To the Graduate Council :

I am submitting herewith a thesis written by Amrish Shah entitled "Collision Avoidance Schemes for Redundant and NonRedundant Manipulators". I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mechanical Engineering.

Dr. Rajiv V. Dubey, Major Professor

We have read this thesis
and recommend its acceptance :

Accepted for the Council :

Associate Vice Chancellor and
Dean of the Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under the rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature 

Date MAY 19th, 1995

COLLISION AVOIDANCE SCHEMES FOR REDUNDANT AND NONREDUNDANT MANIPULATORS

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Amrish Shah

August 1995

Acknowledgments

Although the results achieved and work completed was primarily performed by me, it would not have been possible without the help and guidance of the many people I have come to know during this period.

First and foremost my deep gratitude and appreciation for Dr. Rajiv Dubey for being my major professor and advisor. His guidance in building a subject for this thesis, help and patience, in dealing with my work in the laboratory, has been invaluable. I would also like to thank Dr. Bill Hamel for joining the team in the later half of my thesis and taking time to review my work and providing suggestions on improving it. A note of thanks to Dr. Kawiecki for agreeing to be a member of my defense committee. Tan Fung Chan has probably been the main cause of success of this thesis. His guidance, advice and help is greatly appreciated. I would also like to thank all students in my research group for their continued support during this period.

On the other front, I thank Mr. Winston Holmes of CIEM for maintaining the computer and network facilities working, despite of the mess created by me on many occasions. Thanks to all the people in the departmental office Linda, Janine, Marva, Betty and others for assisting me with the ordering and purchasing of equipment.

Finally, thanks to all my friends Sunil, Kumar ... and my roommate Harsha for their staunch support and appreciation of my work. And last but not the least a sense of indebtedness to my family which has inspired me to come so far.

Abstract

Motion of a manipulator within its workspace is restricted by singularities, joint limits, joint velocity limits and obstacles. To ensure smooth motion of the manipulator, in the presence of the above limitations, the joint motion of the manipulator has to be optimized. To completely control the three position and orientation coordinates of the end-effector in three dimensional space, a minimum of six degrees of freedom are required. A manipulator possessing more than the minimum required degrees of freedom for complete control of the end-effector motion is called a redundant manipulator. Redundant joint motion or self-motion of the manipulator moves the joints of the manipulator without affecting the end-effector position or orientation. A performance criterion is a function of the joint angles of the manipulator for which the manipulator motion is being optimized. The redundant degrees of freedom are used to optimize a performance criterion to overcome the limitations to the manipulator motion. Optimization of the performance criterion is performed by employing schemes to calculate the redundant joint motion in a manipulator. This redundant joint motion or self-motion applied to the manipulator helps in collision avoidance, joint limit avoidance and singularity avoidance.

Collision is an inherent problem with manipulators working in environments clustered with obstacles. This thesis is related to the implementation and testing of two optimization schemes for collision avoidance in manipulators. The two schemes, namely "Variable Dimension Task Space" and "Singularity Robust Inverse"[25,26], provide different approaches to solving the problem of collision in manipulators. The Variable Dimension Task Space utilizes the fact that some motion directions in the task space are more relevant to the task than other directions. Neglecting control of the less important motion directions results in an increase in the redundancy of the manipulator. This

additional redundancy can be used for optimization of a performance criterion to overcome the problem of collision. On the other hand, the Singularity Robust Inverse uses the existing redundancy for optimization. This method sacrifices the accuracy of the solution for the feasibility of solution. A feasible solution, in this case, is one which helps the manipulator avoid obstacles. The amount of error introduced in the trajectory is dependent on the scale factor and weighting matrices used. To study the effect of the scale factor on the error in the motion of the manipulator, constant and variable scale factors are used in the experiments.

The two collision avoidance schemes are implemented and tested on the seven degree of freedom Robotics Research Corporation K2107 manipulator[23]. These schemes are also implemented on the seven degree of freedom Long Reach Manipulator. The Long Reach Manipulator[1-5] is a redundant joint manipulator used in cleanup operations of nuclear waste tanks with hazardous waste. Since the manipulator is operating within the waste tank clustered with obstacles, the tested collision avoidance schemes will help optimize the motion of the Long Reach Manipulator for collision avoidance. Results of experiments for both schemes and their implementation on the RRC K2107 manipulator and the Long Reach Manipulator are presented in this thesis.

Table of Contents

Chapter 1 Introduction **1**

1.1 Thesis Objective 3

1.2 Thesis Organization 4

Chapter 2 Redundancy Resolution Using Gradient Projection Method **6**

2.1 Introduction 6

2.2 Gradient Projection Method of Redundancy Resolution 7

2.2.1 Optimization Using Gradient Projection Method 7

2.3 Obstacle Environment and Performance Criterion for Collision Avoidance 14

2.3.1 Modelling of Collision Objects 15

2.3.2 Performance Criterion for Collision Avoidance 17

2.3.3 Calculation of the Performance Criterion 19

2.3.4 Gradient of the Performance Criterion 20

2.4 RRC K2107 Manipulator 20

2.4.1 Modelling of the RRC K2107 Manipulator for Collision Avoidance 21

Chapter 3 Variable Dimension Task Space for Collision Avoidance **22**

3.1 Introduction 22

3.2 Concept of Variable Dimension Task Space (VDTS) 23

3.2.1 Calculation of Joint Motion using Variable Dimension Task Space 25

3.3 Experimental Setup 26

3.3.1 Experiments and Results 26

3.4 Conclusions 50

Chapter 4 Collision Avoidance Using the Singularity Robust Inverse.....52

4.1 Introduction to the Singularity Robust Inverse (SR-Inverse)[25,26].....52

4.2 Mathematical Equations for the SR-Inverse.....53

4.3 Properties of the SR-Inverse.....56

4.4 SR-inverse in Collision Avoidance.....56

4.5 Experimental Setup.....58

4.5.1 Experiments and Results.....58

4.6 Conclusions.....79

Chapter 5 Design and Control of the Long Reach Manipulator.....81

5.1 Introduction to Tank Waste Retrieval Project.....81

5.2 Tank Waste Retrieval System.....82

5.2.1 Waste Tank.....82

5.2.2 Long Reach Manipulator (LRM).....84

5.2.3 Dexterous Manipulator (DM).....84

5.2.4 Waste and Obstacles Inside The Tank.....85

5.3 Kinematic Design and Redundancy Resolution.....85

5.3.1 Kinematic Design Requirements for the LRM.....85

5.3.2 Kinematic Redundancy in the LRM.....87

5.4 Analysis of Initial Kinematic Designs of LRM.....89

5.4.1 Kinematic Design of LRM.....89

5.4.2 Sequential Entry Type Long Reach Manipulator (REPRPP).....91

5.4.3 Sequential Entry Type Long Reach Manipulator (REPYPP).....94

5.4.4 Comparison of SET Manipulator with Shoulder Roll Joint and Yaw Joint.....95

5.5 SDI/Spar Proposed LRM Design And Modified Design.....97

5.5.1 Initial Studies for the SDI/Spar Proposed LRM Design.....98

5.5.2 Deployment for SDI/Spar Proposed Design and Modified Design....102
5.6 Comparison of SDI/Spar Proposed Design and Modified Design....104
5.7 Conclusions....107
Chapter 6 Computer Program....110
6.1 Graphical User Interface and Popup Menus....110
6.2 Manipulator Control Subroutines....115
6.2.1 Subroutines added to main_pro.c....116
6.2.2 Subroutines added to goalp.c....116
6.2.3 Subroutines added to tools.c....116
6.2.4 Subroutines added to init_graph.c....118
Chapter 7 Conclusions and Recommendations....121
7.1 Conclusions....121
7.2 Recommendations....123
References....125
Appendices....130
Appendix A....131
Appendix B....132
Appendix C....134
Appendix D....140
Option button code added to file main_pro.c....140
k_inv( ) subroutine added to file goalp.c....148
least_normsr11( ) and least_normsr( ) subroutines added to file tools.c....159
cal_j*( ) subroutines added to file tools.c....161
least_norm*( ) subrotuines added to file tools.c....165

get_ortho* ( ) subroutines added to file tools.c.....181

Subroutines added to file init_graph.c.....189

File objenvs.c added to the robot code.....199

Vita.....215

List of Figures

Figure 1.1 Four Degree of Freedom Planar Redundant Manipulator 2

Figure 1.2 RRC K2107 Manipulator In a Clustered Environment 5

Figure 2.1 Rectangular Frame used as Collision Object 15

Figure 2.2 Rectangular Cube 16

Figure 2.3 Hexagonal Wire Frame Tube 16

Figure 2.4 Two Parallel Frames Separated by a Fixed Distance 17

Figure 2.5 Vector Links for the RRC K2107 Manipulator Model 21

Figure 3.1 Redundant Motion of RRC K2107 Manipulator 22

Figure 3.2 Manipulator configurations for least-norm using full Jacobian 27

Figure 3.3 Plot of Cartesian coordinates versus time for full Jacobian 28

Figure 3.4 Least Norm motion for 3x7 Jacobian with orientation neglected 28

Figure 3.5 Least Norm motion for 1x7 Jacobian with all motions except command direction neglected 28

Figure 3.6 Plot of Cartesian coordinates versus time for 3x7 Jacobian 29

Figure 3.7 Plot of Cartesian coordinates versus time for 1x7 Jacobian 29

Figure 3.8 Manipulator motion for full Jacobian with gain factor k=0.1 31

Figure 3.9 Manipulator motion for 3x7 Jacobian and gain factor k=0.1 31

Figure 3.10 Manipulator Configurations for 1x7 Jacobian and gain factor k=0.1 31

Figure 3.11 Plot of Joint angles versus motion time for 3x7 Jacobian and gain factor k=0.1 32

Figure 3.12 Plot of Cartesian Coordinates versus motion time for 3x7 Jacobian and k=0.1 32

Figure 3.13 Plot of norm of dtheta versus time for 6x7 Jacobian and gain factor k=0.1 33

Figure 3.14 Plot of norm of dtheta versus time for 3x7 Jacobian and k=0.1 33
Figure 3.15 Plot of norm of dtheta versus time for 1x7 Jacobian and k=0.1 34
Figure 3.16 Plot of performance criterion versus time for 6x7 Jacobian and k=0.1 34
Figure 3.17 Plot of performance criterion versus time for 3x7 Jacobian and k=0.1 34
Figure 3.18 Plot of performance criterion versus time for 1x7 Jacobian and k=0.1 34
Figure 3.19 Manipulator configurations for 6x7 Jacobian and k=0.01 36
Figure 3.20 Manipulator configurations for 3x7 Jacobian and gain factor 0.01 36
Figure 3.21 Manipulator configurations for 1x7 Jacobian and gain factor 0.01 36
Figure 3.22 Joint angle vs. time 3x7 Jacobian 37
Figure 3.23 Cartesian motion vs. time 3x7 Jacobian 37
Figure 3.24 Manipulator configurations for 6x7 Jacobian and gain k=0.05 37
Figure 3.25 Manipulator configurations for 3x7 Jacobian and k=0.05 39
Figure 3.26 Cartesian motion (6x7 Jacobian) 39
Figure 3.27 Cartesian motion (3x7 Jacobian) 39
Figure 3.28 Manipulator Configurations for full Jacobian 41
Figure 3.29 Manipulator Configurations for 3x7 Jacobian 41
Figure 3.30 Manipulator Configurations for 1x7 Jacobian 41
Figure 3.31 Cartesian Coordinate motion vs. time for 6x7 Jacobian 42
Figure 3.32 Cartesian Coordinate motion vs. time for 3x7 Jacobian 42
Figure 3.33 Norm of dtheta vs. time for 6x7 Jacobian 43
Figure 3.34 Norm of dtheta vs. time for 3x7 Jacobian 43
Figure 3.35 Manipulator Configurations for full Jacobian and k=0.15 44
Figure 3.36 Manipulator Configurations for 3x7 Jacobian and k=0.15 44
Figure 3.37 Cartesian motion (6x7 Jacobian) 45
Figure 3.38 Cartesian motion (3x7 Jacobian) 45

Figure 3.39 Manipulator Configurations for transition from full Jacobian to 3x7 Jacobian 46
Figure 3.40 Cartesian Motion vs. time 47
Figure 3.41 Joint motion vs. time 47
Figure 3.42 Performance Criterion vs. motion time 47
Figure 3.43 Manipulator Configurations for 3x7 Jacobian to full Jacobian 48
Figure 3.44 Cartesian Motion vs. time 49
Figure 3.45 Joint angles vs. time 49
Figure 3.46 Performance Criterion vs. motion time 49
Figure 4.1 Manipulator Configurations Using the Pseudoinverse 59
Figure 4.2 Manipulator Configurations Using SR-inverse and scale factor k_S=0 59
Figure 4.3 Cartesian Motion using the Pseudoinverse 60
Figure 4.4 Cartesian Motion using SR-inverse and scale factor k_S=0 60
Figure 4.5 Performance Criterion for both schemes 61
Figure 4.6 Manipulator Configurations using constant scale factor k_S=0.25 63
Figure 4.7 Joint angles for manipulator using constant scale factor k_S=0.25 63
Figure 4.8 Cartesian motion using constant factor k_S=0.25 64
Figure 4.9 Norm of dtheta using constant scale factor k_S=0.25 64
Figure 4.10 Performance Criterion using constant scale factor k_S=0.25 65
Figure 4.11 Manipulator Configurations using constant scale factor k_S=2.5 65
Figure 4.12 Joint angles for manipulator using constant scale factor k_S=2.5 66
Figure 4.13 Cartesian motion using constant factor k_S=2.5 66
Figure 4.14 Norm of dtheta using constant scale factor k_S=2.5 67
Figure 4.15 Performance Criterion using constant scale factor k_S=2.5 67
Figure 4.16 Manipulator configurations for variable scale factor k_S 69
Figure 4.17 Joint angles for manipulator with variable scale factor k_S 69

Figure 4.18 Cartesian motion using variable scale factor k_S 70
Figure 4.19 Norm of dtheta using variable scale factor k_S 70
Figure 4.20 Performance Criterion using variable scale factor k_S 71
Figure 4.21 Scale factor ks versus motion time 71
Figure 4.22 Manipulator Configurations for constant scale factor k_S=2.5 73
Figure 4.23 Joint angles for manipulator with constant scale factor k_S=2.5 73
Figure 4.24 Cartesian motion using constant scale factor k_S=2.5 74
Figure 4.25 Norm of dtheta using constant scale factor k_S=2.5 74
Figure 4.26 Performance Criterion using constant scale factor k_S=2.5 75
Figure 4.27 Manipulator configurations using variable scale factor k_S 75
Figure 4.28 Joint angles for manipulator using variable scale factor 76
Figure 4.29 Cartesian motion using variable scale factor 76
Figure 4.30 Norm of dtheta using variable scale factor 77
Figure 4.31 Performance Criterion using variable scale factor 77
Figure 4.32 Scale factor ks versus motion time 78
Figure 5.1 Nominal Waste Tank 83
Figure 5.2 Simulation Model for the TWR System 90
Figure 5.3 Long Reach Manipulator 91
Figure 5.4 Deployment for SET Manipulator With Roll Joint 93
Figure 5.5 Collision Avoidance for SET Manipulator (Roll Joint) 94
Figure 5.6 Deployment for SET Manipulator with Yaw Joint 96
Figure 5.7 Collision Avoidance for SET Manipulator (Yaw Joint) 97
Figure 5.8 Long Reach Manipulator 99
Figure 5.9 Workspace Coverage of SDI/Spar Design in Elbow Configuration and Free Space Motion. 100

Figure 5.10 Workspace Coverage of SDI/Spar Design in SCARA Configuration and Free Space Motion 101
Figure 5.11 Deployment of SDI/Spar Proposed LRM in SCARA Configuration 102
Figure 5.12 Deployment for Modified Design of LRM 103
Figure 6.1 Graphical User Interface 110
Figure 6.2 Popup Menu for Choice of Jacobian 111
Figure 6.3 Graphical Popup Button Menu for Collision Object Manipulation 112
Figure 6.4 Graphical Popup Menu for Manipulator Control Options 114
Figure A.1 Frame Attachment for RRC K2107 Manipulator 131
Figure C.1 Rectangular Frame In Manipulator Workspace 134
Figure C.2 Rectangular Cube in the Manipulator Workspace 135
Figure C.3 Hexagonal Tube Position and Orientation in Manipulator Workspace 137

List of Tables

Table 3.1 Manipulator Starting Configuration for Variable Dimension Task Space.....26

Table 4.1 Manipulator Starting Configuration for SR-Inverse.....58

Table 4.2 Test Conditions for the SR-inverse.....62

Table 5.1 Nominal Waste Tank Dimensions.....84

Table 5.2 Comparison of SDI/Spar Manipulator Design and Modified Design.....108

Table 6.1 Buttons on Collision Object Popup Menu.....113

Table 6.2 Buttons on Manipulator Control Options Popup Menu.....115

Table A.1 D-H parameters for the RRC K2107 Manipulator.....131

Table C.1.a Location of Frame Vertices.....135

Table C.1.b Link Lengths and Orientations for Frame.....135

Table C.2.a Location of Cube Vertices.....136

Table C.2.b Link Lengths and Orientations for Cube.....136

Table C.3.a Location of Hexagonal Tube Vertices.....138

Table C.3.b Link Lengths and Orientations for Hexagonal Tube.....139

Chapter 1

Introduction

A robot is an electromechanical device used to perform tedious, time consuming, repetitious tasks that may be unfit for a human being. Robot manipulators are devices which are made up of links attached together in series or parallel and have a mechanical hand or end-effector attached at the end of the chain of links. To move the end-effector to a desired position and orientation, the manipulator makes use of revolute or prismatic joints. A solid object of known dimension can be completely specified in three dimensional space with three independent position and three independent orientation coordinates. Each joint of the manipulator offers a single degree of freedom to the manipulator which in turn can control one of the motion coordinates in three dimensional space. Therefore, theoretically six degrees of freedom are necessary to control the complete motion of the end-effector, in three dimensional space. A manipulator possessing more degrees of freedom than necessary for complete control of the end-effector motion is referred to as a redundant manipulator. For example, take the case of a four joint planar robot shown in figure 1.1. The robot can only move in the plane of the paper and the end-effector can change its orientation in the plane of the paper. The end-effector can be completely defined by two position and one orientation coordinates. To control these three degrees of freedom a minimum of three joints are required. Therefore the fourth joint, an extra joint, is referred to as a redundant joint and such a manipulator is referred to as a redundant manipulator.

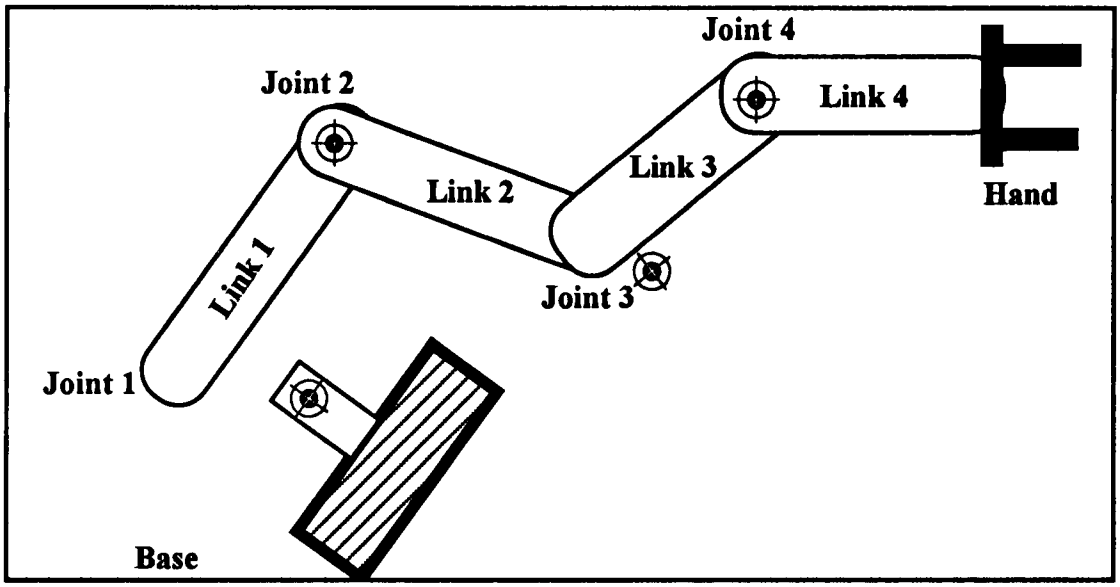


Figure 1.1 Four Degree of Freedom Planar Redundant Manipulator

Manipulators developed for specific repeatable tasks, such as those used in the automotive industry, can be designed for that task and need not have redundant joints. But manipulators working in unstructured and unknown environments, such as nuclear waste tanks and underwater excavation sites, require redundancy to overcome singularities, joint limits, joint velocity limits, and obstacle restrictions. Singularities are locations in the manipulator workspace where certain joint axis of the manipulator get aligned in such a manner that the manipulator loses the ability to move in certain directions.

Motion of a manipulator within its workspace is generally restricted by limits on the range of motion of the joints, limits on joint velocities, singularities and collision obstacles present within the workspace[20]. To overcome the restrictions, it is necessary for a manipulator to possess kinematic redundancy. Redundancy can also help to keep the manipulator in an operating condition upon failure of one of the primary joints.

For non-redundant manipulators in a nonsingular configuration each end-effector motion command results in a single and unique solution of joint commands. On the other hand, a redundant manipulator has an infinite number of joint solutions for a required end-effector motion. Since the Jacobian matrix which relates the end-effector velocities with joint velocities is not a square matrix, the inverse does not exist. In this case, the pseudoinverse[14] of the Jacobian matrix is used to find the minimum norm solution for the desired end-effector motion.

Redundancy in a manipulator can be effectively used to avoid obstacles during motion. Work related to optimizing a performance criterion to resolve the redundancy in a manipulator has been performed by many researchers. Defining and optimizing suitable performance criteria for avoiding singularities, joint limits and collision objects have been discussed in these studies. Optimizing the determinant of product of Jacobian and Jacobian transpose for singularity avoidance was proposed by Yoshikawa[17,24]. Optimizing the norm of the least-norm was proposed by Dubey[9]. Leigois[20] formulated criteria for avoiding joint limits using redundancy. Zhgal[13] defined the problem of collision avoidance for a multiple degree of redundancy robot operating through a window. Work related to optimizing multiple criteria simultaneously was done by Cleary[27], Pamanes[28] and McGhee[22]. Solutions for manipulators in neighborhood of singularities gave rise to the Singularity Robust Inverse developed by Nakamura and Hanafusa[26]. These are a few examples of the work done in optimizing performance criteria for redundancy resolution.

1.1 Thesis Objective

This thesis aims at developing real-time algorithms for collision avoidance in redundant and non-redundant manipulators. Two schemes, namely the "Variable

Dimension Task Space (VDTS)" and "Singularity Robust Inverse (SR-Inverse)"[26] are investigated for effectiveness in collision avoidance. The "Variable Dimension Task Space" scheme used for collision avoidance, increases the existing redundancy of a manipulator by neglecting control of some motion directions. This increased redundancy is then used in optimization for collision avoidance. Different gain factors are used for optimization using the Gradient Projection Method. This method calculates the redundant motion or self-motion of the manipulator by projecting a vector on the null space of the Jacobian matrix. On the other hand, the "Singularity Robust Inverse" scheme[26] sacrifices the accuracy of the solution by introducing error in the trajectory of the manipulator. This error allows the manipulator to change its end-effector orientation and position to maneuver in and around obstacles. The magnitude of error introduced in the trajectory depends on the scale factor used for the SR-inverse. To investigate the effect of the scale factor, constant and variable scale factors are used in the experiments. The algorithms for the different collision avoidance schemes will be implemented and tested on a seven degree of freedom Robotics Research Corporation (RRC) K2107 manipulator[23] in the Robotics Research Laboratory at the University of Tennessee.

The two collision avoidance schemes will also be implemented in the control algorithm of the seven degree of freedom Long Reach Manipulator[1-5] simulation model. Since the Long Reach Manipulator operates in an environment clustered with obstacles, the above collision avoidance schemes will be essential in optimization for collision avoidance. Inverse kinematics (Cartesian and Cylindrical coordinate) with the different collision avoidance schemes will be implemented on the simulation model of the LRM.

1.2 Thesis Organization

Chapter 2 of the thesis gives a background on the "Gradient Projection Method"[20] of optimization with the mathematical equations for real-time redundancy

resolution. This chapter also deals with the performance criterion for collision avoidance, its calculation and modeling of obstacles and the RRC K2107 manipulator for collision avoidance. Chapter 3 introduces the Variable Dimension Task Space scheme for collision avoidance with related equations, experiments, results and simulations. Chapter 4 outlines the Singularity Robust Inverse scheme for collision avoidance. This chapter also gives the equations for this scheme, experiments, results and simulations. Chapter 5 is devoted to the Long Reach Manipulator design and control. Chapter 6 gives a brief description about the special features of the computer code and information on subroutines. This is followed by conclusions and recommendations in Chapter 7. References used for the thesis are attached thereafter. Appendices A, B, and C give the D-H parameters of RRC K2107 manipulator, Gram-Schimdt orthogonolization method, and information on the collision objects used for the study, respectively. The computer code for different collision avoidance schemes and generation of graphical models is attached at the end of the thesis. Figure 1.2 shows a photograph of the RRC K2107 seven degree of freedom manipulator with a JR3 Force-Torque sensor on its end-effector.

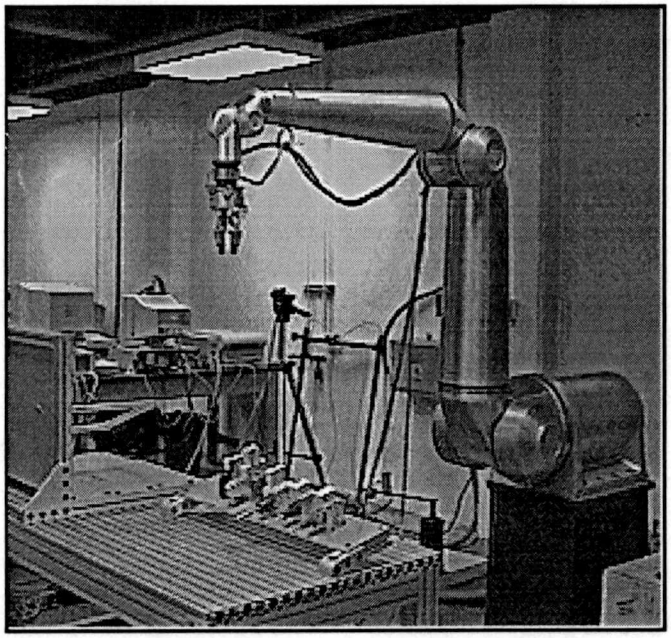


Figure 1.2 RRC K2107 Manipulator In a Clustered Environment

Chapter 2

Redundancy Resolution Using Gradient Projection Method

2.1 Introduction

As mentioned in the introduction, a redundant manipulator is a manipulator which has more degrees of freedom than necessary for control of complete motion of the end-effector in the given task space. For example, a manipulator operating in a three-dimensional task space requires a minimum of six joints to control the motion of the end-effector completely. Using a combination of the motion of all six joints, the end-effector of the manipulator can be positioned at any location and oriented in any desired configuration within the workspace of the manipulator. If this manipulator possesses more than six joints it is referred to as a redundant manipulator. On the other hand for a planar manipulator working only as a positioning device, more than two joints makes the manipulator redundant.

Redundant manipulators have many advantages over non-redundant manipulators. Manipulator motion is generally restricted by limit of motion range of its joints, joint velocity limits, singular configurations[20] within and at the boundary of the workspace and obstacles within the workspace. Redundancy in manipulators can be utilized to optimize the motion of the manipulator in presence of the above mentioned restrictions. To utilize the redundancy an efficient real-time redundancy resolution scheme is implemented. For the purpose optimizing the motion of manipulators in presence of obstacles, the Gradient Projection Method[20] is used for redundancy resolution.

2.2 Gradient Projection Method of Redundancy Resolution

As mentioned earlier, for a manipulator operating in three dimensional work space(3-D) the end-effector of the manipulator has six degrees of freedom namely the three positions and the three orientations. To control the motion of the end-effector in all the six directions the manipulator has to have a minimum of six joints. Any manipulator having more number of joints than the number of degrees of freedom of the end-effector is referred to as a redundant manipulator. If **m** is the degree of freedom of the end-effector and **n** are number of joints of the manipulator and if **n>m**, then **n-m** is the redundancy of the manipulator.

Optimization for the manipulator can be either global or local. Although local optimization may not result in the optimum trajectory for the manipulator end-effector, it is used as it is more feasible for a continuously changing trajectory and for on-line control of the manipulator. The Gradient Projection Method(**GPM**)[20] has been used for optimization. This method utilizes the null space of the Jacobian of the manipulator. Furthermore a performance criterion, depending on the optimization task, defined as a function of joint angles of the manipulator is presented and the gradient of this performance criterion is projected onto the null space of the Jacobian to get the homogenous solution for the joint velocities. The homogenous solution of joint velocities does not change the end-effector position or orientation but moves the manipulator to overcome obstacles, joint limits and singularities.

2.2.1 Optimization Using Gradient Projection Method

The kinematic equation describing the relationship between the end-effector velocities and joint velocities is

$$\dot{x}=J(\theta)\dot{\theta} \tag{2.1}$$

where $\dot{x}$ is a mx1 vector consisting of three linear and three rotational velocity components. $\dot{\theta}$ is a nx1 vector consisting of the joint velocities and $J(\theta)$ is the mxn Jacobian matrix.

For a redundant manipulator $n>m$ and we have infinite number of joint velocity solutions for a given $\dot{x}$. The joint velocity vector can be computed as

$$\dot{\theta}=J^{+}\dot{x}+(I-J^{+}J)\phi \tag{2.2}$$

where J^{+} is the Moore-Penrose generalized inverse[14] of the Jacobian matrix. $J^{+}\dot{x}$ is the least-norm solution of Eq.(2.1). $(I-J^{+}J)\phi$ is the homogenous solution of Eq.(2.1) and $\phi \in R^{n}$ is an arbitrary vector. For the Gradient Projection Method the vector ϕ in Eq.(2.2) is replaced by $k\nabla H$. Then Eq.(2.2) becomes

$$\dot{\theta}=J^{+}\dot{x}+k(I-J^{+}J)\nabla H \tag{2.3}$$

where ∇H is the gradient vector of the performance criterion $H(\theta)$ to be optimized and k is a real scalar coefficient. The scalar constant k is taken to be positive if $H(\theta)$ is to be

maximized and negative if $H(\theta)$ is to be minimized. The value of $\mathbf{k}$ is limited by bounds on joint velocities and actuator torque's.

To use Eq.(2.3) we need to calculate the pseudo-inverse of the Jacobian which is given by

$$\mathbf{J}^{+}=\mathbf{J}^{T}(\mathbf{J}\mathbf{J}^{T})^{-1} \tag{2.4}$$

for a Jacobian with the dimension $\mathbf{n}>\mathbf{m}$.

Determining $\mathbf{J}^{+}$ squares the condition number of the Jacobian matrix and reduces the accuracy of Eq.(2.2) near a singular point. Instead an efficient method of finding the least-norm and the homogenous solution for a redundant manipulator proposed by Dubey et al.[9] is used.

The least-norm solution of Eq.(2.1) can be expressed as

$$\dot{\theta}_{min}=\dot{\theta}_{p}-\dot{\theta}_{h} \tag{2.5}$$

where $\dot{\theta}_{min}$ is the least-norm solution of Eq.(2.1), $\dot{\theta}_{p}$ is the particular solution and $\dot{\theta}_{h}$ is the component of the homogenous solution in direction of the particular solution. The homogenous solution in Eq.(2.5) can be expressed as a combination of linearly independent homogenous solutions as

$$\dot{\theta}_h = \sum_{i=1}^{n-r} k_i \dot{\theta}_{h,i} \tag{2.6}$$

where n is the number of Jacobian matrix columns, r is the rank of the Jacobian matrix and $\dot{\theta}_{h,i}$ are a set of linearly independent homogenous solutions spanning the null space of the Jacobian. The weight coefficients k_i are such that the combination gives us the homogenous solution of Eq.(2.5).

For a rectangular Jacobian matrix with n>m, the end-effector velocity vector has dimension $\dot{x} \in R^m$ and the joint velocity vector has dimension $\dot{\theta} \in R^n$. We assume the Jacobian to be full ranked. Thus we may construct a nonsingular mxm matrix from any m independent columns of J . Eq.(2.1) can be written as

$$\begin{bmatrix} J_c & J_d \end{bmatrix} \dot{\theta} = \dot{x} \tag{2.7}$$

where J_c is an mxm nonsingular matrix containing the linearly independent columns of the Jacobian. The remaining columns of J form an mx(n-m) matrix J_d . To find the particular solution $\dot{\theta}_p$ we assume that the remaining (n-m) elements of $\dot{\theta}_p$ are zero, and the first m elements are solved using the equation

$$[J_c]\dot{\theta}_{pm} = \dot{x} \tag{2.8}$$

where $\dot{\theta}_{pm}$ contains the first m elements of the particular solution $\dot{\theta}_p$.

The set of linearly independent homogenous solutions $\dot{\theta}_{h,i}$ is obtained by setting the right hand of Eq.(2.7) to zero and assuming the last (n-m) elements of $\dot{\theta}_{h,i}$ to be zero except the m+ith element which is assumed to be 1. The first m elements of $\dot{\theta}_{h,i}$ are then solved for using the equation

$$[J_c]\dot{\theta}_{hm,i} = -J_{d,m+i} \quad , \quad i = 1,\ldots.n-m \tag{2.9}$$

where $\dot{\theta}_{hm,i}$ contains the first m elements of $\dot{\theta}_{h,i}$, and $J_{d,m+i}$ is the m+ith column of the Jacobian. The weight coefficients k_i in Eq.(2.6) can be obtained from the condition that the least-norm solution must be orthogonal to any homogenous solution. Hence we get

$$\dot{\theta}_{h,i}^{T}(\dot{\theta}_p - \sum_{j=1}^{n-m} k_i \dot{\theta}_{h,j}) = 0 \tag{2.10}$$

which can be written as

$$\begin{bmatrix} \|\dot{\theta}_{h,i}\|^2 & \dot{\theta}_{h,1}^{T}\dot{\theta}_{h,2} & \cdots & \dot{\theta}_{h,1}^{T}\dot{\theta}_{h,n-m} \\ \dot{\theta}_{h,2}^{T}\dot{\theta}_{h,1} & \|\dot{\theta}_{h,2}\|^2 & \cdots & \dot{\theta}_{h,2}^{T}\dot{\theta}_{h,n-m} \\ \vdots & \vdots & \cdots & \vdots \\ \dot{\theta}_{h,n-m}^{T}\dot{\theta}_{h,1} & \dot{\theta}_{h,n-m}^{T}\dot{\theta}_{h,2} & \cdots & \|\dot{\theta}_{h,n-m}\|^2 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_{n-m} \end{bmatrix} = \begin{bmatrix} \dot{\theta}_{h,1}\dot{\theta}_p \\ \dot{\theta}_{h,2}\dot{\theta}_p \\ \vdots \\ \dot{\theta}_{h,n-m}\dot{\theta}_p \end{bmatrix} \tag{2.11}$$

Solving for k_i from the above equation we can obtain the least-norm solution.

The second part of Eq.(2.3) is calculated by projecting the n dimensional gradient vector ∇H onto the null space of the Jacobian matrix. Let $\dot{\theta}_{\nabla H}$ be the projection of ∇H onto the null space of the Jacobian matrix. $\dot{\theta}_{\nabla H}$ is the linear combination of the set of homogeneous solutions $\dot{\theta}_{h,i}$ and is given by

$$\dot{\theta}_{\nabla H} = \sum_{i=1}^{n-m} w_i \dot{\theta}_{h,i} \tag{2.12}$$

The difference between ∇H and $\dot{\theta}_{\nabla H}$ is the vector that is orthogonal to the set of homogenous solutions $\dot{\theta}_{h,i}$ and can be expressed as

$$\dot{\theta}_{h,i}^{T}(\nabla H - \sum_{i=1}^{n-m} w_i \dot{\theta}_{h,i}) = 0 \tag{2.13}$$

Therefore we can solve for the weights w_i using the equation

$$\begin{bmatrix} \left\|\dot{\theta}_{h,i}\right\|^2 & \dot{\theta}_{h,1}^{T}\dot{\theta}_{h,2} & \cdots & \dot{\theta}_{h,1}^{T}\dot{\theta}_{h,n-m} \\ \dot{\theta}_{h,2}^{T}\dot{\theta}_{h,1} & \left\|\dot{\theta}_{h,2}\right\|^2 & \cdots & \dot{\theta}_{h,2}^{T}\dot{\theta}_{h,n-m} \\ \vdots & \vdots & \cdots & \vdots \\ \dot{\theta}_{h,n-m}^{T}\dot{\theta}_{h,1} & \dot{\theta}_{h,n-m}^{T}\dot{\theta}_{h,2} & \cdots & \left\|\dot{\theta}_{h,n-m}\right\|^2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{n-m} \end{bmatrix} = \begin{bmatrix} \dot{\theta}_{h,1}\nabla H \\ \dot{\theta}_{h,2}\nabla H \\ \vdots \\ \dot{\theta}_{h,n-m}\nabla H \end{bmatrix} \tag{2.14}$$

Substituting these weights in Eq.(2.12) we get the homogenous solution of Eq.(2.3). Therefore the optimized solution is

$$\dot{\theta}=\dot{\theta}_{min}+\dot{\theta}_{VH} \tag{2.15}$$

In real time implementation, we determine $\Delta\theta$, change in the joint angle vector for the time period Δt as

$$\Delta\theta=\Delta\theta_{min}+k\frac{\Delta H_h}{\|\Delta\theta_h\|}n_j \tag{2.16}$$

where the time derivatives are replaced by the corresponding incremental Δ terms. In the above equation $\Delta\theta_{min}$ may be computed using the scheme proposed by Dubey et al., n_j is a unit vector for each homogenous solution, and ΔH_h is obtained using the equation

$$\Delta H_h=H(\theta+\Delta\theta_h)-H(\theta) \tag{2.17}$$

where θ is current joint angle vector. The scalar coefficient k is taken to be positive to maximize the performance criteria for collision avoidance. This numerical

method allows us to optimize a performance criteria without knowing the symbolic form of its gradient.

The above method of calculating the joint velocities for given end-effector velocities does not calculate the pseudo inverse of the Jacobian. In the implementation of the collision avoidance algorithm in this thesis the pseudo inverse is used for calculating the least norm and the homogenous solution for Eqn.(2.2). For a redundant system with one degree of redundancy the method proposed by Dubey et. al. is much faster and efficient than the method of calculating the joint velocities using the pseudo inverse of the Jacobian matrix. But as the redundancy of the system increases the calculation load and time required also increase and in that case the pseudoinverse is faster in performing the calculations. Chan in his thesis has shown that for a system with m=6 and n>9 the method proposed by Dubey et.al. requires more calculation than the pseudoinverse.

2.3 Obstacle Environment and Performance Criterion for Collision Avoidance

To implement the Gradient Projection Method of optimization, described above, a performance criterion for the manipulator is used. The performance criterion is usually a function of joint angles of the manipulator and defines the criterion to be optimized. Since this thesis deals only with collision avoidance in manipulators, the discussion of performance criteria, in the following sections, is limited to the collision avoidance performance criterion. Also to test the different schemes used for collision avoidance, obstacles were modelled and placed within the workspace of the manipulator. The following sections discuss the modelling of collision objects, defining the performance criterion for collision avoidance and calculations involved in evaluating the performance criterion.

2.3.1 Modelling of Collision Objects

Collision objects were modelled on the Silicon Graphics workstation using the executable binary code namely build_environment written by Scott McGhee[ ]. All collision objects are modelled as structures consisting of three-dimensional links attached together to form a defined shape like a cube or a frame. A rectangular frame used for testing the collision avoidance algorithm is shown in figure 2.1.

The above object is constructed using four links namely link AB, link BC, link AD, and link CD. The collision object is first constructed in the collision object environment and then imported to the robot environment. Link length and orientation data for each collision object is kept in a separate file residing in the robot code directory. Other objects modelled were a rectangular cube shown in figure 2.2, hexagonal wire frame tube shown in figure 2.3 and two parallel frames separated by a fixed distance shown in figure 2.4.

