td>
</tr>
</tbody>
</table>

### LABELS

<table>
<thead>
<tr>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>5</td>
<td>AA</td>
<td>6</td>
<td>AA</td>
<td>7</td>
<td>AA</td>
<td>8</td>
<td>AA</td>
<td>100</td>
</tr>
</tbody>
</table>

### FUNCTIONS AND SUBROUTINES REFERENCED

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FINPAT</td>
<td></td>
<td>RDFILE</td>
<td></td>
<td></td>
<td></td>
<td>FORCLOSE</td>
<td></td>
<td>FOROPEN</td>
<td></td>
<td>R*4, GET_CPU_TIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GROW</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>INPUT</td>
</tr>
</tbody>
</table>
block data
real ident(4,4)
integer lindof(2,10)
real d2rad

common /ident/ident
common /lindof/lindof
common /d2rad/d2rad

data ident/1.40.,1.,40.,1.,40.,1./
data lindof/20A0/
data d2rad/.0174533/
end

PROGRAM SECTIONS

Name Bytes Attributes
0 IDENT 64 PIC OVR REL GBL SHR NOEXEC RD WRIT LONG
1 LINDOF 80 PIC OVR REL GBL SHR NOEXEC RD WRIT LONG
2 D2RAD 4 PIC OVR REL GBL SHR NOEXEC RD WRIT LONG

Total Space Allocated 148

VARIABLES

Address Type Name
2-00000000 R*4 D2RAD

ARRAYS

Address Type Name Bytes Dimensions
0-00000000 R*4 IDENT 64 (4, 4)
1-00000000 I*4 LINDOF 80 (2, 10)
subroutine rdfile

Read the contents of the setup files and initialize the variables.
real alpha(4), len(4), dist(4), theta(4)
integer jtype(4), link(4)
common /robot/link, jtype, alpha, len, dist
common /nrdof/n dof, nx dof
common /size/size

Read the contents of the kinematics parameters file.
open(1, file='param.dat', status='old')

Read the number of joint and task coordinates.
read(1,*)
read(1,*) n dof, nx dof

Read the workstation size (units).
read(1,*)
read(1,*) size

Read the robot joint type array.
read(1,*)
read(1,*) (jtype(i), i=1, ndof)

Read the link twists.
read(1,*)
read(1,*) (alpha(i), i=1, ndof)

Read the link lengths.
read(1,*)
read(1,*) (len(i), i=1, ndof)

Read the link joint offsets.
read(1,*)
read(1,*) (dist(i), i=1, ndof)

close(1)

return
end
### PROGRAM SECTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CODE</td>
<td>361</td>
<td>PIC CON REL LCL</td>
</tr>
<tr>
<td>$DATA</td>
<td>11</td>
<td>PIC CON REL LCL</td>
</tr>
<tr>
<td>$LOCAL</td>
<td>72</td>
<td>PIC CON REL LCL</td>
</tr>
<tr>
<td>ROBOT</td>
<td>96</td>
<td>PIC OVR REL GBL</td>
</tr>
<tr>
<td>SIZE</td>
<td>4</td>
<td>PIC OVR REL GBL</td>
</tr>
</tbody>
</table>

**Total Space Allocated**: 552

### ENTRY POINTS

<table>
<thead>
<tr>
<th>Address Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000</td>
<td>RDFILE</td>
</tr>
</tbody>
</table>

### VARIABLES

<table>
<thead>
<tr>
<th>Address Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-00000000</td>
<td>NDOF</td>
</tr>
<tr>
<td>4-00000004</td>
<td>NXDOF</td>
</tr>
<tr>
<td>5-00000000</td>
<td>R4</td>
</tr>
</tbody>
</table>

### ARRAYS

<table>
<thead>
<tr>
<th>Address Type</th>
<th>Name</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-00000020</td>
<td>R4</td>
<td>16 (4)</td>
</tr>
<tr>
<td>3-00000050</td>
<td>R4</td>
<td>16 (4)</td>
</tr>
<tr>
<td>3-00000010</td>
<td>R4</td>
<td>16 (4)</td>
</tr>
<tr>
<td>3-00000030</td>
<td>R4</td>
<td>16 (4)</td>
</tr>
<tr>
<td>3-00000000</td>
<td>R4</td>
<td>16 (4)</td>
</tr>
</tbody>
</table>

### FUNCTIONS AND SUBROUTINES REFERENCED

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Type Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORCLOSE</td>
<td>FOROPEN</td>
</tr>
</tbody>
</table>
subroutine rotnat(kaxis, theta, rotat)
  c  Transform a generalized rotation axis and angle into the equivalent
  c  rotation matrix.
  c  See Paul p. 30 for equations.
  real kaxis(3), theta, rotat(3,3)
  temp1=1.-cos(theta)
  temp2=cos(theta)
  temp3=sin(theta)
  rotat(1,1)=kaxis(1)*temp1+temp2
  rotat(2,1)=kaxis(1)*kaxis(2)*temp1+kaxis(3)*temp2
  rotat(3,1)=kaxis(1)*kaxis(3)*temp1-kaxis(2)*temp3
  rotat(1,2)=kaxis(2)*temp1-kaxis(3)*temp2
  rotat(2,2)=kaxis(2)*kaxis(3)*temp1-kaxis(1)*temp3
  rotat(3,2)=kaxis(2)*kaxis(1)*temp1+kaxis(3)*temp3
  rotat(1,3)=kaxis(3)*temp1+kaxis(1)*temp3
  rotat(2,3)=kaxis(3)*kaxis(2)*temp1-kaxis(1)*temp2
  rotat(3,3)=kaxis(3)*kaxis(3)*temp1+temp2
  return
end

PROGRAM SECTIONS
Name Bytes Attributes
$CODE 210 PIC CON REL LCL SHR EXE RD NONRT LONG
$LOCAL 40 PIC CON REL LCL NOGRT NOEXE RD WRT LONG

Total Space Allocated 250

ENTRY POINTS
Address Type Name
0-00000000 ROTMAT

VARIABLES
Address Type Name Address Type Name Address Type Name Address Type Name
** R*4 TEMP1 ** R*4 TEMP2 ** R*4 TEMP3 AP-000000009 R*4 THETA
ARRAYS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP-0000000040</td>
<td>R*4</td>
<td>KAXIS</td>
<td>12</td>
<td>(3)</td>
</tr>
<tr>
<td>AP-00000000CB</td>
<td>R*4</td>
<td>ROTAT</td>
<td>36</td>
<td>(3, 3)</td>
</tr>
</tbody>
</table>

FUNCTIONS AND SUBROUTINES REFERENCED

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>R*4</td>
<td>MTH6COS</td>
<td>R*4</td>
<td>MTH6SIN</td>
</tr>
</tbody>
</table>

COMMAND QUALIFIERS

FOR / LIST OPT. FOR, RFILE. FOR, INPUT. FOR, GROW. FOR, PINPAT. FOR, SOLVE1. FOR, INT2D. FOR, HULL2D. FOR, CMULT. FOR, JACOB. FOR, COSTT. FOR,

/ CHECK = ( NOBOUND, OVERFLOW, ONEUNDERFLOW )
/ DEBUG = ( NOSYMBOLS, TRACEBACK )
/ STANDARD = ( NOSYNTAX, NOSOURCE_FORM )
/ SHOW = ( NOPREPROCESSOR, NOINCLUDE, MAP, NODICLARATION, SINGLE )
/ WARNINGS = ( GENERAL, NODECLARATION )
/ CONTINUATIONS = 19 / NOCROSS_REFERENCE / NO_LINES / NOEXTEND SOURCE / F77
/ NOC FLOATING / 14 / NOMACHINE CODE / OPTIMIZE

COMPILATION STATISTICS

Run Time: 2.42 seconds
Elapsed Time: 6.12 seconds
Page Faults: 525
Dynamic Memory: 845 pages
subroutine solvel(mat,mdima,ndim,m,n,homog)

Find the projection onto the nullspace of a matrix using the singular value decomposition.