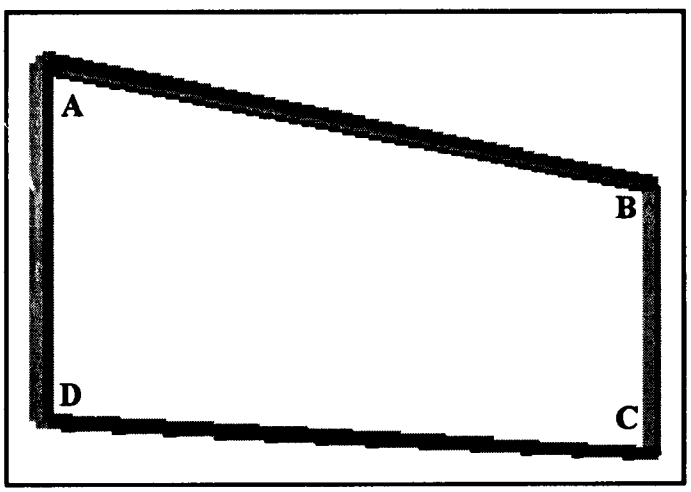


Figure 2.1 Rectangular Frame used as Collision Object

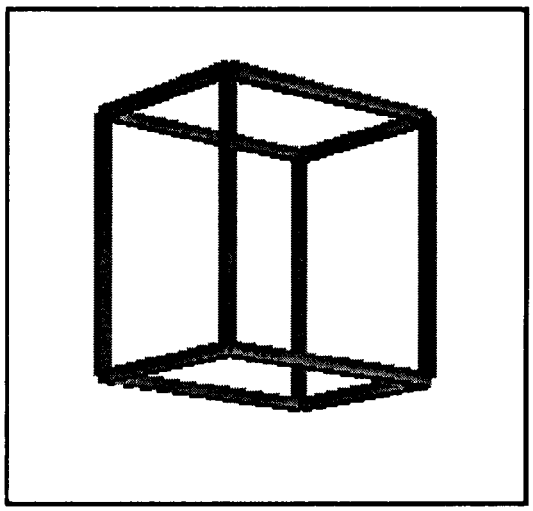


Figure 2.2 Rectangular Cube

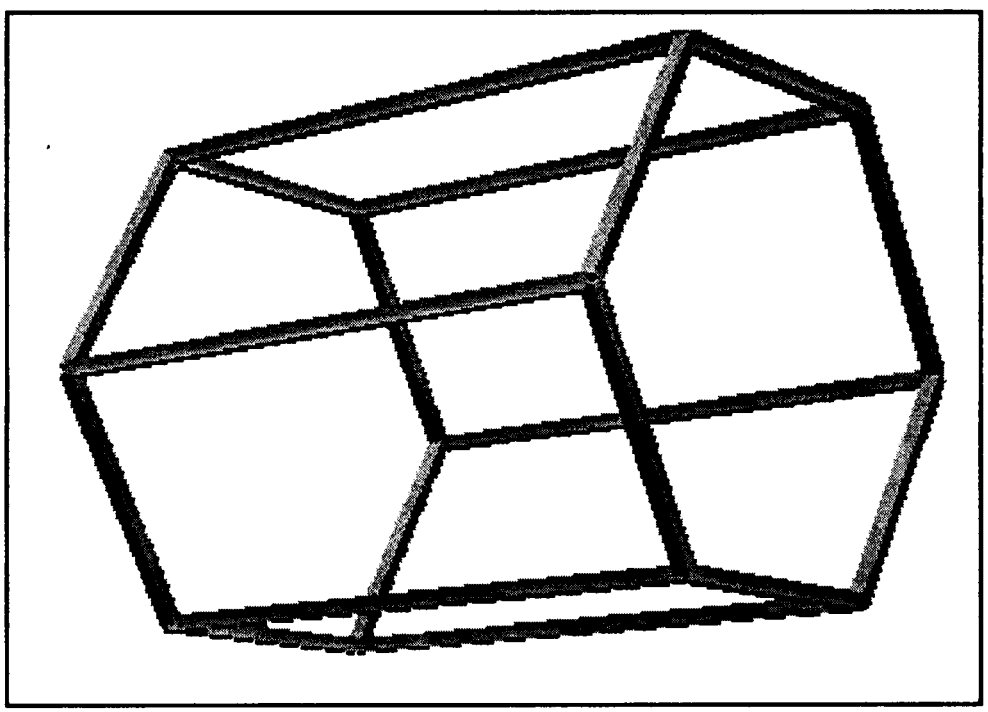


Figure 2.3 Hexagonal Wire Frame Tube

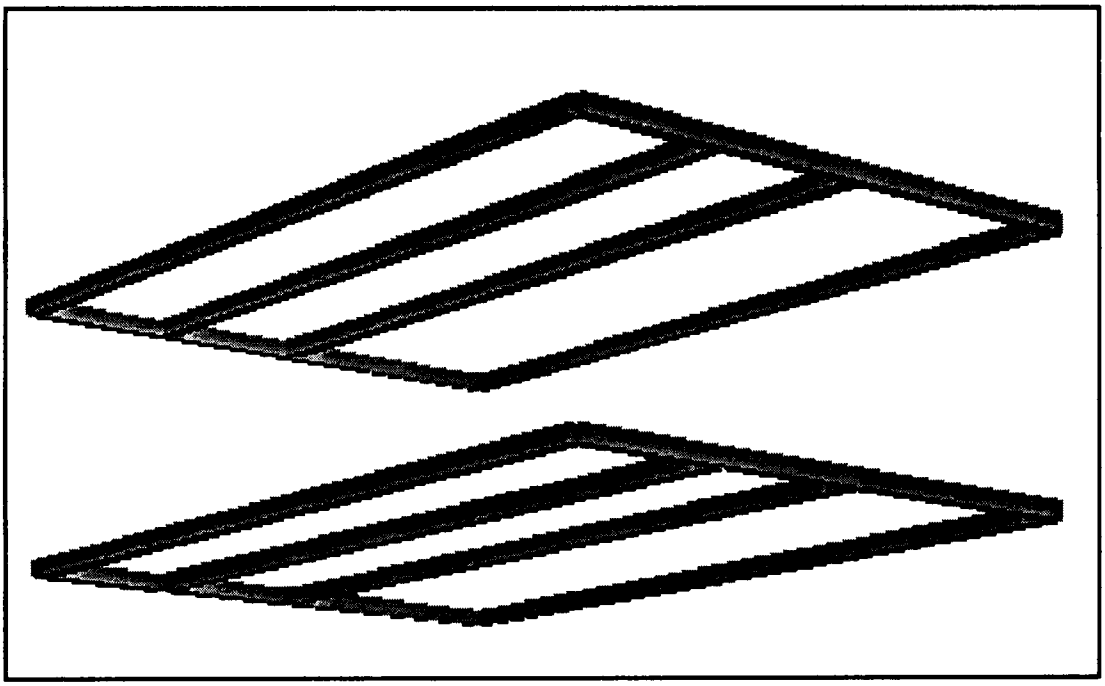


Figure 2.4 Two Parallel Frames Separated by a Fixed Distance

2.3.2 Performance Criterion for Collision Avoidance

Collision between a manipulator and an obstacle occurs when a link of the manipulator comes in contact with any portion of the obstacle within its workspace. Hence, to avoid collision, it is necessary to keep the manipulator far away from the obstacle . It may not always be possible and necessary to keep the manipulator far away from obstacles in an obstacle clustered environment. But maintaining a minimum safety distance between any part of the manipulator and obstacle is necessary for effective collision avoidance.

To optimize the motion of the manipulator a performance criterion is defined. It is obvious the distance between the manipulator link and the obstacle is important for

collision avoidance. The larger the distance between the manipulator links and the obstacles the lesser are the chances of collision. Hence the criterion for collision avoidance should include the distance between the manipulator links and the collision object.

The following performance criteria is used for optimization of motion of the manipulator for collision avoidance,

$$H(\theta)=\sum_{i=1}^{N_a}\sum_{j=1}^{N_o}\frac{1}{S_{D_{i,j}}-D_{stop}} \qquad (2.18)$$

where N_a is the number of manipulator links being considered for collision, N_o is the number of collision objects being modeled and $S_{D_{i,j}}$ is the shortest distance between the i^{th} arm and j^{th} collision object. D_{stop} is the minimum safety distance to be maintained between any arm and collision object to avoid collision. If the shortest distance goes below the minimum safety distance to be maintained then the manipulator stops as collision is eminent.

As mentioned earlier, the obstacles are modelled as objects made up of links. Hence, for the entire manipulator to avoid collision, each link of the manipulator has to avoid collision with each link of the obstacle. The performance criterion defined in Eqn. 2.18 is minimized for collision avoidance. This is because the term on the right hand side of the equation is the summation of the inverse of the shortest distance between each manipulator link and each link of the collision object. If there is no minimum safety distance, it can be seen that the criterion tends to infinity as the manipulator nears the collision object and is about to collide with the obstacle. The minimum safety distance is necessary to prevent any unforeseen damage to the manipulator and always keep the

manipulator at a safe distance from the obstacles. Furthermore, this minimum safety distance stops further motion of the manipulator if the distance between any portion of the manipulator and obstacle falls below the specified safety distance and collision is unavoidable.

2.3.3 Calculation of the Performance Criterion

The performance criterion defined in Eqn. 2.18 involves the calculation of the shortest distance between each manipulator link and each obstacle link[19]. For this purpose the data file for each obstacle and the homogenous transforms for the manipulator is used. Each link of the obstacle and the manipulator is modelled as a vector of fixed length and direction. Since in this study the obstacle is assumed to be stationery, the direction of each obstacle link vector will be constant throughout. The only vectors changing direction will be the manipulator link vectors. The end-points and orientations for each obstacle link can be directly read from the data file for that obstacle while those for the manipulator are calculated by using the information from the homogenous transform from one end of the link to the other.

The shortest distance between any two vectors is a line perpendicular to both the vectors. So finding the shortest distance is just finding a vector whose one endpoint lies on the manipulator link and the other endpoint on the obstacle link and is perpendicular to both these links. The only drawback is that there does not exist any single formulae or expression to calculate the minimum distance between any two vectors in three dimensional space for all cases of orientation and position of the two link vectors. So to calculate the above shortest distance, a new coordinate system is defined and is called the collision coordinate system. All calculations of shortest distance between the manipulator link vector and obstacle link vector are performed in this collision coordinate system. 29

different cases of orientation and positions of the link vectors are possible and each case has to be handled by a different expression for the shortest distance.

The calculation of the minimum distance vector by this method, including the transformations and mappings requires 88 operations in all. This is made up of 60 multiplication's and 28 additions/subtraction's.

2.3.4 Gradient of the Performance Criterion

For optimization using Gradient Projection Method the gradient of the performance criterion is required. We have defined the performance criterion as $\mathbf{H}(\theta)$, a function of the joint angles of the manipulator. The gradient of this performance criterion is defined as $\nabla \mathbf{H}$, where

$$\nabla \mathbf{H} = \left[\frac{\partial \mathbf{H}}{\partial \theta_1} \quad \frac{\partial \mathbf{H}}{\partial \theta_2} \quad \frac{\partial \mathbf{H}}{\partial \theta_2} \quad \cdots\cdots \quad \frac{\partial \mathbf{H}}{\partial \theta_n} \right] \tag{2.19}$$

In real-time implementation the gradient of the performance is calculated using Eqn. 2.17.

2.4 RRC K2107 Manipulator

For the purpose of testing the collision avoidance algorithms, the Robotics Research Corporation K2107[23] seven degree of freedom manipulator was used. This manipulator has a single degree of natural redundancy.

2.4.1 Modelling of the RRC K2107 Manipulator for Collision Avoidance

To implement the collision avoidance scheme on this manipulator, the manipulator links had to be modelled as vectors. For the above mentioned purpose four link vectors encompassing the entire manipulator were identified as shown in figure 2.5.

The four link vectors shown in figure 2.5 are the vector AB the base vector, BC the upper arm vector, CD the forearm vector and DE the wrist vector. But the base vector is not modelled for two reasons namely, the base vector does not change its position but only its orientation when the manipulator is in motion and eliminating the base vector reduces the computational load in the calculation of the shortest distance vectors.

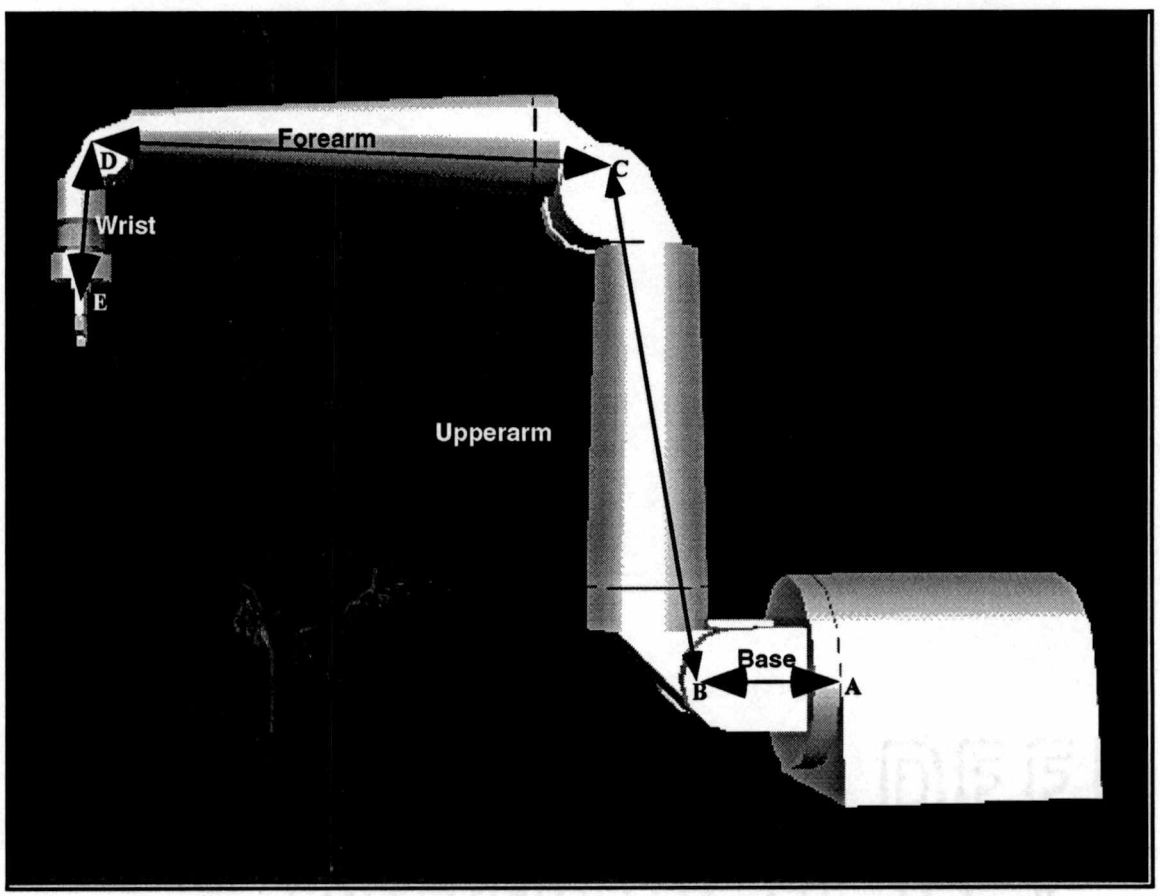


Figure 2.5 Vector Links for the RRC K2107 Manipulator Model

Chapter 3

Variable Dimension Task Space for Collision Avoidance

3.1 Introduction

The previous chapter discussed redundancy resolution for optimizing the motion of manipulators to avoid joint limits, joint velocity limits, singularities and obstacles. A redundant manipulator, unlike a nonredundant manipulator has infinite number of joint solutions for a given end-effector motion command. If there is no optimization, the joint solution with minimum norm is chosen as the feasible solution. This reduces the cost of moving the manipulator, the cost being the energy required to move the joints. For redundant manipulators, one can find a combination of joint motion which will result in no motion of the end-effector of the manipulator. This motion for one degree of redundancy of the RRC K2107 manipulator is shown in figure 3.1

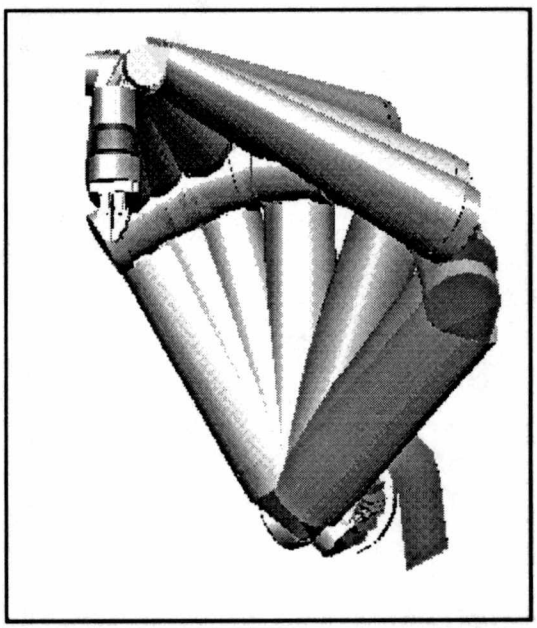


Figure 3.1 Redundant Motion of RRC K2107 Manipulator

In the above case, the end-effector of the manipulator maintains its position and orientation for redundant motion of the manipulator which is multiple elbow configurations of the manipulator.

Since collision avoidance is the main issue being dealt with in this thesis, the analysis will be restricted to that area although it might be true for other criteria. In many cases a single degree of redundancy is not sufficient for optimizing the manipulator motion to avoid obstacles. Also in case of nonredundant manipulators the absence of redundant joints deprives the manipulator of the ability to avoid obstacles. In such cases the only possible solution is to plan the motion of the manipulator from the beginning to the goal point prior to actually moving the manipulator. But this is not possible in case of unstructured environments where precise information on the size and location of the obstacles within the workspace is not available. The question is, what do we do in such cases ? One solution to the above problem is presented in this chapter and is referred to as the Variable Dimension Task Space (VDTS).

3.2 Concept of Variable Dimension Task Space (VDTS)

Since redundancy facilitates optimization, the more the redundancy of the manipulator the greater is its ability to avoid obstacles. For a seven degree of freedom manipulator operating in three dimensional space, six joints are required to position and orient the end-effector of the manipulator. This leaves the manipulator with one degree of natural redundancy. On the other hand for a six degree of freedom manipulator operating in three dimensional space there is no existing redundancy.

Suppose, we are only interested in using the manipulator as a positioning device. The orientation of the end-effector is of no concern to the user. In that case, only three

degrees of freedom of the manipulator are required for controlling the desired motion of the end-effector. This leaves us with four degrees of redundancy in case of a seven degree of freedom manipulator and three degrees of redundancy in case of a six degree of freedom manipulator. On the other hand, suppose the manipulator needs to move only in the XY plane then only two degrees of freedom are required and that leaves us with an additional redundant joint. This "non-natural" redundancy realized by restricting control to certain task space motions and the scheme of introducing this task based redundancy into a manipulator system is called the Variable Dimension Task Space (VDTS).

To cite examples of the concept of Variable Dimension Task Space lets consider a six degree of freedom manipulator holding a welding gun at its end-effector. This manipulator has no existing redundancy. But in most cases the orientation of the welding gun around its own roll axis is unimportant and can be neglected or left uncontrolled. This adds one degree of artificial redundancy into the system. Another example is that of a manipulator carrying a tool to a certain desired location to perform some operation. In this case the orientation of the tool can be initially neglected giving the manipulator extra redundancy. Once the tool has been moved close to the target point, control of all Cartesian coordinates can be resumed and the tool can be oriented in the desired configuration. A camera held by a manipulator tracking an object is another example. The orientation of the camera about its roll axis and its position may be neglected (using the zoom lens may suffice) adding redundancy to the system.

The choice of motion directions to be neglected and those to be controlled depend largely on the type of the manipulator and the type of task being performed by the manipulator. Note that it may not always be feasible and necessary to add artificial redundancy into the manipulator system.

3.2.1 Calculation of Joint Motion using Variable Dimension Task Space

In chapter 2 equations necessary for calculating the joint motion for a redundant manipulator were presented. The real-time method proposed by Dubey et. al. is very efficient for manipulators with a single degree of redundancy. But for manipulators with higher redundancy the pseudoinverse[14] with Gaussian elimination is used to solve for the joint motion corresponding to desired end-effector motion. To find the weights given in equation 2.12 we use equation 2.14. But for higher degrees of redundancy this involves using Gram-Schmidt orthogonalization method to make the vectors orthonormal. Instead, if we use the pseudoinverse given in equation 2.4 the calculations become less intensive in case of higher degrees of redundancy.

To calculate the gradient of the performance criterion either equation 2.17 or if the calculations are done using the pseudoinverse the following method is used. Since the symbolic form of the gradient given by equation 2.19 is not known, a numerical method is used. The gradient at the current joint angles is calculated. The angles are then perturbed by a small amount $d\theta$. The value of the gradient is then calculated for the perturbed angles. The gradient of the performance criterion is then given as

$$\frac{\partial H}{\partial \theta_i} = \frac{H(\theta + d\theta) - H(\theta)}{d\theta_i} \tag{3.1}$$

where i=1,....,n is the number of joints for the manipulator.

Calculation of joint angles for high degree of redundancy using both the real-time method and using the pseudoinverse is performed for comparison and results are presented later in the chapter.

3.3 Experimental Setup

To verify the collision avoidance scheme presented in this chapter, the code was tested on the seven degree of freedom RRC K2107 manipulator. Collision objects modelled were placed at known positions within the workspace of the manipulator and the manipulator was made to move from the home position to a point beyond the obstacle in a way such that the manipulator had to encounter the obstacle and avoid it successfully to reach the final position and orientation.

Three collision objects were used in the experiments, a rectangular frame, a rectangular cube and a hexagonal tube. The details for position, orientation and dimensions of these collision objects is presented in appendix C. For all the cases tested below the manipulator end-effector velocity was maintained at 30 cm/sec and starting configuration was the home configuration given in table 3.1.

Table 3.1 Manipulator Starting Configuration for Variable Dimension Task Space

Configuration	X (cms.)	Y (cms.)	Z (cms.)	Roll (degrees)	Pitch (degrees	Yaw (degrees)
Home	0.00	126.512	75.385	90.0	0.0	180.0

3.3.1 Experiments and Results

To prove the effectiveness of optimization in manipulator motion, the least-norm solutions of the manipulator for the frame for three different manipulator task dimension space is shown below.

Figure 3.2 shows the manipulator motion for a single degree of redundancy with no optimization. The manipulator starting configuration is the home position of the manipulator and command motion direction is the X-direction.

Figure 3.3 shows the plot of Cartesian coordinates against motion time for the above case. The Y, Z, Roll, Pitch, and Yaw all remain unchanged as the manipulator is moving in the X direction. Similar figure and plots are shown for cases when the orientation of the end-effector is neglected and when all coordinate motions except the command direction are neglected.

Figures 3.4 and 3.5 show the simulation configurations for the manipulator using only the least-norm solution when orientation is neglected and all motion directions except the command direction is neglected and figures 3.6 and 3.7 show the corresponding Cartesian plots .

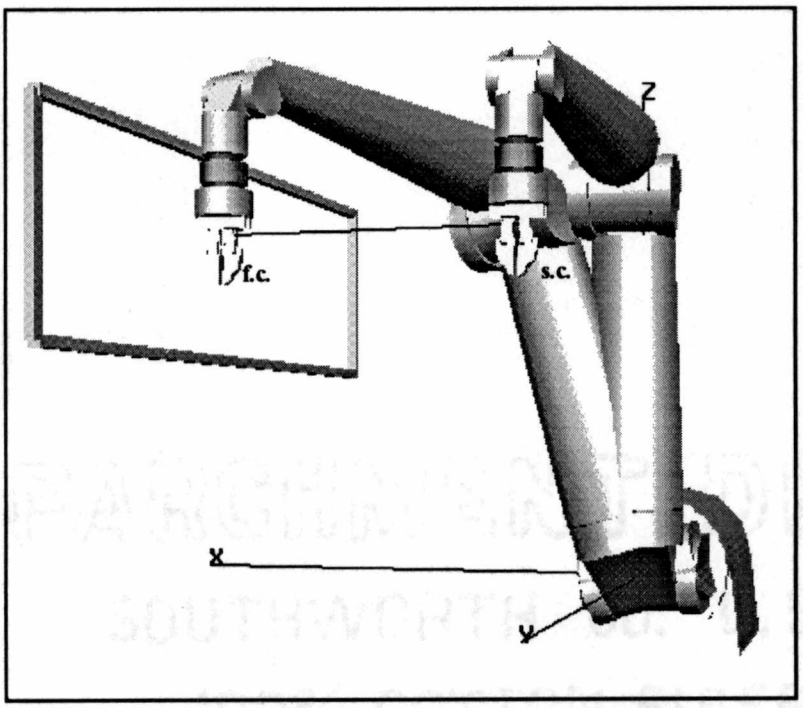


Figure 3.2 Manipulator configurations for least-norm using full Jacobian

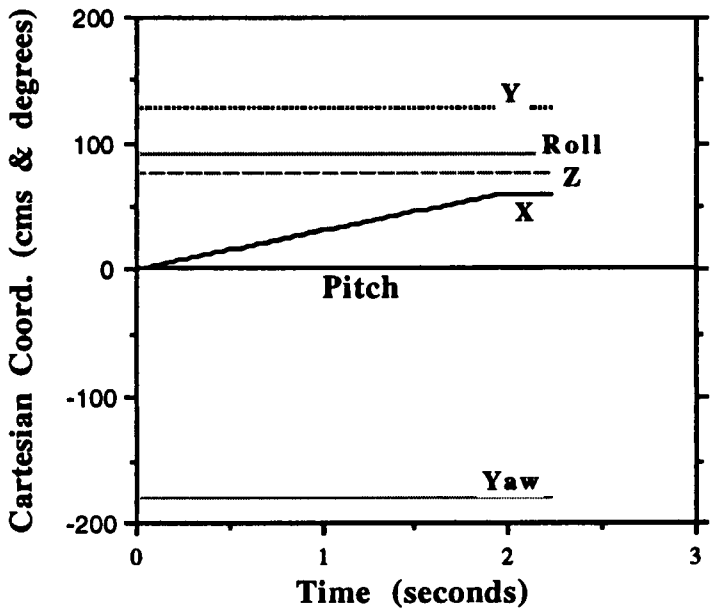


Figure 3.3 Plot of Cartesian coordinates versus time for full Jacobian

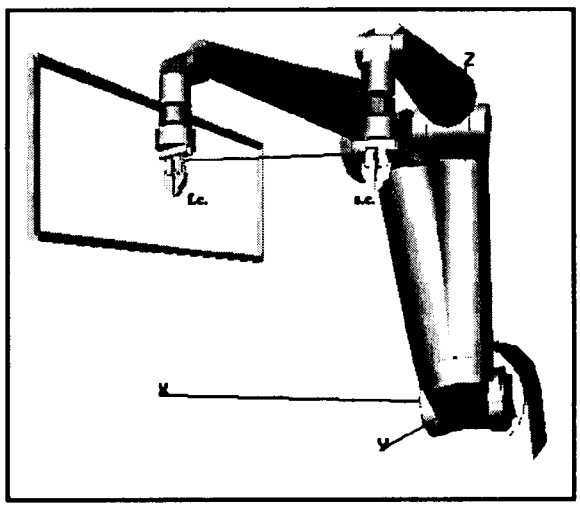


Figure 3.4 Least Norm motion for 3x7 Jacobian with orientation neglected

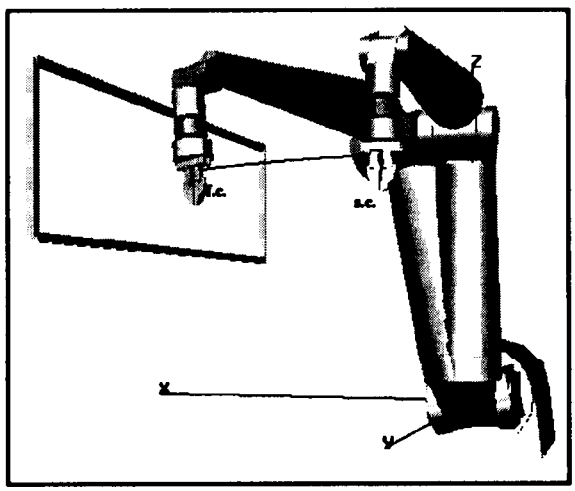


Figure 3.5 Least Norm motion for 1x7 Jacobian with all motions except command direction neglected

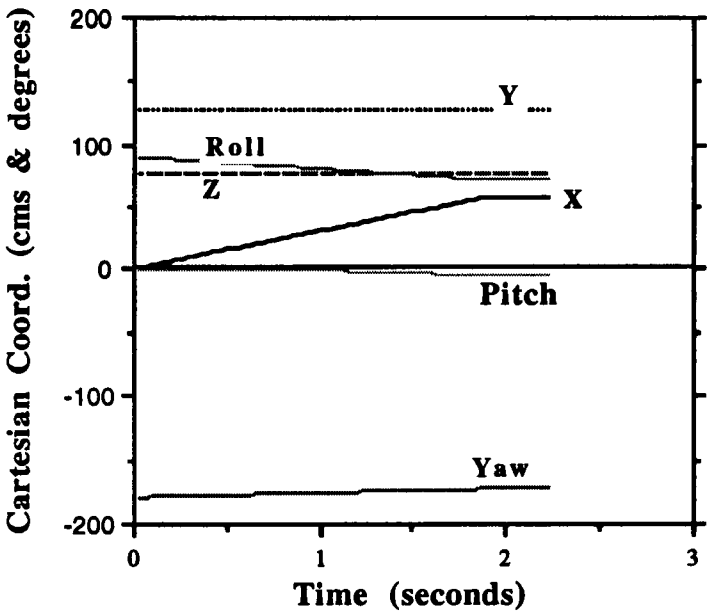


Figure 3.6 Plot of Cartesian coordinates versus time for 3x7 Jacobian

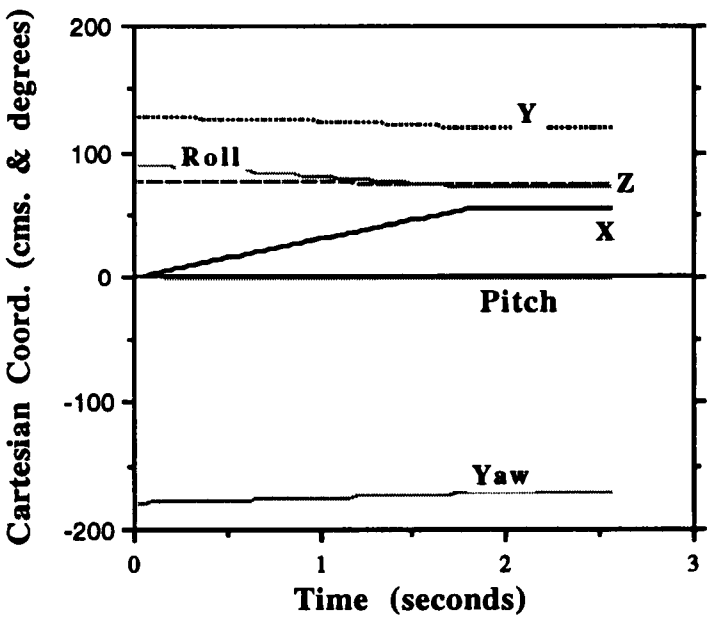


Figure 3.7 Plot of Cartesian coordinates versus time for 1x7 Jacobian

Now the results for optimized motion of the manipulator for different dimensions of the task space are presented. The three task spaces considered for the study were the following

a) Six dimension task space where all positions and orientations are controlled.

b) Three dimension task space where only the three positions are controlled and orientations neglected.

c) One dimensional task space where only command motion direction is controlled and all other motion directions neglected.

I) Collision Object : Rectangular frame

In case of the rectangular frame, the Variable Dimension Task Space was tested for three different cases. All three cases used the Gradient Projection Method of optimization. The first case the scalar gain factor for optimization was 0.1 and in the second case it was 0.01. In both these cases the pseudoinverse was used to perform the calculations. In the third case the gain factor was 0.05 and the method proposed by Dubey et. al. was used for calculation of joint angles. The simulation results and plots for joint angles, norm of dtheta and performance criterion for the three cases are presented in the following pages.

Case 1 : Calculations using pseudoinverse and gain factor k=0.1

Figures 3.8, 3.9, and 3.10 show the manipulator motion configurations for cases when a full Jacobian, Jacobian without orientation control and Jacobian with only command direction control is used. Plots for joint angles and Cartesian Coordinate motion for the 3x7 Jacobian case are shown in figure 3.11 and 3.12. The plots for performance criterion and dtheta are shown in figures 3.13 to 3.18.

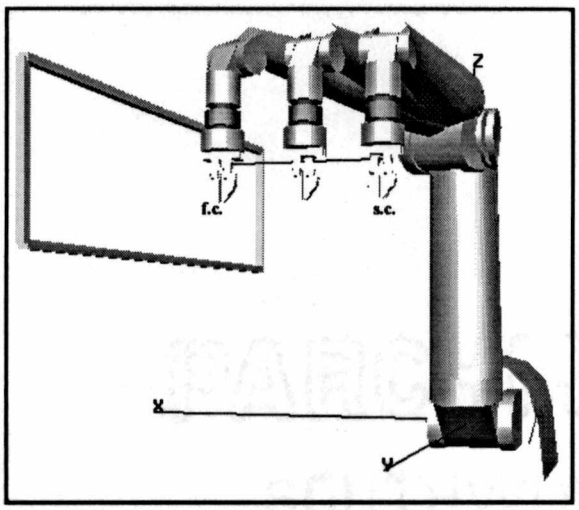


Figure 3.8 Manipulator motion for full Jacobian with gain factor k=0.1

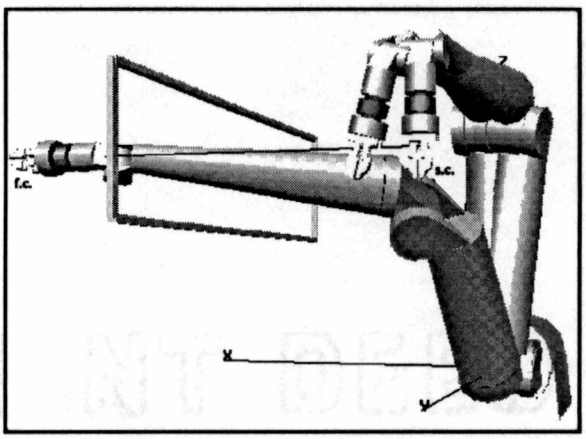


Figure 3.9 Manipulator motion for 3x7 Jacobian and gain factor k=0.1

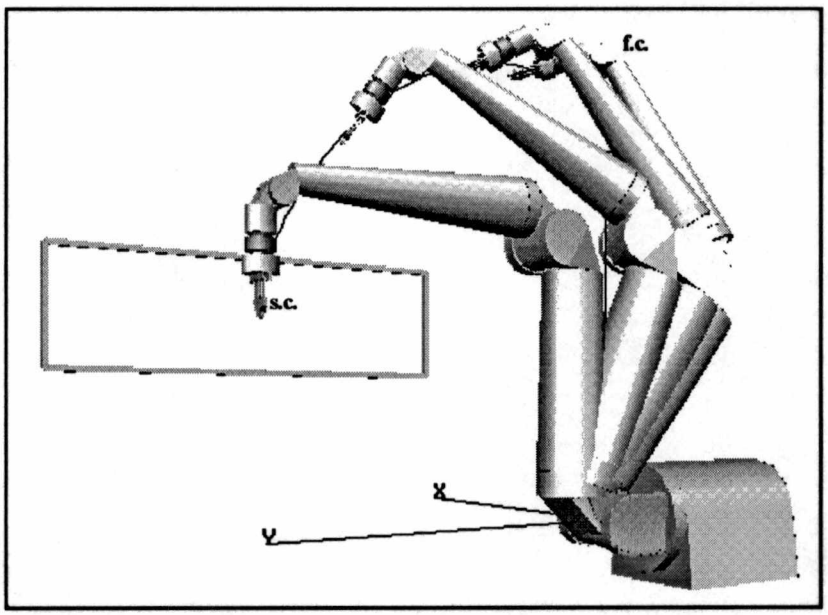


Figure 3.10 Manipulator Configurations for 1x7 Jacobian and gain factor k=0.1

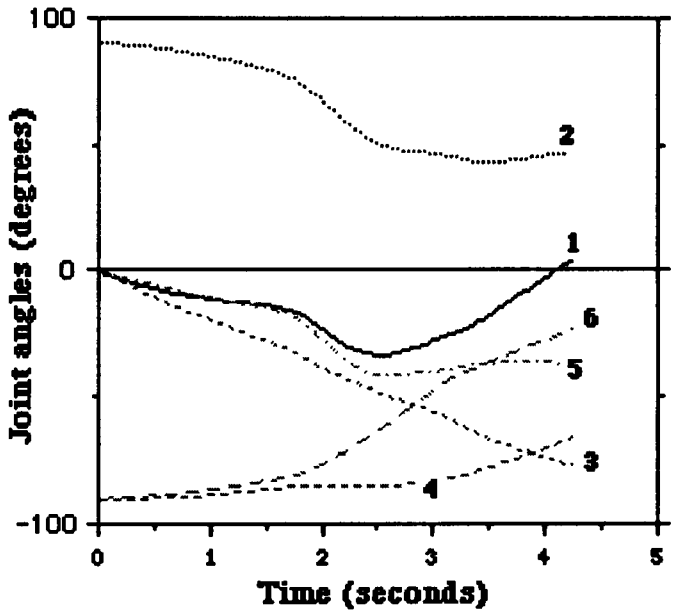


Figure 3.11 Plot of Joint angles versus motion time for 3x7 Jacobian and gain factor k=0.1

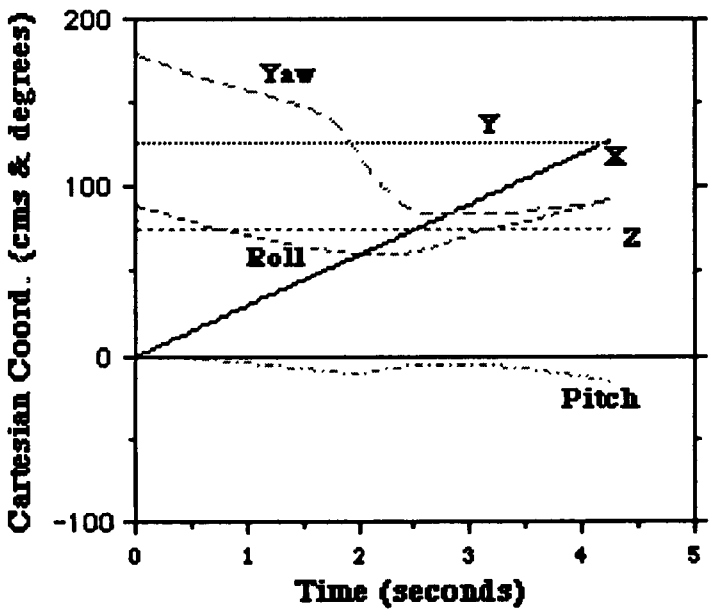


Figure 3.12 Plot of Cartesian Coordinates versus motion time for 3x7 Jacobian and k=0.1

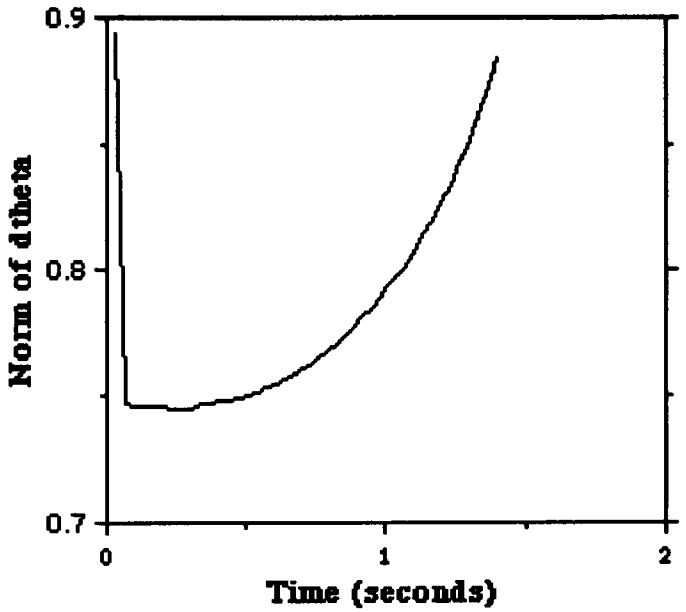


Figure 3.13 Plot of norm of dtheta versus time for 6x7 Jacobian and gain factor k=0.1

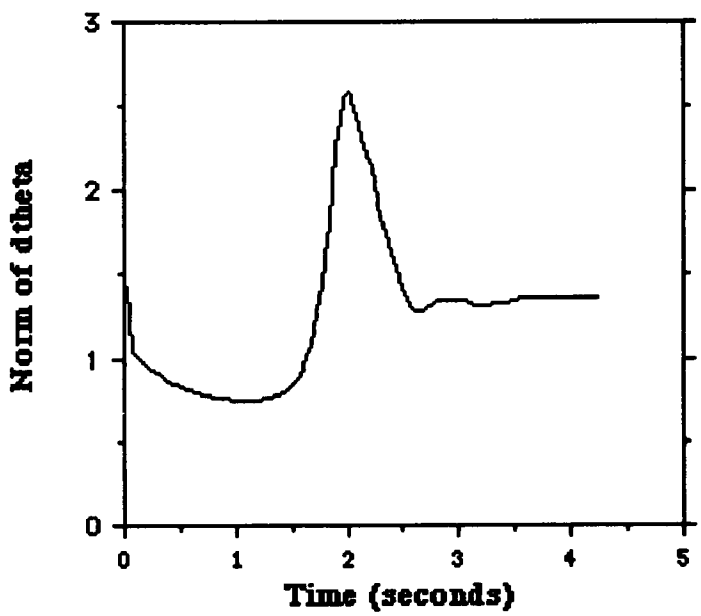


Figure 3.14 Plot of norm of dtheta versus time for 3x7 Jacobian and k=0.1

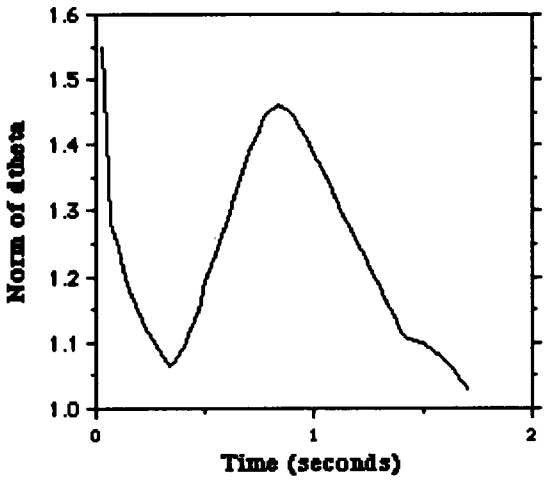


Figure 3.15 Plot of norm of dtheta versus time for 1x7 Jacobian and k=0.1

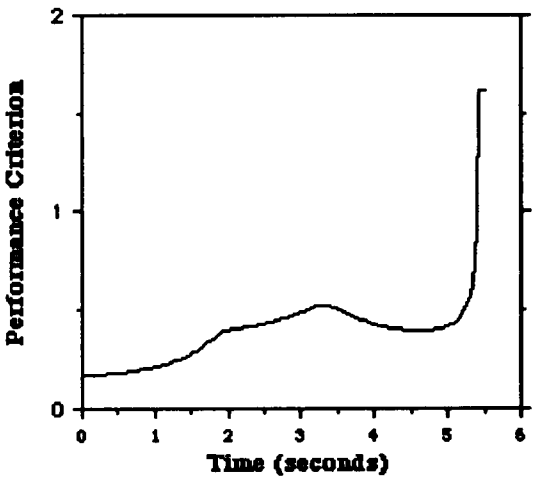


Figure 3.17 Plot of performance criterion versus time for 3x7 Jacobian and k=0.1

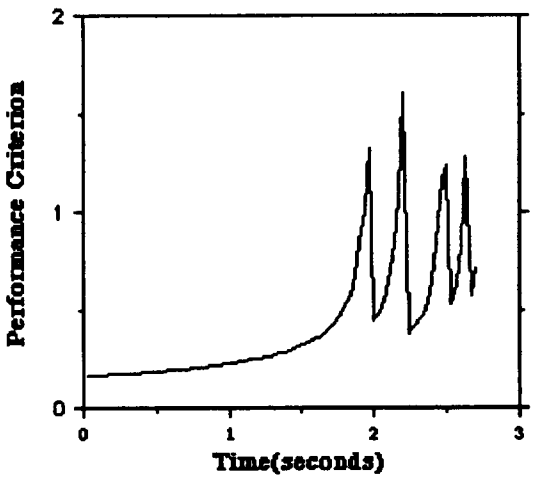


Figure 3.16 Plot of performance criterion versus time for 6x7 Jacobian and k=0.1

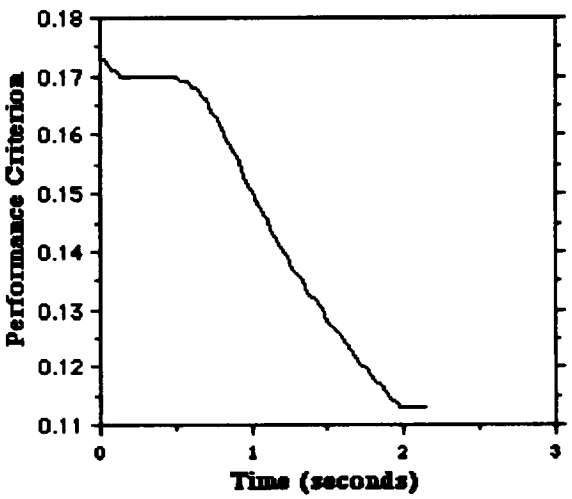


Figure 3.18 Plot of performance criterion versus time for 1x7 Jacobian and k=0.1

The plot in figure 3.12 shows the variation of X, Y, Z, Roll, Pitch and Yaw coordinates with motion time. Since orientation is neglected in this case Roll, Pitch, and Yaw coordinates do not remain constant during motion. X is the motion command direction and Y and Z remain constant.

Figures 3.13, 3.14, and 3.15 show plot of norm of dtheta versus motion time for full Jacobian, 3x7 Jacobian with orientation neglected and 1x7 Jacobian. In the case of the full Jacobian it is seen that there is no obstacle avoidance and although a lot of self motion is used for optimization the self motion is not sufficient for collision avoidance and the manipulator stops it motion after reaching the obstacle. But in the other two cases, the self motion is sufficient for obstacle avoidance and once the manipulator avoids the obstacle the amount of self motion added to the joint motion is very small as the gradient is very small. The performance criterion is shown in figures 3.16, 3.17 and 3.18 for each of the above cases. The performance criterion for the full Jacobian is shown in figure 3.16. Since collision is unavoidable the performance criterion increases at a very high rate till the manipulator stops motion. On the other hand with a 3x7 Jacobian, optimization helps the manipulator avoid obstacles and the performance criterion although increasing is lower than in the full Jacobian case. In case of 1x7 Jacobian the manipulator instantly moves away from the obstacle resulting in a decrease in performance criterion.

Case 2 : Calculations using pseudoinverse and gain factor k=0.01

The manipulator starting configuration was the same as before, its home position. The scalar gain factor used for optimization was 0.01. Figures 3.19, 3.20, and 3.21 show the motion configurations for the three different Jacobian cases. For the full Jacobian the result is given in figure 3.19 and figure 3.20 and 3.21 give results for 3x7 Jacobian and 1x7 Jacobian.

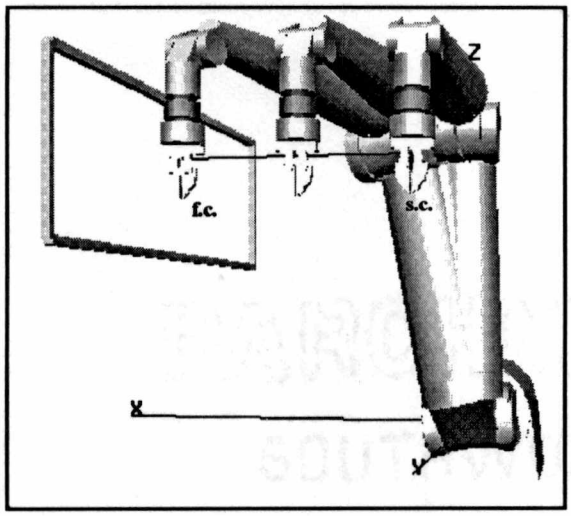


Figure 3.19 Manipulator configurations for 6x7 Jacobian and k=0.01

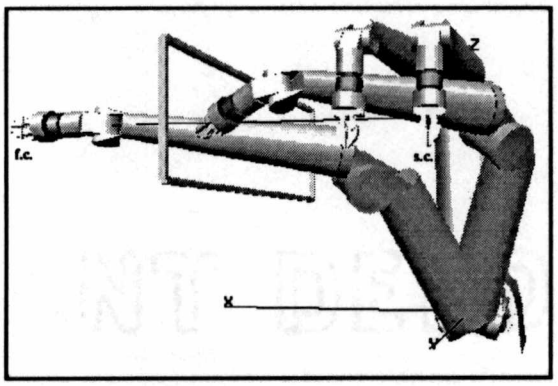


Figure 3.20 Manipulator configurations for 3x7 Jacobian and gain factor 0.01

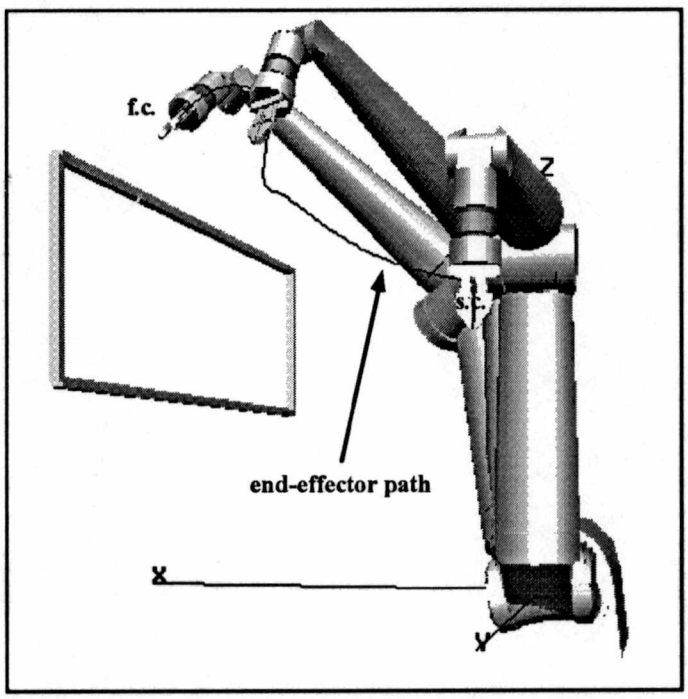


Figure 3.21 Manipulator configurations for 1x7 Jacobian and gain factor 0.01

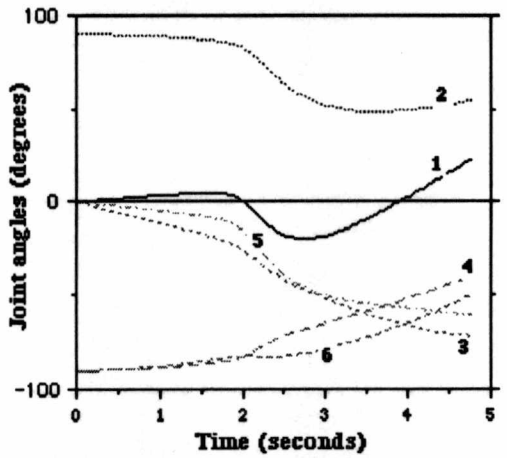


Figure 3.22 Joint angle vs. time 3x7 Jacobian

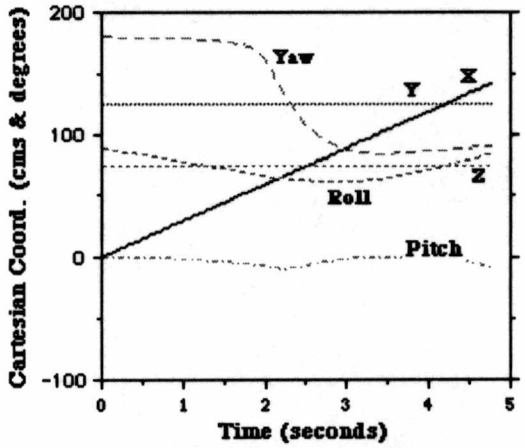


Figure 3.23 Cartesian motion vs. time 3x7 Jacobian

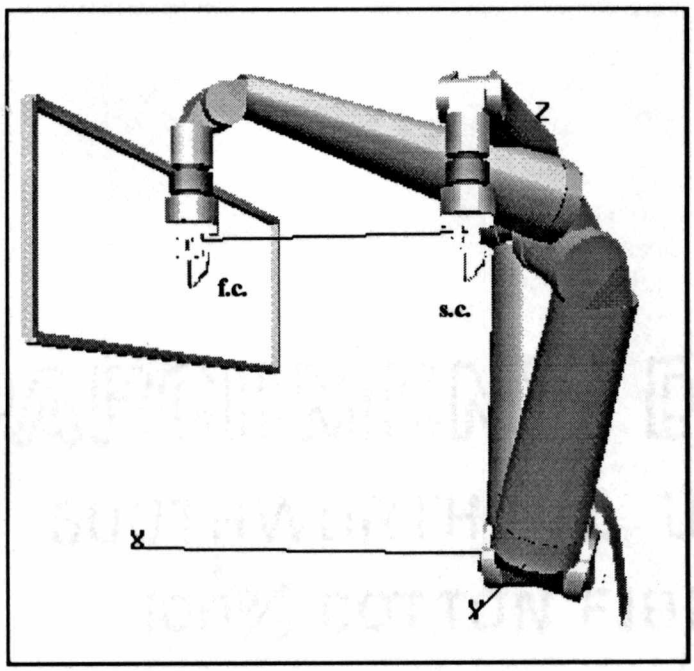


Figure 3.24 Manipulator configurations for 6x7 Jacobian and gain k=0.05

From the simulations for the 6x7 Jacobian it is shown that the manipulator cannot avoid obstacles with the single natural degree of redundancy and additional redundancy helps the manipulator to avoid obstacles and maintain uninterrupted motion in the command direction. The Joint angle motion and Cartesian coordinate motion for the 3x7 Jacobian case is shown in figures 3.22 and 3.23 respectively.

In all the above cases the calculations of joint motion was calculated using the pseudoinverse. The scheme was also tested using the real-time optimization scheme proposed by Dubey et.al. The conditions for the experiment remained the same and the collision objects used were the same. The results below are for the full Jacobian case and the 3x7 Jacobian.

Figure 3.24 shows the configuration of the manipulator when single degree of freedom is used for optimization. The manipulator cannot avoid collision and stops as soon as it reaches the collision object. In figure 3.25 the manipulator motion for 3x7 Jacobian is shown. The effectiveness of the Variable Dimension Task Space collision avoidance scheme is obvious from the above results and shows the increased dexterity of the manipulator around obstacles. Plots for the Cartesian motion of both cases are given in figures 3.26 and 3.27.

The following results are for the other two collision objects the rectangular cube and the hexagonal tube.

II) Collision Object : Rectangular cube

The manipulator motion for collision avoidance was tested with three different Jacobian cases the full Jacobian, 3x7 Jacobian and 1x7 Jacobian. The scalar gain factor k was set at 0.25 for all the three cases and calculations were done using the pseudoinverse. Figure 3.28, 3.29, and 3.30 show the simulation results for the full Jacobian, 3x7 Jacobian and 1x7 Jacobian respectively.

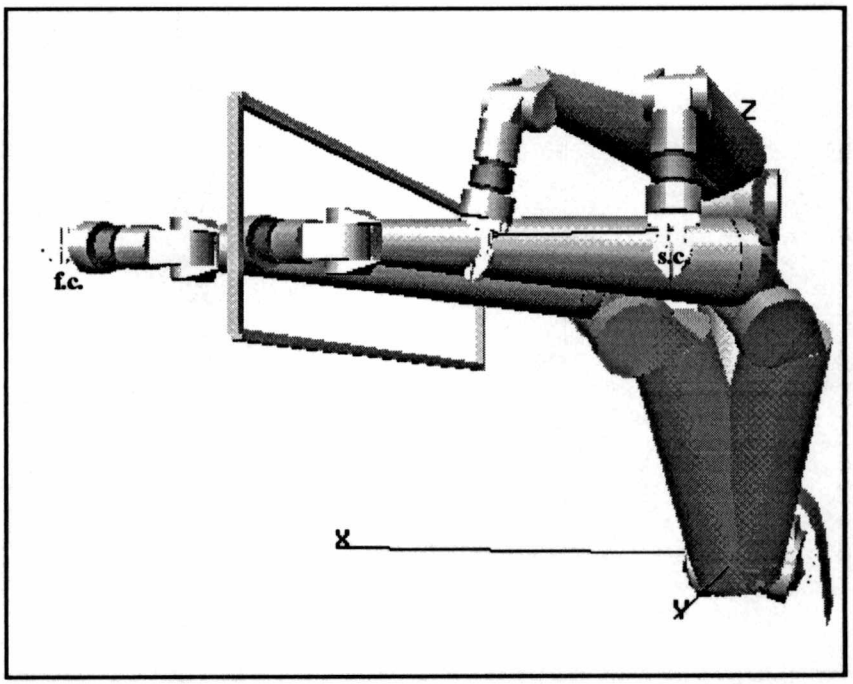


Figure 3.25 Manipulator configurations for 3x7 Jacobian and k=0.05

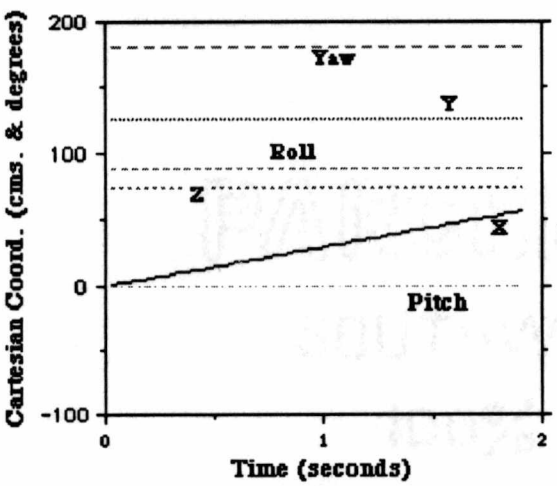


Figure 3.26 Cartesian motion (6x7 Jacobian)

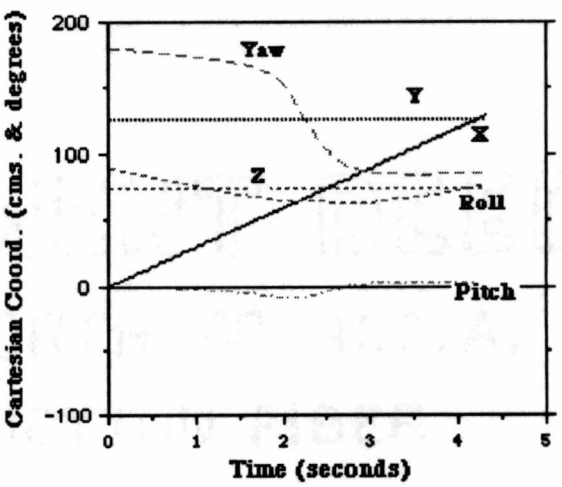


Figure 3.27 Cartesian motion (3x7 Jacobian)

It can be seen from figure 3.28, that the manipulator is not allowed to change the orientation of the end-effector and thus it cannot avoid the obstacle and continue its motion. But if the orientation of the end-effector is neglected, as in the second case, then the manipulator can maneuver through the collision object and complete its motion. In figure 3.30, where all motion directions except command direction are neglected, the manipulator avoids the obstacle completely and takes a path far away from the obstacle links. The Cartesian motion for the full Jacobian and 3x7 Jacobian is plotted against motion time and shown in figures 3.31 and 3.32. In case of the full Jacobian the manipulator has to maintain all coordinates except the command direction coordinate constant. This is clear from the graph in figure 3.31. In the 3x7 Jacobian case the orientations are neglected and can be seen changing from the starting configuration to the final configuration. The norm of joint angle motion for the above case is shown in figures 3.33 and 3.34. In figure 3.34 the band at the end is due to oscillation of the manipulator because of its nearness to the collision object and high gain factor. Results for the 1x7 Jacobian case are not shown because of the repetitiveness of the results. The behavior of the manipulator is the same for each collision object for each different Jacobian case.

III) Collision Object : Hexagonal Tube

As mentioned earlier the results for each collision object in every different Jacobian case is similar in behavior and so only few important results are included in this chapter. This section gives the configurations, Cartesian coordinate motion for the manipulator with hexagonal tube as the collision object. Figures 3.35 and 3.36 show the manipulator configurations for the full Jacobian case and 3x7 Jacobian case respectively. The plots for Cartesian motion in both these cases is given in figures 3.37 and 3.38. The explanation of the results is similar to the previous collision objects.

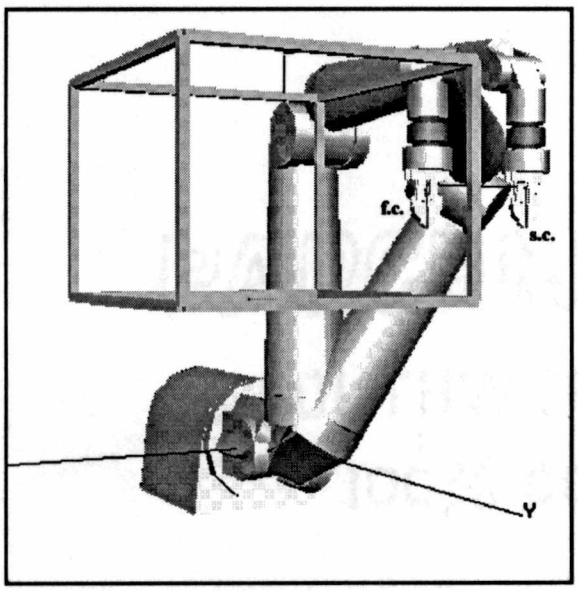


Figure 3.28 Manipulator Configurations for full Jacobian

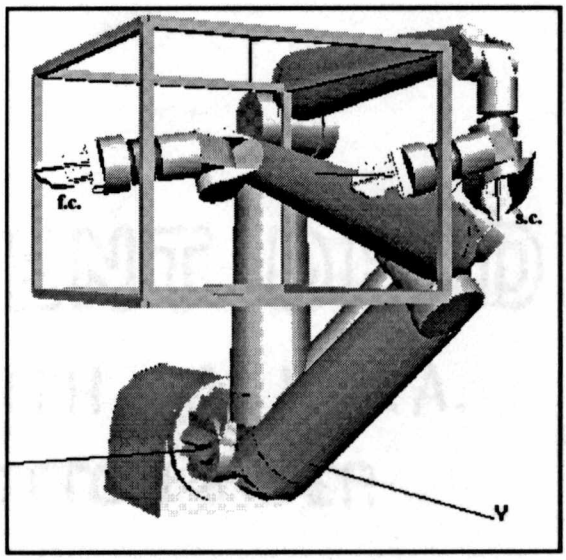


Figure 3.29 Manipulator Configurations for 3x7 Jacobian

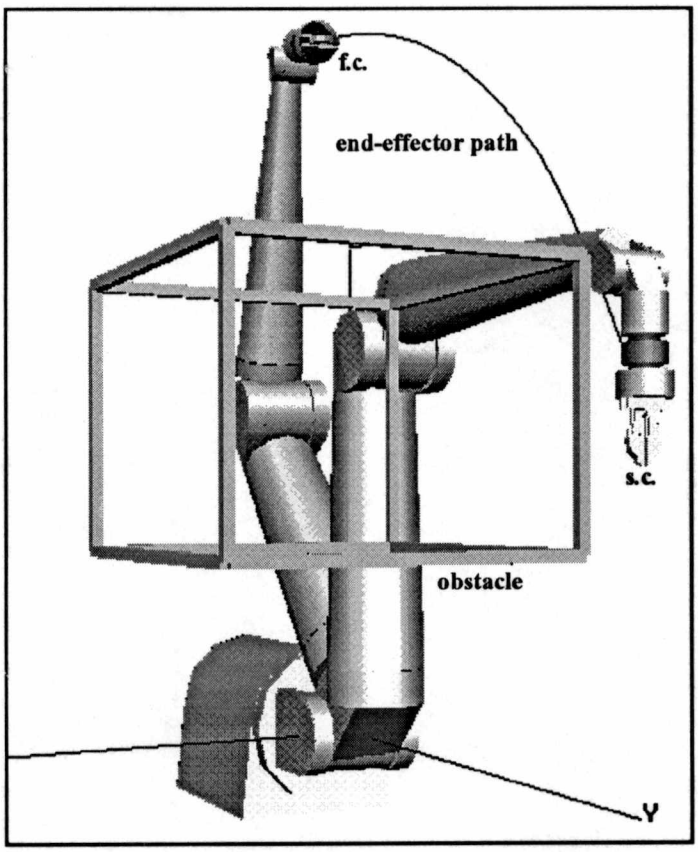


Figure 3.30 Manipulator Configurations for 1x7 Jacobian

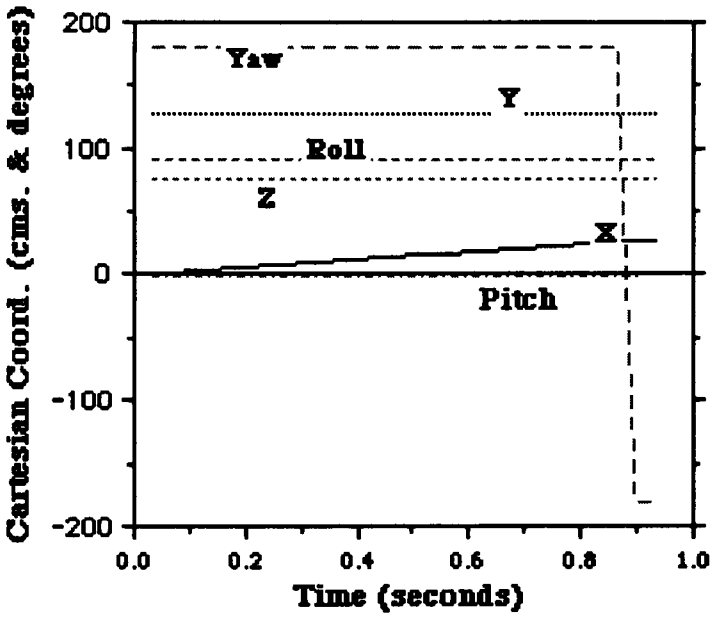


Figure 3.31 Cartesian Coordinate motion vs. time for 6x7 Jacobian

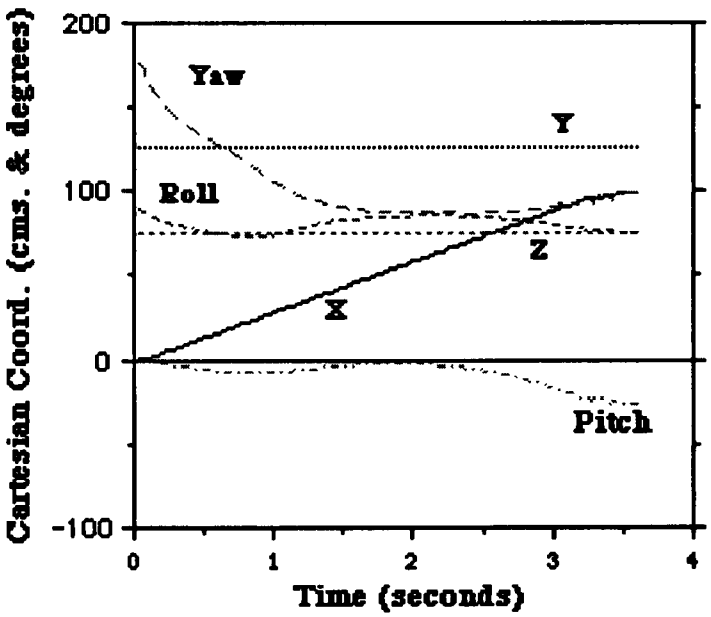


Figure 3.32 Cartesian Coordinate motion vs. time for 3x7 Jacobian

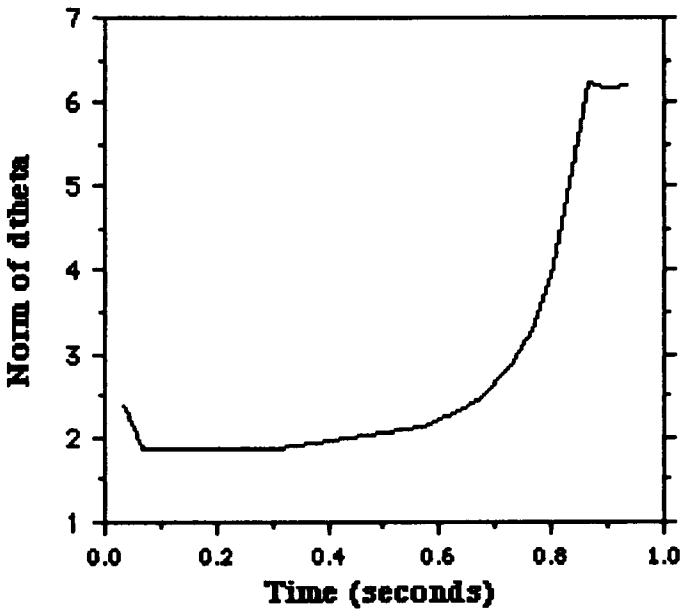


Figure 3.33 Norm of dtheta vs. time for 6x7 Jacobian

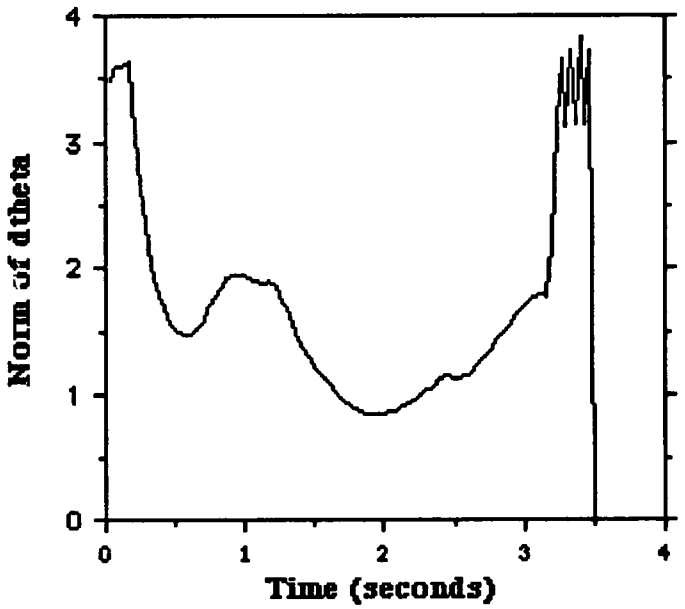


Figure 3.34 Norm of dtheta vs. time for 3x7 Jacobian

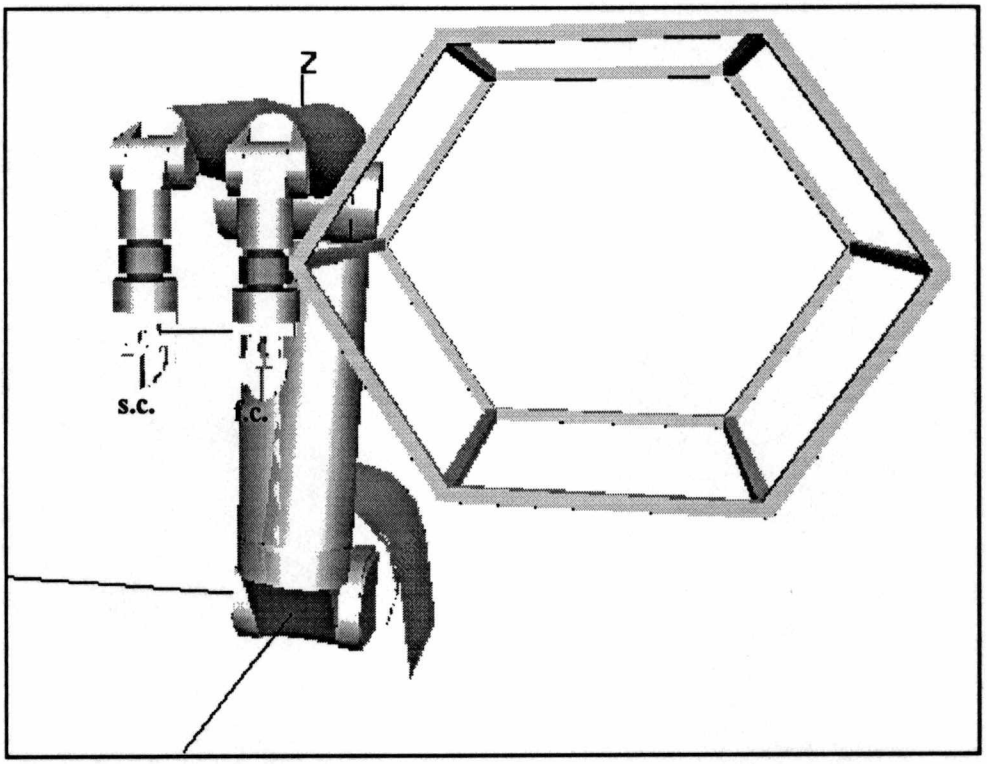


Figure 3.35 Manipulator Configurations for full Jacobian and k=0.15

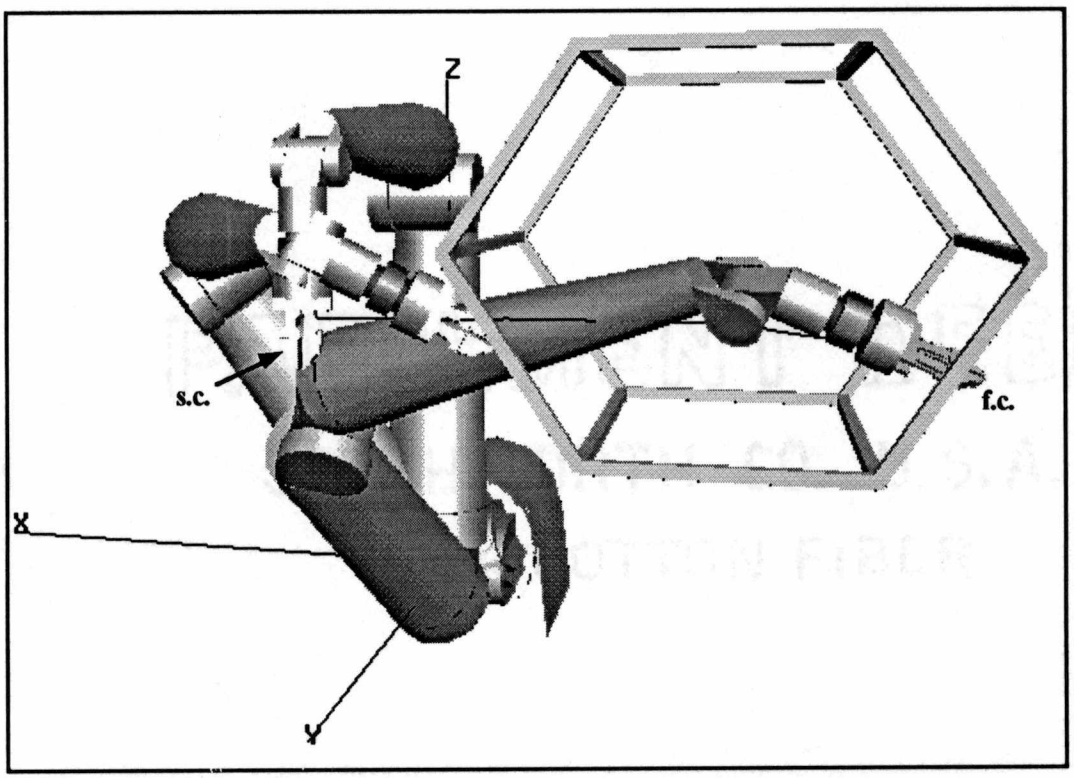


Figure 3.36 Manipulator Configurations for 3x7 Jacobian and k=0.15

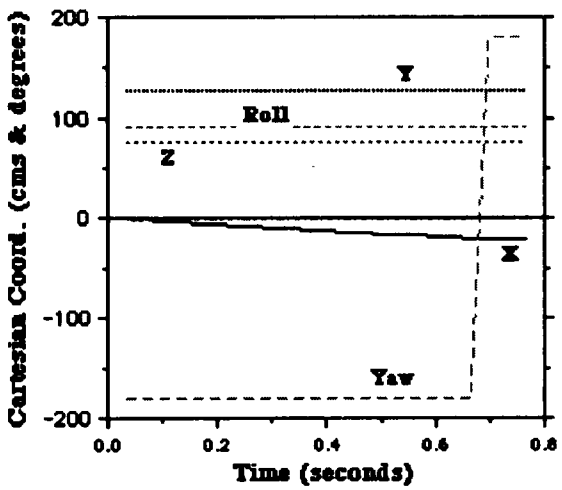


Figure 3.37 Cartesian motion (6x7 Jacobian)

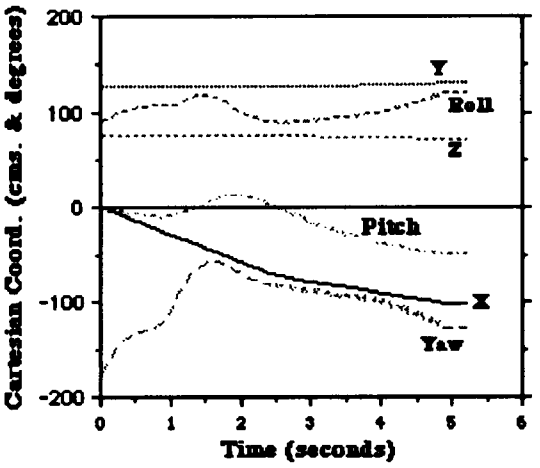


Figure 3.38 Cartesian motion (3x7 Jacobian)

In all the above cases only a single type of Jacobian or a single type of task space was chosen and the manipulator was moved in that particular task space. But in most cases it is necessary to use a combination of different task space dimensions during the motion of the manipulator. The user may need to start with a full Jacobian with complete control over all coordinate motion directions but would like to switch over to a partial task space where the orientation of the end-effector is neglected. The following results are for cases when the motion of the manipulator is started with some dimension task space but switched to another dimension task space sometime during the motion of the manipulator. The collision object used in this case was the rectangular frame and gain factor for optimization was 0.1.

a) Motion started with full Jacobian and switched to 3x7 Jacobian

Figure 3.39 shows the motion of the manipulator for this case. It is seen that as long as the full task space is controlled the manipulator end-effector maintains its orientation as well as position but when the motion is switched to a partially controlled

task space the orientation of the end-effector is allowed to change to accommodate collision avoidance.

I.C. is the configuration where the dimension of the task space is changed form six to three and orientation is neglected thereafter. But throughout the motion the manipulator maintains the end-effector position. The manipulator successfully avoids collision with the rectangular frame and moves through the rectangular frame to its final configuration.

The plot for Cartesian coordinate motion is shown in figure 3.40. It is clearly seen that for the initial part all coordinates are held constant except the command motion coordinate but once a switch is made to the partial Jacobian the Roll, Pitch and Yaw coordinates start changing and only Y and Z coordinates are held constant thereafter. Figure 3.41 shows the joint angle plot for the same case. It is seen that at the point where the Jacobians are switched there is no sudden change in joint angles and the transition is smooth. The performance criterion is shown in figure 3.42.

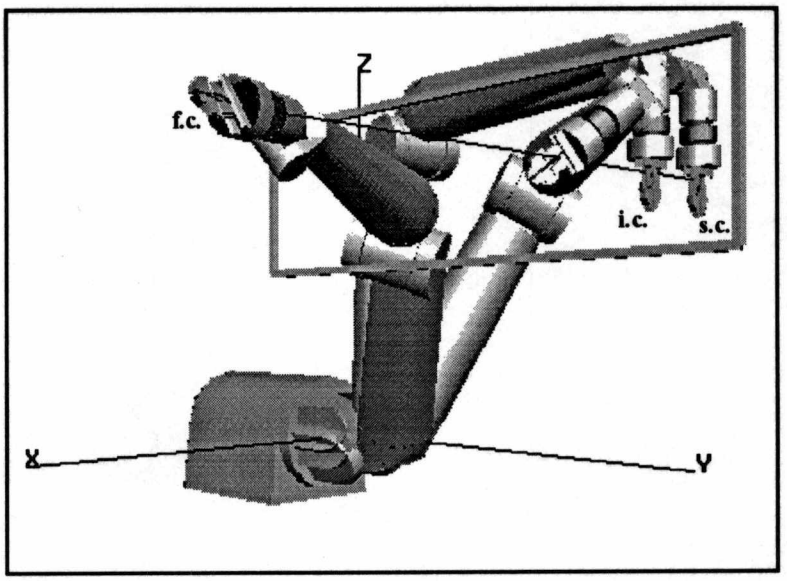


Figure 3.39 Manipulator Configurations for transition from full Jacobian to 3x7 Jacobian

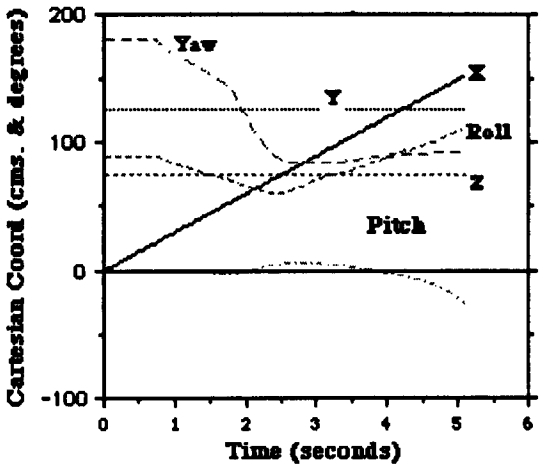


Figure 3.40 Cartesian Motion vs. time

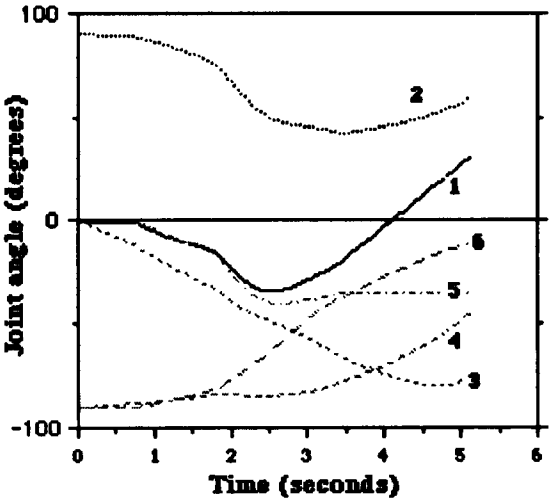


Figure 3.41 Joint motion vs. time

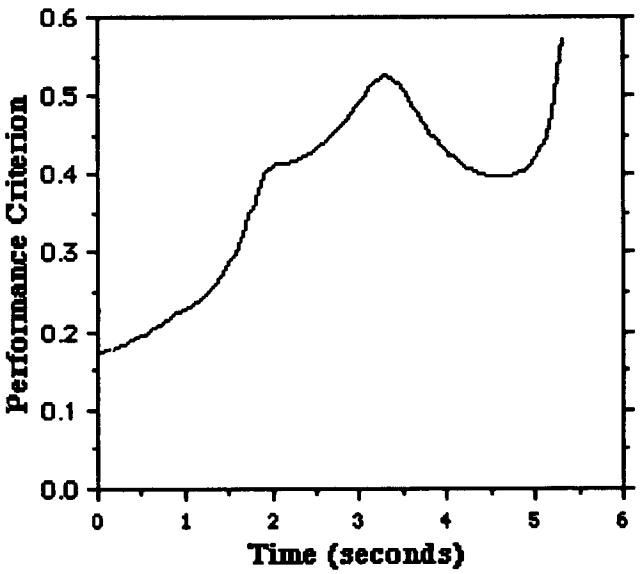


Figure 3.42 Performance Criterion vs. motion time

b) Motion started with 3x7 Jacobian and switched to full Jacobian

The manipulator configurations in this case are shown in figure 3.43 and plots for Cartesian coordinate motion, Joint angles and performance criterion are shown in figures 3.44, 3.45, and 3.46 respectively.

It is seen in figure 3.43 that the manipulator starts changing the orientation of the end-effector as soon as it starts its motion. This is because the orientation is not under control initially. Once past the obstacle the control is switched over to a full Jacobian at the intermediate configuration. The plot in figure 3.44 shows the roll, pitch and yaw coordinates changing initially but held constant during the latter half of the motion. The transition from a partial Jacobian to a full Jacobian is smooth as in the previous case and can be verified from figure 3.45.

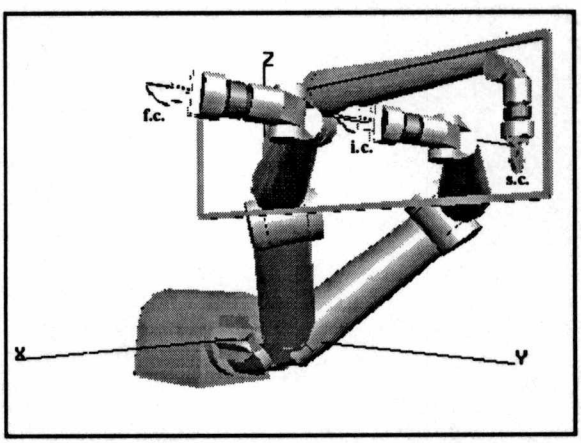


Figure 3.43 Manipulator Configurations for 3x7 Jacobian to full Jacobian

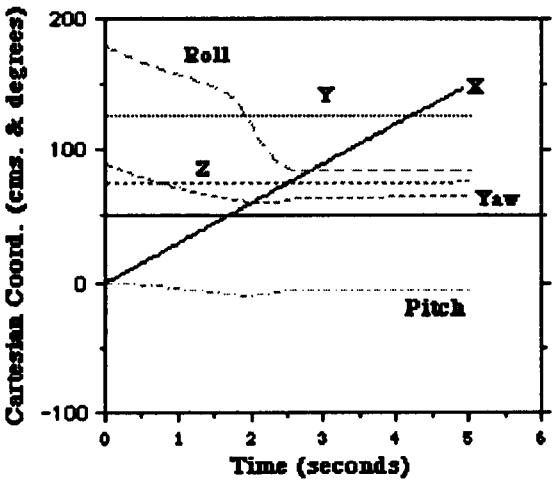


Figure 3.44 Cartesian Motion vs. time

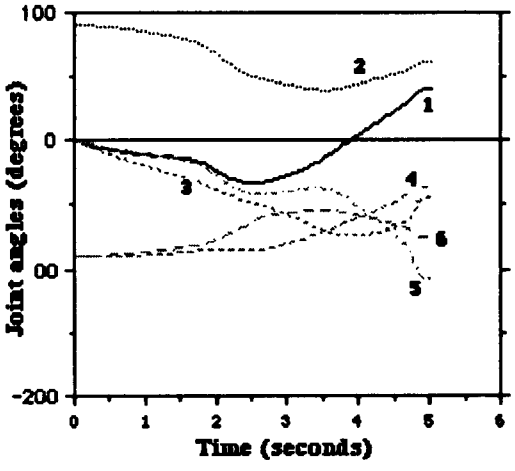


Figure 3.45 Joint angles vs. time

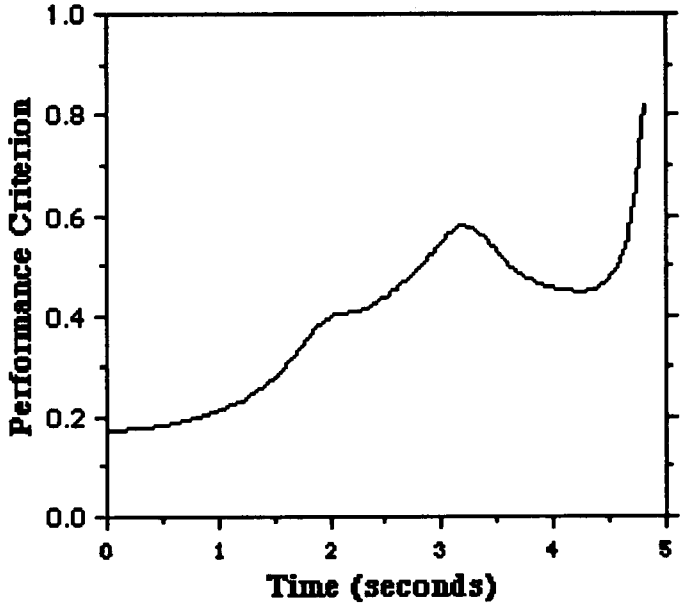


Figure 3.46 Performance Criterion vs. motion time

3.4 Conclusions

This chapter successfully demonstrated the effectiveness of the Variable Dimension Task Space scheme in dealing with the problem of collision avoidance. Results of experiments using this scheme are given in detail in this Chapter.