Parameters:

mat - the (m x n) matrix
mdima - the first dimension of array 'mat'
ndim - the dimension of 'homog'
m,n - the dimensions of the matrix
homog - the projection onto the nullspace of 'mat'

real mat(mdima,*)
s(10),vd(10,10)
real vdt(10,10)
real homog(ndim,ndim)

min=min0(m,n)

Find the singular value decomposition of the matrix.
call svd(mat,s,vd,mdima,10,m,n,0,0,min)
c Calculate the projection matrix which projects any vector onto the null space of the Jacobian.

T

The projection is ( I - vd vd ).
do 885 i=1,n
do 886 j=1,n
vdt(i,j)=vd(i,j)
886 continue
885 continue

call gmult(vd,vdt,homog,n,m,10,10,ndim)
do 39 i=1,n
homog(i,i)=homog(i,i)+1.
39 continue

return
end
PROGRAM SECTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 @CODE</td>
<td>239</td>
<td>PIC CON REL LCL SHR EXE RD NOSHR LONG</td>
</tr>
<tr>
<td>1 @UPDATA</td>
<td>8</td>
<td>PIC CON REL LCL SHR NOEXE RD NOSHR LONG</td>
</tr>
<tr>
<td>2 @LOCAL</td>
<td>1044</td>
<td>PIC CON REL LCL NOSHR NOEXE RD NRT LONG</td>
</tr>
</tbody>
</table>

Total Space Allocated: 1291

ENTRY POINTS

Address  Type  Name
0-00000000  SOLVE1

VARIABLES

Address  Type  Name
** AA  IA4 I
2-00000348 IA4 MIN

ARRAYS

Address  Type  Name  Bytes  Dimensions
AP-000000180 R*4 HOMOG ** (*, *)
AP-000000060 R*4 MAT ** (*, *)
2-000000060 R*4 S 40 (10)
2-00000028 R*4 VD 400 (10, 10)
2-00000188 R*4 VDT 400 (10, 10)

LABELS

Address  Label  Address  Label  Address  Label
**  19  **  885  **  886

FUNCTIONS AND SUBROUTINES REFERENCED

Type  Name  Type  Name
GMULT  SVD
subroutine solve2(isode, mat, mdima, ndim, n, m, nhomog, thedot)

Find the minimum norm inverse kinematic solution.
Solve the system $A \cdot x = b$ where $A$ ($m \times n$), $x$ ($n \times 1$), $b$ ($m \times 1$).

Parameters:
- isode: if .true., return the projection 'homog'.
- mat: the matrix ($A$) to be solved.
- mdima: The first dimension of array 'mat'
- ndim: The dimension of the solution 'thedot' (b).
- m,n: The dimensions of A
- homog: The projection onto the null space of A
- thedot: The minimum norm solution.

real mat(mdima,*), thedot(ndim)
real vd(10,10), s(10), vdt(10,10)
real homog(ndim,ndim)
real toler=.00001
integer min0(m,n)

Perform singular value decomposition.
call avd(mat, a, ud, vd, adlma, 10, m, n, l, 0, mln)

Derive the solution from the singular values.
do 10 i=1,n
   thedot(i)=0.
do 15 j=1,m
   if (s(j)/s(1).gt.stoler) then
      thedot(i)=thedot(i)+vd(i,j)/s(j)*mat(j,n+1)
   endif
15 continue
10 continue
if (isode.eq.1) then
do 35 i=1,n
   thedot(i)=0.
do 36 j=1,n
   vdt(i,j)=-vd(j,i)
36 continue
35 continue
call gmult(vd, vdt, homog, n, m, n, 10, 10, ndim)
do 100 i=1,n
   homog(i,i)=1.+homog(i,i)
100 continue
endif
PROGRAM SECTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 CODE</td>
<td>383</td>
<td>PIC CON REL LCL SHR EXE RD Nوم GT LONG</td>
</tr>
<tr>
<td>1 PDATA</td>
<td>12</td>
<td>PIC CON REL LCL SHR NONEXE RD Nوم GT LONG</td>
</tr>
<tr>
<td>2 LOCAL</td>
<td>1092</td>
<td>PIC CON REL LCL NOSH R NONEXE RD N 모 T LONG</td>
</tr>
</tbody>
</table>

Total Space Allocated: 1487

ENTRY POINTS

<table>
<thead>
<tr>
<th>Address Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000</td>
<td>SOLVE2</td>
</tr>
</tbody>
</table>

VARIABLES

<table>
<thead>
<tr>
<th>Address Type</th>
<th>Name</th>
<th>Address Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>AP-0000000048</td>
<td>I4</td>
</tr>
<tr>
<td>AA I4 I</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>AP-0000000348</td>
<td>I4</td>
</tr>
<tr>
<td>AA I4 I</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2-0000000348</td>
<td>I4</td>
</tr>
<tr>
<td>AA I4 I</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2-000000034C</td>
<td>R4</td>
</tr>
<tr>
<td>AA R4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2-00000000</td>
<td>R4</td>
</tr>
<tr>
<td>AA R4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2-00000190</td>
<td>R4</td>
</tr>
<tr>
<td>AA S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2-00000200</td>
<td>R4</td>
</tr>
<tr>
<td>AA THE DOT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2-00000000</td>
<td>R4</td>
</tr>
<tr>
<td>AA VD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2-00000100</td>
<td>R4</td>
</tr>
<tr>
<td>AA VDOT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ARRAYS

<table>
<thead>
<tr>
<th>Address Type</th>
<th>Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HOMOC</td>
<td>AA</td>
<td>(A, A)</td>
</tr>
<tr>
<td></td>
<td>MAT</td>
<td>AA</td>
<td>(A, A)</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>40</td>
<td>(10, 10)</td>
</tr>
<tr>
<td></td>
<td>THE DOT</td>
<td>AA</td>
<td>(A)</td>
</tr>
<tr>
<td></td>
<td>VD</td>
<td>400</td>
<td>(10, 10)</td>
</tr>
<tr>
<td></td>
<td>VDOT</td>
<td>400</td>
<td>(10, 10)</td>
</tr>
</tbody>
</table>

LABELS

<table>
<thead>
<tr>
<th>Address Type</th>
<th>Label</th>
<th>Address Type</th>
<th>Label</th>
<th>Address Type</th>
<th>Label</th>
<th>Address Type</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>10</td>
<td>AA</td>
<td>15</td>
<td>AA</td>
<td>35</td>
<td>AA</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>
FUNCTIONS AND SUBROUTINES REFERENCED

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMULT</td>
<td>SVD</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

COMMAND QUALIFIERS

FOR/LIST OPT.FOR, UDIFILE.FOR, INPUT.FOR, GROW.FOR, FIPNAT.FOR, SOLVE1.FOR, INTD2D.FOR, HULL2D.FOR, GMULT.FOR, JACOB.FOR, COSTT.FOR,

/CHECK= (NOBOUNDS, OVERFLOW, UNDERFLOW)
/DEBUG= (NOSTMBOLS, TRACEBACK)
/STANDARD= (NOSYNTAX, NO SOURCE FORM)
/SHOW= (NOPREPROCESSOR, NO INCLUDE, MAP, NO DICTIONARY, SINGLE)
/MESSAGES= (GENERAL, NO DECLARATIONS)
/CONTINUATIONS= 19 / NOCROSS_REFERENCE / MOD_LINES / NOEXTEND_SOURCE / F77
/NOG Floating / I4 / NOMACHINE_CODE / OPTIMIZE

COMPILATION STATISTICS

<table>
<thead>
<tr>
<th>Run Time</th>
<th>2.95 seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed Time</td>
<td>7.45 seconds</td>
</tr>
<tr>
<td>Page Faults</td>
<td>3890</td>
</tr>
<tr>
<td>Dynamic Memory</td>
<td>845 pages</td>
</tr>
</tbody>
</table>
subroutine sort(frac,tmax1,tmax2,vlist,ftable)

Test all the penetration vectors and flag those which are larger than 'frac' times the largest.