The Variable Dimension Task Space scheme is an optimization scheme which can be used for redundant as well as nonredundant manipulators. For redundant manipulators it can be used to add to the existing redundancy of the manipulator whereas for nonredundant manipulators it helps in creating redundancy in the system. But to gain the advantage of accquiring redundancy some sacrifices have to be made. The user loses control over the motion directions neglected and used for redundancy. This may not be desirable in cases where complete control of position and orientation of the end-effector is desired.

On the other hand, if there is a task where certain motion coordinates can be neglected, this scheme is one of the best ways to add redundancy and improve the performance of the manipulator. The addition of redundancy and choice of directions to be controlled or neglected is totally up to the user and the task being performed. The choice of the scalar gain factor k for optimization is an issue that needs to be investigated. As shown, too large a gain factor leads to oscillations in the manipulator and too small a gain factor results in poor optimization of motion.

Another point to be kept in mind is that the increased redundancy may not always help in collision avoidance. The effectiveness of the manipulator to avoid obstacles also depends in the type of redundancy added to the manipulator. For example, if a manipulator has multiple joints oprating in the same plane, then if a single degree of

redundancy does not help in collision avoidance, an increase in the same type of redundancy will not help in collision avoidance.

The increase in redundancy is a decision to be made by the user, the task being performed and the obstacles present in the manipulator workspace.

Chapter 4

Collision Avoidance Using the Singularity Robust Inverse

4.1 Introduction to the Singularity Robust Inverse (SR-Inverse)[25,26]

The relationship between the joint velocities and the end-effector velocities of a manipulator is given by equation 2.1. In case of a nonredundant manipulator the joint velocities are usually computed using the inverse of the Jacobian matrix and in case of redundant manipulators using the pseudoinverse of the Jacobian matrix. Singular points for manipulators exist not only at workspace boundaries but also within the workspace of the manipulator. A singular location is a point where the end-effector of the manipulator cannot move in certain directions i.e. it is not possible to find a set of joint motion that would produce the required end-effector motion. The Jacobian matrix of the manipulator at singular points is not full rank and the inverse for this matrix is nonexistent.

In case of redundant manipulators and singular locations the pseudoinverse of the Jacobian matrix is used to compute the joint velocities. The pseudoinverse[14] of the Jacobian matrix finds the least-squares solution to equation 2.1. The solution provides the minimum norm solution for the joint velocities from all the solutions that satisfy equation 2.1. There are two issues that need to be dealt with for finding an appropriate joint velocity solution for required end-effector motion. Those are

a) Accuracy

b) Feasibility

The solution using the pseudoinverse only considers one of the above criteria namely the accuracy of the solution. In the neighborhood of singular points and at singular points the pseudoinverse gives a set of joint motion that will produce the required end-effector motion. This results in very high joint velocities which cannot be realized by the manipulator hardware and results in very high error and further degeneration of motion.

To overcome the limitation of the pseudoinverse at or around singular locations Nakamura and Hanafusa[26] defined an alternate inverse called the Singularity Robust Inverse (SR-Inverse) which provides feasible solutions at or around singular locations.

4.2 Mathematical Equations for the SR-Inverse

As mentioned in the previous section the SR-Inverse considers both accuracy and feasibility factors for finding a solution for a manipulator at or around singular positions. The mathematical formulation for the SR-inverse[25,26] is given in this section. The evaluation criterion for the SR-inverse is given by equation 4.1

$$\min \|\delta e\|_W \quad \text{.....(4.1)}$$

where δe is given by

$$\delta e \equiv \begin{pmatrix} \delta r - J(\theta)\delta\theta \\ \delta\theta \end{pmatrix} \quad \text{.....(4.2)}$$

$\| * \|_W$ is the weighted Euclidean norm for *. The weighting matrix W is given by equation 4.3 as follows

$$W = \begin{pmatrix} W_1 & 0 \\ 0 & W_2 \end{pmatrix} \quad \text{.....(4.3)}$$

where $W \in R^{(m+n)\,x\,(m+n)}$, $W_1 \in R^{m\,x\,m}$, and $W_2 \in R^{n\,x\,n}$. The matrices W_1 and W_2 are both symmetric positive definite matrices. Then equation 4.1 satisfies the following relationship

$$\|\delta e\|_W^2 = \|\delta r - J(\theta)\delta\theta\|_{W_1}^2 + \|\delta\theta\|_{W_2}^2 \quad \text{.....(4.4)}$$

Hence the solution to equation 2.1 for the evaluation criterion given by equation 4.1 is given by equation 4.5

$$\begin{aligned} \delta\theta &= J^*(\theta)\delta r \\ J^*(\theta) &= \left(J^T W_1 J + W_2\right)^{-1} J^T W_1 \end{aligned} \quad \text{.....(4.5)}$$

$J^*(\theta)$ is the definition for the SR-inverse of the Jacobian matrix and is given in equation 4.5.

W_1 and W_2 are represented by equation 4.6 as follows

$$W_1 = N_r^T N_r$$
$$W_2 = k_s N_\theta^T N_\theta \quad \text{.....(4.6)}$$

where N_r and N_θ are symmetric positive definite matrices, and scale factor $k_s > 0$. If N_r and N_θ are taken as identity matrices then the calculations become much simpler and the SR-inverse of the Jacobian matrix can be written as

$$J^*(\theta) = \left(J^T J + k_s I\right)^{-1} J^T \quad \text{.....(4.7)}$$

The SR-inverse is then also equivalent to the form given in equation 4.8

$$J^*(\theta) = J^T \left(JJ^T + k_s I\right)^{-1} \quad \text{.....(4.8)}$$

It is much more efficient to use equation 4.8 instead of equation 4.7 because equation of the cost involved in computing the bracketed terms in either equation.

4.3 Properties of the SR-Inverse

The various properties and advantages of the SR-inverse have been analyzed previously by Nakamura and Hanafusa[26] and are listed below in brief.

a) Gives a feasible solution in neighborhood of singular locations of a manipulator within its workspace.

b) Solution using SR-inverse does not disturb the redundant space of the Jacobian matrix of the manipulator.

c) Solution by SR-inverse belongs to the orthogonal complement of the redundant space as well as the solution by the pseudoinverse.

d) If the scale factor $\mathbf{k_s}$ for the SR-inverse is zero then the SR-inverse is equal to the pseudoinverse of the Jacobian matrix.

e) The SR-inverse considers both accuracy and feasibility while determining a solution for a manipulator. The scale factor $\mathbf{k_s}$ decides on which factor is given more importance. Smaller the value of $\mathbf{k_s}$ greater is the importance of accuracy of the solution.

The next section describes the application of the SR-inverse in collision avoidance for manipulators.

4.4 SR-inverse in Collision Avoidance

The SR-inverse was developed specifically for the problem of singularity in robot manipulators. In this thesis the concept is further extended to address the problem of collision avoidance in manipulators.

The SR-inverse sacrifices the accuracy of a solution in order for it to be a feasible solution. This achieved mainly by using an appropriate scale factor and weighting matrices for the SR-inverse. In collision avoidance the manipulator motion is restricted by obstacles within the workspace of the manipulator. If the accuracy of motion of the manipulator is sacrificed then the manipulator can achieve collision avoidance, and that concept is proved using the SR-inverse.

The scale factor k_s used in the SR-inverse should only come into effect when the manipulator is near obstacles and not far away from obstacles. To obtain this result the scale factor is changed dynamically in the code depending on the distance of the manipulator from the collision obstacles. The scale factor is kept zero when collision avoidance is not necessary for the manipulator and the pseudoinverse is used for calculating the joint motion of the manipulator. As soon as the manipulator gets close to a obstacle the scale factor kicks in and the SR-inverse is used for calculating the joint motion and helping in collision avoidance. The scale factor k_s is calculated as follows

$$k_s = \begin{cases} k_0\left(1 - \dfrac{d}{d_0}\right) \ldots\ldots\ldots\ldots\ldots\ldots d < d_0 \\ 0 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots d \geq d_0 \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots (4.9)$$

where d is the distance of the manipulator from the obstacle and d_0 is a threshold distance. If the actual distance of the manipulator falls below a certain predetermined value then the SR-inverse is used for calculations of joint velocities. As long as the manipulator is far away from obstacles the scale factor k_s is maintained zero and the SR-inverse is equal to the pseudoinverse. k_0 is the nominal value of the scale factor.

4.5 Experimental Setup

The effectiveness of SR-inverse in collision avoidance was tested on the simulation model of the RRC K-2107 seven degree of freedom redundant manipulator. Two collision objects namely the rectangular frame and the rectangular cube were used in the experiments. The manipulator end-effector velocity was maintained at 30 cm/sec and the starting configuration for the manipulator was chosen as the home position given in table 4.1.

Table 4.1 Manipulator Starting Configuration for SR-Inverse

Configuration	X (cms.)	Y (cms.)	Z (cms.)	Roll (degrees)	Pitch (degrees	Yaw (degrees)
Home	0.00	126.512	75.385	90.0	0.0	180.0

4.5.1 Experiments and Results

Since the SR-inverse sacrifices the accuracy of solution for collision avoidance, it is desirable to use the pseudoinverse for calculation of joint velocities whenever obstacle avoidance is not needed for the manipulator. If the scale factor $\mathbf{k_s}$ for the SR-inverse is zero then the SR-inverse is equal to the pseudoinverse.

The equality of the pseudoinverse and the SR-inverse is shown in figures 4.1 and 4.2 which show the manipulator motion using the pseudoinverse and SR-inverse with scale factor $\mathbf{k_s}$ zero. Plots for the Cartesian motion coordinates and the performance criterion are given in figures 4.3 - 4.5. Since all dimensions of the end-effector are controlled using this scheme, it is seen that collision avoidance is not possible with the available one degree of redundancy. The Gradient Projection Method is used for optimization and the gain factor is taken as 0.1 for all cases.

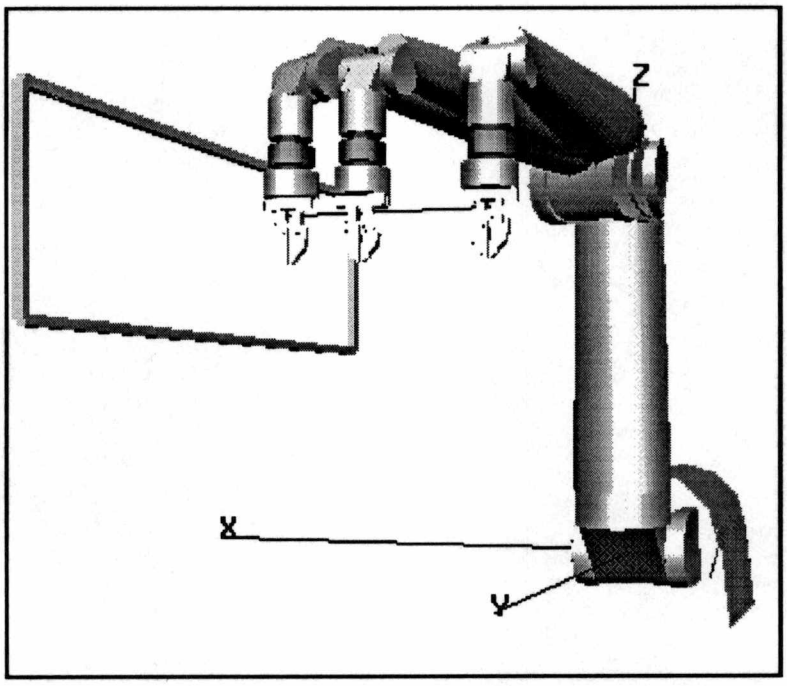


Figure 4.1 Manipulator Configurations Using the Pseudoinverse

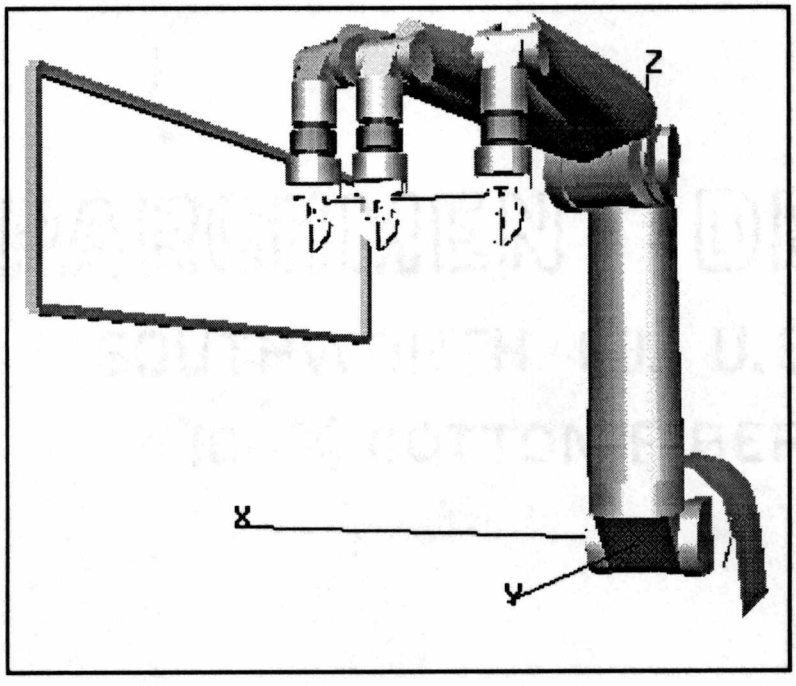


Figure 4.2 Manipulator Configurations Using SR-inverse and scale factor k_S=0

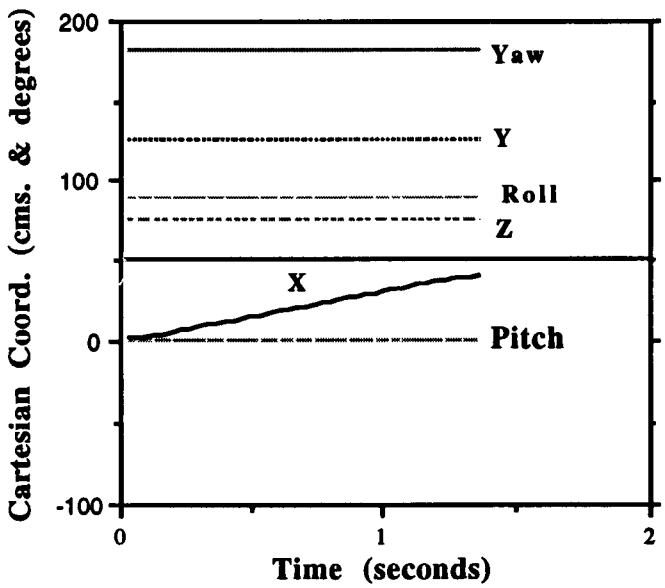


Figure 4.3 Cartesian Motion using the Pseudoinverse

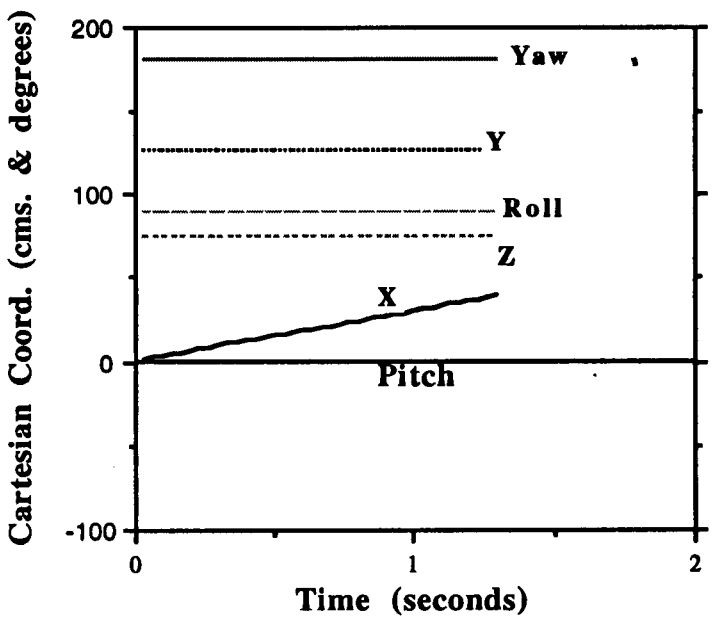


Figure 4.4 Cartesian Motion using SR-inverse and scale factor k_s=0

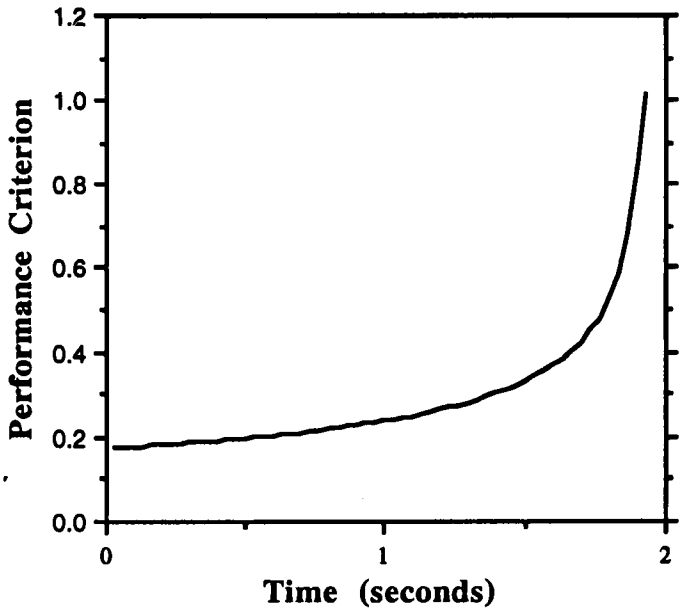


Figure 4.5 Performance Criterion for both schemes

Although the performance of the SR-inverse involves two factors namely the weighting or normalizing matrices W_1 and W_2 and the scale factor k_s, for the purpose of this study the weighting matrices are chosen to be identity matrices and the behavior of the SR-inverse with change in scale factor is focused on.

I) Collision Object Rectangular Frame

In this case the SR-inverse was tested for three different cases. In two cases the scale factor is kept at a constant value, during the entire motion period, while in the third case the scale factor changes dynamically during the motion of the manipulator.

II) Collision Object Rectangular Cube

As in the case of the rectangular frame, the SR-inverse was tested for the same three conditions. Additionally, one more case with the scale factor coming into effect later than in previous case is also tested.

The following configuration pictures and plots give the results for the tests using the SR-inverse for collision avoidance in case of a rectangular frame and rectangular cube. Table 4.2 gives the conditions for each test.

Table 4.2 Test Conditions for the SR-inverse

Case	Scale Factor k_s	Gain Factor for GPM
1	0.25	0.1
2	2.5	0.1
3 & 4	Variable	0.1

Case 1 & 2: Scale Factor k_s=0.25 and 2.5 constant

Figures 4.6 to 4.10 show the results for the motion of the manipulator with the rectangular frame as the collision object and figures 4.11 to 4.15 show those with the same collision object but a different scale factor.

The scale factor k_s is maintained constant at 0.25 for the first case and 2.5 for the second case. The SR-inverse is used for calculation of joint velocities from the starting position till the final position of the manipulator. Since the full Jacobian is used for motion, the Cartesian coordinates, other than the command direction, would all remain constant if the pseudoinverse is used. But the SR-inverse introduces error in the motion of the manipulator i.e. sacrifices accuracy for feasibility. Feasibility of solution in this case is the ability of the manipulator to avoid obstacles during motion.

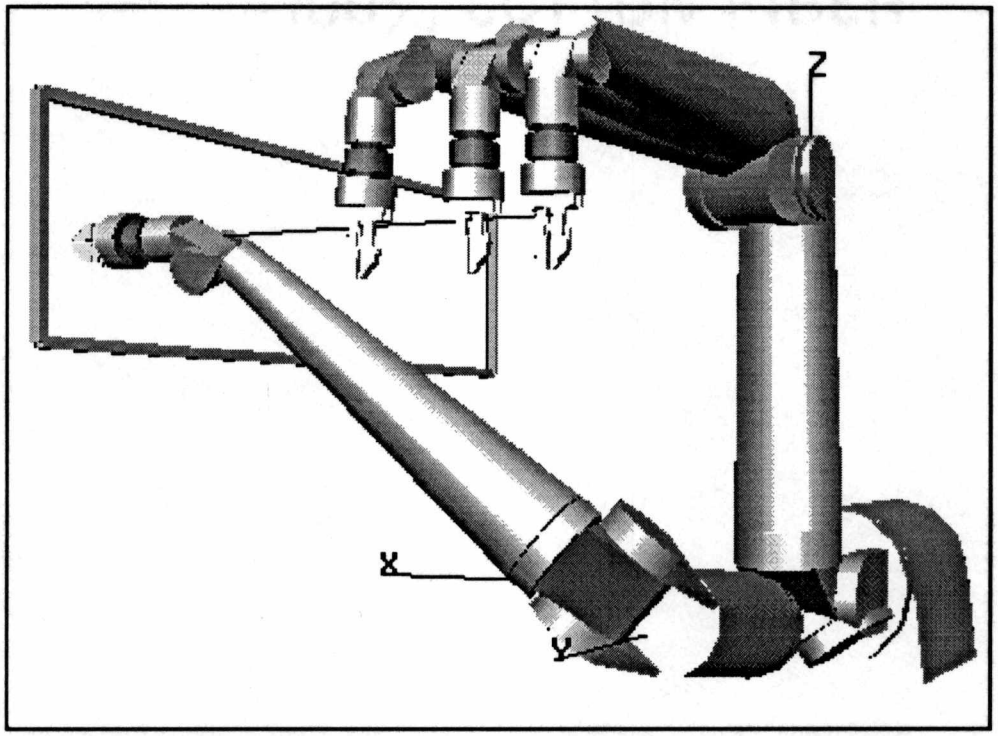


Figure 4.6 Manipulator Configurations using constant scale factor k_S=0.25

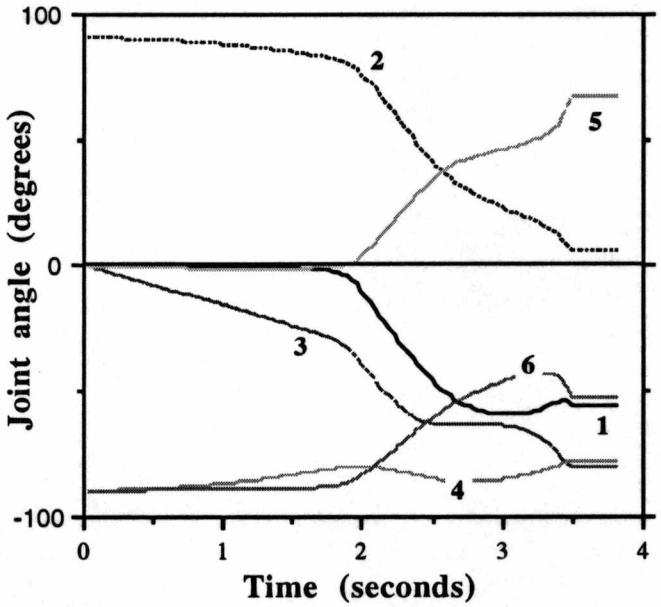


Figure 4.7 Joint angles for manipulator using constant scale factor k_S=0.25

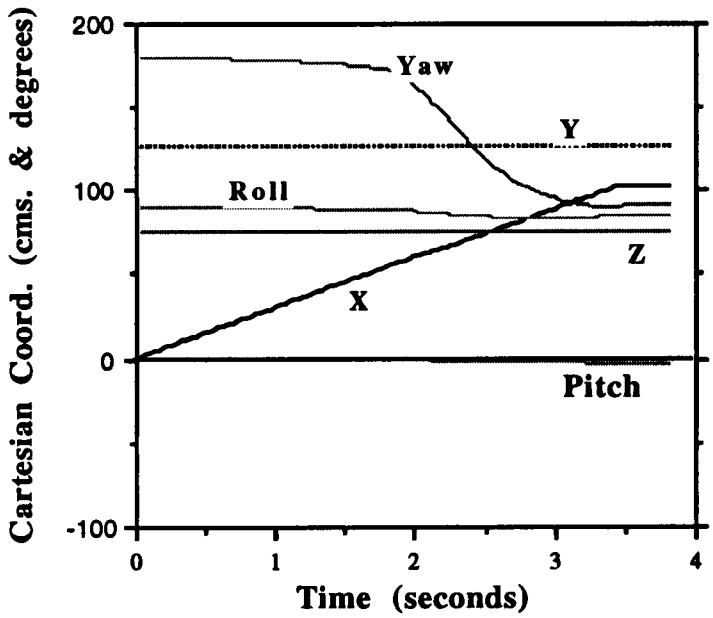


Figure 4.8 Cartesian motion using constant factor k_s=0.25

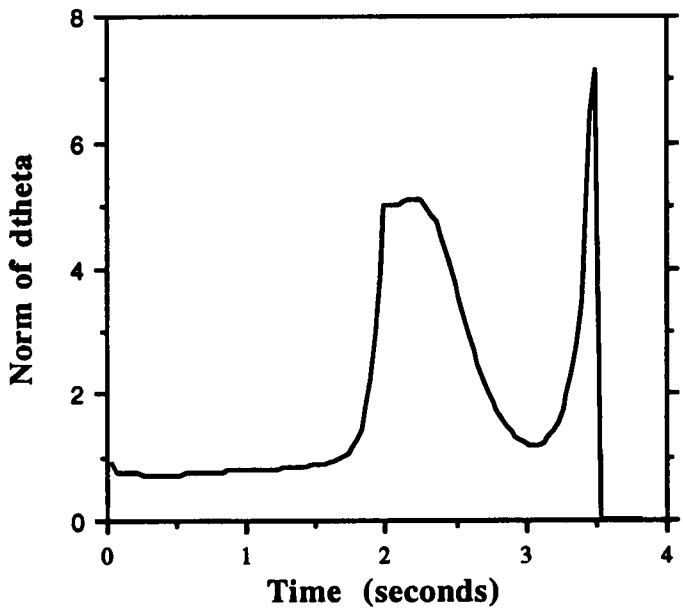


Figure 4.9 Norm of dtheta using constant scale factor k_s=0.25

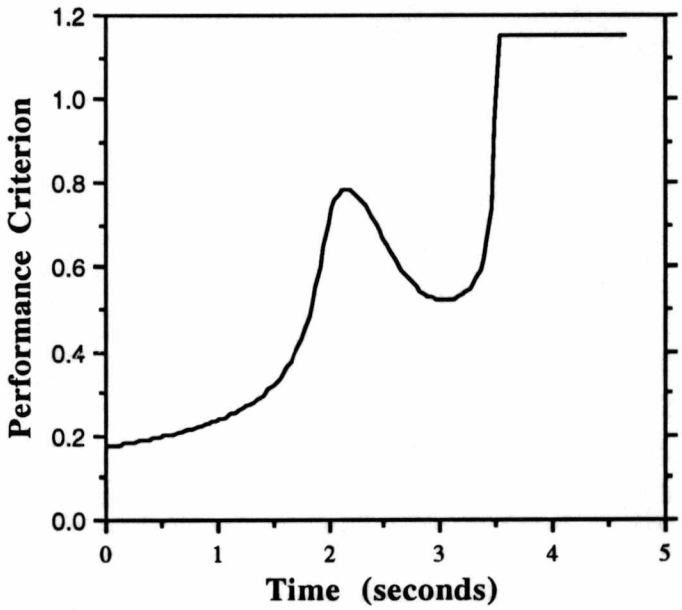


Figure 4.10 Performance Criterion using constant scale factor k_S=0.25

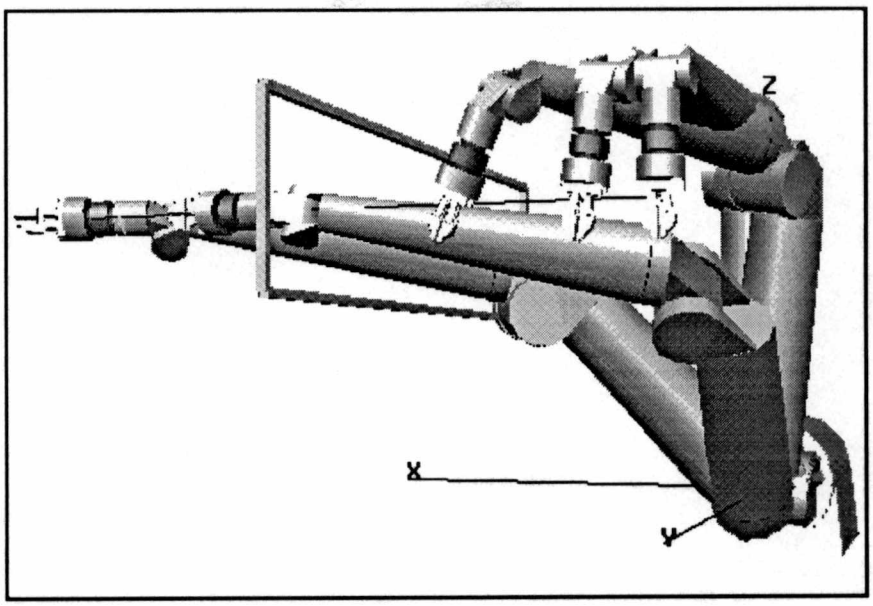


Figure 4.11 Manipulator Configurations using constant scale factor k_S=2.5

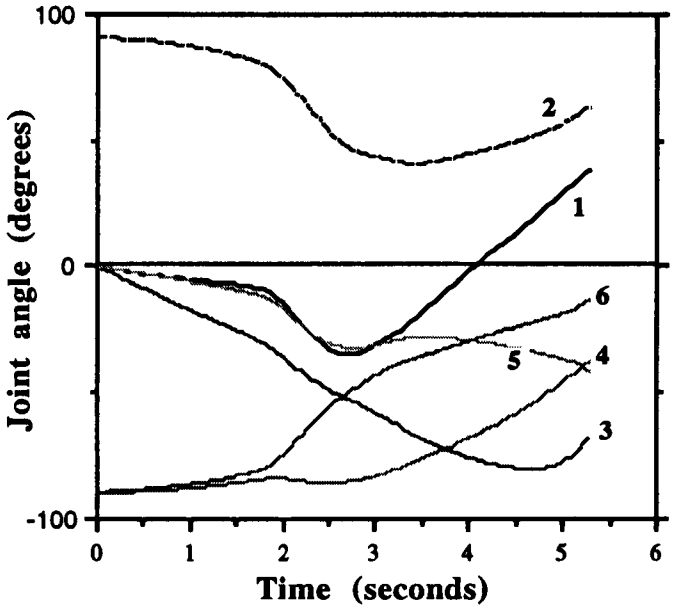


Figure 4.12 Joint angles for manipulator using constant scale factor k_s=2.5

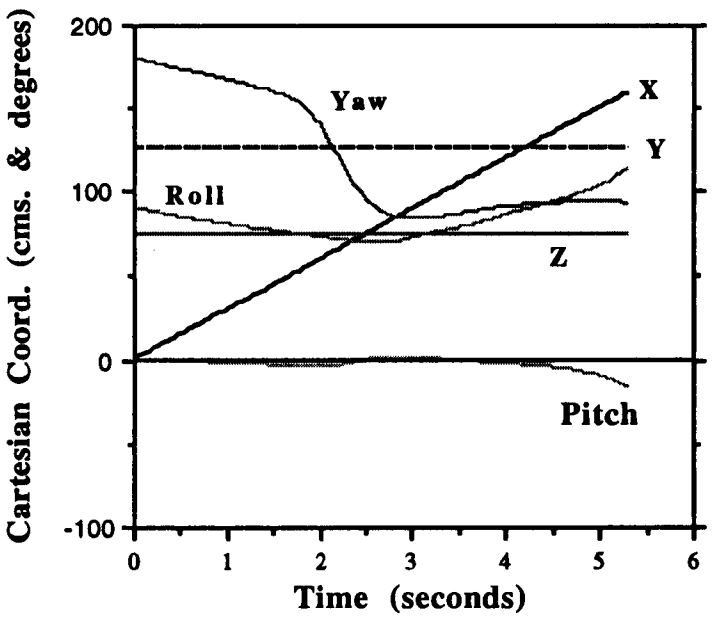


Figure 4.13 Cartesian motion using constant factor k_s=2.5

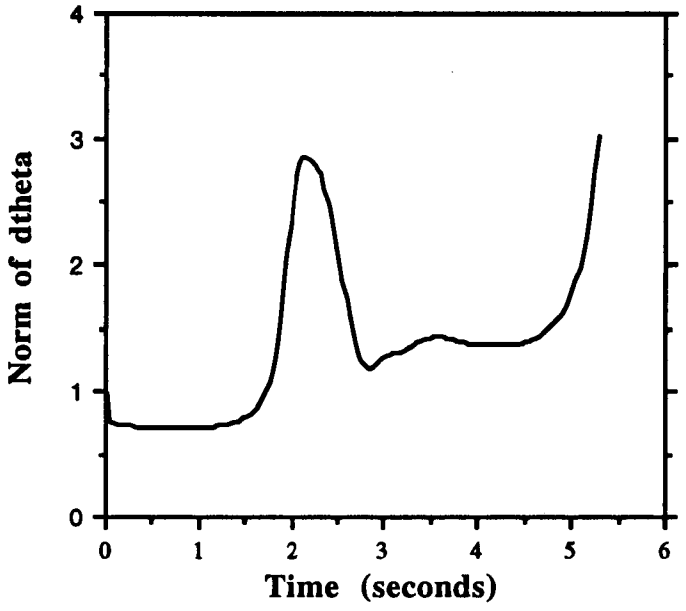


Figure 4.14 Norm of dtheta using constant scale factor k_s=2.5

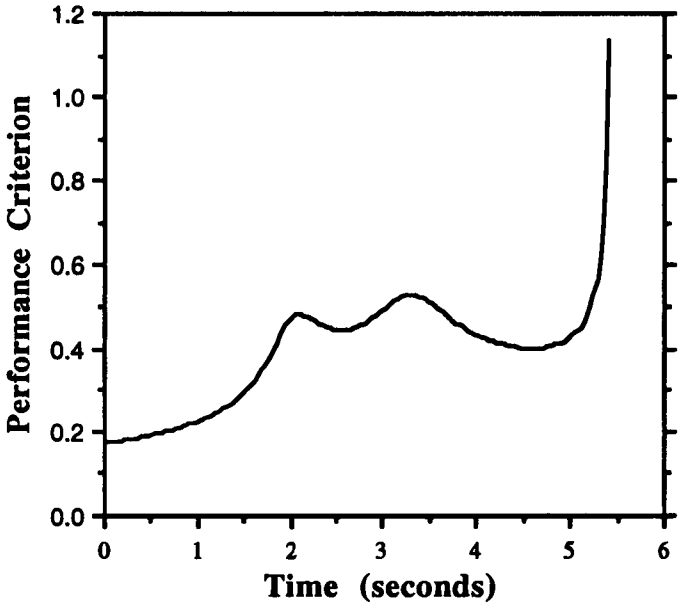


Figure 4.15 Performance Criterion using constant scale factor k_s=2.5

In figures 4.6 to 4.15 the scale factor $\mathbf{k_s}$ is kept constant. This ensures that collision avoidance is always active for the manipulator and SR-inverse is used for joint velocity computations throughout the motion of the manipulator.

Figure 4.6 shows that the manipulator is successful in avoiding the obstacle and manages to go through the window presented by the rectangular frame. This is made possible mainly due to a change in orientation of the manipulator using elbow motion. This is also evident in the joint motion plot shown in figure 4.7. Near the obstacle there is a large motion of joints through 6. Figure 4.8 shows the Cartesian coordinate motion for this case. It is seen that the major change that helps the manipulator in collision avoidance is that in the orientation of the end-effector. The error in the position coordinates is negligible compared to that for the orientation. The plot for norm of dtheta shows a large dtheta near the obstacle indicating the change in orientation of the end-effector. Figure 4.10 shows the change in performance criterion throughout the motion period. Initially the manipulator is far away from the obstacle and the performance criterion is very low. But as the manipulator approaches the obstacle the performance criterion increases and is seen decreasing as collision avoidance is successful.

With a higher scale factor for the SR-inverse similar results are obtained. Collision avoidance is better in this case as there is allowance for higher error in the solution. The change in orientation of the end-effector is higher in this case shown in figure 4.13. The performance criterion shown in figure 4.15 also indicates better collision avoidance in the case of a higher scale factor for collision avoidance.

Case 3: Scale Factor k is variable

Figures 4.16 to 4.21 show the manipulator configurations and motion plots for this case.