Parameters:

frac - the fraction of the largest penetration to be used as a threshold.
tmax1,tmax2 - the magnitudes of the two largest penetrations.

vlist(1,j) - link id
vlist(2,j) - obstacle id
vlist(3,j) - current or previous penetration vector
vlist(4,j) - number of degrees of freedom of link

ftable(i,j) - table of penetration vectors larger than the threshold
i = link id, j = obst id
values:
1 - current penetration only
2 - previous penetration only
3 - both penetrations

real frac
integer ftable(10,10)
real vlen1(10,10),vlen2(10,10)
integer lindof(2,10)
real vlen2(2),vcts(2,2),points(2,2)
integer vlist(4,2)
common /lindof/lindof
common /nobjs/nlinks,nobst
common /nrdof/ndof,nxdof

Find the largest and second largest penetration vectors, and find the lengths of all the penetration vectors.

real tmax1=0.
tmax2=0.
do 10 ind=1,ndof
   istart=lindof(1,ind)
inumb=lindof(2,ind)
   if (istart.ne.0) then
      do 20 ilink=istart,istart+inumb-1
         do 25 ibst=1,nobst
            call cvect(1,ilink,ibst,vlen,iects,points)
         enddo 20
         do 25 ibst=1,nobst
            vlen([ilink,ibst]=vlen(1)
               if (vlen(ind2).gt.tmax1) then
                  tmax2=tmax1
                  tmax1=vlen(ind2)
                  do 55 i=1,4
                     vlist(i,2)=vlist(i,1)
                  continue
elseif (vlen(1,ind).gt.test) then
    teap=vlen(1,ind)
    teap2=vlen(2,ind)
    if (teap.ge.test) then
        if (teap2.ge.test) then
            iftable(1,1,ind)=1
        elseif (teap2.ge.test1) then
            iftable(2,1,ind)=2
        else
            iftable(1,1,ind)=0
        endif
    elseif (teap.ge.test) then
        iftable(1,1,ind)=3
    endif
    continues
endif

beginf
continue
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endi
**PROGRAM SECTIONS**

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $CODE</td>
<td>447</td>
<td>FIC CON REL LCL SHR EXE RD NOHRT LONG</td>
</tr>
<tr>
<td>1 $DATA</td>
<td>4</td>
<td>FIC CON REL LCL SHR NOEXE RD NOHRT LONG</td>
</tr>
<tr>
<td>2 $LOCAL</td>
<td>936</td>
<td>FIC CON REL LCL NOHRT NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>3 $INDOF</td>
<td>80</td>
<td>FIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>4 NOBS</td>
<td>8</td>
<td>FIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>5 NRDOF</td>
<td>8</td>
<td>FIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
</tbody>
</table>

**Total Space Allocated** 1483

**ENTRY POINTS**

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000</td>
<td>SORT</td>
<td></td>
</tr>
</tbody>
</table>

**VARIABLES**

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP-00000004</td>
<td>R*4</td>
<td>FRACT</td>
<td>2-00000340</td>
<td>I*4</td>
<td>ILINK</td>
<td>2-00000348</td>
<td>I*4</td>
<td>IND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>I*4</td>
<td>IND2</td>
<td>2-00000344</td>
<td>I*4</td>
<td>IOBST</td>
<td></td>
<td>I*4</td>
<td>ISTART</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-00000000</td>
<td>I*4</td>
<td>RDOF</td>
<td>4-00000004</td>
<td>I*4</td>
<td>NLINKS</td>
<td>4-00000004</td>
<td>I*4</td>
<td>NOBST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>R*4</td>
<td>TEMP1</td>
<td></td>
<td>R*4</td>
<td>TEMP2</td>
<td></td>
<td>R*4</td>
<td>TEST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AP-0000000C</td>
<td>R*4</td>
<td>TMX2</td>
<td></td>
<td></td>
<td></td>
<td>AP-0000000B</td>
<td>R*4</td>
<td>TMX1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ARRAYS**

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP-00000004</td>
<td>I*4</td>
<td>FTABLE</td>
<td>400</td>
<td>(10, 10)</td>
</tr>
<tr>
<td>3-00000000</td>
<td>I*4</td>
<td>LINDOF</td>
<td>80</td>
<td>(2, 10)</td>
</tr>
<tr>
<td>2-00000038</td>
<td>R*4</td>
<td>POINTS</td>
<td>16</td>
<td>(2, 2)</td>
</tr>
<tr>
<td>2-00000032</td>
<td>R*4</td>
<td>VECTS</td>
<td>16</td>
<td>(2, 2)</td>
</tr>
<tr>
<td>2-00000000</td>
<td>R*4</td>
<td>VLEN1</td>
<td>400</td>
<td>(10, 10)</td>
</tr>
<tr>
<td>2-00000019</td>
<td>R*4</td>
<td>VLEN2</td>
<td>400</td>
<td>(10, 10)</td>
</tr>
<tr>
<td>2-00000032</td>
<td>R*4</td>
<td>VLEN3</td>
<td>8</td>
<td>(2)</td>
</tr>
<tr>
<td>AP-00000010</td>
<td>I*4</td>
<td>VLIST</td>
<td>32</td>
<td>(4, 2)</td>
</tr>
</tbody>
</table>

**LABELS**

<table>
<thead>
<tr>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td>25</td>
<td></td>
<td>50</td>
<td></td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td></td>
<td></td>
<td>125</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
FUNCTIONS AND SUBROUTINES REFERENCED

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CVECT</td>
</tr>
</tbody>
</table>

COMMAND QUALIFIERS

FOR/LIST OPT.FOR, RDFILE.FOR, INPUT.FOR, GROW.FOR, FINPAT.FOR, SOLVE1.FOR, INT2D.FOR, HULL2D.FOR, GMULT.FOR, JACOB.FOR, COSTT.FOR,

/CHECK*(NOBOUNDS, OVERFLOW, ROUNDERRFLOW)
/DEBUG*(NOSYMBOLS, TRACEBACK)
/STANDARD*(NOSYNTAX, NOSOURCE_FORM)
/SHOW*(NOPREPROCESSOR, NOINCLUDE, MAP, NODICTIONARY, SINGLE)
/WARNINGS*(GENERAL, NODECLARATIONS)
/CONTINUATIONS*19 /NCROSS_REFERENCE /NOEXTEND_SOURCE /F77
/NOFLOATING /14 /NOMACHINE_CODE /OPTIMIZE

COMPILATION STATISTICS

Run Time: 4.23 seconds
Elapsed Time: 10.89 seconds
Page Faults: 624
Dynamic Memory: 845 pages
program straj
0003  c Interactively generate a proposed sequence of intermediate positions
0004  c defining a starting guess for the optimization algorithm.
0005
0006  real theta(4,10)
0007  real box1(2,10,10),box2(2,10,10)
0008  integer links(4)
0009  logical ierase,stop
0010  real xdot(3),xgoal(3),xcurr(3)
0011  integer jtype(4),link(4)
0012  logical ixdot(6)
0013  real kgoal(3)
0014  real theta1(4)
0015  real theta2(4)
0016  real rotdes(3,3)
0017  real xpos(3)
0018  logical save,first,final
0019
0020  data kgoal/0.,0.,1/
0021
0022  common /robot/link,jtype,alpha,len,theta,dist
0023  common /nrdof/ndof,nxdof
0024
0025  c Read the kinematic parameters from a file.
0026
0027  call rdfile
0028
0029  ilast=1
0030
0031  write(10,*) 'Enter the number of task coordinates (2 or 3)'
0032  read(10,*) nxdof
0033
0034  if (nxdof.eq.3) ixdot(3)=.true.
0035  ixdot(4)=.true.
0036  ixdot(5)=.true.
0037
0038  d2rad=.0174533
0039
0040  c Read the geometry of the links and obstacles from a file.
0041
0042  open(1, file='box.dat', status='old')
0043  read(1,*) nlinks
0044  do 29 k=1,nlinks
0045    read(1,*) nvert,links(k)
0046  29 continue
0047  do 13 i=1,nvert
0048    read(1,*) (box1(j,i,k),j=1,2)
0049  13 continue
0050  continue
0051  do 19 k=1,nobst
0052    read(1,*) nvert
0053    do 14 i=1,nvert
0054      read(1,*) (box2(j,i,k),j=1,2)
0055  14 continue
0056  continue
0057  continue
0058    close(1)
0059    npos=1
0061    write(A,A) 'Enter initial configuration angles'
0062    read(A,A) (theta(i,npos),i=1,ndof)
0064    do 25 i=1,ndof
0065      theta(i)=theta(i,npos)
0066 25 continue
0068    c Initialize the graphics environment.
0069    call grstrt(l,4027)
0070    call newpag
0071    call window(-4.,4.,-4.,4.)
0072    call vviewport(0.,100.,0.,100.)
0074    first=.true.
0076    ierase=.true.
0077    final=.true.
0079 100 continue
0081    if (.not.first) then
0082      call cclose
0084      write(A,A) 'Save position? (T/F)'
0085      read(A,A) save
0086    if (.not.save) then
0087      npos=max(0(1,npos-1)
0088    else
0089      final=.false.
0090 endif
0092    write(A,A) 'Quit? (T/F)'
0093    read(A,A) stop
0095    if (final) then
0096      write(A,A) 'Enter goal position and orientation'
0097      read(A,A) xgoal(1),xgoal(2),angoal
0098    else
0099      if (stop) then
0101      xpos(1)=xgoal(1)
0102      xpos(2)=xgoal(2)
0103      angle=angoal*d2rad
0104    else
0105      write(A,A) 'Enter intermediate position and orientation'
0106      read(A,A) xpos(l),xpos(2),angle
0107      angle=angle*d2rad
0108 endif
0109    endif
0110    c Find an optimal inverse solution for the desired position.
0111    call rotmat(kgoal,angle,rotdes)
0113    call invers(len,alpha,dist,theta1,jtype,ndof,nxdof,ixdot,rotdes, 
0114       xpostheta2)
write(*,*) 'X,Y pos, angle ',(xpos(i),i=1,2),angle
npos=npos+1
do 23 i=1,ndof
  thetal(i)=theta2(i)
  theta(i,npos)=theta2(i)
23  continue
endif
write(*,*) 'Erase screen? (T/F)' read(*,*) ierase
endif
if (ierase) then
  call newpag
c Draw the obstacles.
do 22 k=1,nobst
  call move(box2(1,1,k),box2(2,1,k))
call draw(box2(1,2,k),box2(2,2,k))
call draw(box2(1,3,k),box2(2,3,k))
call draw(box2(1,4,k),box2(2,4,k))
call draw(box2(1,1,k),box2(2,1,k))
22  continue
c Draw final position.
call tridnt(.false.)
call transl(xgoal(1),xgoal(2))
call rotate(anggoal,anggoal)
call move(boxl(1,1,ilast),boxl(2,1,ilast))
call draw(boxl(1,2,ilast),boxl(2,2,ilast))
call draw(boxl(1,3,ilast),boxl(2,3,ilast))
call draw(boxl(1,4,ilast),boxl(2,4,ilast))
call draw(boxl(1,1,ilast),boxl(2,1,ilast))
c Draw the link positions.
do 15 ipos=npos
  call tridnt(.false.)
  call transl(xpos(ipos),xpos(ipos))
call rotate(angle,angle)
call transl(len(ipos),0.)
call move(0.,0.)
call dashpt(1)
call draw(-len(ind),0.)
call dashpt(0)
if (ind.eq.links(ilink)) then
call move(box1(1,1,ilink),box1(2,1,ilink))
call draw(box1(1,2,ilink),box1(2,2,ilink))
call draw(box1(1,3,ilink),box1(2,3,ilink))
call draw(box1(1,4,ilink),box1(2,4,ilink))
call draw(box1(1,1,ilink),box1(2,1,ilink))
endif
ilink=ilink+1
continue
continue
first=.false.
if (.not.stop) goto 100
continue
call grstop
c Save the positions in a file.
open(1,filena'traj.dat',status='new')
write(1,A) xgoal(1),xgoal(2),xgoal(3),angoal
write(1,A) npos
do 500 i=1,npos
write(1,A) (theta(j,1),j=1,4)
continue
500 close(1)
stop
### PROGRAM SECTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $CODE</td>
<td>1890</td>
<td>PIC CON REL LCL SHR EXE RD NONRT LONG</td>
</tr>
<tr>
<td>1 $DATA</td>
<td>271</td>
<td>PIC CON REL LCL SHR NOEXE RD NONRT LONG</td>
</tr>
<tr>
<td>2 $LOCAL</td>
<td>2208</td>
<td>PIC CON REL LCL NOCSH NODEX RD WRT LONG</td>
</tr>
<tr>
<td>3 ROBOT</td>
<td>240</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>4 NRDOF</td>
<td>8</td>
<td>PIC OVR REL GBL SHR NOEXE RD WRT LONG</td>
</tr>
</tbody>
</table>

Total Space Allocated: 4617

### ENTRY POINTS

<table>
<thead>
<tr>
<th>Address Type Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000000 STRAJ</td>
</tr>
</tbody>
</table>

### VARIABLES

<table>
<thead>
<tr>
<th>Address Type Name</th>
<th>Address Type Name</th>
<th>Address Type Name</th>
<th>Address Type Name</th>
<th>Address Type Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-0000070C R4 ANGLE</td>
<td>2-00000708 R4 ANGOAL</td>
<td>2-000006EC LA4 STOP</td>
<td>2-00000670 R4 DZRAD</td>
<td>2-000006FB LA4 FINAL</td>
</tr>
<tr>
<td>2-000006F4 LA4 FIRST</td>
<td>** IA4 I</td>
<td>2-000006EB LA4 IERASE</td>
<td>** IA4 J</td>
<td>2-00000674 R4 KGOAL</td>
</tr>
<tr>
<td>** IA4 ILINK</td>
<td>** IA4 X</td>
<td>2-000006FC IA4 NLINKS</td>
<td>** IA4 J</td>
<td>2-00000600 IA4 THETA1</td>
</tr>
<tr>
<td>** IA4 I4 NPOS</td>
<td>2-00000200 IA4 XCURH</td>
<td>2-00000000 IA4 NXDOF</td>
<td>2-00000004 IA4 NOBST</td>
<td></td>
</tr>
</tbody>
</table>

### ARRAYS

<table>
<thead>
<tr>
<th>Address Type Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-00000020 R4 ALPHA</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00000000 R4 BOX1</td>
<td>800</td>
<td>(2, 10, 10)</td>
</tr>
<tr>
<td>2-00000320 R4 BOX2</td>
<td>800</td>
<td>(2, 10, 10)</td>
</tr>
<tr>
<td>3-000000E0 R4 DIST</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00000674 LA4 IDXOT</td>
<td>24</td>
<td>(6)</td>
</tr>
<tr>
<td>3-00000010 IA4 JTYPE</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-0000068C R4 KGOAL</td>
<td>12</td>
<td>(3)</td>
</tr>
<tr>
<td>3-00000030 R4 LEN</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>3-00000000 IA4 LINK</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00000640 IA4 LINKS</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-0000068B R4 ROUTES</td>
<td>36</td>
<td>(3, 3)</td>
</tr>
<tr>
<td>3-00000040 R4 THETA</td>
<td>160</td>
<td>(4, 10)</td>
</tr>
<tr>
<td>2-00000678 R4 THETA1</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-000006A8 R4 THETA2</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00000668 R4 XCURR</td>
<td>12</td>
<td>(3)</td>
</tr>
<tr>
<td>2-00000650 R4 XDOT</td>
<td>12</td>
<td>(3)</td>
</tr>
<tr>
<td>2-0000065C R4 KGOAL</td>
<td>12</td>
<td>(3)</td>
</tr>
<tr>
<td>2-000006EC R4 XPOS</td>
<td>12</td>
<td>(3)</td>
</tr>
</tbody>
</table>
### LABELS

<table>
<thead>
<tr>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>10</td>
<td>AA</td>
<td>13</td>
<td>AA</td>
<td>14</td>
<td>**</td>
<td>15</td>
</tr>
<tr>
<td>AA</td>
<td>23</td>
<td>**</td>
<td>29</td>
<td>**</td>
<td>19</td>
<td>**</td>
<td>22</td>
</tr>
</tbody>
</table>