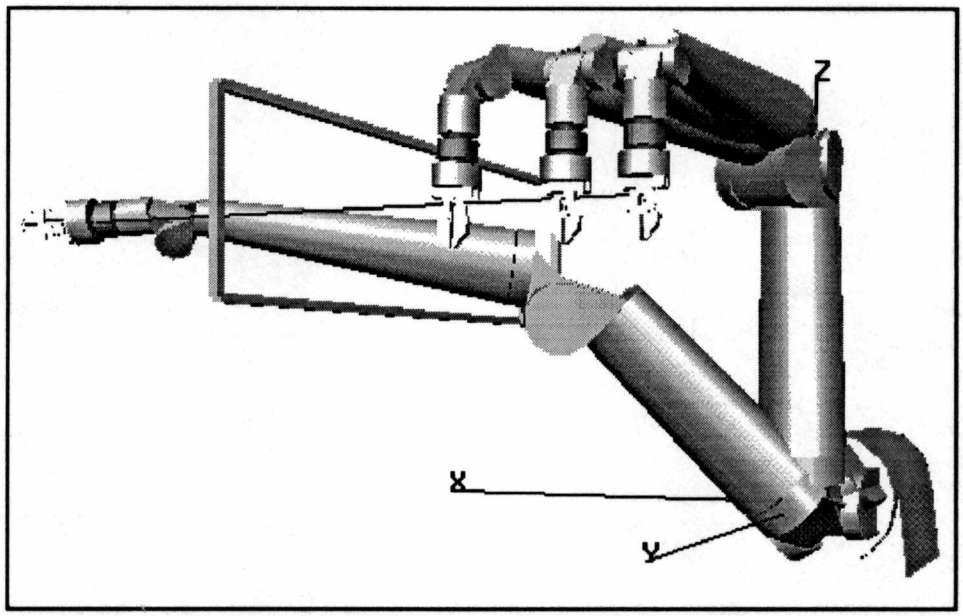


Figure 4.16 Manipulator configurations for variable scale factor k_s

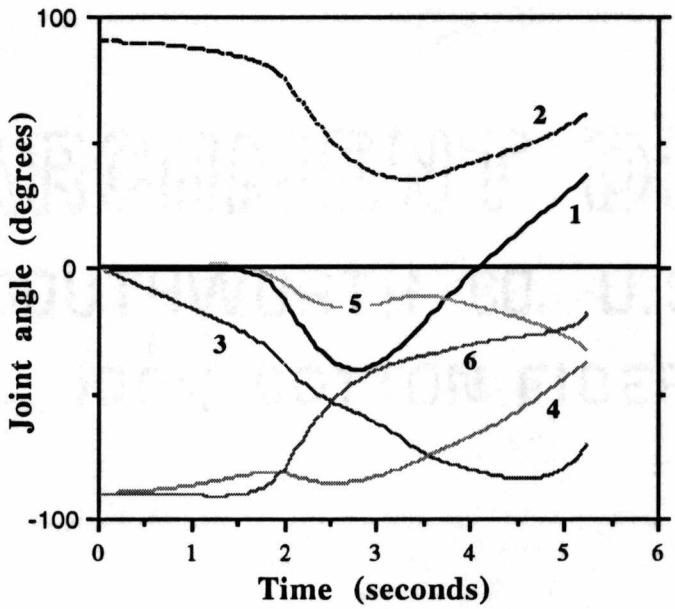


Figure 4.17 Joint angles for manipulator with variable scale factor k_s

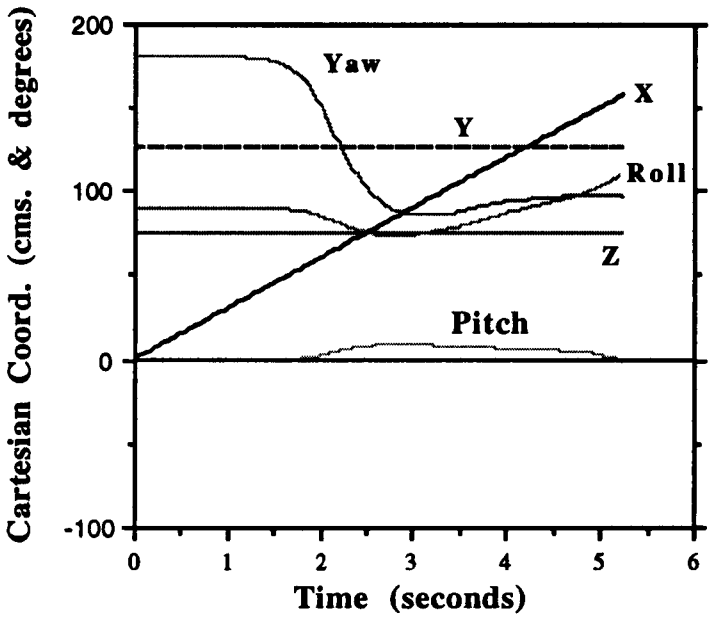


Figure 4.18 Cartesian motion using variable scale factor k_s

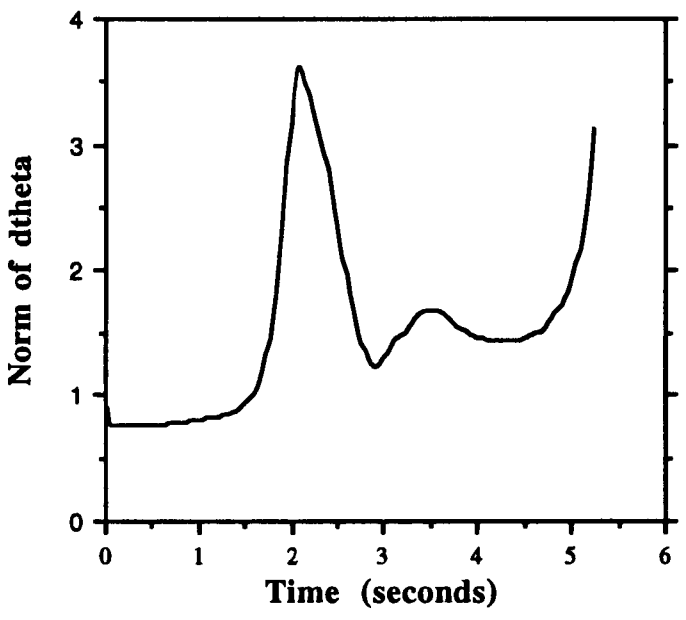


Figure 4.19 Norm of dtheta using variable scale factor k_s

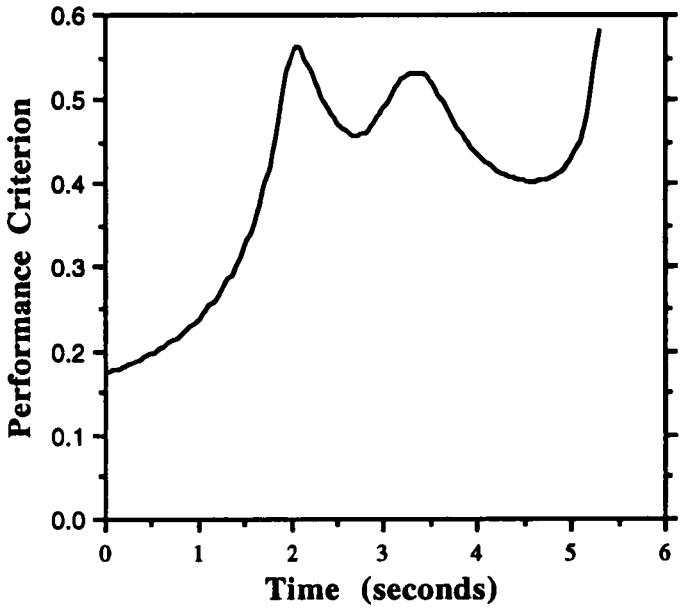


Figure 4.20 Performance Criterion using variable scale factor k_s

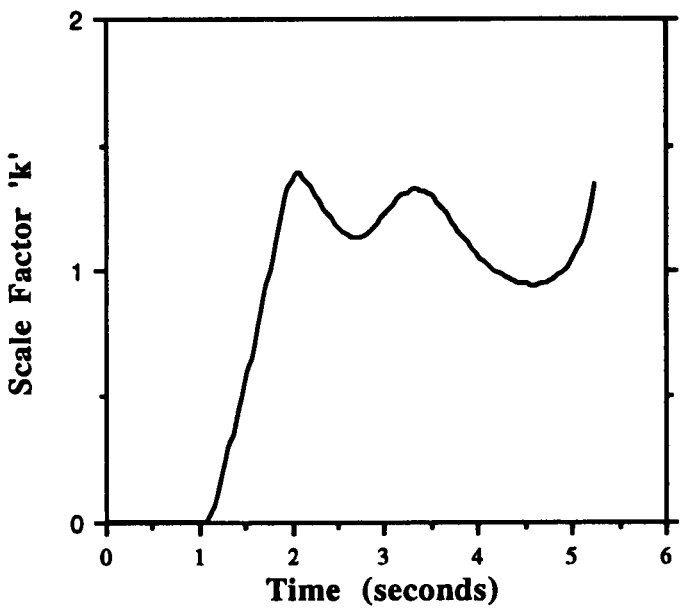


Figure 4.21 Scale factor k_s versus motion time

It is important to realize that the solution obtained using the SR-inverse is not an exact solution. If the scale factor $\mathbf{k_s}$ in the SR-inverse is zero then the SR-inverse is equal to the pseudoinverse as shown earlier in this chapter. To make use of this property of the SR-inverse a variable scale factor is used.

The scale factor remains zero until collision avoidance is not needed and comes into play only when the manipulator is close to the obstacle and collision avoidance scheme is to be employed. This ensures that the solution using the SR-inverse is exact for the motion period where collision avoidance is unnecessary. Figure 4.21 shows us the change in scale factor with an increase in motion time of the manipulator. The scale factor is zero for about the initial 1.1 seconds of the motion and then changes according to the distance of the manipulator from the obstacle. The Cartesian motion plot in figure 4.18 show that all coordinates except command direction remain constant during this period. As soon as the manipulator requires collision avoidance, error starts creeping into the solution and helps in collision avoidance.

The scale factor $\mathbf{k_s}$ is set to change dynamically within the robot code. This is achieved using equation 4.9. The nominal distance d_0 is chosen depending on the size and complexity of the obstacle. This is the major factor which decides between choice of the SR-inverse and pseudoinverse for calculation of joint velocities.

The scheme was also tested for the rectangular cube as the collision object. Manipulator configurations and motion plots for two cases, namely

a) Constant scale factor $\mathbf{k_s}$=2.5

b) Variable scale factor

are shown in figures 4.22 to 4.32.

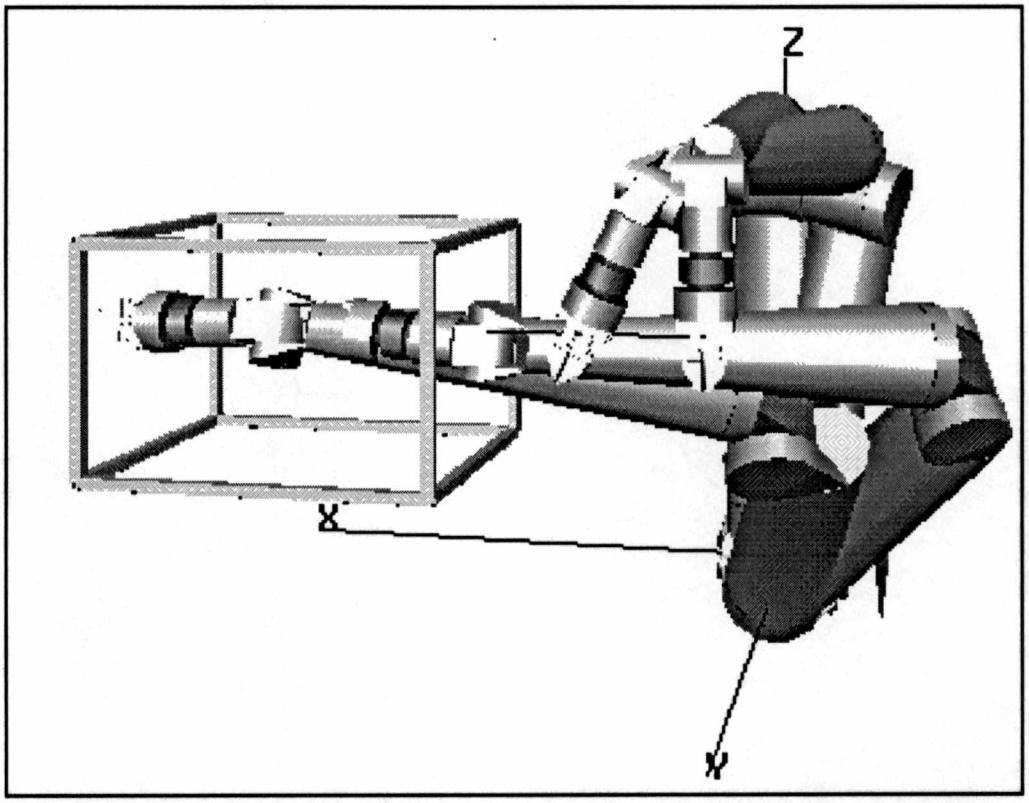


Figure 4.22 Manipulator Configurations for constant scale factor k_S=2.5

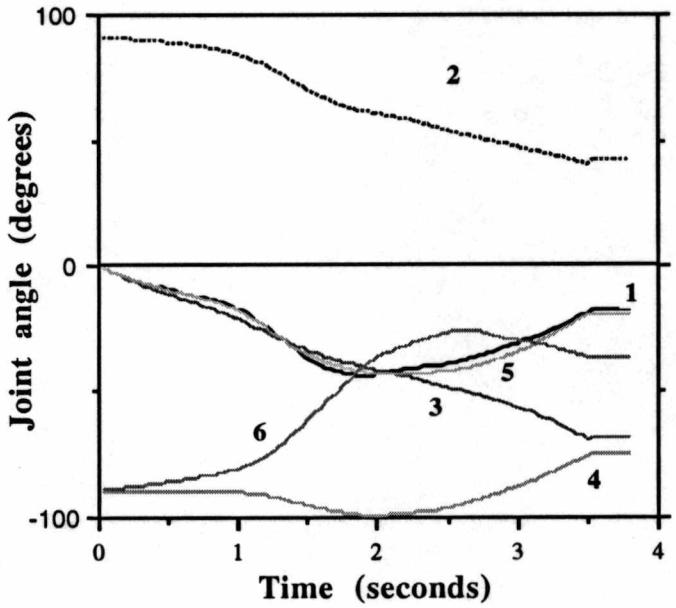


Figure 4.23 Joint angles for manipulator with constant scale factor k_S=2.5

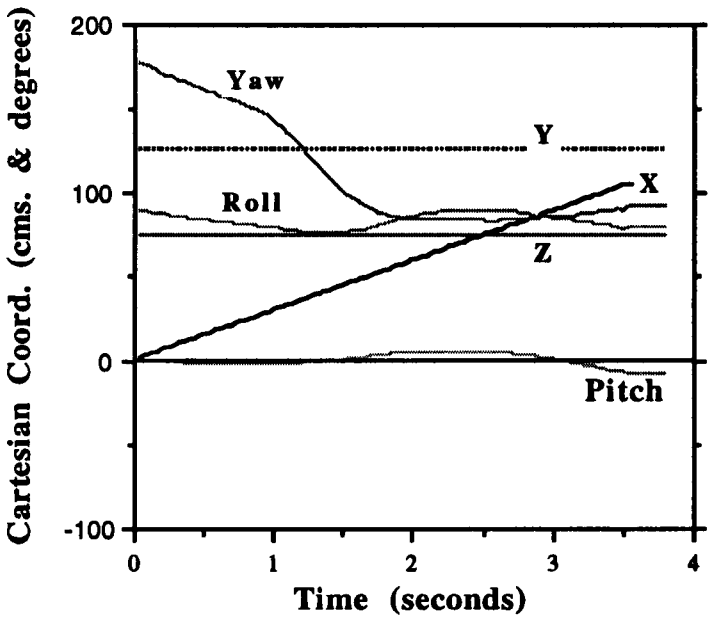


Figure 4.24 Cartesian motion using constant scale factor k_S=2.5

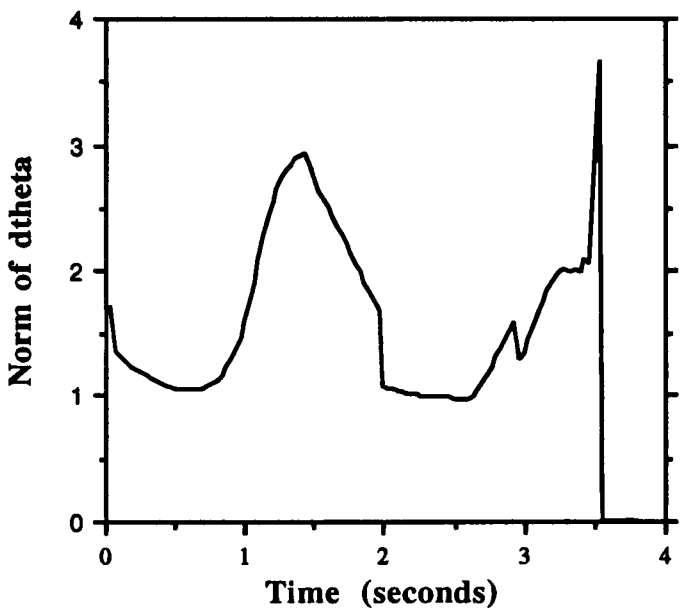


Figure 4.25 Norm of dtheta using constant scale factor k_S=2.5

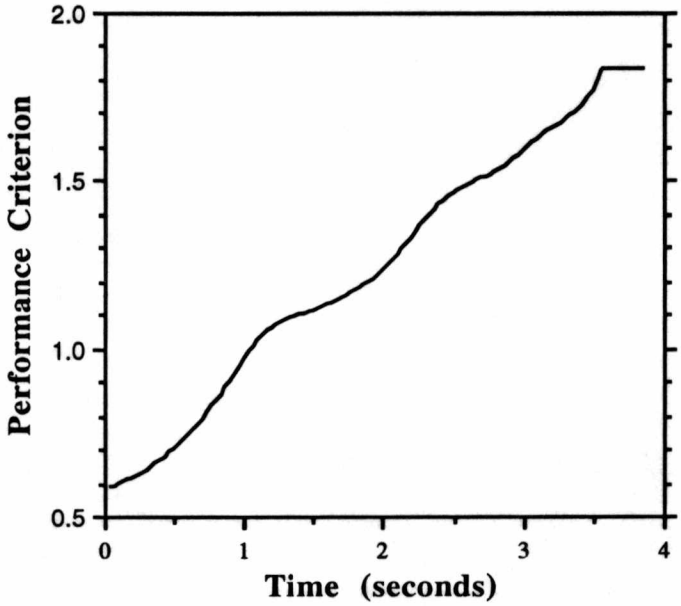


Figure 4.26 Performance Criterion using constant scale factor k_S=2.5

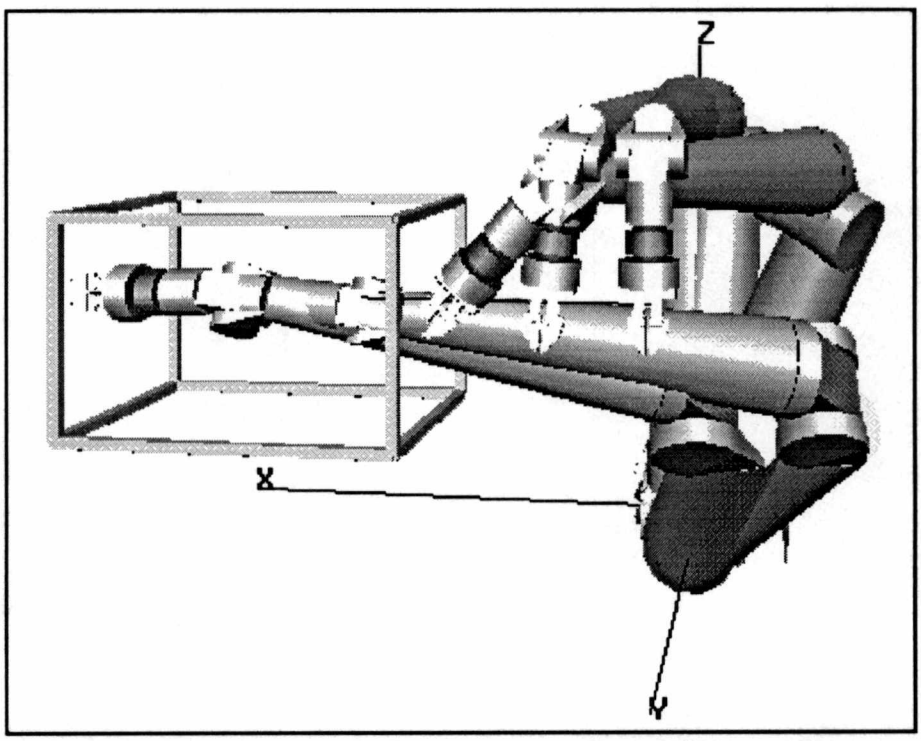


Figure 4.27 Manipulator configurations using variable scale factor k_S

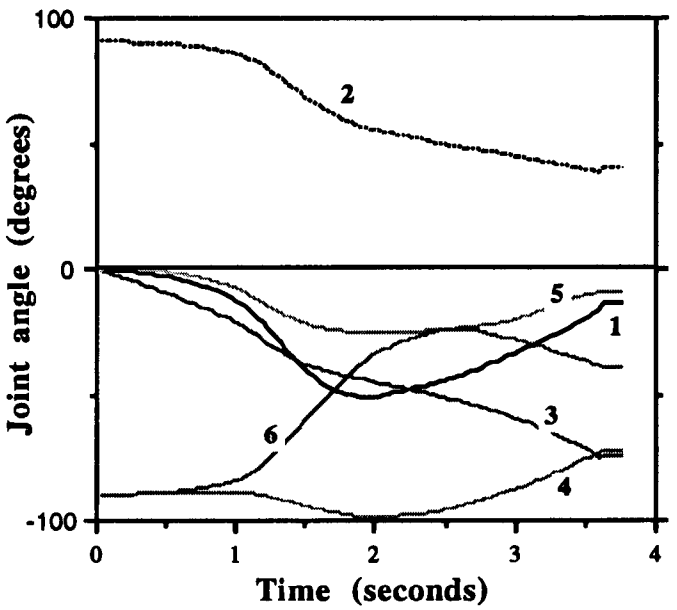


Figure 4.28 Joint angles for manipulator using variable scale factor

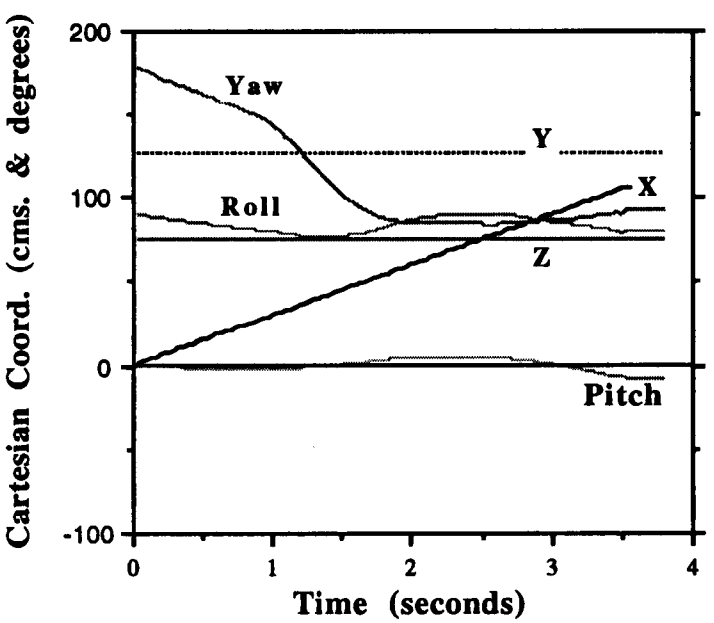


Figure 4.29 Cartesian motion using variable scale factor

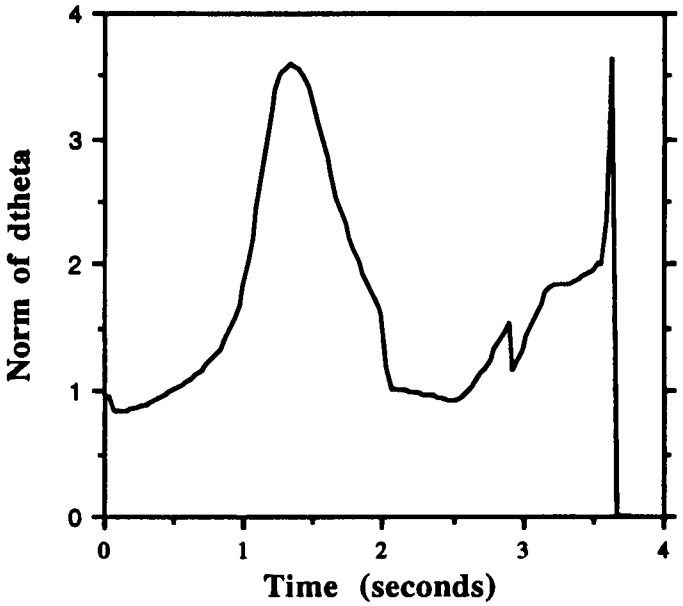


Figure 4.30 Norm of dtheta using variable scale factor

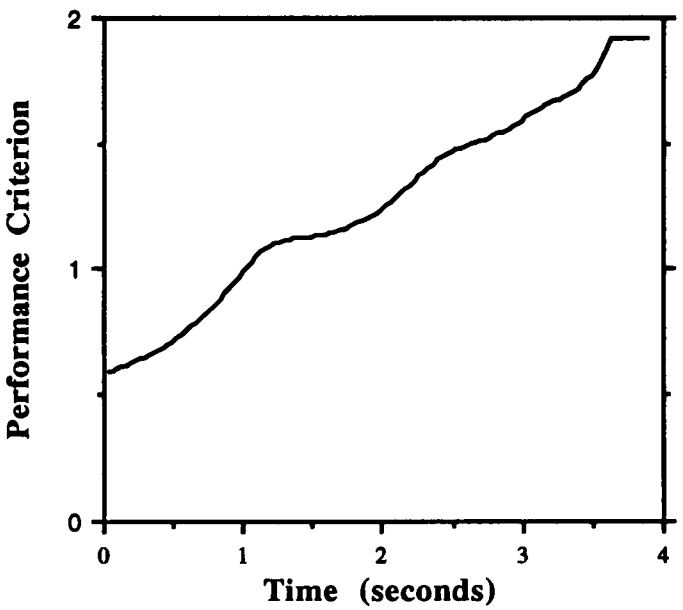


Figure 4.31 Performance Criterion using variable scale factor

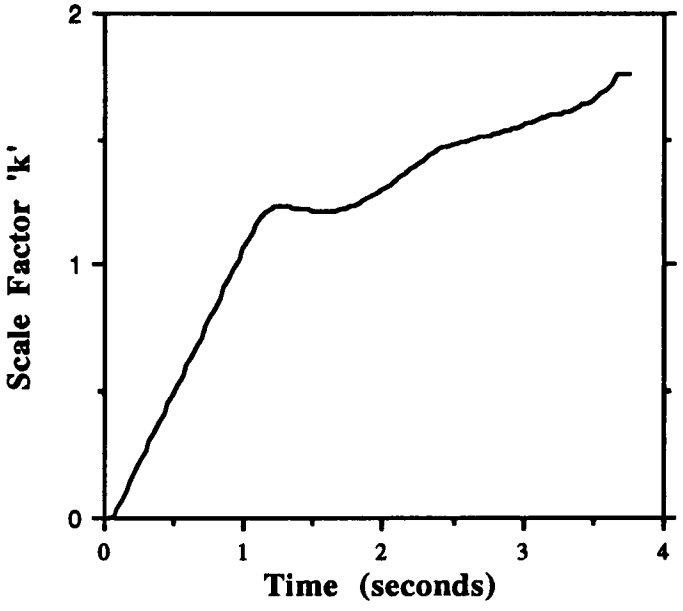


Figure 4.32 Scale factor $\mathbf{k_S}$ versus motion time

The explanation of results obtained for the rectangular cube are similar to that for the rectangular frame. Use of a constant scale factor for the entire motion period results in calculation of joint velocities using the SR-inverse. This is evident from figure 4.24 for Cartesian coordinates which show an error in the orientation from the starting configuration of the manipulator. The norm of dtheta plot in figure 4.25 show the large motion of the joints shortly before entering the collision object. After the manipulator has cleared the front side of the cube, the change in orientation of the manipulator end-effector is very small, thus indicating a decrease in the total motion of the joints.

With a dynamically changing scale factor similar results are obtained. But in this case the scale factor is zero for a small period of motion time, at the beginning. During this period the pseudoinverse is automatically used, in place of the SR-inverse, for calculations. There is no error in the motion of the manipulator, position or orientation,

during this period. Once the SR-inverse is applied for calculations the error starts creeping in the solution and optimization for collision avoidance takes place. It should be noted that even during the use of the pseudoinverse there is optimization of the single degree of redundancy in the manipulator. But as shown in the previous chapter this redundancy is not sufficient for the manipulator to avoid obstacles. The plot for scale factor versus motion time in figure 4.32 shows the change of scale factor with time.

4.6 Conclusions

This chapter has demonstrated the effectiveness of the SR-inverse for collision avoidance in manipulators. The scheme was tested on the seven degree of freedom RRC K-2107 manipulator.

This scheme, unlike the Variable Dimension Task Space scheme presented in the previous chapter, does not neglect any motion directions to add to the redundancy of the system. The SR-inverse is based on two factors governing the solution to the inverse kinematics of the manipulator. The two factors are the accuracy and the feasibility of the solution. Unlike the pseudoinverse, where only the accuracy of the solution is considered, the SR-inverse gives the user a choice to decide which of the two factors is more important for the task being performed by the manipulator.

The scale factor k_s plays a very important role in the success of the SR-inverse used for collision avoidance. This scale factor is the determining factor by which the user decides the weighting between the accuracy and the feasibility of the solution. A higher scale factor leads to higher errors in the solution but is more effective in collision avoidance and vice versa. For the purpose of this study the weighting matrices, in the SR-

inverse, are assumed to be identity matrices of suitable dimension. The properties and effect of these matrices on the solution needs to be investigated.

Choosing the scale factor is also a subject under debate. Two different approaches, one keeping the scale factor a constant and the other having a variably changing scale factor are investigated. Both approaches are effective in collision avoidance but give rise to different results. For a constant scale factor there is always an error in the solution but in the variable scale factor case the error is only introduced when the scale factor is non-zero. The point of change from an exact to an approximate solution depends on the type of obstacle and location of obstacle in the manipulator workspace. Although a higher scale factor results in higher error in the solution, the amount of error and the motion directions in which error is to be introduced cannot be controlled by the scale factor.

Collision avoidance is an example of redundancy being used to optimize the motion of the manipulator. In general, redundancy will be used for joint limit avoidance, joint velocity limit avoidance and singularity avoidance. The next chapter describes a seven degree of freedom redundant Long Reach Manipulator on which the two different schemes for collision avoidance are implemented.

Chapter 5

Design and Control of the Long Reach Manipulator

5.1 Introduction to Tank Waste Retrieval Project

The previous chapters introduced redundancy and optimization of performance criteria for collision avoidaqnce using redundant joint motion of a manipulator. It has been mentioned that the optimization schemes presented in this thesis can be used for both redundant and nonredundant manipulators. The Long Reach Manipulator(LRM)[1-5] is a redundant joint manipulator. Since the LRM will be used for cleanup of waste tanks in the presence of obstacles, the collision avoidance schemes will play an important role in its control. Joint limit avoidance schemes also play an important role in controlling the LRM because of the small motion range of the joints. This chapter represents the implementation of the collision avoidance schemes for control of the LRM. The collision avoidance schemes help the LRM move around obstacles to perform the cleanup tasks.

Many nuclear waste storage tanks around the country are single shell tanks and have outlived their design lifetime. Deterioration in the condition of the tank has resulted in leakage of hazardous material, which requires immediate attention. To prevent the leakage of the contaminated waste from the tanks, a reliable and efficient manipulator system to clean the tanks and dispose of the waste, is needed.

The Tank Waste Retrieval(TWR)[12] manipulation system consists of a Long Reach Manipulator(LRM), a dexterous manipulator and different waste clean up tools. The Long Reach Manipulator is to be used as a positioning device for multiple waste

clean-up tools and the dexterous manipulator, mounted on its end-effector. The LRM designs proposed have more than the minimum number of joints required, making it a redundant joint manipulator. The LRM will have to move around obstacles inside the tank, work within prescribed joint limits and minimize vibrations during the clean-up task. Effective real-time redundancy resolution will be of great importance in control of the LRM. Different performance criteria for optimization are defined and optimized for collision avoidance, joint limit avoidance, minimizing vibrations during motion and attaining optimal configurations for maximum mechanical advantage.

The following sections describe the analysis done for various proposed designs of the LRM, results of the analysis, and redundancy resolution schemes. Section 5.2 gives a brief background of the whole system and its components. It is followed by the kinematic design requirements of the LRM, redundancy resolution schemes and performance criteria for optimization in Section 5.3. Section 5.4 give the results of the kinematic analysis performed for two proposed designs of the LRM. The results of comparisons for the SDI/Spar proposed LRM design(REPPPPP) and Modified Design(REPYPYY) are discussed in sections 5.5 and 5.6. Section 5.7 gives conclusions of the comparison between the SDI/Spar proposed design and the Modified design. This section also describes the future work required for effective redundancy resolution in the LRM.

5.2 Tank Waste Retrieval System

This section gives a background on the setup of the Tank Waste Retrieval system and its components.

5.2.1 Waste Tank

The waste tank dimensions are shown in figure 5.1.

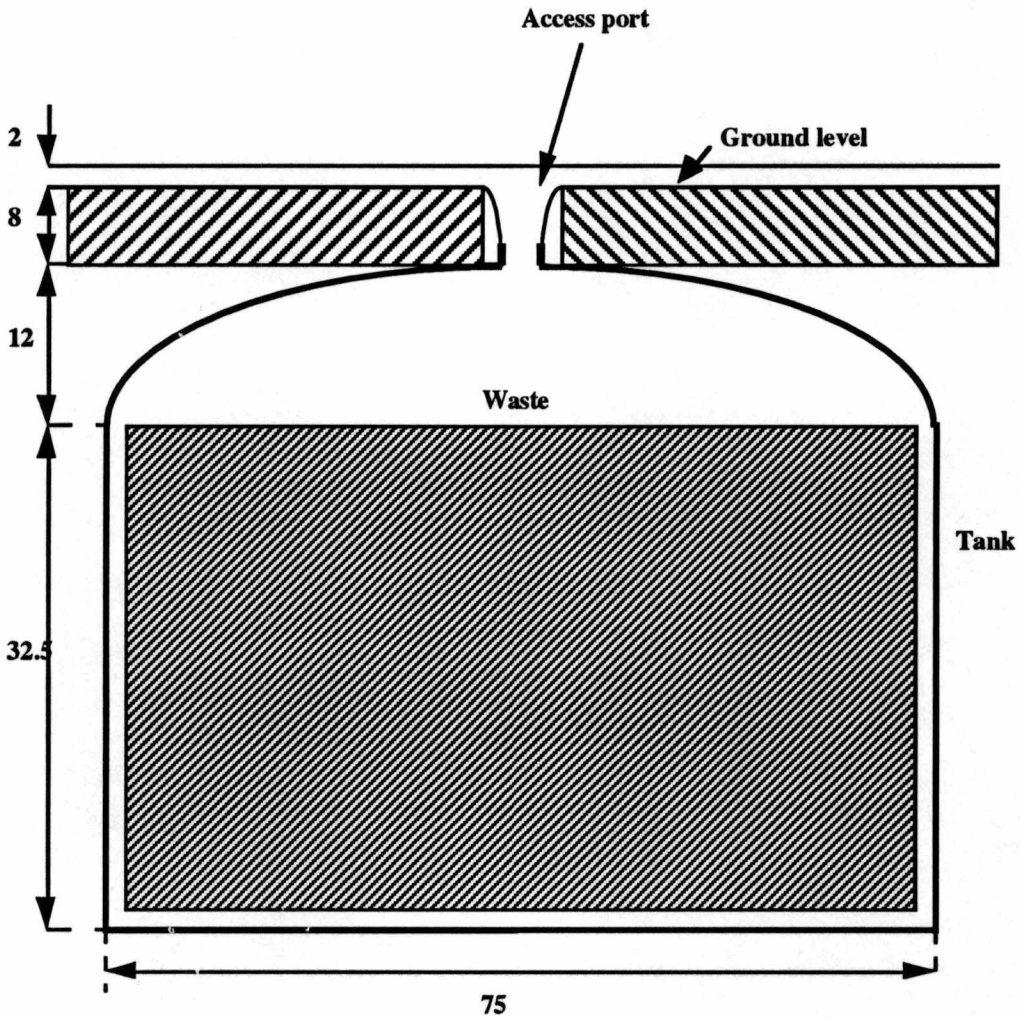


Figure 5.1 Nominal Waste Tank

Table 5.1 Nominal Waste Tank Dimensions

Height of cylindrical portion	**32.5 ft.**
Diameter of cylindrical tank	**75 ft.**
Dome height	**12 ft.**
Depth beneath ground	**8 ft.**
Clearance above ground	**2 ft.**

Table 5.1 lists the nominal waste tank dimensions. It is assumed that the manipulator will be deployed through the existing riser 42 in. in diameter. The waste tank is 75 feet in diameter with a access port 42 in. diameter at the center. The top of the tank is 8 feet below the ground level and the clearance in the dome area at the top of the tank is 12 feet.

5.2.2 Long Reach Manipulator (LRM)

The Long Reach Manipulator is a positioning device for the dexterous manipulator and the different waste clean-up tools. Since the LRM has more than the minimum three joints required of a positioning device, it is a redundant manipulator. Redundancy resolution and the design requirements for the LRM will be discussed in section 5.3.

5.2.3 Dexterous Manipulator (DM)

The dexterous manipulator proposed by SDI/Spar is a six degree of freedom manipulator. The dexterous manipulator design is a slight modification of the Atlas 8F manipulator[12]. This manipulator is both a positioning and orientation device. If a DM with more than six degrees of freedom is used then redundancy resolution schemes will be used for the DM. The DM will be mounted at the end-effector of the LRM and will be

used for fine articulate tasks. If the DM is not used for clean-up it will be replaced by waste clean-up tools mounted on the LRM end-effector.

5.2.4 Waste and Obstacles Inside The Tank

The waste inside the tank is assumed to be a mixture of solid waste and liquid waste. The solid waste will first have to be mined and converted to a form in which it can be removed from within the tank using vacuum pick-up nozzles called the sludge sucker. Obstacles inside the tank are mainly vertical pipes extending from the bottom of the tank to the dome. The LRM will have to move around these obstacles while cleaning up the waste inside the tank.

5.3 Kinematic Design and Redundancy Resolution

5.3.1 Kinematic Design Requirements for the LRM

Design requirements for the LRM, are classified into three categories[1-5]

a) Functional requirements

b) Kinematic requirements

c) Performance requirements

The functional requirements of the LRM include operation both in telerobotic mode as well as robotics mode. If force reflection is used in the telerobotic mode, then the LRM can be used by itself for tasks requiring force control(i.e. without the dexterous manipulator attached).Trajectory planning should be done to avoid exciting the manipulator's vibration modes. The Tank Waste Retrieval system should be such that it

does not load the tank structure itself and contact between the manipulator and the access port should be avoided. In case the manipulator fails inside the tank, provisions should be made for retrieval of the manipulator or repairing the manipulator in the tank itself. Since one single system will be used for multiple tanks the manipulator must be designed such that easy and quick installation and removal of the manipulator is possible.

Kinematic requirements include workspace coverage, deployment, degrees of freedom. The LRM should have a workspace large enough to cover the whole volume of the tank, considering the fact that some tasks may be performed without attaching the dexterous manipulator. The manipulator design has to be such that it can use the existing access ports on the tank which are 24-42 in. in diameter. The LRM may be required to be deployed within the dome clearance above the waste height (shown in Figure 5.1) in the tank. For gross positioning the LRM needs to have at least 3 degrees of freedom. Additional degrees of freedom are needed for joint limit avoidance, obstacle avoidance and reaching positions all over the tank volume. Tower or column(mast) height will be affected by tank configurations, workspace and manipulator deployment.

On the other hand performance requirements dictate issues like position accuracy and repeatability, no-load and full load velocity and static and dynamic performance characteristics. Position and accuracy requirements will depend on the tasks being performed. For initial waste removal, position accuracy required will not be high but for the bottom 10-15% waste high position accuracy will be required. Also repeatability should be high due to fact that end-effectors may be changed in the robotic mode. There will be an upper bound on the velocity so as to prevent collision and allow good dynamic response by the manipulator. Effective damping of the vibrations of the manipulator due to its motion or the motion of the dexterous manipulator mounted on it can reduce the overall time required for the tasks to be performed and the performance of those tasks.

5.3.2 Kinematic Redundancy in the LRM

As mentioned earlier the LRM is a redundant joint manipulator. For effective operation of the LRM, the redundant degrees of freedom will have to be resolved using efficient real-time control and optimization schemes discussed in the previous Chapters. This sub-section discusses the need for these redundant degrees of freedom and their optimization using the Gradient Projection Method[20].

Redundancy in the LRM

Because of the nature of the task to be performed, and motion ranges of the joints, the LRM cannot accomplish the clean-up tasks with the minimum number of joints required for positioning the end-effector. Additional redundant degrees of freedom are necessary for the LRM to accomplish the clean-up task with the above mentioned constraints. Redundant degrees of freedom should be added only if they significantly improve the performance of the manipulator because extra joints mean lower stiffness, lower reliability , difficulty in failure recovery and higher cost of the manipulator .

Once the degrees of freedom for the manipulator have been decided, effective schemes for utilizing the redundancy to optimize the motion of the manipulator are to be developed. Redundant degrees of freedom are of little use if optimization is not effective. To resolve the redundancy in the system various performance criteria will have to be developed for constraints like obstacle avoidance, joint limit avoidance, singularity avoidance, maximizing the mechanical advantage of the manipulator and minimize vibrations.

Most of the obstacles inside the tank are vertical risers. It is undesirable to cut these risers after every few feet of mining and so the manipulator motion will be optimized using the collision avoidance performance criterion. This will allow the

manipulator to work around obstacles facilitating the removal of these risers at a later stage. Due to the prescribed joint limits, the manipulator may not be able to complete all awkward tasks without running into joint limits. Joint limit avoidance will prevent the manipulator from reaching the joint limits. Optimization will also help to keep the manipulator away from singular configurations. Linear actuators provide non-uniform torque throughout the joint motion range. Redundancy will be used to keep manipulator in configurations where minimum torque is required at all joints. This will maximize the mechanical advantage of the LRM. Mechanical advantage is defined as the ratio energy spent on driving a joint to the torque produced at the joint. Due to flexibility of the LRM, vibration damping will play a major role in performance of the manipulator. The vibrations of the arm during motion can be minimized utilizing redundancy to keep manipulator in configurations where the joints can be used to damp out the vibrations. A trade-off between mechanical advantage and vibration damping will be necessary depending on which constraint is more important. Redundancy Resolution using Gradient Projection Method was discussed in Chapter 2.

Performance Criterion for Joint Limit Avoidance

The joints of the manipulator have a upper limit and a lower limit. The manipulator should always remain within these joint motion ranges. To optimize the motion of the manipulator for joint limit avoidance, we define the following performance criterion

$$H_i(\theta)=\sum_{i=1}^{N_p} \frac{\theta_{i,Max}-\theta_{i,Min}}{(\theta_{i,Max}-\theta_i)(\theta_i-\theta_{i,Min})} \qquad (5.1)$$

where $\theta_{i,Max}$ and $\theta_{i,Min}$ are maximum and minimum permissible joint angles and θ_i is the current joint angle for the i^{th} joint.

5.4 Analysis of Initial Kinematic Designs of LRM

This section discusses the kinematic analysis done for the initial designs proposed for the LRM. Figure 5.2 shows the initial simulation model developed for the TWR system. This model was developed using the simulation software on the Silicon Graphics Workstation. After initial analysis the models are also implemented on the IGRIP simulation software.

5.4.1 Kinematic Design of LRM

Initial studies were performed on the Sequential Entry Type Manipulator[1-5] (SET). The SET manipulator has a joint configuration Roll-Extend-Pitch-Roll-Pitch-Pitch.

After an initial study of the SET manipulator, we modified the design of the SET manipulator by replacing the pitch joint with a yaw joint. This yaw joint will improve the dexterity of the manipulator and also help in collision avoidance. In case of the originally proposed SET manipulator, the total number of joints could be reduced from six to five. But this option is only valid when extra initial waste removal is permissible. Simulation models for both manipulators were developed on the existing simulation software. Figure 5.3 shows the kinematic configuration for these two models. The link sizes and joint limits are the identical in both the models. Simulation models for collision avoidance for both models are also presented in this section.

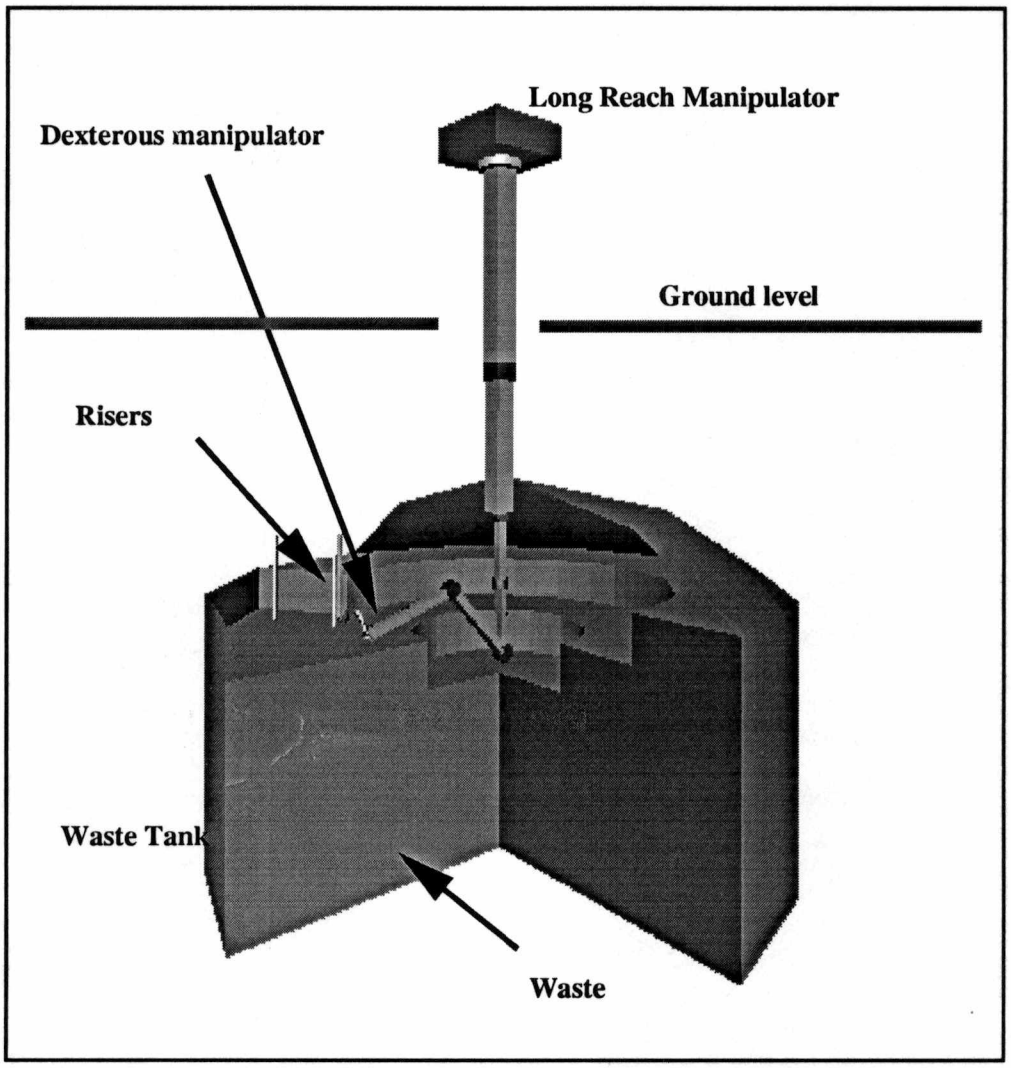


Figure 5.2 Simulation Model for the TWR System

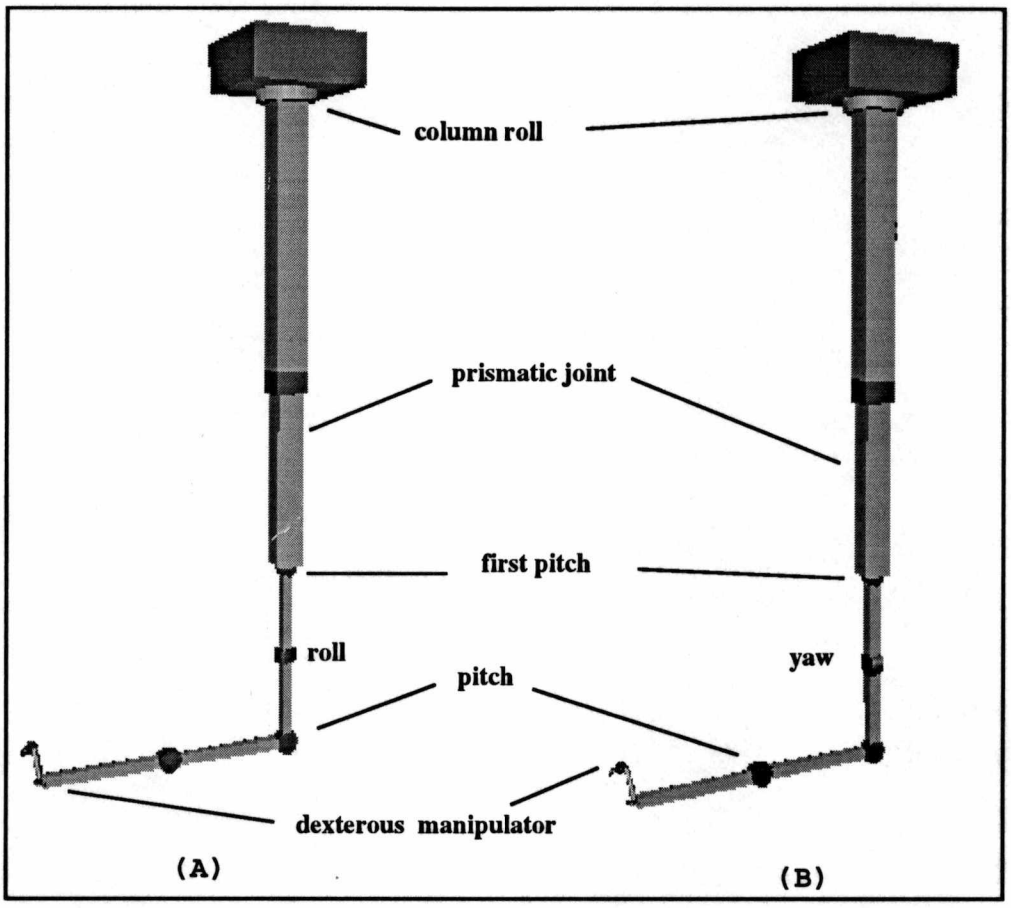


Figure 5.3 Long Reach Manipulator

(A) SET Manipulator (Roll joint) (B) SET Manipulator (Yaw joint)

5.4.2 Sequential Entry Type Long Reach Manipulator (REPRPP)

The SET manipulator was the first proposed design analyzed for the performance requirements of the LRM. It consists of the column roll and prismatic joint, shoulder pitch and roll, elbow pitch and the last pitch joint. The arm can work in an elbow configuration as well as SCARA configuration. In elbow configuration, the arm has good vertical motion and the column need not be used for vertical motion of the end-effector. The shoulder link length is 16 ft. and the remaining two links are 11 ft. each in length. The shoulder pitch joint has motion range -90^{o} to 50^{o}, elbow pitch -140^{o} to 140^{o} with the

shoulder roll and the last pitch -70° to 70°. In this case the complete tank workspace volume is covered if free space motion of the arm is considered and there is no waste inside the tank.

Arm Deployment

Deployment of the manipulator plays a very important role in the performance of the manipulator. The task of deploying the manipulator should be done in minimum time to reduce the overall time needed for clean-up of the entire tank. Since the dome clearance area is only 12 ft. some initial waste clean-up will have to be performed to deploy the arm completely into the tank.

Indexing for the arm will be done before deployment and once inside the tank the arm can work either in a SCARA or elbow configuration. The deployment for this type of manipulator is shown in Figure 5.4. The last link is first deployed into the tank and used for initial waste clean-up. After sufficient waste has been cleaned up all links are deployed sequentially and mining work can be started. It can be seen from the figure that in stage 1 the last link of the manipulator is deployed and used for initial waste clean-up. Stage 2 shows the deployment of the third link after the initial waste clean-up. Finally the shoulder link is inserted into the tank and the arm is completely deployed. In this case the arm is shown deployed in an elbow configuration. If the arm is to avoid obstacles then the roll joint will have to be indexed before deployment to allow arm to work in SCARA configuration.

Collision Avoidance for SET Manipulator with Roll Joint

Because of the shoulder roll joint, the SET manipulator can work either in an elbow configuration or SCARA configuration. Since most of the obstacles in the tank are vertical risers, the arm will have to work in a SCARA configuration to avoid these obstacles.

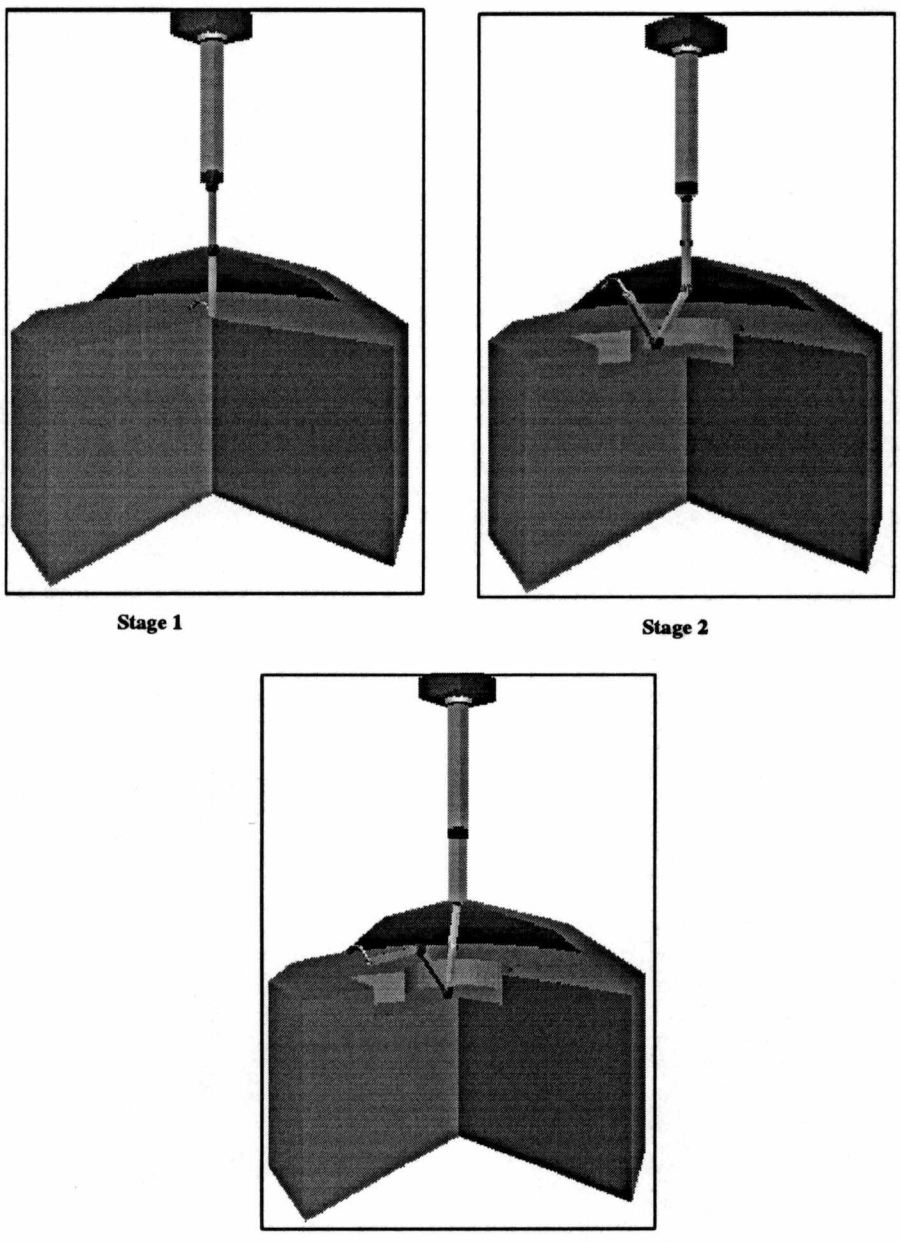
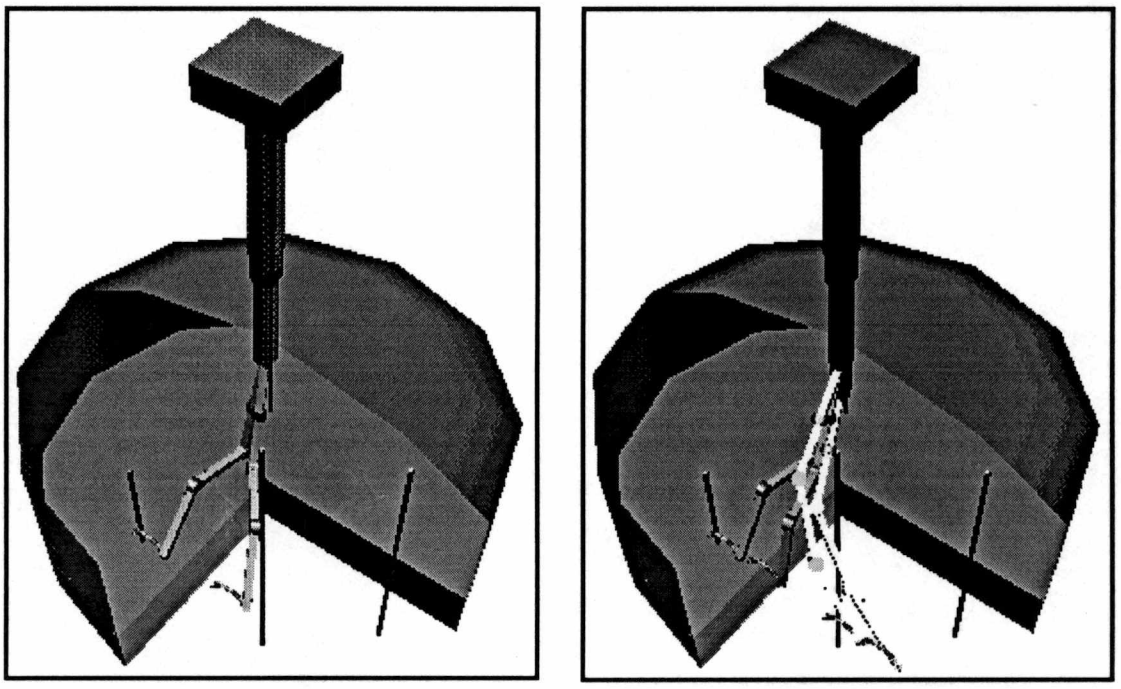


Figure 5.4 Deployment for SET Manipulator With Roll Joint



a) Without optimization

b) With optimization

Figure 5.5 Collision Avoidance for SET Manipulator (Roll Joint)

Figure 5.5 shows the collision avoidance capability for this type of manipulator. Figure 5.5a shows motion of the arm without any optimization and Figure 5.5b shows motion with optimization for collision avoidance. Redundancy resolution is achieved by optimization of the collision avoidance performance criteria using the Gradient Projection Method used in Chapter 3.

5.4.3 Sequential Entry Type Long Reach Manipulator (REPYPP)

After studying the initial design of the LRM, the University of Tennessee team suggested a study of a slightly modified design of the LRM. The design was different from the former design in that the shoulder roll joint was replaced by a yaw joint. The yaw joint increased the dexterity of the manipulator and also provided a cheaper alternative to the complex roll joint. Performance of this arm was tested for workspace coverage and collision avoidance.

Arm Deployment

Deployment sequence for this arm is similar to the SET manipulator with the roll joint. The last link is first introduced to perform the initial clean-up work essential for deployment of the entire arm. If the link sizes are the same as that in the previous case then initial mining work will not be needed as all the links can be deployed in the 12 feet clearance area in the dome of the tank. Figure 5.6 shows the deployment sequence for SET manipulator with yaw joint.

Collision Avoidance for SET Manipulator with Yaw joint

The simulation for collision avoidance is shown in Figure 5.7. Figure 5.7a shows motion with no optimization and Figure 5.7b shows the optimized motion of the arm. Optimization of motion is done using the collision avoidance performance criterion and Gradient Projection Method for Optimization discussed in Chapter 3.

5.4.4 Comparison of SET Manipulator with Shoulder Roll Joint and Yaw Joint

For a hydraulic actuator, a roll joints is more difficult to design than a pitch or yaw joint. Roll joints have less torque generating capability than a pitch or yaw joint. So it is desired to replace the roll joint by a yaw joint. Replacing the roll with a pitch joint would reduce the manipulators collision avoidance capability. The yaw joint can be actuated with the same linear actuators as those used for the pitch joints. No initial waste removal is required if the link sizes remain the same and the manipulator can be deployed in the initial clearance area of the dome of the tank. No indexing of the arm will be required as in case of a discrete roll joint. If the SET manipulator with roll joint works in a SCARA configuration then the arm can only move in the horizontal plane and vertical motion will have to be produced by the column prismatic joint or the shoulder pitch joint.

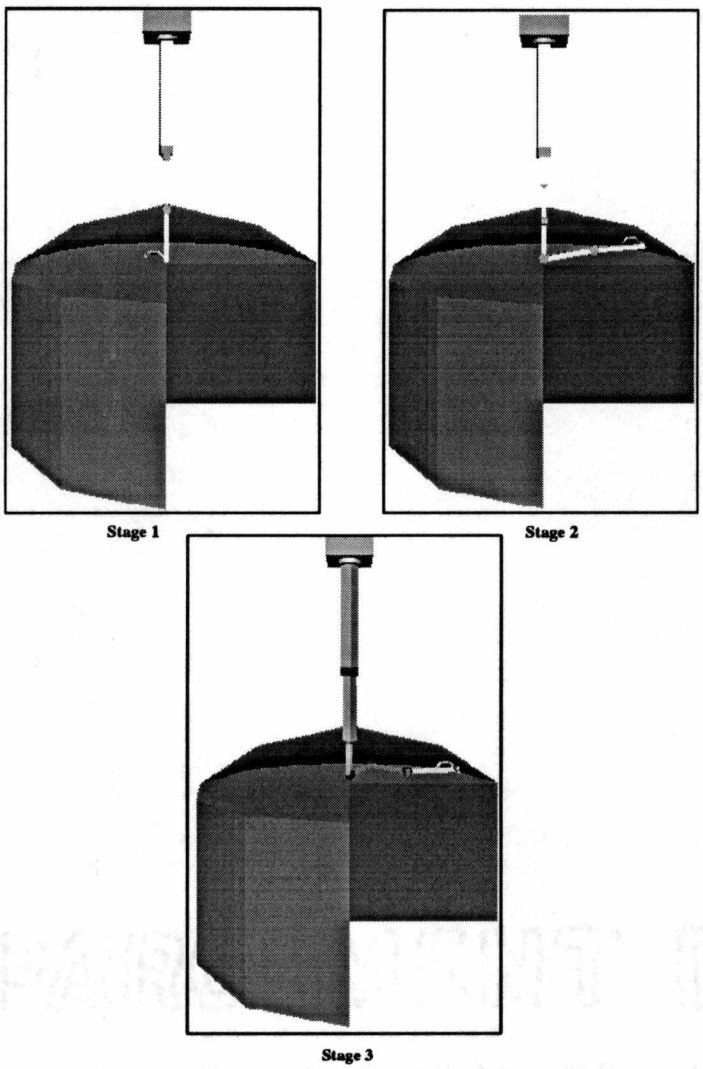
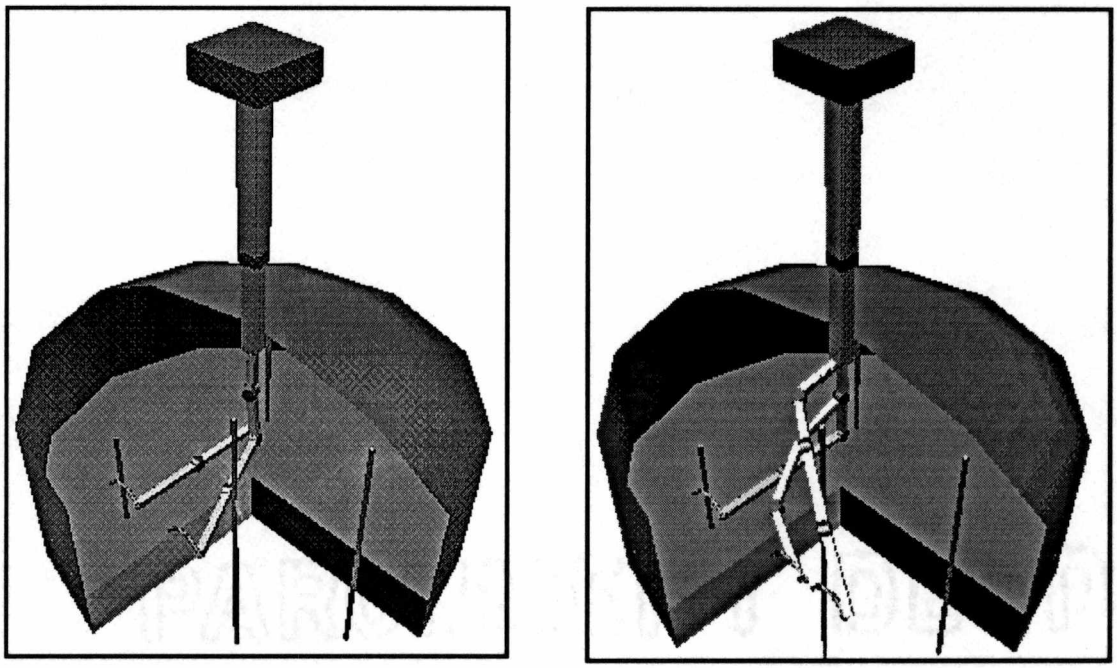


Figure 5.6 Deployment for SET Manipulator with Yaw Joint



a) Without optimization **b) With optimization**

Figure 5.7 Collision Avoidance for SET Manipulator (Yaw Joint)

The shoulder pitch cannot be used for vertical motion because of closeness of arm to waste and use of the column is undesirable because of it's high inertia. The yaw joint increases the dexterity of the manipulator. These advantages of the SET manipulator with yaw joint over that using a roll joint, favor the change of the shoulder roll joint to a yaw joint.

5.5 SDI/Spar Proposed LRM Design And Modified Design

SDI/Spar recommended a LRM design with joint configurations Roll-Extend-Pitch-Pitch-Pitch-Pitch-Pitch[12]. There are also two indexable roll joints, one after the first pitch and the other a wrist roll after the fourth pitch joint with discrete positions 0, 90 and -90^{o}. The first discrete roll joint enables the manipulator to work in an elbow down configuration or a SCARA configuration. The mono mast has a vertical motion range of 53 feet and the discrete roll joints can be indexed to positions 0, 90 and -90^{o}. All

the pitch joints except the last pitch have joint limits -5^{o} to 105^{o}. The wrist pitch has motion range 0^{o} to 90^{o}. The shoulder link or the upper boom is 13.5 feet in length and remaining links are 9.5 feet long except the wrist boom which is 13.5 feet in length. An analysis of this model has revealed some limitations to the LRM with regard to workspace coverage and deployment. Keeping in mind the problem of waste removal needed for deployment, workspace coverage due to waste retrieval sequence , use of prismatic joint and collision avoidance capability with the SDI/Spar proposed design of the LRM, we suggested a modification in the kinematic configuration of the SDI/Spar proposed design and this new configuration is referred to as the Modified Design of the LRM.

It is suggested to change the mid elbow pitch joint to a yaw joint. For convenience of manufacturing, this minor modification can be achieved by rotating the mid elbow pitch joint by 90^{o}. The link sizes and joint limits remain unchanged and the manipulator joint configuration now is Roll-Extend-Pitch-Pitch-Yaw-Pitch-Pitch with two discrete roll joints at the same locations as in the SDI/Spar proposed design.

Simulation models of the SDI/Spar proposed LRM design and the Modified design are shown in Figures 5.8a and 5.8b respectively. Model 5.8b was developed by rotating the pitch joint by 90 degrees into a yaw joint.

5.5.1 Initial Studies for the SDI/Spar Proposed LRM Design

A kinematic analysis of the SDI/Spar arm was conducted to check the workspace coverage of the LRM. Figure 5.9 shows the front view of the tank with the arm working in an elbow down configuration. Figure 5.10 shows the plan or top view of the same arm working in a SCARA configuration.

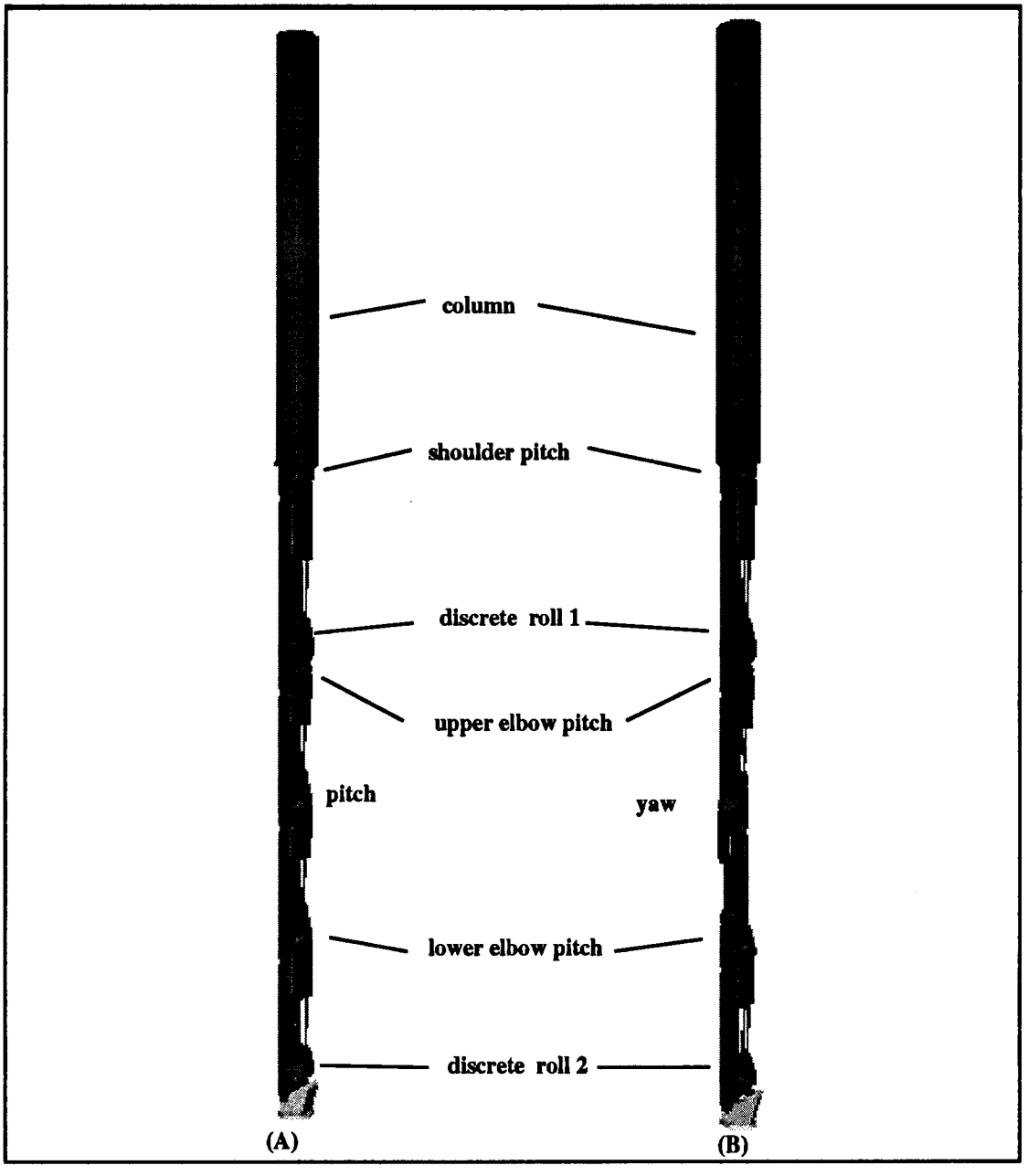


Figure 5.8 Long Reach Manipulator

(A) SDI/Spar Proposed Design (B) Modified Design

For the initial kinematic analysis, free space motion of the arm is assumed i.e. there are no obstacles or waste inside the tank. The arm can either be deployed in elbow down configuration or SCARA configuration. In Figure 5.9 the workspace volume covered by the arm moving in an empty tank is shown. It is seen that the central waste enclosed within the solid curve directly below the column cannot be reached by the end-effector of the LRM. Since the workspace is symmetric around the vertical axis, the volume of the workspace obtained by rotating this curve around the vertical axis, is inaccessible by the LRM working in the elbow down configuration. The only possible way to reach this volume with given joint limits and link sizes is to straighten out the links and move the mast for vertical motion in this region.

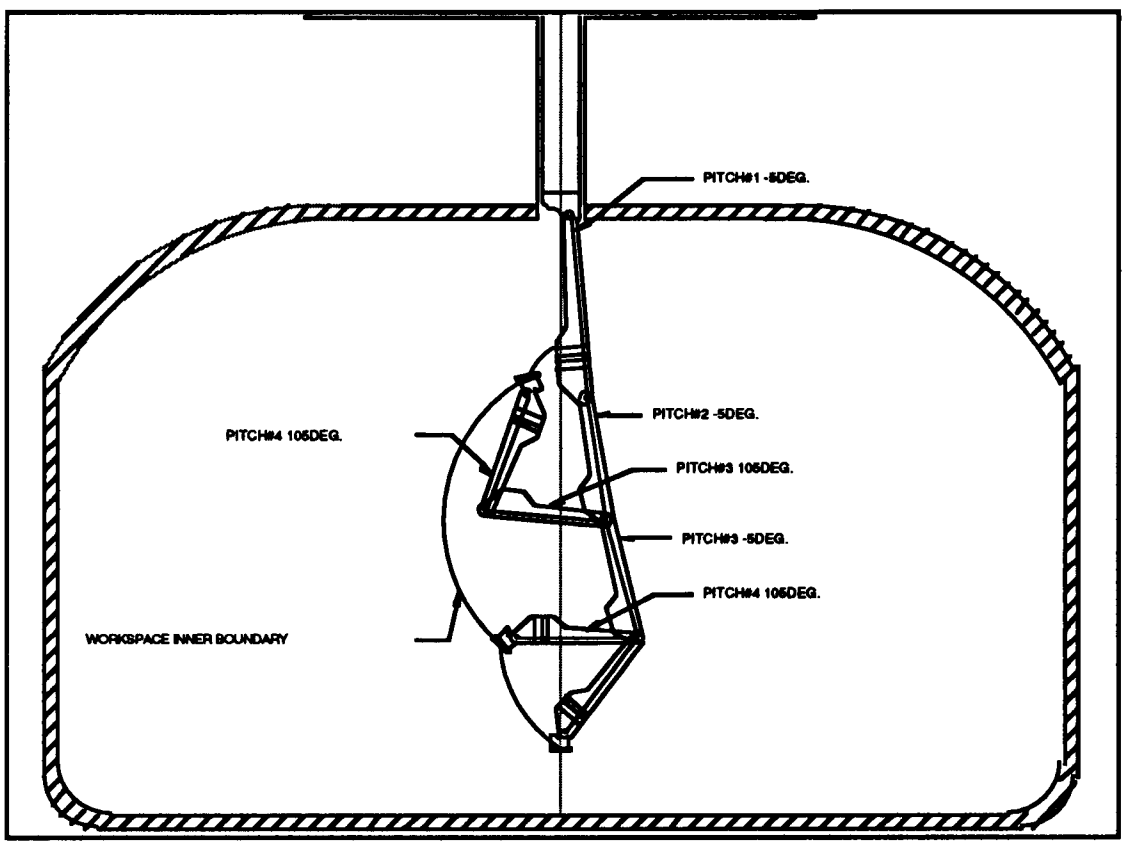


Figure 5.9 Workspace Coverage of SDI/Spar Design in Elbow Configuration and Free Space Motion

On the other hand if the arm is deployed in a SCARA configuration as shown in Figure 5.10, the arm can only move in a horizontal plane. Hence for vertical motion of the end-effector, the column will have to be used. In this case the arm cannot reach a cylindrical volume of radius approximately 2 feet directly below the mast all the way from top of liner to the bottom of the tank. This volume is shown by a hatched circle in the center of the tank. Although the arm can theoretically reach this volume, the end-effector will collide with the column before it can reach the center of the tank.

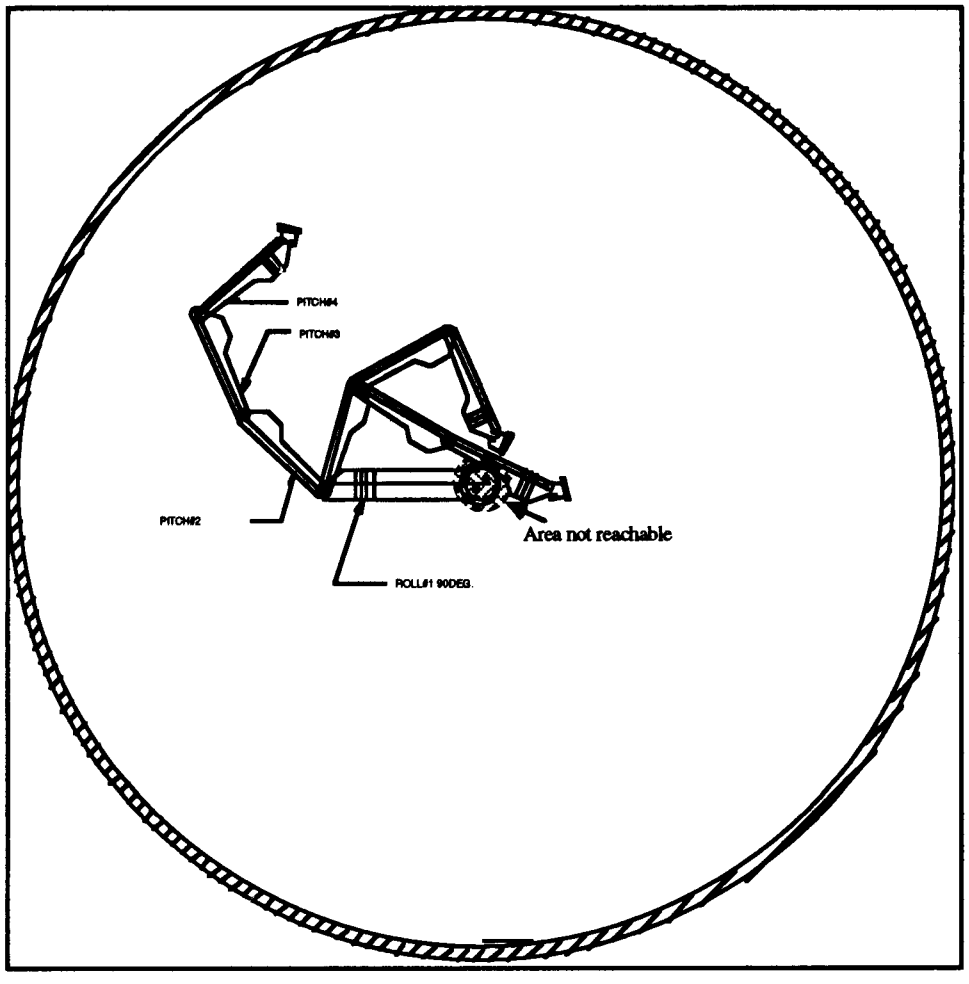


Figure 5.10 Workspace Coverage of SDI/Spar Design in SCARA Configuration and Free Space Motion

5.5.2 Deployment for SDI/Spar Proposed Design and Modified Design

Deployment for SDI/Spar Proposed Design

Since the shoulder boom is 13.5 feet long, it is not possible to deploy the manipulator without some initial waste clean-up. To perform the initial waste clean-up, the wrist boom is first deployed and used for initial clean up of some central waste. From the analysis done, a quarter of a cylindrical volume of central 4 feet high and 24 feet radius will have to be cleaned up initially to enable full deployment of arm in a SCARA configuration. After the initial clean-up the remaining links can be deployed and mining work can be continued without the use of the column prismatic joint. Initial waste clean-up and deployment configuration of arm is shown in Figure 5.11.

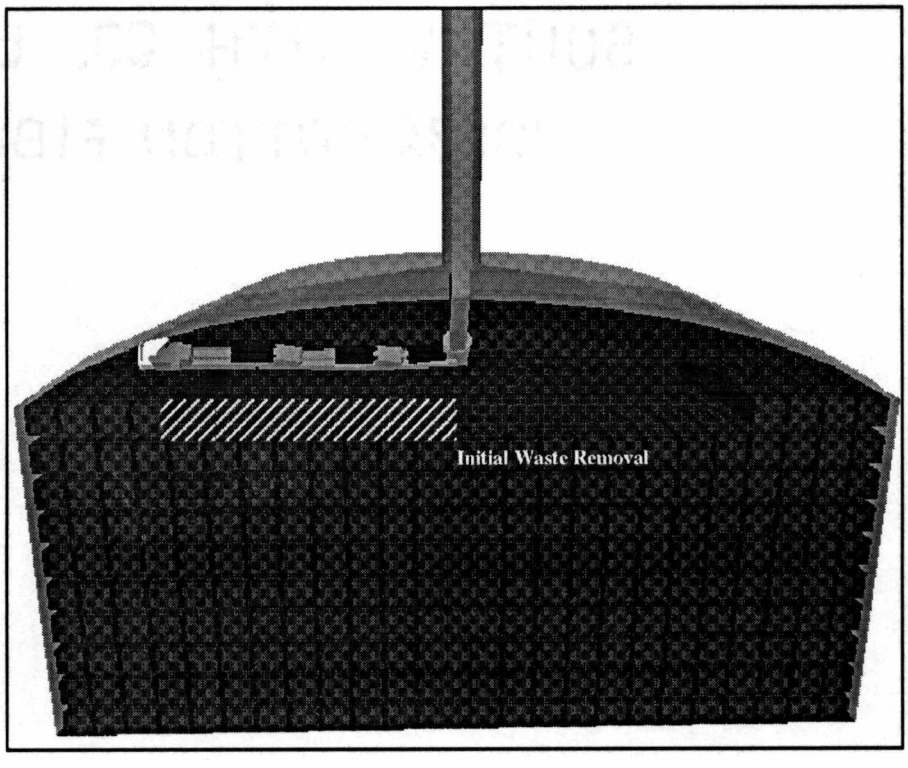


Figure 5.11 Deployment of SDI/Spar Proposed LRM in SCARA Configuration

Indexing will enable the arm to work in an elbow down or a SCARA configuration. Due to strength considerations of the discrete roll joint and limited free space inside the tank, the arm will have to be indexed before deployment.

Deployment for Modified Design

The arm is first indexed by indexing the first discrete roll joint to 90°. The last link is deployed into the tank to perform initial waste clean-up for deployment. After sufficient waste has been mined the entire arm is deployed inside the tank and mining work can continue without the use of the column prismatic joint for vertical motion.

Figure 5.12 shows the deployment of this arm. The waste removed is enough for deployment. In this case initial waste clean-up of a quarter of a cylindrical volume 4 feet high and 12 feet radius is necessary to enable deployment.

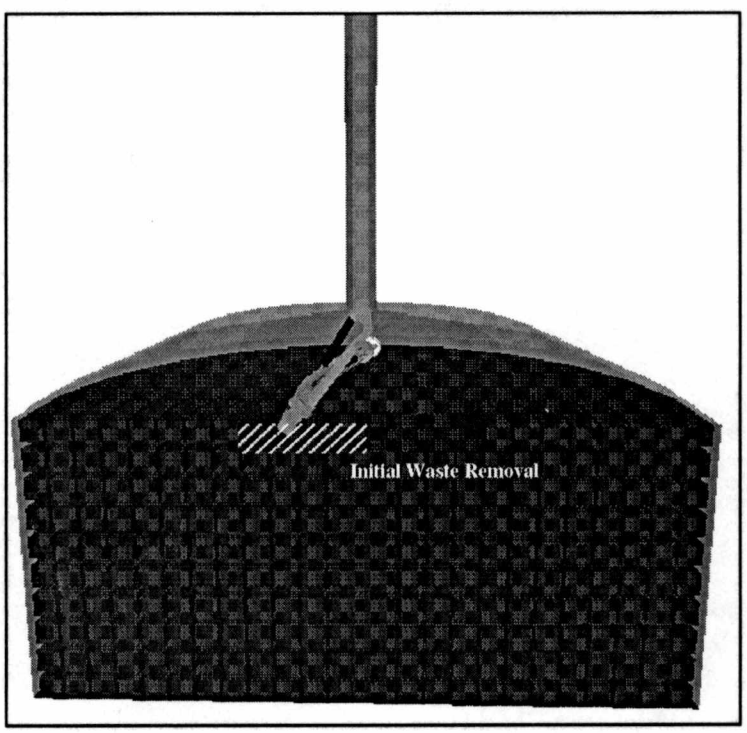


Figure 5.12 Deployment for Modified Design of LRM

5.6 Comparison of SDI/Spar Proposed Design and Modified Design

From the above analysis and results it is clear that the SDI/Spar proposed design for the LRM does not meet all the requirements of the LRM. There is a problem with workspace coverage due to waste removal sequence to be followed by the arm and the prismatic joint has to be used for mining. To overcome these limitations in the SDI/Spar design a Modified design for the LRM is proposed. This section gives a comparison between the SDI/Spar proposed design and the Modified design

a) Initial waste removal for deployment

The SDI/Spar proposed design consists of five pitch joints after the column roll and prismatic joint, and two indexable roll joint with discrete positions -90, 0 and 90^{o}. Waste removal of a quarter of a cylindrical volume approximately 4 feet in height and 24 feet radius is required, before the shoulder boom can be deployed inside the tank. For arm to be deployed in elbow configuration much more waste removal is required and this configuration is not suitable for initial mining work.

On the other hand, with the same link lengths and joint motion ranges as that of the SDI/Spar design, the modified design requires initial waste removal of a quarter of a cylindrical volume of 4 feet in height and 12 feet radius, before the entire arm can be deployed inside the tank in a SCARA configuration. Thus , the amount of waste removal required in this case is 25% of the amount to be removed for the original design manipulator to be deployed.

b) Mining Sequence for each Manipulator

SDI/SPAR Manipulator: For the SDI/Spar proposed manipulator, the arm is indexed and deployed in SCARA type configuration. A cylindrical volume of approximately 2 feet

radius, directly below the column, in the second layer cannot be cleaned. This is because the end-effector cannot reach this waste without colliding with the arm. Now the arm cannot move vertically downwards because of the waste directly beneath the column. To further proceed with the mining operation the arm will have to be retracted back out of the tank and central portion will have to be mined using the prismatic joint for vertical motion.

If the arm is deployed in the elbow down configuration initially, only a small area near the outer perimeter of the tank can be cleaned after which the arm will have to be retracted back out and some vertical cleaning will have to be done using the prismatic joint. For the elbow down configuration to be viable for cleaning the entire tank, initial mining of a very large cylindrical volume of waste is necessary, requiring the use of the prismatic joint during mining.

Modified Design After deployment, a cylindrical portion of diameter approximately 2 feet radius and height 11 feet, directly below the mast, cannot be cleaned. But the waste surrounding this volume can all be cleaned out. The arm can also clean up the central waste below the top 11 feet of unmined central waste. This will result in a collapse of all the unmined central waste. It can be seen that more than 70% of the entire waste inside the tank can be cleaned after the initial deployment. The prismatic joint is locked after the deployment and all this waste is cleaned without the use of the prismatic joint. The remaining waste around the bottom can then be cleaned up by lowering the arm a little more inside the tank. If collapse of the unmined waste is undesirable, then the arm can be allowed to retract back out the tank and this central waste can be cleaned out. From our analysis, we also find that the 11 feet of central waste can also be mined right after deployment if the length of the upper boom is reduced from 9.5 feet to 7 feet approximately.

c) Prismatic Joint and Its use for Mining:

The prismatic joint has a travel of 53' and has a very high inertia. The use of this joint during the mining operation is not recommended. The SDI/Spar proposed design requires frequent use of the prismatic joint to move the arm in the vertical direction, retract and deploy the arm after a couple of layers of waste is cleaned. To use the arm in an elbow configuration a large central area will have to be mined out initially using the prismatic joint.

On the other hand, the modified design can clean up more than 70% of the tank waste once the arm is deployed into the tank. Throughout the mining operation the prismatic joint is locked and not used for vertical motion. The prismatic arm will then have to be moved only once, to lower the end-effector in the tank, to clean up the waste at the bottom outer surface of the tank.

d) Collision Avoidance:

The SDI/Spar proposed manipulator can avoid obstacles only if it works in a SCARA configuration. Because of the yaw joint , the modified manipulator design possesses collision avoidance capability even if the discrete roll joint is not indexed.