### FUNCTIONS AND SUBROUTINES REFERENCED

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Type Name</th>
<th>Type Name</th>
<th>Type Name</th>
<th>Type Name</th>
<th>Type Name</th>
<th>Type Name</th>
<th>Type Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMCL08</td>
<td>CMOPEN</td>
<td>CMOPEN</td>
<td>CMOPEN</td>
<td>CMOPEN</td>
<td>CMOPEN</td>
<td>CMOPEN</td>
<td>CMOPEN</td>
</tr>
<tr>
<td>GRSTOP</td>
<td>GRSTRT</td>
<td>ROSET</td>
<td>ROSET</td>
<td>ROSET</td>
<td>ROSET</td>
<td>ROSET</td>
<td>ROSET</td>
</tr>
<tr>
<td>ROTATE</td>
<td>ROTMAT</td>
<td>TRANSL</td>
<td>TRANSL</td>
<td>TRANSL</td>
<td>TRANSL</td>
<td>TRANSL</td>
<td>TRANSL</td>
</tr>
<tr>
<td>DRAME</td>
<td>MOVE</td>
<td>MOVE</td>
<td>MOVE</td>
<td>MOVE</td>
<td>MOVE</td>
<td>MOVE</td>
<td>MOVE</td>
</tr>
<tr>
<td>FDRSCLOSE</td>
<td>NEHFAG</td>
<td>VWWPORT</td>
<td>VWWPORT</td>
<td>VWWPORT</td>
<td>VWWPORT</td>
<td>VWWPORT</td>
<td>VWWPORT</td>
</tr>
<tr>
<td>FOR@CLOSE</td>
<td>FOR@OPEN</td>
<td>FOR@OPEN</td>
<td>FOR@OPEN</td>
<td>FOR@OPEN</td>
<td>FOR@OPEN</td>
<td>FOR@OPEN</td>
<td>FOR@OPEN</td>
</tr>
<tr>
<td>RDFIE</td>
<td></td>
<td></td>
<td></td>
<td>WINDOW</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SUBROUTINE SVD(A,Q,U,V,ND,NDV,M,N,P,NU,NV)
DIMENSION A(ND,1),Q(N),U(ND,1),V(NDV,1),E(100)
REAL A,SS,DA,DB,DSORT
INTEGER P
DATA TOL,EPS/2.E-30,5.E-7/
C THIS ROUTINE IS CODED FROM THE ALGOL PROCEDURES BY GOLUB AND REINSCH
C IN CONTRIBUTION I/10, PAGE 134, OF 'HANDBOOK FOR AUTOMATIC
C COMPUTATION', EDITED BY WILKINSON AND REINSCH.
C THIS ROUTINE COMBINES PROCEDURES SVD AND MINFIT. M>N IF MATRIX
C REQUIRED. OTHERWISE NO RESTRICTION ON M. IF SOLN. OF AX=B
C REQUIRED AND M<N, THEN RANK(A)=N.
GO TO 10
ENTRY SING(A,Q,ND,M,N)
P=0
NU=0
NV=0
GO TO 10
ENTRY SOLSVD(A,Q,V,ND,NDV,M,N,P)
NU=0
NV=N
10 NP=N+P
NP=NP+P
IER=0
M
C REDUCE U TO BIDIAGONAL FORM BY HOUSEHOLDER TRANSFORMATIONS
IF (M.GE.N) GO TO 20
MP1=M+1
DO 15 I=MP1,N
DO 15 J=1,N
A(I,J)=0.
20 CONTINUE
G=0.0
X=0.0
DO 30 I=1,M
DA=A(I,I)
30 SS=SS+DA*DA
F=SQRT(SS)
IF(F.GE.EPS) G=-G
H=FAG-SS
A(I,I)=F-G
IF (L.GT.NP) GO TO 60
DO 55 J=L,NP
SS=SS+DA*DA
55 CONTINUE
G=0.0
DO 60 I=1,N
F=A(I,I)
60 G*DGSRT(SS)
IF (F.GE.0.00) G=-G
H=FAG-SS
A(I,I)=F-G
IF (L.GT.NP) GO TO 60
DO 55 J=L,NP
SS=SS+DA*DA
55 CONTINUE
G=0.0
DO 60 I=1,N
F=A(I,I)
60 DA=A(I,I)
DB=A(I,J)
SS=SS+DA*DB
F=SS/N
DO 50 K=I,M
50 DA=A(K,I)
50 DB=A(K,J)
SS=SS+DA*DB
F=SS/N
DO 50 K=I,M
0058  50 A(K,J)=A(K,J)+F*A(K,I)
0059  55 CONTINUE
0060  60 Q(I)=G
0061     SS=0.D0
0062    IF(L.GT.N) GO TO 120
0063    IF(I.GT.N) GO TO 75
0064    DO 70 J=L,N
0065    DA=A(I,J)
0066    70 SS=SS+DA*DA
0067    75 G=0.0
0068    IF(SS.LE.TOL) GO TO 120
0069    F=A(I,L)
0070    G=DSQRT(SS)
0071    IF(F.GE.0.0) G=-G
0072    H=F*G-SS
0073    HI=1.D0/H
0074    A(I,L)=F-G
0075    DO 85 J=L,N
0076    85 E(J)=A(I,J)*HI
0077    DO 100 J=L,N
0078    100 SS=SS+DA*DB
0079    CONTINUE
0080  85 G=A(J,K)+SS*E(K)
0081  100 CONTINUE
0082  90 SS=SS+DA*DB
0083    DO 95 K=L,N
0084    95 G=A(J,K)+SS*E(K)
0085    100 CONTINUE
0086  120 Y=ABS(Q(I))+ABS(E(I))
0087    IF(Y.GT.X) X=Y
0088  130 CONTINUE
0089  C
0090  C ACCUMULATE RIGHT HAND TRANSFORMATIONS.
0091  IF(NV.EQ.0) GO TO 210
0092    V(N,N)=1.0
0093    G=E(N)
0094    IF(N.EQ.1) GO TO 210
0095    DO 200 I=2,N
0096    I=NP1-I1
0097    L=I1
0098    IF(G.EQ.0.0.OR.I.GT.N) GO TO 180
0099    H=A(I,L)*G
1000    DO 140 J=L,N
1001    140 V(J,I)=A(I,J)/H
1002    DO 170 J=L,N
1003    SS=0.D0
1004    DO 150 K=L,N
1005    DA=A(I,K)
1006    DB=V(K,J)
1007    150 SS=SS+DA*DB
1008    DO 160 K=L,N
1009    160 V(K,J)=V(K,J)+SS*V(K,I)
1010    170 CONTINUE
1011    180 DO 185 J=L,N
1012    V(I,J)=0.0
1013    185 V(J,I)=0.0
1014    190 V(I,I)=1.0
G = E(I)
200 CONTINUE
C ACCUMULATE LEFT HAND TRANSFORMATIONS
210 IF(NU.EQ.0) GO TO 310
211 IF(NU.LE.N) GO TO 215
212 DO 214 I = I+1,NU
213 U(J,I) = 0.0
214 U(I,I) = 1.0
215 DO 300 II = 1, M
216 I = I+1
217 L = I-1
218 G = Q(I)
219 IF(L.GT.NU) GO TO 223
220 DO 220 J = L, NU
221 U(I,J) = 0.0
222 IF(G.EQ.0.0 OR I.GT.M) GO TO 280
223 IF(L.GT.NU) GO TO 255
224 DO 250 J = L, M
225 SS = SS + A(K,J)*DB(I,J)
226 F = SS/H
227 U(K,J) = U(K,J) + F*A(K,I)
228 DO 250 CONTINUE
229 255 G1 = 1.0/G
230 DO 260 II = I, M
231 260 U(J,I) = U(J,I) + G1*II
232 CONTINUE
C CODE FOR C = 0.0
233 DO 285 J = I, M
234 285 U(J,I) = 0.0
235 U(I,I) = U(I,I) + 1.0
236 CONTINUE
DIAGONALISE THE BIDIAGONAL FORM
310 CONTINUE
EPSX = EPS*X
311 IF(M.GE.N) GO TO 315
312 IF(NP.EQ.N) GO TO 315
313 MP1 = MP1+1
314 DO 318 J = J+1, MP1
315 A(I,J) = 0.0
316 DO 315 II = 0, 1000
317 KK = II+1
318 NITER = 0
319 K = NP1 - KK
320 KP1 = KP1+1
321 DO 320 LL = 1, K
322 EPSX = EPS*X
323 IF(ABS(E(L)).LE.EPSX) GO TO 450
324 IF(L.EQ.1) GO TO 450
325 CONTINUE
0172 IF(ABS(Q(L-1)).LE.EPSX) GO TO 400
0173 CONTINUE
0174 C
0175 C CANCEL E(L) - ONLY IF L>1
0176 400 C=0.0
0177 S=1.0
0178 LM1=L-1
0179 DO 440 I=LM1,K
0180 F=S*E(I)
0181 E(I)=C*E(I)
0182 IF(ABS(F).LE.EPSX) GO TO 450
0183 G=Q(I)
0184 IF(ABS(F).GT.ABS(G)) GO TO 410
0185 H=2.0*ABS(G)*SQRT(.25+1.5*F/G)**2)
0186 C TO 415
0187 410 H=2.0*ABS(F)*SQRT(.25+(.5A/F/G)**2)
0188 415 Q(I)=H
0189 C=G/H
0190 IF(NU.EQ.0) GO TO 430
0191 DO 420 J=1,M
0192 Y=U(J,LM1)
0193 Z=U(J,I)
0194 U(J,LM1)=Y*C+Z*3
0195 U(J,I)=-Y*S+Z*C
0196 420 CONTINUE
0197 430 CONTINUE
0198 IF(NP.EQ.N) GO TO 440
0199 DO 435 J=NP1,NP
0200 Y=A(LM1,J)
0201 Z*A(I,J)
0202 A(LM1,J)=C*Y+S*Z
0203 435 A(I,J)=-S*Y+C*Z
0204 440 CONTINUE
0205 C
0206 C TEST F CONVERGENCE
0207 450 Z=Q(K)
0208 IF(L.EQ.K) GO TO 900
0209 NITER=NITER+1
0210 IF(NITER.LT.40) GO TO 455
0211 IER=IER+1
0212 GO TO 1000
0213 455 CONTINUE
0214 C
0215 C SHIFT FROM BOTTOM 2 BY 2 MINOR
0216 X=Q(L)
0217 Y=Q(L-1)
0218 C=E(X-1)
0219 H=E(K)
0220 F=0.5((Y-Z)*(Y+Z)+(G-H)*(G+H))/(H*Y)
0221 G=SQRT(F+F+1.0)
0222 TD=F*G
0223 IF(F.LT.0.0) TD=F-G
0224 F=((X-Z)*(X+Z)+H*(Y/TD-H))/X
0225 C
0226 C NOW DO THE QR TRANSFORMATION
0227 C=1.0
0228 S=1.0
SVD