Also if the SDI/Spar proposed arm works in a SCARA configuration, the entire arm is very close to the contaminated waste during the entire mining process. The modified design for the LRM keeps the entire arm away from the waste during the mining process.

Table 5.2 compares the SDI/Spar design and the Modified design of the LRM.

5.7 Conclusions

In case of the SDI/Spar design, only the SCARA configuration for the arm is considered. This is because the arm is assumed to be completely deployed inside the tank and should not be retracted out of the tank during mining. If retraction of the arm is allowable then the mid pitch joint can be removed without hampering the performance of the manipulator, thereby reducing the cost of the manipulator. In elbow configuration, the arm cannot perform any clean-up tasks unless a very large central volume of waste is removed initially. This will necessitate the use of the column prismatic joint for vertical motion of the end-effector, which is undesirable.

The Modified design for the LRM can cover the entire volume of the tank after one deployment of the manipulator. With this modification, the prismatic joint need not be used at all during the mining process. The yaw joint also gives the manipulator the required collision avoidance capability and workspace coverage of the entire tank volume while staying within the prescribed joint limits. The arm need not be retracted out of the tank as in the SDI/Spar proposed design, thus reducing the risk of contaminants coming out of the tank. A major portion of the arm is always away from the contaminated waste and hence reduces the chance of the whole arm coming in contact with the waste.

The modifications required in the SDI/Spar proposed design are minor. One is to rotate the mid elbow pitch joint by 90^{o} to make it a yaw joint. Another modification which is desired but not necessary, is to reduce the length of the upper boom from 9.5 feet to approximately 7 feet. This will enable the new design to clean up the top 11 feet of unmined central waste which otherwise will have to be collapsed by cleaning the waste beneath it.

Table 5.2 Comparison of SDI/Spar Manipulator Design and Modified Design

Description	SDI/Spar Manipulator Design	Modified Design
Configuration	R-E-P-P-P-P-P	R-E-P-P-Y-P-P
Waste Removal for Deployment	Approx. a quarter of 4' height and 24' radius when arm works in SCARA configuration	Approx. a quarter of 4' height and 12' radius. 25% the volume of the original design.
Workspace Coverage after deployment	Without moving arm back out of the tank, the workspace coverage is less than two layers of waste.	Can clean up **more than 70% of tank volume** after deployment and locking the column.
Use of Prismatic Joint	Have to use prismatic joint during mining and to retract and deploy arm after cleaning two layers of waste.	Have to use prismatic joint only twice during the whole operation. Not used for mining.
Collision Avoidance Capability	Can have collision avoidance only in SCARA configuration.	Collision Avoidance capability both in horizontal and vertical direction.
Working Configuration	Cannot work in elbow configuration until nearly 75% of waste under mast is cleaned	Arm can work in an indexed or unindexed roll joint configuration.
Closeness to waste	Entire arm close to waste when cleaning due to SCARA configuration requirement.	Arm always away from waste.
Retraction of arm out of tank	Arm needs to be retracted after every two layers of waste are cleaned.	Arm can be left inside after deployment until entire tank is cleaned.
Time for clean-up	Considerably more time required for clean-up of entire tank, because of frequent retraction and deployment.	Much faster in cleaning up the entire tank, because arm needs to be deployed only once.

While the waste tank, waste material and Modified arm models were developed at UT, the SDI/Spar arm model was developed at PNL[16]. The waste material has been modeled in the form of layers. Each layer consists of concentric rings of waste. It is assumed once the end-effector of the LRM reaches a certain position in a ring the entire ring can be cleaned out using the column roll because of the symmetry of the workspace around the vertical axis.

Chapter 6

Computer Program

6.1 Graphical User Interface and Popup Menus

The code for the simulation and control of the RRC K2107 manipulator consists of graphics as well as kinematics and was developed on the Silicon Graphics GTX 340 workstation. The entire graphics as well as kinematics code is written in the C programming language and use of the Silicon Graphics GL library is made use of for modelling the manipulator and collision objects. Figure 6.1 shows the graphical user interface for the simulation and control of the RRC K2107 manipulator.

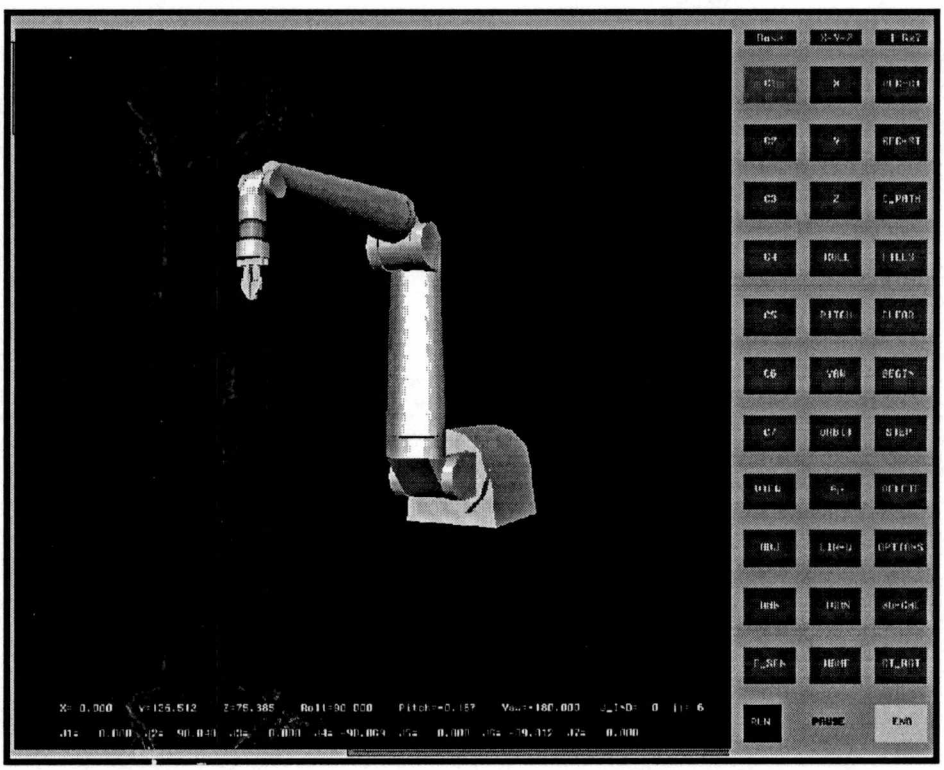


Figure 6.1 Graphical User Interface

The initial user interface and simulation program was developed by Chan[ ]. Every user has modified the program according to his needs by adding graphical utilities and various subroutines to perform certain calculations. The figures shown below explain the three major graphic utilities added to the code for better handling and ease of operation of the simulation program.

Figure 6.2 shows the popup menu for choosing the dimension of the Jacobian to be used for the inverse kinematic subroutine in the code. The user has five choices from 1x7 to 5x7. The default choice is a full rank Jacobian 6x7 and this Jacobian is used when the program is run initially. The choice of Jacobian is indicated in the upper right hand corner of the teach pendant.

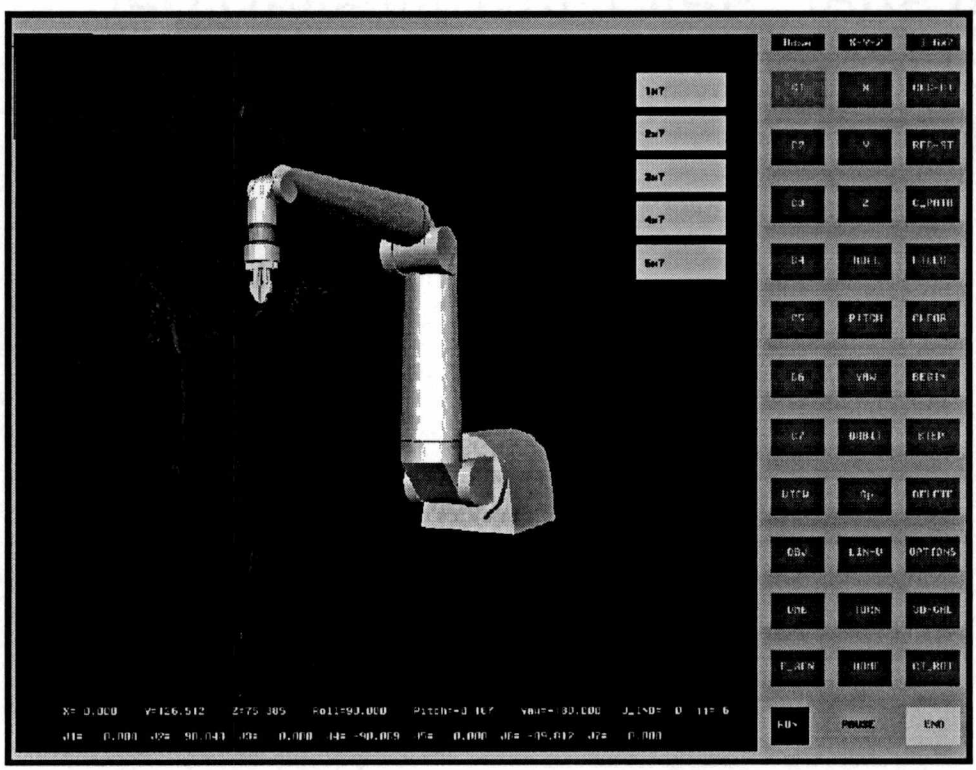


Figure 6.2 Popup Menu for Choice of Jacobian

Figure 6.3 shows the graphical interface for the collision avoidance object environment. The popup menu in this case has 12 buttons. These buttons help modify the collision object in the manipulator environment as required by the operator.

Table 6.1 lists the function of each button on the popup menu of the collision object. The button order is from the top of the menu to the bottom of the menu. For each popup menu the color of the buttons is different so as to help the user identify the menu being used at that moment.

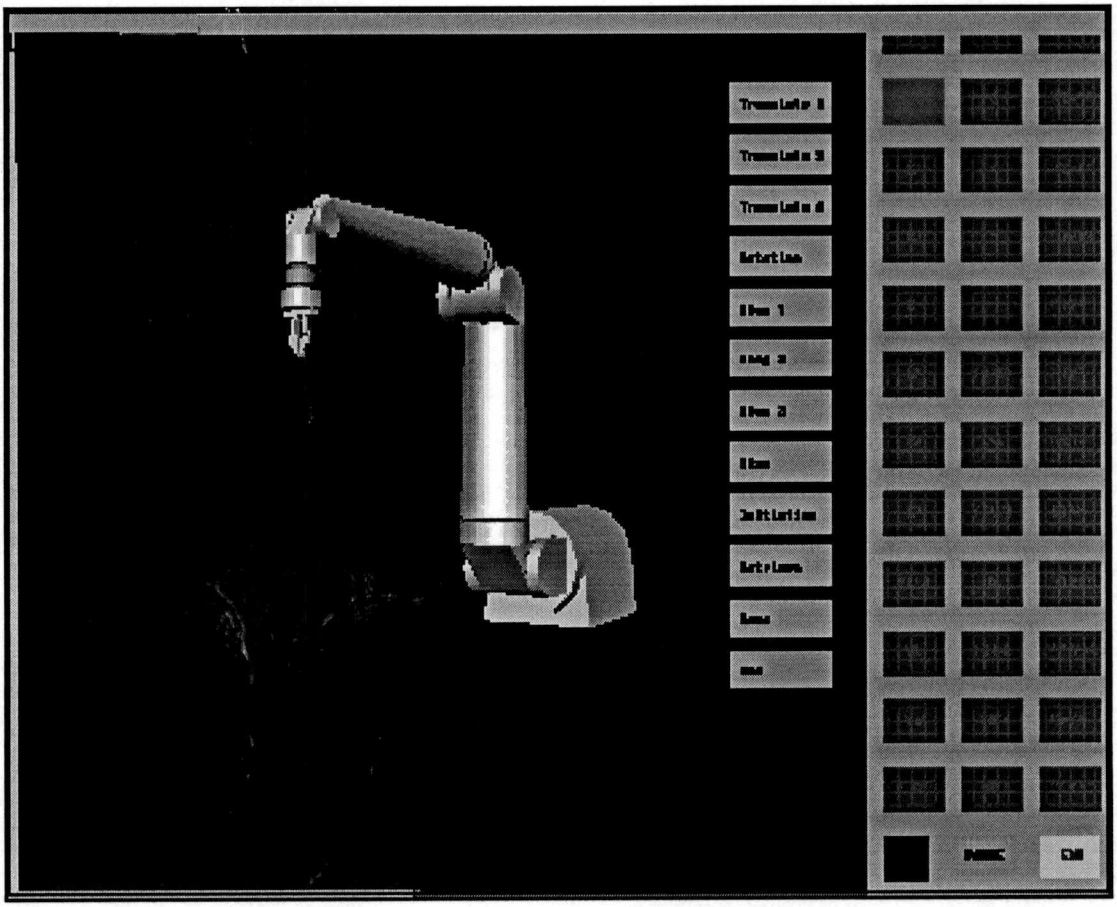


Figure 6.3 Graphical Popup Button Menu for Collision Object Manipulation

Table 6.1 Buttons on Collision Object Popup Menu

Button	Action
Translate 1	Translate Object in the X direction
Translate 2	Translate Object in the Y direction
Translate 3	Translate Object in the Z direction
Rotation	Rotate the object about its major axis
Diag 1	Move object in a diagonal XY direction
Diag 2	Move object in a diagonal YZ direction
Diag 3	Move object in a diagonal ZX direction
Size	Change the size of the collision object
Initiation	Restores object to the initial condition undoing all changes
Retrieve	Gives the user a choice of collision objects which can be used
Save	Saves the modified collision object data to a specified file
End	Returns user to the main manipulator program

Figure 6.4 and table 6.2 give an explanation about the popup menu for the manipulator control. These include buttons for selecting the gain factor for optimization, defining new joint limits and joint velocity limits, changing the control scheme, controlling

the manipulator in real time and some other keys. Each button on a popup menu of any kind, when selected using the mouse button, calls a subroutine and executes the commands in that subroutine.

The popup menu in this case also has 12 buttons. The name and function of each button is explained in table 6.2 given on the next page.



Figure 6.4 Graphical Popup Menu for Manipulator Control Options

Table 6.2 Buttons on Manipulator Control Options Popup Menu

Button	Action
Gain	Allows user to input gain factor for optimization scheme
Scheme	Allows choice of inverse kinematics scheme to be used
Radius	Define radius of circular path
E.E. Path	Show end-effector path
Trajectory	Repeat the trajectory continuously
Color	Change the color attributes of the graphics
Store	Store trajectory and path data
S.G. Time	Change the graphics update time
Limit	Change various limits for joints
Teleop	Gives the user a choice of performing teleoperation
Coll. Av.	Allows user to use some extra functions in manipulating the collision avoidance environment
Exit	Returns user to the main manipulator program

6.2 Manipulator Control Subroutines

For realization of the redundancy optimization schemes developed earlier in the thesis, several subroutines were added to the original C codes. The programs developed

for control of the RRC K2107 manipulator and the Long Reach Manipulator are main_pro.c, tools.c, goalp.c, collision.c, objenvs.c, draw_k2107.c, draw_ltm.c, objects.c, init_graph.c and optimization.c.

The following give the C programming language subroutines added to various files and the functions they perform. The listing of these subroutines is attached at the end of the thesis.

6.2.1 Subroutines added to main_pro.c

a) Option Button :

Upon selection of this button on the teach pendant, the user is presented with a popup menu shown in the previous section of this chapter. A choice of a button in this menu calls various subroutines in other files allowing user to change variables like color of user interface, gain factors, joint limit values and scheme of optimization.

6.2.2 Subroutines added to goalp.c

a) k_inv( ) function:

This routine is similar to the k_inverse routine written by Chan[ ]. This routine is accessed by the main_pro.c program when the user moves the manipulator using a reduced Jacobian scheme. This routine contains calculations for all the optimization scheme used for this study.

6.2.3 Subroutines added to tools.c

Most of the subroutines added to this file perform calculations for different inverse kinematic schemes tested in this thesis. The other subroutines check the joint limits and carry out the Gram-Schmidt orthogonalization method

a) least_normsr11( ) and least_normsr( ):

These routines are accessed by the k_inv and k_inverse routines of the goalp.c program. These routines perform the actual calculation of joint solutions using the Singularity Robust Inverse scheme.

b) least_norm71( ) and least_norm72( ):

These perform the inverse kinematics calculation using the pseudoinverse for calculation of joint velocities.

c) least_norm1( ), least_norm2( ).....least_norm5( ) :

These are routines used to perform the inverse kinematics using the real-time scheme proposed by Dubey et. al[ ]. There are five Jacobian choices the user can make and hence the five routines corresponding to the Jacobian chosen.

d) All cal_j*( ) routines :

These routines are small routines used to perform the Jacobian and weighting matrix or Jacobian and gradient multiplication's. The results of these routines are used in the least_norm routines.

e) get_ortho1( )..... get_ortho5( ) routines :

These routines are used to perform the Gram-Schmidt orthogonalization scheme on the set of homogenous vectors calculated by the least_norm routines.

f) j_v_limit1( ) :

This routine checks the joint velocity limits after a solution is found and defines the magnitude of the self-motion.

6.2.4 Subroutines added to init_graph.c

The routine added to this file mainly perform all the graphics related functions of the code.

a) view_butn1( ) :

This routine draws the object popup menu when the Obj. button on the teach pendant is selected.

b) view_butn2( ) :

Draws the popup menu for changing the size of the collision object. This routine is accessed by the size button on the object popup menu.

c) view_butn3( ) :

Draws the popup menu for the user to choose a Jacobian. This routine is accessed by the upper rightmost button on the teach pendant.

d) view_butn4( ) :

This draws the options popup menu and is accessed by the main_pro.c program when the options button in the teach pendant is chosen.

e) get_obj_size( ) :

This routine actually modifies the size of the collision object depending upon the button chosen in the object size popup menu.

f) move_coll_obj( ):

This routine also manipulates the collision object by moving it around in the manipulator workspace. This routine performs the X, Y, Z, and diagonal translations of the collision objects in the workspace.

g) get_jacob( ):

This routine actually decides on the type of Jacobian chosen by the user in the Jacobian popup menu and returns the value to the main program.

h) get_case( ):

Decides the choice made by the user in the options popup menu and calls the appropriate routines required by the user.

i) draw_all( ):

This routine accesses the manipulator configurations from a memory or a data file and draws all configurations simultaneously. It also changes the background color of the main graphic window to white. This routine is accessed by using the draw_all button on the teach_pendant.

j) store_data( ):

Stores the joint and Cartesian position values in a data file for plotting or for later access. The robot program can read these values and play the recording at any time as requested by the user.

These are subroutines added to the existing robot code developed by Chan[ ]. In addition a new file was added to the library of source code for the RRC K2107

manipulator and two new files to the Long Reach manipulator code. They were the objenvs.c file and the draw_lrm1.c and draw_lrm2.c files respectively.

The objenvs.c file modifies the collision object environment and also manipulates, saves and retrieves collision objects. The draw_lrm files for the Long Reach Manipulator perform the same function as the draw_k2107 and draw_LTM files in the RRC K2107 code. These include modelling the manipulators, initializing the views and other parameters for the manipulator and finally calculating the Jacobian matrix, homogenous transform, and shortest distances between manipulator and collision object links for the manipulators.

Chapter 7

Conclusions and Recommendations

7.1 Conclusions

The purpose of this thesis was to test and implement two different collision avoidance schemes for robot manipulators. While the "Variable Dimension Task Space" scheme can be used for both redundant and nonredundant manipulators, the "Singularity Robust Inverse" scheme[25,26] can only be used for redundant manipulators.

The Variable Dimension Task Space scheme relies on the fact that for a particular task being performed, some task space motion coordinates may be more important than others. Sacrificing control of the unimportant motion directions results in an increase in the redundancy of the system. This increased redundancy assists the manipulator in collision avoidance. The scheme was tested on simulation model of the seven degree of freedom RRC K2107 manipulator. The full 6x7 Jacobian for all motion directions controlled was used along with the 3x7 Jacobian and the 1x7 Jacobian. While for the 3x7 Jacobian the orientation of the end-effector was neglected, in case of the 1x7 Jacobian all motion directions except the command direction were neglected. Three collision objects with gain factors of 0.01, 0.05, 0.1, 0.15 and 0.25 were utilized for the Gradient Projection Method[20]. Large gain factors resulted in better optimization but lead to oscillations in the manipulator configuration. Very small gain factors resulted in poor optimization and manipulator was unable to avoid obstacles. These factors were chosen depending on the complexity of the obstacles with larger gain factors used for more complex obstacles. The scheme was also tested for a combination of different task spaces.

The manipulator started motion with a complete control over all motion directions and then the control was switched over to a partial Jacobian with certain motion directions neglected. This control strategy had the advantage of control over all motion directions when collision avoidance was not necessary. The case with orientation neglected and only position controlled appear to be the most effective in collision avoidance. The end-effector maintained its position throughout the entire motion period and followed the desired trajectory. On the other hand, only controlling the command direction and neglecting all other motion directions was less preferred to the condition where only orientation was neglected. The manipulator in this case can change its position and orientation arbitrarily, and did not follow a prescribed path. Unnecessary self-motion is added to the manipulator motion in this case and resulted in large joint motions.

On the other hand, the "Singularity Robust Inverse" scheme achieves collision avoidance by sacrificing the accuracy of the solution for feasibility. Feasibility in this case is defined as the ability to avoid obstacles. The choice of the scale factor for the SR-inverse, which influences the error introduced in the solution, was discussed, and experiments were performed with constant and variable scale factors. It was shown that a larger scale factor introduced a higher error in the trajectory, but the scale factor $\mathbf{k_s}$ had no influence on the motion direction in which error was introduced. The case of a dynamically changing scale factor was the most useful for collision avoidance. In this case, the manipulator had no error in its trajectory until it came within range of collision with an obstacle. As soon as collision avoidance was necessary, a non zero scale factor $\mathbf{k_s}$ was introduced in the SR-inverse and an error was introduced in the solution. The scale factor $\mathbf{k_s}$ varied dynamically, depending on the distance of the manipulator arm from the obstacle and the magnitude of optimization needed. This ensured that the error introduced in the solution was zero or very small, when collision was not imminent. The equality of

the SR-inverse and the pseudoinverse for zero scale factor was also shown in the results for the SR-inverse scheme.

Cartesian coordinate control and Cylindrical coordinate control with the collision avoidance schemes was implemented on the Silicon Graphic simulation models of the Long Reach Manipulator[1-5,12]. The effectiveness of the collision avoidance schemes implemented on the Long Reach Manipulator was shown using simulations. Some discussion and simulations related to the design of this manipulator were also presented[.

It is important to note that an increase in redundancy does not always imply efficient optimization. Increased redundancy results in increased self-motion for the manipulator but may not help in collision avoidance. Hence it is necessary to increase redundancy in the system such that it helps in optimization.

The computer code for control of the RRC K2107 manipulator and the Long Reach Manipulator was written using the C programming language. This code in conjunction with the code written by Chan[21] provides a user friendly graphical interface for the control of redundant joint manipulators.

7.2 Recommendations

The two optimization schemes presented are related to collision avoidance. These schemes can be extended to optimization for the joint limit problem and singularity problem. A single degree of redundancy, as in case of the RRC K2107 manipulator, is generally not sufficient to avoid obstacles. The "Variable Dimension Task Space" scheme presented in this thesis can be applied in cases where an increase in redundancy for optimization is desired. The decision of increasing the redundancy of the system, by neglecting motion directions, is dependent on the nature of the task being performed. Hence, the choice of optimization scheme to be used, is made depending on the nature of

the task and the type of manipulator being used. Studies involving the combined effect of the weighting matrix and the scale factor on the error in the SR-inverse solution need to be performed. The weighted least-norm scheme[8,21] of optimization should be modified to work for collision avoidance. This would solve the problem of choosing an appropriate gain factor for optimization as in case of Gradient Projection Method[20]. It may also solve the problems of oscillations in neighborhood of optimal configurations. Finally this study can be extended to incorporate optimization of several multiple criteria simultaneously[7,22].

In this study the minimum distance between the manipulator and obstacles is calculated by model based methods. Since the location of the obstacle is completely known to the user, the minimum distance is calculated by finding a line perpendicular to both the obstacle link and the manipulator link. In unstructured environments the location of the obstacle may not be exactly known. A sensor based system for calculating distances between manipulators and obstacles needs to be developed.

References

References

[1] "Key Design Requirements for Long Reach Manipulators", by Kwon D. S., March-Leuba S., Babcock S. M. and Hamel W. R., September 1993, ORNL/TM-12251.

[2] "Implications of Access Hole Size On Tank Waste Retrieval System Design And Cost", by Babcock S. M., Kwon D. S., Burks B. L., Stoughton R. S. and Evans M. S., May 1994, ORNL/TM-12751.

[3] "Status of Long-Reach Manipulator Control Development at the Oak Ridge National Laboratory", July 20-21, 1993.

[4] "Long-Reach Manipulation for Waste Storage Tank Remediation", Jansen J. F., March-Leuba S., Kwon D. S., Babcock S. M., Burks B. L., Kress R. L. and Hamel W. R., FY 1991 Report, ORNL/TM-11999.

[5] "Parametric Design Studies of Long Reach Manipulators", by Kwon D. S., March-Leuba S., Babcock S. M., Burks B. L., and Hamel W. R., Fifth Topical Meeting on Robotics and Remote Handling, April 1993.

[6] Chan, T. F. and Dubey, R. V., " Design and Experimental Studies of a Generalized Bilateral Controller for a Teleoperator System with a Six Dof Master and a Seven Dof Slave",1994 IEEE International Conference on Robotics and Automation, Vol. 3 , pp. 2612.

[7] McGhee S. M., Chan, T. F., Dubey, R. V., and Kress R. L., " Probability Based Weighting cf Performance Criteria for a Redundant Manipulator.",1994 IEEE International Conference on Robotics and Automation, Vol. 3 , pp. 1887.

[8] Chan, T. F. and Dubey, R. V., " A Weighted Least-Norm Solution Based Scheme for Avoiding Joint Limits for Redundant Manipulators",1993 IEEE International Conference on Robotics and Automation, Vol. 3 , pp. 395.

[9] Dubey R. V., Euler J. A. and Babcock S. M.," Real Time Implementation of an Optimization Scheme for Seven Degree of Freedom Redundant Manipulators," IEEE Transactions on Robotics and Automation, Vol. 7, No. 5, October 1991.

[10] Dubey, R. V. and Luh, J. Y. S.(1988) " Redundant Robot Control Using Task Performance Based Measures.", Journal of Robotic Systems, 5(5), 409-432,October.

[11] Dubey, R. V. and Luh, J. Y. S.(1987) " Redundant Robot Control for Higher Flexibility", IEEE Int. Conf. on Robotics and Automation, March,Vol. 3 , pp. 1066-1072.

[12] "Tank Waste Retrieval Test Manipulator System", by Schilling, October 1994.

[13] Zghal, H., Dubey, R., and Euler, J., "Collision Avoidance of a Multiple Degree of Freedom Redundancy Manipulator Operating Through a Window," Journal of Dynamic Systems, Measurement, and Control, pp. 717-721, Dec. 1992.

[14] Albert, A., Regression and the Moore-Penrose Pseudo-inverse, Academic Press, 1972.

[15] Zghal, H., Dubey, R., and Euler, J., "Efficient Gradient Projection Optimization for Manipulators with Multiple Degrees of Redundancy," Proc. of the IEEE International Conference on Robotics and Automation, May 13-18, Cincinnati, Ohio, Vol. 2, pp. 1006-1011, 1990.

[16] IGRIP model for TWR system developed at PNL.

[17] Nakamura, Y., Hanafusa, H., and Yoshikawa, T., " Task-Priority based Redundancy Control of Robot Manipulators," The Int. Journal of Robotics Research, 6, pp. 3-15, 1987.

[18] Borenstein, J., and Koren, Y., "Real-Time Obstacle Avoidance for Fast Mobile Robots in Cluttered Environments," IEEE Journal of Robotics and Automation, pp. 572-577, 1990.

[19] Zghal, H., "Collision Avoidance of a Multiple-Redundant-Joint Manipulator Operating through a Window," M.S. Thesis, Univ. of Tennessee, Knoxville, Dec. 1989.

[20] Leigois, A., "Automatic Supervisory Control of the Configuration and Behavior of Multibody Mechanisms," IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-7(12), pp. 868-871, December 1977.

[21]Chan, T.F., "Simulation and Real-Time Control of A Seven-Degree-of-Freedom Manipulator," M.S. Thesis, Univ. of Tennessee, Knoxville, May 1992.

[22] McGhee, B. S., "Collision Avoidance and Multiple Criteria Optimization for a Redundant Manipulator," M.S. Thesis, Univ. of Tennessee, Knoxville, December 1992.

[23] Robotics Research Corporation "User Manual", of the K-series robot arms type 11 robot controllers, Pub. No. UM-100-87, 1986.

[24] Yoshikawa, T., Foundation of Robotics, Analysis and Control, The MIT Press, 1990.

[25] Nakamura, Y., Advanced ROBOTICS, Redundancy and Optimization, Addison-Wesley Publishing Company, Inc., 1991.

[26] Nakamura, Y., and Hanafusa, H., 1986,"Inverse Kinematic Solutions with Singularity Robustness for Robot Manipulator Control," J. Dyn. Sys., Meas., Contr. 108 (3): 163-171.

[27] Cleary, K., "Incorporating Multiple Criteria in the Operation of Redundant Manipulators," Proc. 1990 IEEE International Conference on Robotics nad Automation, 618-624.

[28] Pamanes, J. A. and Zeghloul, S., "Optimal Placement of Robotic Manipulators Using Multiple Kinematic Criteria," Proc. 1991 IEEE International Conference on Robotics and Automation, 933-938.

Appendices

Appendix A

D-H Parameters for K2107 Manipulator

Figure A.1 and table A.1 show the joint frame attachment and the D-H parameters for the RRC K2107 manipulator.

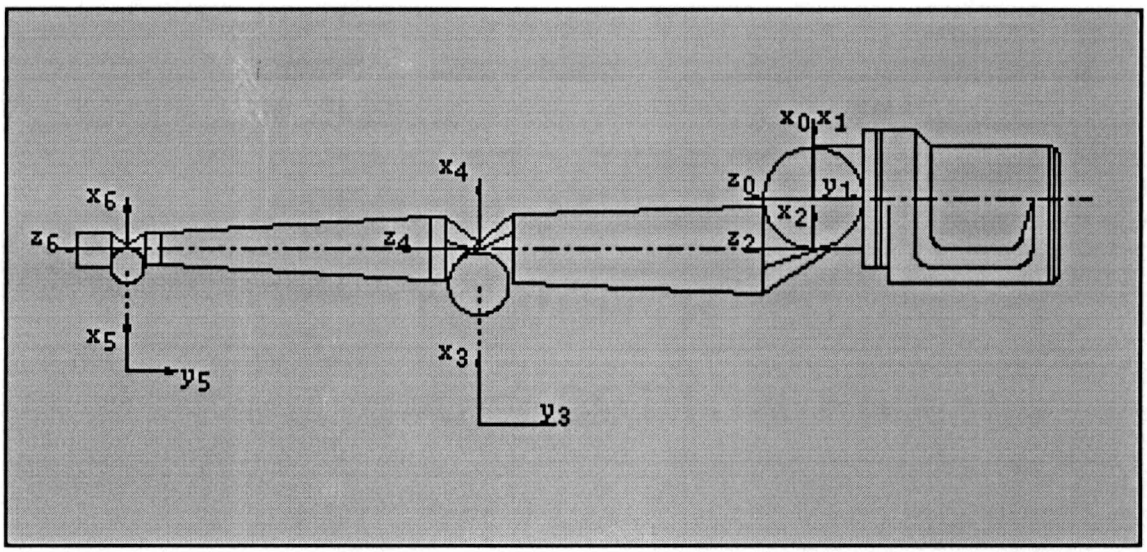


Figure A.1 Frame Attachment for RRC K2107 Manipulator

Table A.1 D-H parameters for the RRC K2107 Manipulator

α_i (degrees)	a_i (meters)	d_i (meters)	θ_i (degrees)
-90	0.00	0.00	$-180 < \theta_1 < 180$
90	-0.1429	0.00	$-45 < \theta_2 < 135$
-90	-0.1080	0.9648	$-180 < \theta_3 < 180$
90	0.1080	0.00	$-180 < \theta_4 < 0$
-90	-0.0492	0.9651	$-360 < \theta_5 < 360$
90	0.0492	0.00	$-180 < \theta_6 < 0$
0	0.00	0.2583	$-720 < \theta_7 < 720$

Appendix B

Gram-Schmidt Orthogonalization Method

The Gram-Schimdt orthogonalization method finds a basis of orthonormal vectors from a given set of independent vectors. Suppose there is a set of n independent vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots\ldots\ldots\ldots, \mathbf{a}_n$ and we require a set of n orthonormal vectors $\mathbf{q}_1, \ldots\ldots\ldots\ldots\ldots, \mathbf{q}_n$ having the same basis as the independent vectors $\mathbf{a}_i$. The Gram-Schmidt method provides a easy way to find the basis of orthonormal vectors. This orthogonolization method takes the first vector as given and makes it a unit vector(normalizes the vector). It then finds the projection of the second vector onto this normalized vector and subtracts this projection from the original second vector. This leaves us with a second vector perpendicular to the first vector having the same basis as the original second vector. The new second vector is then normalized. To find the third vector it subtracts from the original third vector the projection onto the first two normalized orthogonal vector. This gives us the third orthogonal vector which is then normalized. This procedure is carried out for all n vectors resulting in an orthonormal basis of vectors for the independent set of vectors.

The following equations clarify the Gram-Schmidt method. Lets start with an independent set of n vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots\ldots\ldots\ldots, \mathbf{a}_n$. We will find a set of orthonormal vectors with the same basis as the former vectors. Let the set of orthonormal vectors be called $\mathbf{q}_1, \ldots\ldots\ldots\ldots\ldots, \mathbf{q}_n$.

First let,

$$\mathbf{q}_1 = \frac{\mathbf{a}_1}{\|\mathbf{a}_1\|} \tag{B.1}$$

This is the first orthonormal vector. The second orthogonal vector is then given by

$$q_2' = a_2 - (q_1^T a_1) q_1 \tag{B.2}$$

The vector in Eqn. B.2 is then normalized to give us an orthonormal vector to the first vector. This process is carried on for n vectors to get a set of n orthonormal vectors.

The n^{th} orthonormal vector will be the normalized form of the vector

$$q_n' = a_n - (q_1^T a_n) q_1 - (q_2^T a_n) q_2 \ldots\ldots\ldots - (q_{n-1}^T a_n) q_{n-1} \tag{B.3}$$

This procedure is very useful in calculating a orthonormal set of homogenous solutions from an independent set of homogenous solutions given by Eqn. 2.9. This is necessary to simplify the calculation of the scalar coefficients k_i and w_i in Eqns. 2.6 and 2.12 respectively. If we have only one degree of redundancy then there is only one homogenous solution and the scalar coefficients are found easily. But in case of higher degrees of redundancy it is necessary to use this normalization method to find a set of orthonormal homogenous solution vectors for use in the Gradient Projection Method of optimization.

Appendix C

Data for Obstacles Used In Experiments

This appendix contains information on the different obstacles used in experiments for the Variable Dimension Task Space and SR-Inverse methods for collision avoidance. The same obstacles were used for both the schemes to allow a comparison between the two schemes. The manipulator starting configuration and final configurations are given in the experiments and results sections of the corresponding chapters. Three obstacles were used for the experiments and they are rectangular frame, rectangular cube and hexagonal tube.

Data for rectangular frame:

The rectangular frame shown in figure C.1 is made of four link vectors. Table C.1.a gives the position for the frame in the manipulator workspace and Table C.1.b gives the orientation and link lengths for the frame.

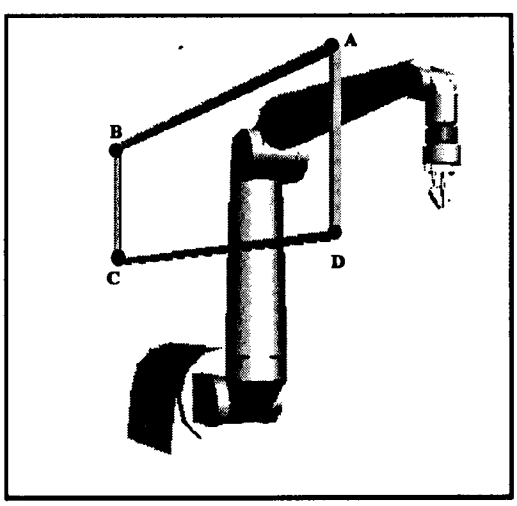


Figure C.1 Rectangular Frame In Manipulator Workspace

Table C.1.a Location of Frame Vertices

Vertex	X (cms.)	Y (cms.)	Z (cms.)
A	68.0	173.0	93.0
B	68.0	36.0	93.0
C	68.0	36.0	53.0
D	68.0	173.0	53.0

Table C.1.b Link Lengths and Orientations for Frame

Link	Length (cms.)	Orientation in base frame
AB	137.0	Perpendicular to XZ plane
BC	40.0	Perpendicular to XY plane
CD	137.0	Perpendicular to XZ plane
DA	40.0	Perpendicular to XY plane

Data for Rectangular Cube :

The rectangular cube is modelled with 12 link vectors as shown in Figure C.2. The black dots indicate the vertices

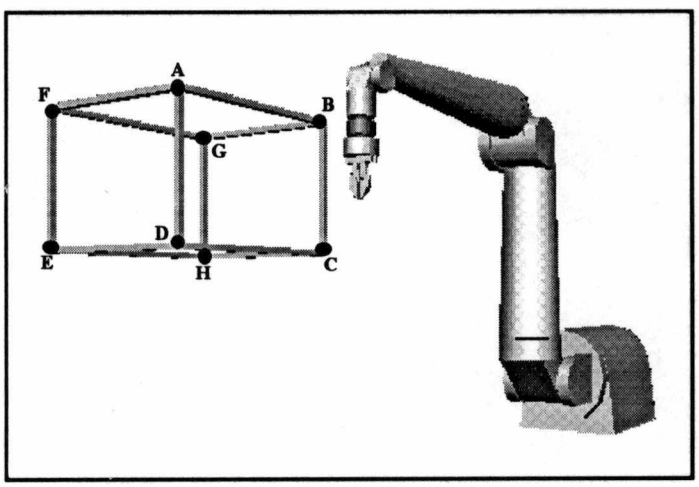


Figure C.2 Rectangular Cube in the Manipulator Workspace

Tables C.2.a lists the position of the vertices for the cube in the base frame of the manipulator. Table C.2.b lists the link lengths and their orientations in the same reference frame.

Table C.2.a Location of Cube Vertices

Vertex	X (cms.)	Y (cms.)	Z (cms.)
A	42.0	148.0	92.0
B	42.0	100.0	92..0
C	42.0	100.0	52.0
D	42.0	148.0	52.0
E	104.0	148.0	52.0
F	104.0	148.0	92.0
G	104.0	100.0	92.0
H	104.0	100.0	52.0

Table C.2.b Link Lengths and Orientations for Cube

Link	Length (cms.)	Orientation in base frame
AB	48.0	Perpendicular to XZ plane
BC	40.0	Perpendicular to XY plane
CD	48.0	Perpendicular to XZ plane
DA	40.0	Perpendicular to XY plane
EF	40.0	Perpendicular to XY plane
FG	48.0	Perpendicular to XZ plane
GH	40.0	Perpendicular to XY plane
HE	48.0	Perpendicular to XZ plane
AF	62.0	Perpendicular to YZ plane
DE	62.0	Perpendicular to YZ plane
BG	62.0	Perpendicular to YZ plane
CH	62.0	Perpendicular to YZ plane

Data for Hexagonal Tube :

Figure C.3 denotes the nomenclature and position of the hexagonal tube in the manipulator workspace. As in the previous two cases the black dots represent the vertices.

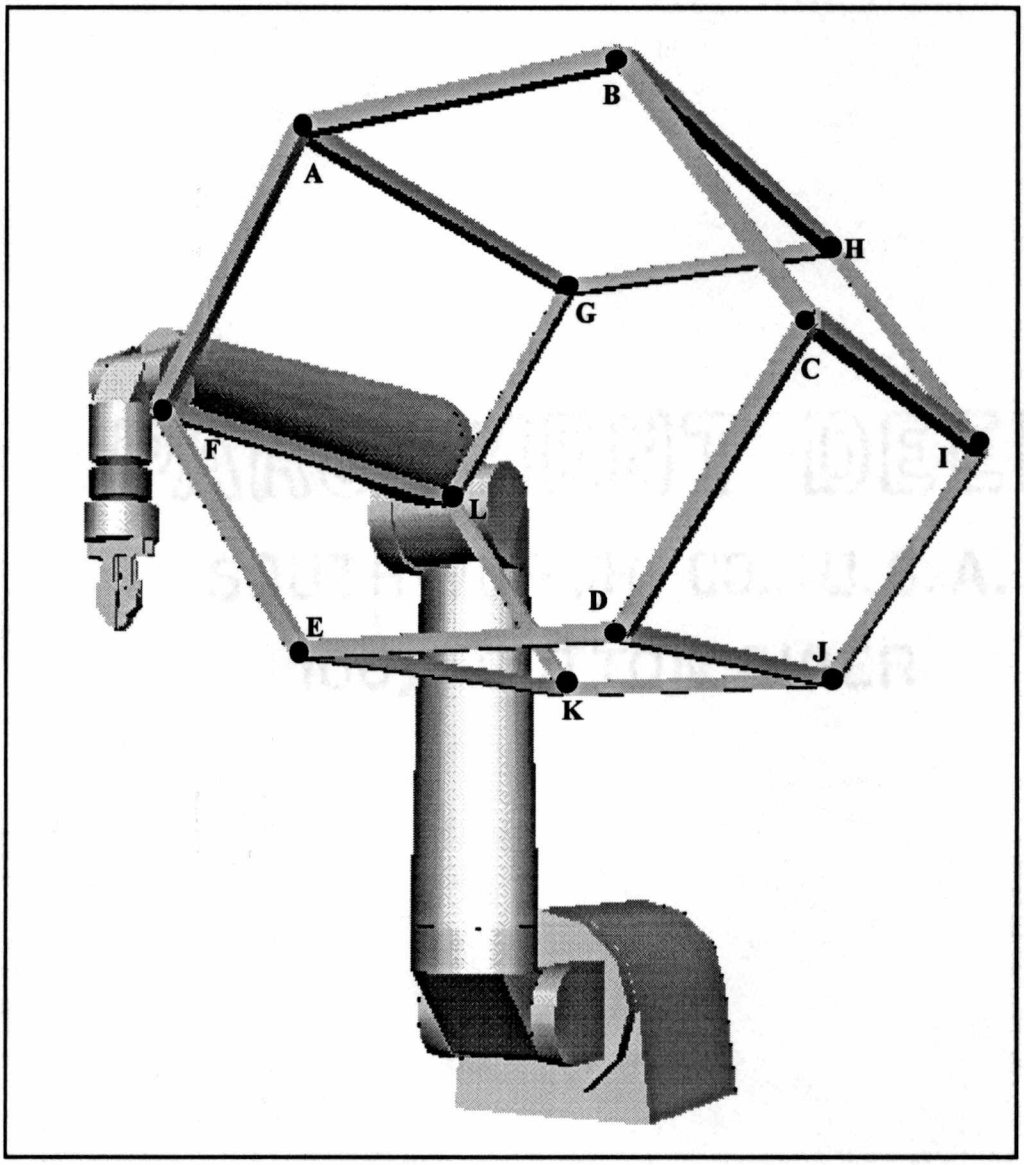


Figure C.3 Hexagonal Tube Position and Orientation in Manipulator Workspace

Tables C.3.a lists the position of the vertices for the hexagonal tube in the base frame of the manipulator.