LP1=L+1
0230 IF(LP1.GT.K) GO TO 510
0231 DO 500 I=LP1,K
0232 IM1=I-1
0233 G=E(I)
0234 Y=Q(I)
0235 H=SG
0236 C=SG
0237 IF(ABS(F).GT.ABS(H)) GO TO 456
0238 Z=2.0*ABS(H)*SQRT(.25+(.5*F/H)**2)
0239 GO TO 457
0240 456 Z=2.0*ABS(F)*SQRT(.25+(.5*H/F)**2)
0241 457 E(IM1)=Z
0242 C=F/Z
0243 S=H/Z
0244 F=X*C+G*SG
0245 G=-X*SG+G*SG
0246 H=SG
0247 X=Y*SG
0248 IF(MU.EQ.0) GO TO 465
0249 DO 460 J=1,N
0250 X=V(J,IM1)
0251 Y=V(J,I)
0252 V(J,IM1)=X*C+Z*S
0253 V(J,I)=-Y*S+Z*C
0254 460 CONTINUE
0255 465 IF(ABS(F).GT.ABS(H)) GO TO 470
0256 IF (ABS(H).EQ.0.) GO TO 472
0257 Z=2.0*ABS(H)*SQRT(.25+(.5*F/H)**2)
0258 GO TO 475
0259 470 Z=2.0*ABS(F)*SQRT(.25+(.5*H/F)**2)
0260 GO TO 475
0261 472 Z=0.
0262 C=1./SQRT(2.)
0263 S=C
0264 Q(I-1)=0.
0265 GO TO 476
0266 475 Q(I-1)=Z
0267 C=F/Z
0268 S=H/Z
0269 476 F=SG+C*S
0270 G=SG+C*SG
0271 IF(MU.EQ.0) GO TO 490
0272 IF (IM1.GT.MU .OR.I.GT.MU) GO TO 490
0273 DO 480 J=1,M
0274 U(J,IM1)=C*Y+S*Z
0275 U(J,I)=-S*Y+CAZ
0276 480 CONTINUE
0277 490 IF(N.EQ.NP) GO TO 500
0278 DO 495 J=NP1,NP
0279 Y=A(I,J)
0280 Z=A(I,J)
0281 A(IM1,J)=C*Y+S*Z
0282 A(I,J)=S*Y+C*Z
0283 495 A(I,J)=S*Y+C*Z
0284 500 CONTINUE
FUNCTIONS AND SUBROUTINES REFERENCED

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>R=8</td>
<td>MTHDSQRT</td>
<td>R=4</td>
<td>MTHDSQRT</td>
</tr>
</tbody>
</table>

COMMAND QUALIFIERS

FOR/LIST OPT FOR,FILE FOR,INPUT FOR,COMMAND FOR,FINPATS, FOR, SOLVE1 FOR, INT2D FOR, HULL2D FOR, GMULT FOR, JACOB FOR, COSTT FOR,

/ CHECK* (NOBOUND5, OVERFLOW, ROUNDERROR)
/ DEBUG* (NOSTACKS, TRACEBACK)
/ STANDARD* (NOSYNTAX, NOSOURCE FORM)
/ SHOW* (NOPREPROCESSOR, NOINCLUDE, MAP, NODICTIO NARY, SINGLE)
/ WARNINGS* (GENERAL, NODECLARATIONS)
/ CONTINUATIONS* 19 / NOKROS CROSS_REFERENCE / MOD_LINES / NO EXTEND SOURCE / F77
/ NOG FLOATING / 14 / NOMICHEL_CODE / OPTIMIZE

COMPILATION STATISTICS

Run Time: 24.50 seconds
Elapsed Time: 81.99 seconds
Page Faults: 2828
Dynamic Memory: 845 pages
C CHECK IF ANY SING. VALUES NOT FOUND TO REQD. ACCURACY IN 40
0301 C ITERATIONS. IER>0 IF SO.
0302 IF(IER.EQ.0) GO TO 700
0303 WRITE(6,710) IER
0304 710 FORMAT('0',I5,3X,'SINGULAR VALUES NOT FOUND IN 40 ITERATIONS')
0305 RETURN
0306 C
0307 C SORT SINGULAR VALUES
0308 700 DO 600 K=1,N
0309 G=1.0
0310 J=K
0311 DO 610 I-K,N
0312 IF(Q(I).LE.G) GO TO 610
0313 G=Q(I)
0314 J=I
0315 610 CONTINUE
0316 IF(J.EQ.K) GO TO 600
0317 Q(J)=Q(K)
0318 Q(K)=G
0319 IF(NV.EQ.0) GO TO 630
0320 DO 620 I=1,N
0321 H=V(I,J)
0322 V(I,J)=V(I,K)
0323 620 V(I,K)=H
0324 630 IF(NV.EQ.0) GO TO 650
0325 DO 640 I=1,N
0326 H=U(I,J)
0327 U(I,J)=U(I,K)
0328 640 U(I,K)=H
0329 650 IF(NV.EQ.NP) GO TO 660
0330 DO 660 I=NP1,NP
0331 H=A(J,I)
0332 A(J,1)=A(K,1)
0333 660 A(K,1)=H
0334 600 CONTINUE
0335 1100 RETURN
0336 END
### Variables

<table>
<thead>
<tr>
<th>Address Type</th>
<th>Name</th>
<th>Address</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA 4</td>
<td>C</td>
<td>2-0000019B</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>RA 4</td>
<td>EPSX</td>
<td>2-0000011D</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>RA 4</td>
<td>H</td>
<td>2-00000264</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>RA 4</td>
<td>LL</td>
<td>2-00000234</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>RA 4</td>
<td>LM</td>
<td>2-0000024C</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>RA 4</td>
<td>N</td>
<td>2-00000298</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>RA 4</td>
<td>NP</td>
<td>2-000002CB</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>RA 4</td>
<td>TOL</td>
<td>2-000001A0</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

### Entry Points

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-0000003E</td>
<td></td>
<td>SING</td>
<td>0-0000006E</td>
<td>SOLSVD</td>
<td>0-00000000</td>
<td>SVD</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Arrays

<table>
<thead>
<tr>
<th>Address Type</th>
<th>Name</th>
<th>Address</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA 4</td>
<td>A</td>
<td>AP-00000048</td>
<td>AA</td>
<td>(A, 1)</td>
</tr>
<tr>
<td>RA 4</td>
<td>E</td>
<td>2-00000000</td>
<td>400</td>
<td>100</td>
</tr>
<tr>
<td>RA 4</td>
<td>Q</td>
<td>AP-0000006B</td>
<td>AA</td>
<td>()</td>
</tr>
<tr>
<td>RA 4</td>
<td>U</td>
<td>AP-000000CB</td>
<td>AA</td>
<td>(A, 1)</td>
</tr>
<tr>
<td>RA 4</td>
<td>V</td>
<td>2-00000294B</td>
<td>AA</td>
<td>(A, 1)</td>
</tr>
</tbody>
</table>

### Labels

<table>
<thead>
<tr>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-00000080</td>
<td>10</td>
<td>AA</td>
<td>15</td>
<td>0-0000010D</td>
<td>20</td>
<td>AA</td>
<td>30</td>
<td>0-00000171</td>
<td>35</td>
<td>AA</td>
<td>45</td>
</tr>
<tr>
<td>50</td>
<td>AA</td>
<td>55</td>
<td>0-00000277</td>
<td>60</td>
<td>AA</td>
<td>70</td>
<td>0-0000022C</td>
<td>75</td>
<td>AA</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>AA</td>
<td>95</td>
<td>100</td>
<td>0-0000041B</td>
<td>120</td>
<td>AA</td>
<td>130</td>
<td>AA</td>
<td>140</td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>AA</td>
<td>160</td>
<td>170</td>
<td>0-000005CD</td>
<td>180</td>
<td>AA</td>
<td>185</td>
<td>AA</td>
<td>190</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>0-0000062C</td>
<td>210</td>
<td>213</td>
<td>AA</td>
<td>214</td>
<td>0-000006B9</td>
<td>215</td>
<td>AA</td>
<td>220</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0-00000729</td>
<td>225</td>
<td>AA</td>
<td>240</td>
<td>245</td>
<td>AA</td>
<td>250</td>
<td>0-00000803</td>
<td>255</td>
<td>AA</td>
<td>260</td>
<td></td>
</tr>
<tr>
<td>280</td>
<td>AA</td>
<td>285</td>
<td>0-0000087D</td>
<td>290</td>
<td>AA</td>
<td>300</td>
<td>0-00000989</td>
<td>310</td>
<td>AA</td>
<td>312</td>
<td></td>
</tr>
<tr>
<td>0-00000909</td>
<td>315</td>
<td>0-00000934</td>
<td>330</td>
<td>AA</td>
<td>340</td>
<td>0-000009BC</td>
<td>400</td>
<td>0-00000A14</td>
<td>410</td>
<td>0-00000A32</td>
<td>415</td>
</tr>
</tbody>
</table>
subroutine trandb(values)
   Transform the geometric database specifying kinematic chains of rigid links by performing a chain of transformations on each link. The transformations are in a standard form, and are functions of given joint variables.

   VARIABLES:
   param values- the values of the joint variables (degrees).

   SUBROUTINES CALLED:
   gmult

   Called from:

   Variables:
   len, dist, alpha, theta - Denavit-Hartenberg specifications for the kinematic chains
   jtype - array specifying each joint as prismatic or revolute
   ndof - the number of degrees of freedom for each kinematic chain

integer ndof, nxdof
integer jtype(4)
integer link(4)
real len(4), dist(4), alpha(4), theta(4)
real values(4)
real ident(4,4)
real temp(4,4)
real pretra(4), postra(4)
real tmat(4,4), tprod(4,4)
real temp(4,4), temp2(4,4), temp3(4)
integer lindof(2,10)
integer boxes(2,10)
real sverts(2,100), tverts(2,100)
common /robot/link, jtype; alpha, len, theta, dist
common /ident/ident
common /ndof/ndof, nxdof
common /objs/boxes, sverts, tverts
common /nobjs/nlinks, nobst
common /lindof/lindof

d2rad=.0174533

do 500 i=1,4
   do 501 j=1,4
      tprod(j,i)=ident(j,i)
   continue
   continue
500
icount=1
Transform the part clusters constituting the links of all the open kinematic chains in the system. Each link is related to the previous link by the Denavit-Hartenberg transformation. The net transformation is the product of these transformations.

\[
\begin{align*}
\text{tmat}(3,1) &= 0. \\
\text{tmat}(4,1) &= 0. \\
\text{tmat}(4,2) &= 0. \\
\text{tmat}(4,3) &= 0. \\
\text{tmat}(4,4) &= 1.
\end{align*}
\]

Transform each link of the kinematic chain by the Denavit-Hartenberg transformation.

\[
\begin{align*}
\text{do } & 20 \text{ nd}2=1, \text{ndof} \\
\text{temp1}(\text{ind}2) &= \alpha(\text{ind}2) \times \text{d2rad} \\
\text{temp2}(\text{ind}2) &= \theta(\text{ind}2) \times \text{d2rad} \\
\text{temp3}(\text{ind}2) &= \text{value}(\text{ind}2) \times \text{d2rad} \\
\text{if } & (j\text{type}(\text{ind}2) .eq. 0) \text{ then} \\
\text{Revolute joint.} \\
\text{tmat}(1,1) &= \cos(\text{temp3}(\text{ind}2)) \\
\text{tmat}(2,1) &= \sin(\text{temp3}(\text{ind}2)) \\
\text{tmat}(1,2) &= -\sin(\text{temp3}(\text{ind}2)) \times \cos(\text{temp1}(\text{ind}2)) \\
\text{tmat}(2,2) &= -\cos(\text{temp3}(\text{ind}2)) \times \cos(\text{temp1}(\text{ind}2)) \\
\text{tmat}(3,2) &= -\sin(\text{temp1}(\text{ind}2)) \\
\text{tmat}(3,3) &= \cos(\text{temp1}(\text{ind}2)) \\
\text{tmat}(3,4) &= \text{len}(\text{ind}2) \times \cos(\text{temp3}(\text{ind}2)) \\
\text{tmat}(3,5) &= \text{len}(\text{ind}2) \times \sin(\text{temp3}(\text{ind}2)) \\
\text{tmat}(2,4) &= \text{len}(\text{ind}2) \times \sin(\text{temp3}(\text{ind}2)) \\
\text{tmat}(3,5) &= \cos(\text{temp2}(\text{ind}2)) \\
\text{tmat}(2,5) &= \sin(\text{temp2}(\text{ind}2)) \\
\text{tmat}(3,5) &= -\cos(\text{temp2}(\text{ind}2)) \times \sin(\text{temp1}(\text{ind}2)) \\
\text{tmat}(2,6) &= -\cos(\text{temp2}(\text{ind}2)) \times \cos(\text{temp1}(\text{ind}2)) \\
\text{tmat}(3,6) &= \text{len}(\text{ind}2) \times \cos(\text{temp3}(\text{ind}2)) \\
\text{tmat}(3,7) &= \text{len}(\text{ind}2) \times \sin(\text{temp3}(\text{ind}2)) \\
\text{tmat}(3,7) &= \cos(\text{temp2}(\text{ind}2)) \\
\text{tmat}(2,8) &= \sin(\text{temp2}(\text{ind}2)) \\
\text{tmat}(3,8) &= -\cos(\text{temp2}(\text{ind}2)) \times \sin(\text{temp1}(\text{ind}2)) \\
\text{tmat}(2,9) &= -\cos(\text{temp2}(\text{ind}2)) \times \cos(\text{temp1}(\text{ind}2)) \\
\text{tmat}(3,10) &= \text{len}(\text{ind}2) \times \cos(\text{temp3}(\text{ind}2)) \\
\text{tmat}(3,11) &= \text{len}(\text{ind}2) \times \sin(\text{temp3}(\text{ind}2)) \\
\text{tmat}(3,12) &= \text{temp3}(\text{ind}2) \\
\text{endif}
\end{align*}
\]
CALL GMULT(TPROD,TMAT,TMP,4,4,4,4)

DO 300 I=1,4
    DO 301 J=1,4
        TPROD(J,I)*TMP(J,I)
    CONTINUE
300 CONTINUE

C If the link is modelled, transform its vertices in the data base.

IF (LINDOF(1,IND2).NE.0) THEN
    DO 111 IND=LINDOF(1,IND2),LINDOF(1,IND2)+LINDOF(2,IND2)-1
        DO 101 J=BOXES(1,IND),BOXES(1,IND)+BOXES(2,IND)-1
            DO 102 K=1,2
                PRETRA(K)=AVERTS(K,J)
            CONTINUE
102 CONTINUE
101 CONTINUE
    CONTINUE
111 CONTINUE
    IF (LINDOF(2,IND2).NE.0) THEN
        DO 300 IND=LINDOF(2,IND2),LINDOF(2,IND2)+LINDOF(1,IND2)-1
            DO 301 J=BOXES(2,IND),BOXES(2,IND)+BOXES(1,IND)-1
                DO 302 K=1,2
                    PRETRA(K)=AVERTS(K,J)
                CONTINUE
302 CONTINUE
301 CONTINUE
300 CONTINUE
    CONTINUE
    END
### PROGRAM SECTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Bytes</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 CODE</td>
<td>563</td>
<td>PIC COM REL LCL SHR EEX RD NONRT LONG</td>
</tr>
<tr>
<td>1 PDATA</td>
<td>8</td>
<td>PIC COM REL LCL SHR NOEXE RD NONRT LONG</td>
</tr>
<tr>
<td>2 LOCAL</td>
<td>376</td>
<td>PIC COM REL LCL NOBR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>3 ROBOT</td>
<td>96</td>
<td>PIC OVR REL GRL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>4 IDENT</td>
<td>64</td>
<td>PIC OVR REL GRL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>5 RXDF</td>
<td>8</td>
<td>PIC OVR REL GRL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>6 OBJS</td>
<td>1680</td>
<td>PIC OVR REL GRL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>7 NDOF</td>
<td>8</td>
<td>PIC OVR REL GRL SHR NOEXE RD WRT LONG</td>
</tr>
<tr>
<td>8 LINDOF</td>
<td>80</td>
<td>PIC OVR REL GRL SHR NOEXE RD WRT LONG</td>
</tr>
</tbody>
</table>

**Total Space Allocated**: 2883

### ENTRY POINTS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-000000000</td>
<td></td>
<td>TRANDB</td>
</tr>
</tbody>
</table>

### VARIABLES

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Address</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-0000000000</td>
<td>I4</td>
<td>NLINKS</td>
<td>7-00000004</td>
<td>I4</td>
<td>NOBST</td>
<td>5-000000004</td>
<td>I4</td>
<td>NDOF</td>
</tr>
<tr>
<td>AA I4 I</td>
<td></td>
<td></td>
<td>AA I4 ICOUNT</td>
<td></td>
<td></td>
<td>AA I4 IND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AA I4 IND2</td>
<td></td>
<td></td>
<td>AA I4 K</td>
<td></td>
<td></td>
<td>5-000000000</td>
<td>I4</td>
<td>NDOF</td>
</tr>
</tbody>
</table>

### ARRAYS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Name</th>
<th>Bytes</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-00000020</td>
<td>R4</td>
<td>ALPHA</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>6-000000000</td>
<td>I4</td>
<td>BOXES</td>
<td>80</td>
<td>(2, 10)</td>
</tr>
<tr>
<td>3-00000050</td>
<td>R4</td>
<td>DIST</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>4-000000000</td>
<td>R4</td>
<td>IDENT</td>
<td>64</td>
<td>(4, 4)</td>
</tr>
<tr>
<td>4-00000010</td>
<td>I4</td>
<td>JTYPE</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>3-00000030</td>
<td>R4</td>
<td>LEN</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>3-000000000</td>
<td>I4</td>
<td>LINDOF</td>
<td>80</td>
<td>(2, 10)</td>
</tr>
<tr>
<td>3-000000000</td>
<td>I4</td>
<td>LINK</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-000000050</td>
<td>R4</td>
<td>POSTRA</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-00000040</td>
<td>R4</td>
<td>PRETRA</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>6-000000050</td>
<td>R4</td>
<td>SVERTS</td>
<td>800</td>
<td>(2, 100)</td>
</tr>
<tr>
<td>2-000000000</td>
<td>R4</td>
<td>TEMP</td>
<td>64</td>
<td>(4, 4)</td>
</tr>
<tr>
<td>2-000000000</td>
<td>R4</td>
<td>TEMP1</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-000000000</td>
<td>R4</td>
<td>TEMP2</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-000000100</td>
<td>R4</td>
<td>TEMP3</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>3-000000040</td>
<td>R4</td>
<td>THETA</td>
<td>16</td>
<td>(4)</td>
</tr>
<tr>
<td>2-000000040</td>
<td>R4</td>
<td>TMAAT</td>
<td>64</td>
<td>(4, 4)</td>
</tr>
<tr>
<td>2-000000040</td>
<td>R4</td>
<td>TPROD</td>
<td>64</td>
<td>(4, 4)</td>
</tr>
<tr>
<td>6-000000000</td>
<td>R4</td>
<td>TVERTS</td>
<td>800</td>
<td>(2, 100)</td>
</tr>
<tr>
<td>3-000000040</td>
<td>R4</td>
<td>VALUES</td>
<td>16</td>
<td>(4)</td>
</tr>
</tbody>
</table>
LABELS

<table>
<thead>
<tr>
<th>Address</th>
<th>Label</th>
<th>Address</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>AA</td>
<td>101</td>
<td>AA</td>
</tr>
<tr>
<td>301</td>
<td>AA</td>
<td>500</td>
<td>AA</td>
</tr>
<tr>
<td>102</td>
<td>AA</td>
<td>103</td>
<td>AA</td>
</tr>
<tr>
<td>111</td>
<td>AA</td>
<td>300</td>
<td></td>
</tr>
</tbody>
</table>

FUNCTIONS AND SUBROUTINES REFERENCED

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Type Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMULT</td>
<td>RA4</td>
</tr>
<tr>
<td>MTHASIN</td>
<td>RA4</td>
</tr>
<tr>
<td>MTHCOS</td>
<td>RA4</td>
</tr>
<tr>
<td>MTHSIN</td>
<td>RA4</td>
</tr>
</tbody>
</table>

COMMAND QUALIFIERS

FOR/LIST OPT.FOR, REDLIST.FOR, INPUT.FOR, GROM.FOR, FINPAT.FOR, SOLVE1.FOR, INTZD.FOR, HULLZD.FOR, GMULT.FOR, JACOB.FOR, COSTT.FOR,

/ CHECK* (NOBOUNDS, OVERFLOW, ROUNDERRFLOW)
/ DEBUG* (NOSYMBOLS, TRACEBACK)
/ STANDARD* (NOFORMAT, NOOPTIONFLOH)
/ SHOW* (NOPREFIX, NOINCLUDE, MAP, MODIFICATION, SINGLE)
/ WARNINGS* (GENERAL, NODECLARATIONS)
/ CONTINUATIONS* (NOREFERENCE, NOINDEX, SOURCE, NOEXTEND_SOURCE)
/ NOG_FLOATING /14 /NOMACHINE_CODE /OPTIMIZE

COMPILATION STATISTICS

Run Time: 9.56 seconds
Elapsed Time: 16.63 seconds
Page Faults: 690
Dynamic Memory: 845 pages