Table C.3.a Location of Hexagonal Tube Vertices

Vertex	X (cms.)	Y (cms.)	Z (cms.)
A	-52.0	160.0	118.0
B	-92.0	160.0	118.0
C	-112.0	160.0	88.0
D	-92.0	160.0	58.0
E	-52.0	160.0	58.0
F	-32.0	160.0	88.0
G	-52.0	85.0	118.0
H	-92.0	85.0	118.0
I	-112.0	85.0	88.0
J	-92.0	85.0	58.0
K	-52.0	85.0	58.0
L	-32.0	85.0	88.0

For the sake of experiments and simplicity of calculations all the collision objects have been kept as simple as possible. The link lengths are integer values and the objects are symmetric about the three perpendicular planes passing through the center of gravity of the objects except in case of the hexagonal tube where 8 links are oriented at an angle to the plane and have whole numbers as link lengths. It can be seen that the orientations of the links in all three objects are perpendicular to one of the three coordinate planes. Finally, link lengths and their orientations for the hexagonal tube in the manipulator base reference frame are listed in table C.3.b.

Table C.3.b Link Lengths and Orientations for Hexagonal Tube

Link	Length (cms.)	Orientation in base frame
AB	40.0	Perpendicular to YZ plane
BC	36.0555	Perpendicular to Y axis
CD	36.0555	Perpendicular to Y axis
DE	40.0	Perpendicular to YZ plane
EF	36.0555	Perpendicular to Y axis
FA	36.0555	Perpendicular to Y axis
GH	40.0	Perpendicular to YZ plane
HI	36.0555	Perpendicular to Y axis
IJ	36.0555	Perpendicular to Y axis
JK	40.0	Perpendicular to YZ plane
KL	36.0555	Perpendicular to Y axis
LG	36.0555	Perpendicular to Y axis
AG	75.0	Perpendicular to XZ plane
FL	75.0	Perpendicular to XZ plane
EK	75.0	Perpendicular to XZ plane
DJ	75.0	Perpendicular to XZ plane
CI	75.0	Perpendicular to XZ plane
BH	75.0	Perpendicular to XZ plane

The gain factors and starting and final configurations of the manipulator for the different experiments are given in the respective chapters.

Appendix D

Computer Code

The following are the subroutines added to the robot control code running on the Silicon Graphics workstation.

a) Option button code added to file main_pro.c

```
/* Main Menu of Options */
            okay=0;
            while ( okay==0)
                  {
           view_butn4();
           draw_robot(theta_rd);
           get_case(choice);
           /* Option to type in choice */
              /*system("clear");
          tpon();
               printf("AVAILABLE OPTIONS: \n\n");

          printf("  [ 1 ] Change the gain factors.                    \n");
          printf("  [ 2 ] Select a control scheme.                    \n");
          printf("  [ 3 ] Change the turning radius.                  \n");
          printf("  [ 4 ] Change the parameter for 3-D plot.          \n");
          printf("  [ 5 ] Display End-Effector path.                  \n");
          printf("  [ 6 ] Allow repeat of trajectory.                 \n");
          printf("  [ 7 ] Change the color defaults.                  \n");
          printf("  [ 8 ] Select a manipulator.                       \n");
          printf("  [ 9 ] Change swap graphics time.                  \n");
          printf("  [ 10 ] Set joint and vel. limit.                  \n");
          printf("  [ 11 ] Using hand controller.                     \n");
           printf("  [ 12 ] Configure collision avoidance.            \n");
           printf("  [ 13 ] Configure to optimize multiple criteria.  \n");
          printf("  [ 14 ] Exit the 'OPTIONS'.\n");

             printf("\n YOUR SELECTION: ");
          pop_cursor();
          fgets(buffer, 10, stdin);
          i=atoi(buffer);*/
          tpon();
          i= choice[0][0];
          okay = choice[1][1];
              switch( i )
               {
         case 1:

                 okay=0;
                      system("clear");
```

```
   printf(" [1] Change the force sensor gain factors\n");
    printf(" [2] Change the joint velocity weights\n");
  fgets(buffer, 10, stdin);                                    /* Get user input */
  j = atoi(buffer);
  if (j == 1)
    {
      printf("The current gain factors are:\n");
     for (k=1; k<=6; k++)
         printf("kf(%d)=%8e\n",k,kf[k]);
      printf("Enter the new gain factors : \n");
     for (k=1; k<=6; k++)
       {
        printf("kf(%d)= ",k);
        gets(buffer);
        if (strlen(buffer) > 0)
           kf[k]=atof(buffer);
       }
    }
  else if (j == 2)
    {
       printf("The current joint velocity weights are:\n");
     for (k=1; k<=7; k++)
          printf("joint_weight(%d)=%8e\n",k,joint_weight[k]);
      printf("Enter the new gain factors : \n");
     for (k=1; k<=7; k++)
       {
        printf("joint_weight(%d)= ",k);
        gets(buffer);
        if (strlen(buffer) > 0)
           joint_weight[k]=atof(buffer);
       }
    }
   tpoff();
  break;

case 2:

       okay=0;
            system("clear");
            printf("[1] Using gradient projection method. \n");
            printf("[2] Using change joint velocity weights method.\n");
  printf("[3] Using combination of GPM & WLN. \n  ");
  j = atoi(fgets(buffer, 20, stdin));

  if (j == 1)
    {
              printf(" Available Control Methods/ Scehemes :          \n");
              printf("[1] Minimize the Norm of Least-Norm Solution.    \n");
               printf("[2] Minimize the Inverse of tne Manipulability.  \n");
              printf("[3] Minimize the Inverse of the Minimum S. V. (a) \n");
              printf("[4] Minimize the Inverse of the Minimum S. V. (b) \n");
             printf("[5] Joint Limit Avoidance. (a)                \n");
             printf("[6] Joint Limit Avoidance. (b)                \n");
             printf("[7] Collision Avoidance                       \n");
              printf("[8] Simultaneously Optimize Multiple Criteria    \n");

              printf(" All other keys give least norm & no optimization \n\n");
             printf(" Your choice: ");
     scheme_index = atoi(fgets(buffer, 20, stdin));

    for (j=1; j<=8; j++)
       joint_weight[j] = 1;
```

```
        if ((scheme_index < 1) || (scheme_index > 8))
          scheme_index = 0;
        else
          {
                    printf("\n\n Gain factor: ");
          fgets(buffer, 20, stdin);
          scheme_gain = atof(buffer);
          }
        }
      else if (j == 2)
        {
                  printf(" Available Control Methods/ Scehemes :          \n");
                   printf("[1] Joint Limit Avoidance. \n");
                   printf("[2] Joint velocity Limit Avoidance. \n");
         printf("[3] Collision Avoidance. \n");
         scheme_index = 8 + atoi(fgets(buffer, 20, stdin));
        if (scheme_index == 9)
           jlimit_weights(theta_rd);


         else if ((scheme_index < 9) || (scheme_index > 11))
           scheme_index = 0;




        }
      else if (j == 3)
      {
        printf(" Available Control Methods/ Scehemes :          \n");
        printf("[1] Combination of WLN & GPM for Joint Limit Avoidance. \n");
        printf("[2] Combination of WLN & GPM for  Collision Avoidance. \n");
         printf("[3] SR inverse for Collision Avoidance(with gpm).\n");
         printf("[4] Pseudo-inverse for collision avoidance(with gpm). \n");
         printf("[5] SR inverse for Collision Avoidance(without gpm).\n");
         printf("[6] Pseudo-inverse for collision avoidance(without gpm). \n");
         printf("[7] Pseudo-inverse + SR-inverse(GPM) collision avoidance.\n");
         printf("[8] Pseudo_inverse for Joint Limit Avoidance(with gpm).\n");

         scheme_index = 11 + atoi(fgets(buffer, 20, stdin));



        if ((scheme_index < 12) || (scheme_index > 19))
           scheme_index = 0;


        if (((scheme_index >= 12) && (scheme_index <=15)) || (scheme_index ==
18) || (scheme_index == 19))
      {
         printf("\n\n Gain factor: ");
           fgets(buffer, 20, stdin);
           scheme_gain = atof(buffer);

      }
        if ((scheme_index == 14) || (scheme_index == 16))
        {
          printf(" Please give the vector srk");
          for (j=1; j<=6; j++)
          {
            printf("\nsrk[%d]=",j);
            fgets(buffer,20,stdin);
```

```
            srk[j] = atof(buffer);
            srk[j] = sqrt(srk[j]);
          }
           printf(" \nPlease give the gain for joint velocities:\n");
            printf("\ngains =");
            fgets(buffer,20,stdin);
            gains = atof(buffer);
            printf(" Please give the vector srk1");
          for (j=1; j<=7; j++)
          {
            printf("\nsrk1[%d]=",j);
            fgets(buffer,20,stdin);
            srk1[j] = atof(buffer);
          }
      }
    }
     if (( scheme_index == 7 ) || (scheme_index >= 12))
    {
     for (i=0; i<=10; i++)
      tp_thet[i] = theta_rd[i];
      jacobian(tp_thet, h_transf, jc,0);
     o_motion(jc, dthet);
     new_theta(tp_thet, dthet);
     for (i=0; i<=10; i++)
     theta_rd[i] = tp_thet[i];
     draw_robot(theta_rd);
    }
     tpoff();
   break;


case 3:

          okay=0;
                system("clear");
      printf("Enter the trajectory blending radius: ");
    fgets(buffer, 20, stdin);
   temp1 = atof(buffer);
   if (temp1 > 0)
      r = temp1;
    tpoff();
   break;

case 4:

          okay=0;
                system("clear");
     printf(" [ 1 ]  Change the reference joint.\n");
     printf(" [ 2 ]  Record the joint velocity norm\n ");
     printf(" [ 3 ]  Record the maximum joint velocity\n ");
    fgets(buffer, 20, stdin);
   if (buffer[0] == '1')
     {
       printf("Enter the reference joint number: ");
      fgets(buffer, 20, stdin);
     j = atoi(buffer);
     if ((j > 0) && (j<8))
        ref_joint = j;
     }
   else if (buffer[0] == '2')
     {
       printf("Record the norm for 3-D ploat (y/n): ");
      fgets(buffer, 20, stdin);
```

```
      if ((buffer[0] == 'y') || (buffer[0] == 'Y'))
          record_criterion = 1;
      else
          record_criterion = 0;
      }
    else if (buffer[0] == '3')
      {
         printf("Record the maximum joint velocity (y/n): ");
       fgets(buffer, 20, stdin);
       if ((buffer[0] == 'y') || (buffer[0] == 'Y'))
          record_criterion = 2;
      else
          record_criterion = 0;
      }
       tpoff();
   break;

case 5:

          okay=0;
                system("clear");
     printf("Show end-effector path (y/n): ");
    fgets(buffer, 20, stdin);
    if ((buffer[0]=='y') || (buffer[0]=='Y'))
      {
       e_traj_count= -1;
       show_path=1;
      }
   else
       show_path = 0;
      tpoff();
   break;

case 6:


          okay=0;
                system("clear");
      printf("Repeat trajectory continually (y/n)? ");
    fgets(buffer, 20, stdin);
    if ((buffer[0]=='y') || (buffer[0]=='Y'))
       repeat=1;
   else
       repeat=0;
      tpoff();
   break;

case 7:

          okay=0;
                system("clear");
                 printf("This option allows online color updates. \n");
                printf("To change setting -> type in new value.  \n");
               printf("To skip setting   -> type RETURN         \n");
                printf("\n\n");
     printf("Background color=%x\n", color_back);
   gets(buffer);
   i = strlen(buffer);
  if (i > 0)
       color_back=strtol(buffer,(char **) NULL, 16);
     printf("Sub_window background color=%x\n", color_back2);
   gets(buffer);
   i = strlen(buffer);
```

```
if (i > 0)
    color_back2=strtol(buffer, (char **) NULL, 16);
  printf("output data color=%x\n", color_data);
gets(buffer);
i = strlen(buffer);
if (i > 0)
    color_data=strtol(buffer, (char **) NULL, 16);
   printf("teach pendant button color=%x\n", color_button);
gets(buffer);
i = strlen(buffer);
if (i > 0)
     color_button=strtol(buffer, (char **) NULL, 16);
  printf("title color=%x\n", color_title);
gets(buffer);
i = strlen(buffer);
if (i > 0)
    color_title=strtol(buffer, (char **) NULL, 16);
  printf("color_bt_char=%x\n", color_bt_char);
gets(buffer);
i = strlen(buffer);
if (i > 0)
    color_bt_char=strtol(buffer, (char **) NULL, 16);
 printf("frame color=%x\n", color_frame);
gets(buffer);
i = strlen(buffer);
if (i > 0)
    color_frame=strtol(buffer, (char **) NULL, 16);
fullscrn();
init_rob_win();
draw_robot(theta_rd);
drawing = 1;
tpoff();
break;

case 8:

      okay=0;
   system("clear");
    printf("Available Manipulators in Library: \n");
            printf("    [ 1 ] Robotic's Research k-2107 \n");
            printf("    [ 2 ] Laboratory Telerobotic Manipulator (LTM)\n");
            printf("\n\n Your selection: ");

   fgets(buffer, 20, stdin);
  i = atoi(buffer);

 if ((i != robot_index) && ( i >=1) && (i<=3))
  {
    save_parameters(robot_index);
   robot_index = i;
    select_robot(robot_index);
   draw_robot(theta_rd);

   transf(theta_rd, h_transf);
                for ( j=1; j<=3; j++ )        trajectory[0][(j-1)]=h_transf[j][4];
   e_traj_count = 0;
    if ((robot_index == 1) && (robot_index == 3))
     {
       z_axis[2] = y_axis[1] = x_axis[0] = 125;
     }
   else if (robot_index == 2)
       z_axis[2] = y_axis[1] = x_axis[0] = 65;
  }
```

```
 fm_print();
 draw_robot(theta_rd);
 drawing = 1;
 tpoff();

 tpon();
 system("clear");
  printf("Open or close files: \n");
                printf("    [ 1 ] Open new file. \n");
                printf("    [ 2 ] Close file.\n");
                 printf("\n\n Your selection: ");
  i =0;
   while((i<1) || (i>2))
  {
    printf("Choose [1] or [2]\n");
    fgets(buffer, 20, stdin);
   i = atoi(buffer);
  }

 if (i== 1)
{
 stor = 1;
    printf("Give data file for storing joint angle, position and orientation data.  :\n");
 gets(buffer);
  if (0==(fil_name[1]=fopen(buffer,"a")))
  printf("Can't open the file!\n");
}
 else if ( i== 2)
{
  stor = 0;
  tim = 0.0;
   fclose(fil_name[1]);
 }

 tpoff();
 ringbell();
 break;

case 9:
    printf("swap graphics interval=%d\n The new value: ", pic_delay);
  fgets(buffer, 20, stdin);
  if (buffer[0] >= 48)
    {
     j = atoi(buffer);
     if ((j > 0) && (j != pic_delay))
       {
         temp1 = pic_delay;
        temp2 = j;
         temp2 /= temp1;
         joint_v *= temp2;
          linear_v_limit *= temp2;
         djoint_v *= temp2;
         dlinear_v *= temp2;
        pic_delay = j;
         swapinterval(pic_delay);
       }
    }
  tpoff();
  break;
case 10:
        okay=0;
              system("clear");
   printf("[ 1 ]  Set joint limit.\n");
```

```
    printf("[ 2 ]  Set joint velocity limit.\n");
    printf("[ 3 ]  Set acceleration limit.\n");
   gets(buffer);
   if (buffer[0] == '1')
     {
       printf("Set joint limit (y/n)? ");
      gets(buffer);
      if ((buffer[0]=='y') || (buffer[0]=='Y'))
          joint_limit_f = 1;
     else
          joint_limit_f = 0;
     }
   else if (buffer[0] == '2')
     {
       printf("Set joint velocity limit (y/n)? ");
      gets(buffer);
      if ((buffer[0]=='y') || (buffer[0]=='Y'))
          joint_vlimit_f = 1;
     else
          joint_vlimit_f = 0;
     }
   else if (buffer[0] == '3')
     {
       printf("The acceleration factor is : %f\n", linear_a);
       printf("Enter the new value (must > 0): ");
      gets(buffer);
      linear_a = atof(buffer);
     }
  tpoff();
  break;

 case 11:

        okay=0;
    master_switch = talk_to_master();
  tpoff();
  break;

 case 12:
  okay = FALSE;    /* Set exit flag */
              configure_collision_avoidance();
  tpoff();
  break;

 case 13:
  okay = FALSE;    /* Set exit flag */
              configure_multiple_criteria();
  tpoff();
  break;

 case 14:
  okay = 1;
      }
    }

      /* Finishing Options */
tpoff();
ringbell();
 update_bit3_ctl_variable();
sleep(1);
if (control_robot == 1)
 {
  real_time_switch(1);
```

```
            reset_time_delay();
          }
        if (master_switch == 1)
          {
           command_master();
           drawing = 0;
          }

        else
          {
           draw_robot(theta_rd);
           draw_robot(theta_rd);
           drawing = 1;
          }
    }
```

b) k_inv( ) subroutine added to file goalp.c

```
k_inv(draw, scheme_gain, theta_rd, vels, origin)
int draw;
float *theta_rd, *vels, scheme_gain, *origin;
{
    extern FILE *joint_vel[MAT_DIM], *joint_th[MAT_DIM];
    extern int record_jt;
    extern struct obtctl;
    extern float jv_lim[MAT_DIM];
    extern float dth_lim[MAT_DIM], th_lim1[MAT_DIM], th_lim2[MAT_DIM],      joint_weight[MAT_DIM];
     extern float joint_weight_old[MAT_DIM];
     extern float total_cost, path_distance, h_proj_mat[MAT_DIM][MAT_DIM];
    extern float control_path[TSTEPS][3], h_transf[5][5], jaco[MAT_DIM][MAT_DIM];
     extern int steps, steps1, ref_joint, scheme_index, win_state, record_criterion;
     extern int robot_index, scheme_index, servo_ctl_mode;
    extern float f_vector[13][8];
    extern int j_ind,jj;
    float scheme_gain1;
     extern float gradat;

    int okay_to_move;
     float copy_of_orginal_angles[MAT_DIM];


    static float sigma6;
     int i, j, joint_v_lim;
   float tp[MAT_DIM], dp1[MAT_DIM], dp2[MAT_DIM], dp3[MAT_DIM], er[MAT_DIM];
    float  norm2[MAT_DIM], norm3, norm_k[MAT_DIM], nk_max, x0, x1, y0, y1, x, kk, norm1;
    float jsqr[MAT_DIM][MAT_DIM], temp[8], eigen[MAT_DIM], position[8];
   float dtheta_h[MAT_DIM][MAT_DIM], h[5][5], jc[MAT_DIM][MAT_DIM];
     float  vect11[4], vect12[4], vect21[4], vect22[4], link_p1[4], link_p2[4];
    static float dth[MAT_DIM];
   float tp1[MAT_DIM],tp2[MAT_DIM],tp3[MAT_DIM],tp4[MAT_DIM],tp5[MAT_DIM],tp6[MAT_DIM];
   float h1[MAT_DIM],h2[MAT_DIM],h3[MAT_DIM],h4[MAT_DIM],h5[MAT_DIM],
h6[MAT_DIM];
   float d1[MAT_DIM][MAT_DIM],d2[MAT_DIM][MAT_DIM];
    float gr1,gr2,gr3,gr4,gr5,gr6,gr;
    float theta_alt[MAT_DIM],dp11[MAT_DIM];
```

```
int copy_size;
float dpm[4][MAT_DIM], gradients[MAT_DIM],gradients1[MAT_DIM], tp_t[MAT_DIM],
    vels_t[MAT_DIM];
char buffer[10];
float costing;

scheme_gain1 =  (scheme_gain/20.0);

 memcpy( &copy_of_orginal_angles[0], &theta_rd[0], 32);
if (servo_ctl_mode == 1)
  {
    bit3_end_effector_pos(origin);
  if (draw == 1)
      draw_robot(theta_rd);
   if (win_state==1)
     {
      steps1=steps;
     draw_curv();
     }
  return;
  }

if (steps >= TSTEPS )
  {
    control_path[0][0] = control_path[(steps - 1)][0];
    control_path[0][1] = control_path[(steps - 1)][1];
    control_path[0][2] = control_path[(steps - 1)][2];
   steps=1;
  }
 control_path[steps][0] = path_distance*10;
  control_path[steps][1]=theta_rd[ref_joint]/0.017453293;
 control_path[steps][2] = 0;
for (i=1; i<=T_JOINT; i++)
    tp[i]=theta_rd[i];
 jacobian(theta_rd,h,jaco,1);
 if ((scheme_index == 0) || ((scheme_index >=9) && (scheme_index <=11)) || (scheme_index >= 14))
  {
   if (steps == 1)
       for (i=1; i<=7; i++)
          dth[i] = 0;
  if (scheme_index == 9)
     {
       jlimit_weights(theta_rd);
     }
   else if (scheme_index == 10)
     {
      for (i=1; i<=7; i++)
         joint_weight[i] = 1;
       least_norm(jaco, vels, dp1);
      for (i=1; i<=7; i++)
        {
          temp[0] = fabs(dp1[i]) - fabs(jv_lim[i]);
         if (temp[0] > 0)
           {
              joint_weight[i] = temp[0]/jv_lim[i];
           }
         else
             joint_weight[i] = 1e-30;
        }
     }
    else if (( scheme_index == 11)||(scheme_index == 14) || (scheme_index == 15))
     {
      for ( i=1; i<=7; i++)
```

```
    {
      tp_t[i] = 0.0017453293 ;
     }
        evaluate_gradient(1, 11, theta_rd, tp_t, h1, gradients);
      gradat=0.0;
      for (i=1; i<=7; i++)
       gradat += (gradients[i]*gradients[i]);
      gradat = sqrt(gradat);
       collavd_weights(gradients);
        collision_cost(theta_rd, &costing);
   }
 else if (( scheme_index == 16)||(scheme_index == 17))
    {
     for ( i=1; i<=7; i++)
      gradients[i] = 0.0;
       gradat=0.0;
    }
else if (scheme_index == 19)
   {
     for ( i=1; i<=7; i++)
    {
      tp_t[i] = 0.0017453293 ;
     }
        evaluate_gradient(1, 5, theta_rd, tp_t, h1, gradients);
    }
  if (scheme_index <= 11)
{
  least_norm71(jaco, vels, dp1);
 new_theta(tp, dp1);
  difference(tp, h_transf, origin, temp);
 jacobian(tp, h, jc,1);
  least_norm71(jc, temp, dp2);
 new_theta(tp, dp2);
 new_theta(dp1, dp2);
}
 else if ((scheme_index == 14) || (scheme_index == 16))
{
   least_normsr(jaco, vels,gradients,costing, dp1);
 new_theta(tp, dp1);
  difference(tp, h_transf, origin, temp);
 jacobian(tp, h, jc,1);
   least_normsr(jc, temp,gradients,costing, dp2);
 new_theta(tp, dp2);
 new_theta(dp1, dp2);
}
 else if ((scheme_index == 15) || (scheme_index == 17) || (scheme_index == 19))
{
   least_norm72(jaco, vels,gradients, dp1);
 new_theta(tp, dp1);
  difference(tp, h_transf, origin, temp);
 jacobian(tp, h, jc,1);
  least_norm72(jc, temp,gradients, dp2);
 new_theta(tp, dp2);
 new_theta(dp1, dp2);
}
else if (scheme_index == 18)
{
 for (i=1; i<=7; i++)
     dp1[i] = 0.0;
}
 /*if ( jj == 1)
  least_norm1(jaco, vels, d1);
 else if ( jj == 2)
```

```
    least_norm2(jaco, vels, d1);
   else if ( jj == 3)
    least_norm3(jaco, vels, d1);
   else if ( jj == 4)
    least_norm4(jaco, vels, d1);
   else if ( jj == 5)
    least_norm5(jaco, vels, d1);*/
    /* least_norm71(jaco,vels,dp1);*/

   /*for (i=1; i<=7; i++)
   dp1[i] = d1[i][7];*/
   /*new_theta(tp, dp1);
   difference(tp, h_transf, origin, temp);
   jacobian(tp,h,jc, 1);*/

   /*if ( jj == 1 )
    least_norm1(jc, temp, d2);
   else if ( jj == 2)
    least_norm2(jc, temp, d2);
   else if ( jj == 3)
    least_norm3(jc, temp, d2);
   else if ( jj == 4)
    least_norm4(jc, temp, d2);
   else if ( jj == 5)
    least_norm5(jc, temp, d2);*/
     /*least_norm71(jaco,temp,dp2);*/
  /*  for (i=1; i<=7; i++)
    dp2[i] = d2[i][7];*/
    /* new_theta(tp, dp2);*/

        get_j_lim(&norm1, theta_rd);

    for (i=1; i<=T_JOINT; i++)
      {
       theta_rd[i]+=dp1[i];
       dth[i] = dp1[i];
         control_path[steps][2]+=dp1[i]*dp1[i];
      }
/* ------------------------------------ */
 /*  If collision avoidance is active, check    */
 /*  for a collision before sending next angles */
/* ------------------------------------ */

 if ( collision.is_active )
 {
    check_for_collision( theta_rd, robot_index, &okay_to_move );
  if ( !okay_to_move )
   {
     /* In trouble, restore orginal angles */
     memcpy( &theta_rd[0], &copy_of_orginal_angles[0],  32);
   }
 }
    if (record_criterion != 2)
          control_path[steps][2]=sqrt(control_path[steps][2])*1145.9156;
   else
      {
        control_path[steps][2] = 0;
       for (i=1; i<=7; i++)
            if (fabs(control_path[steps][2]) < fabs(dp1[i]))
                control_path[steps][2] = fabs(dp1[i])*1145.9156*3;
      }
     total_cost += control_path[steps][2];
   if (draw == 1)
```

```
            draw_robot(theta_rd);
        if (win_state==1)
          {
           steps1=steps;
          draw_curv();
          }
        return;
        }
else
{
   if ( scheme_index == 12 )
   {
    jlimit_weights(theta_rd);
   }
    else if (scheme_index == 13 )
   {
     for ( i=1; i<=7; i++)
          {
            tp_t[i] = 0.0017453293 ;
          }
       evaluate_gradient(1, 11, theta_rd, tp_t, h1, gradients);
       collavd_weights(gradients);
   }
    else if (( scheme_index == 5 ) || ( scheme_index == 6 ) || ( scheme_index == 7 ))
   {
    for (i=1; i<=7; i++)
    joint_weight[i] = 1.0;
   }
    if ( jj == 1)
     least_norm1(jaco, vels, d1);
    else if ( jj == 2)
     least_norm2(jaco, vels, d1);
    else if ( jj == 3)
     least_norm3(jaco, vels, d1);
    else if ( jj == 4)
     least_norm4(jaco, vels, d1);
    else if ( jj == 5)
     least_norm5(jaco, vels, d1);

    for ( i=1; i<=7; i++)
{
dp1[i] = d1[i][7];
h1[i] = 0.01*d1[i][1];
h2[i] = 0.01*d1[i][2];
h3[i] = 0.01*d1[i][3];
h4[i] = 0.01*d1[i][4];
h5[i] = 0.01*d1[i][5];
h6[i] = 0.01*d1[i][6];
dp2[i] = d1[i][8];
}

for (i=1; i<=7; i++)
    tp1[i] = tp2[i] = tp3[i] = tp4[i] = tp5[i] = tp6[i] =tp[i];
     new_theta(tp1, h1);
     new_theta(tp2, h2);
     new_theta(tp3, h3);
     new_theta(tp4, h4);
     new_theta(tp5, h5);
     new_theta(tp6, h6);

gr1 = gr2 = gr3 = gr4 = gr5 = gr6 = 0.0;
if (( scheme_index == 5 ) || ( scheme_index == 6 ) || ( scheme_index == 7 ) || (scheme_index == 12) ||
(scheme_index == 13))
```

```
{
  if ( jj == 1)
  {
   norm2[1] = 0.0;
   for (i=1; i<=T_JOINT; i++)
       norm2[1] += h1[i]*h1[i];

       copy_size = (T_JOINT+1)*sizeof(float);
      memcpy( &tp_t[0], &tp1[0], copy_size);
       evaluate_gradient(0, scheme_index, theta_rd, tp_t, h1, gradients);
       norm_k[1] = gradients[0]*scheme_gain1;
       gr1 = gradients[0]*sqrt(norm2[1])/scheme_gain1;

    norm2[2] = 0.0;
   for (i=1; i<=T_JOINT; i++)
       norm2[2] += h2[i]*h2[i];

       copy_size = (T_JOINT+1)*sizeof(float);
      memcpy( &tp_t[0], &tp2[0], copy_size);
       evaluate_gradient(0, scheme_index, theta_rd, tp_t, h2, gradients);
       norm_k[2] = gradients[0]*scheme_gain1;
         gr2 = gradients[0]*sqrt(norm2[2])/scheme_gain1;

    norm2[3] = 0.0;
   for (i=1; i<=T_JOINT; i++)
       norm2[3] += h3[i]*h3[i];

       copy_size = (T_JOINT+1)*sizeof(float);
      memcpy( &tp_t[0], &tp3[0], copy_size);
       evaluate_gradient(0, scheme_index, theta_rd, tp_t, h3, gradients);
       norm_k[3] = gradients[0]*scheme_gain1;
        gr3 = gradients[0]*sqrt(norm2[3])/scheme_gain1;

    norm2[4] = 0.0;
   for (i=1; i<=T_JOINT; i++)
       norm2[4] += h4[i]*h4[i];

       copy_size = (T_JOINT+1)*sizeof(float);
      memcpy( &tp_t[0], &tp4[0], copy_size);
       evaluate_gradient(0, scheme_index, theta_rd, tp_t, h4, gradients);
       norm_k[4] = gradients[0]*scheme_gain1;
         gr4 = gradients[0]*sqrt(norm2[4])/scheme_gain1;

   norm2[5] = 0.0;
   for (i=1; i<=T_JOINT; i++)
       norm2[5] += h5[i]*h5[i];

       copy_size = (T_JOINT+1)*sizeof(float);
      memcpy( &tp_t[0], &tp5[0], copy_size);
       evaluate_gradient(0, scheme_index, theta_rd, tp_t, h5, gradients);
       norm_k[5] = gradients[0]*scheme_gain1;
        gr5 = gradients[0]*sqrt(norm2[5])/scheme_gain1;

    norm2[6] = 0.0;
   for (i=1; i<=T_JOINT; i++)
       norm2[6] += h6[i]*h6[i];

       copy_size = (T_JOINT+1)*sizeof(float);
      memcpy( &tp_t[0], &tp6[0], copy_size);
       evaluate_gradient(0, scheme_index, theta_rd, tp_t, h6, gradients);
       norm_k[6] = gradients[0]*scheme_gain1;
         gr6 = gradients[0]*sqrt(norm2[6])/scheme_gain1;
```

```
    }

else if ( jj == 2)
{
 norm2[1] = 0.0;
   for (i=1; i<=T_JOINT; i++)
       norm2[1] += h1[i]*h1[i];

       copy_size = (T_JOINT+1)*sizeof(float);
      memcpy( &tp_t[0], &tp1[0], copy_size);
       evaluate_gradient(0, scheme_index, theta_rd, tp_t, h1, gradients);
       norm_k[1] = gradients[0]*scheme_gain1;
       gr1 = gradients[0]*sqrt(norm2[1])/scheme_gain1;

    norm2[2] = 0.0;
   for (i=1; i<=T_JOINT; i++)
       norm2[2] += h2[i]*h2[i];

       copy_size = (T_JOINT+1)*sizeof(float);
      memcpy( &tp_t[0], &tp2[0], copy_size);
       evaluate_gradient(0, scheme_index, theta_rd, tp_t, h2, gradients);
       norm_k[2] = gradients[0]*scheme_gain1;
         gr2 = gradients[0]*sqrt(norm2[2])/scheme_gain1;

    norm2[3] = 0.0;
   for (i=1; i<=T_JOINT; i++)
       norm2[3] += h3[i]*h3[i];

       copy_size = (T_JOINT+1)*sizeof(float);
      memcpy( &tp_t[0], &tp3[0], copy_size);
       evaluate_gradient(0, scheme_index, theta_rd, tp_t, h3, gradients);
       norm_k[3] = gradients[0]*scheme_gain1;
        gr3 = gradients[0]*sqrt(norm2[3])/scheme_gain1;

    norm2[4] = 0.0;
   for (i=1; i<=T_JOINT; i++)
       norm2[4] += h4[i]*h4[i];

       copy_size = (T_JOINT+1)*sizeof(float);
      memcpy( &tp_t[0], &tp4[0], copy_size);
       evaluate_gradient(0, scheme_index, theta_rd, tp_t, h4, gradients);
       norm_k[4] = gradients[0]*scheme_gain1;
         gr4 = gradients[0]*sqrt(norm2[4])/scheme_gain1;

   norm2[5] = 0.0;
   for (i=1; i<=T_JOINT; i++)
       norm2[5] += h5[i]*h5[i];

       copy_size = (T_JOINT+1)*sizeof(float);
      memcpy( &tp_t[0], &tp5[0], copy_size);
       evaluate_gradient(0, scheme_index, theta_rd, tp_t, h5, gradients);
       norm_k[5] = gradients[0]*scheme_gain1;
        gr5 = gradients[0]*sqrt(norm2[5])/scheme_gain1;

norm_k[6] = 0.0;
}
else if ( jj == 3)
{
 norm2[1] = 0.0;
   for (i=1; i<=T_JOINT; i++)
       norm2[1] += h1[i]*h1[i];

      copy_size = (T_JOINT+1)*sizeof(float);
```

```
        memcpy( &tp_t[0], &tp1[0], copy_size);
        evaluate_gradient(0, scheme_index, theta_rd, tp_t, h1, gradients);
        norm_k[1] = gradients[0]*scheme_gain1;
        gr1 = gradients[0]*sqrt(norm2[1])/scheme_gain1;

      norm2[2] = 0.0;
     for (i=1; i<=T_JOINT; i++)
        norm2[2] += h2[i]*h2[i];

        copy_size = (T_JOINT+1)*sizeof(float);
       memcpy( &tp_t[0], &tp2[0], copy_size);
        evaluate_gradient(0, scheme_index, theta_rd, tp_t, h2, gradients);
        norm_k[2] = gradients[0]*scheme_gain1;
          gr2 = gradients[0]*sqrt(norm2[2])/scheme_gain1;

      norm2[3] = 0.0;
     for (i=1; i<=T_JOINT; i++)
        norm2[3] += h3[i]*h3[i];

        copy_size = (T_JOINT+1)*sizeof(float);
       memcpy( &tp_t[0], &tp3[0], copy_size);
        evaluate_gradient(0, scheme_index, theta_rd, tp_t, h3, gradients);
        norm_k[3] = gradients[0]*scheme_gain1;
         gr3 = gradients[0]*sqrt(norm2[3])/scheme_gain1;

      norm2[4] = 0.0;
     for (i=1; i<=T_JOINT; i++)
        norm2[4] += h4[i]*h4[i];

        copy_size = (T_JOINT+1)*sizeof(float);
       memcpy( &tp_t[0], &tp4[0], copy_size);
        evaluate_gradient(0, scheme_index, theta_rd, tp_t, h4, gradients);
        norm_k[4] = gradients[0]*scheme_gain1;
          gr4 = gradients[0]*sqrt(norm2[4])/scheme_gain1;

norm_k[5]= 0.0;
norm_k[6] = 0.0;
}
else if ( jj == 4)
{
 norm2[1] = 0.0;
    for (i=1; i<=T_JOINT; i++)
        norm2[1] += h1[i]*h1[i];

        copy_size = (T_JOINT+1)*sizeof(float);
       memcpy( &tp_t[0], &tp1[0], copy_size);
        evaluate_gradient(0, scheme_index, theta_rd, tp_t, h1, gradients);
        norm_k[1] = gradients[0]*scheme_gain1;
        gr1 = gradients[0]*sqrt(norm2[1])/scheme_gain1;


      norm2[2] = 0.0;
     for (i=1; i<=T_JOINT; i++)
        norm2[2] += h2[i]*h2[i];

        copy_size = (T_JOINT+1)*sizeof(float);
       memcpy( &tp_t[0], &tp2[0], copy_size);
        evaluate_gradient(0, scheme_index, theta_rd, tp_t, h2, gradients);
        norm_k[2] = gradients[0]*scheme_gain1;
          gr2 = gradients[0]*sqrt(norm2[2])/scheme_gain1;

      norm2[3] = 0.0;
     for (i=1; i<=T_JOINT; i++)
```

```
        norm2[3] += h3[i]*h3[i];

        copy_size = (T_JOINT+1)*sizeof(float);
       memcpy( &tp_t[0], &tp3[0], copy_size);
        evaluate_gradient(0, scheme_index, theta_rd, tp_t, h3, gradients);
        norm_k[3] = gradients[0]*scheme_gain1;
         gr3 = gradients[0]*sqrt(norm2[3])/scheme_gain1;

norm_k[4]= 0.0;
norm_k[5]=0.0;
norm_k[6] = 0.0;
}
else if ( jj == 5)
{
 norm2[1] = 0.0;
   for (i=1; i<=T_JOINT; i++)
        norm2[1] += h1[i]*h1[i];

       copy_size = (T_JOINT+1)*sizeof(float);
       memcpy( &tp_t[0], &tp1[0], copy_size);
        evaluate_gradient(0, scheme_index, theta_rd, tp_t, h1, gradients);
        norm_k[1] = gradients[0]*scheme_gain1;
        gr1 = gradients[0]*sqrt(norm2[1])/scheme_gain1;

    norm2[2] = 0.0;
   for (i=1; i<=T_JOINT; i++)
        norm2[2] += h2[i]*h2[i];

        copy_size = (T_JOINT+1)*sizeof(float);
       memcpy( &tp_t[0], &tp2[0], copy_size);
        evaluate_gradient(0, scheme_index, theta_rd, tp_t, h2, gradients);
        norm_k[2] = gradients[0]*scheme_gain1;
          gr2 = gradients[0]*sqrt(norm2[2])/scheme_gain1;

norm_k[3] = 0.0;
norm_k[4] = 0.0;
norm_k[5] = 0.0;
norm_k[6] = 0.0;
}
     }
  else if ( scheme_index == 8)
  {
    norm2[1] = 0.0;
   for (i=1; i<=T_JOINT; i++)
        norm2[1] += h1[i]*h1[i];

       copy_size = (T_JOINT+1)*sizeof(float);
       memcpy( &tp_t[0], &tp1[0], copy_size);
        evaluate_gradient(0, scheme_index, theta_rd, tp1, h1, gradients);
        norm_k[1] = gradients[0]*scheme_gain1;
        gr1 = gradients[0]/scheme_gain1;

    norm2[2] = 0.0;
   for (i=1; i<=T_JOINT; i++)
        norm2[2] += h2[i]*h2[i];

        copy_size = (T_JOINT+1)*sizeof(float);
       memcpy( &tp_t[0], &tp2[0], copy_size);
        evaluate_gradient(0, scheme_index, theta_rd, tp2, h2, gradients);
        norm_k[2] = gradients[0]*scheme_gain1;
        gr2 = gradients[0]/scheme_gain1;

    norm2[3] = 0.0;
```

```
    for (i=1; i<=T_JOINT; i++)
        norm2[3] += h3[i]*h3[i];

        copy_size = (T_JOINT+1)*sizeof(float);
       memcpy( &tp_t[0], &tp3[0], copy_size);
        evaluate_gradient(0, scheme_index, theta_rd, tp3, h3, gradients);
        norm_k[3] = gradients[0]*scheme_gain1;
        gr3 = gradients[0]/scheme_gain1;

     norm2[4] = 0.0;
    for (i=1; i<=T_JOINT; i++)
        norm2[4] += h4[i]*h4[i];

        copy_size = (T_JOINT+1)*sizeof(float);
       memcpy( &tp_t[0], &tp4[0], copy_size);
        evaluate_gradient(0, scheme_index, theta_rd, tp4, h4, gradients);
        norm_k[4] = gradients[0]*scheme_gain1;
        gr4 = gradients[0]/scheme_gain1;

    norm2[5] = 0.0;
    for (i=1; i<=T_JOINT; i++)
        norm2[5] += h5[i]*h5[i];

        copy_size = (T_JOINT+1)*sizeof(float);
       memcpy( &tp_t[0], &tp5[0], copy_size);
        evaluate_gradient(0, scheme_index, theta_rd, tp5, h5, gradients);
        norm_k[5] = gradients[0]*scheme_gain1;
        gr5 = gradients[0]/scheme_gain1;

     norm2[6] = 0.0;
    for (i=1; i<=T_JOINT; i++)
        norm2[6] += h6[i]*h6[i];

        copy_size = (T_JOINT+1)*sizeof(float);
       memcpy( &tp_t[0], &tp6[0], copy_size);
        evaluate_gradient(0, scheme_index, theta_rd, tp6, h6, gradients);
        norm_k[6] = gradients[0]*scheme_gain1;
          gr6 = gradients[0]/scheme_gain1;
   }
/* If give the homo. solution a limit */
   for (i=1; i<=7-jj; i++)
{
    norm2[i] = sqrt(norm2[i]);
    if (robot_index == 1)
      {
       if (norm_k[i]>0.1/norm2[i])
          norm_k[i]=0.1/norm2[i];
       else if (norm_k[i]<-0.1/norm2[i])
          norm_k[i]=(-0.1/norm2[i]);
      }
    else if (robot_index == 2)
      {
       if (norm_k[i]>0.2/norm2[i])
          norm_k[i]=0.2/norm2[i];
       else if (norm_k[i]<-0.2/norm2[i])
          norm_k[i]=(-0.2/norm2[i]);
      }
}

    for (i=1; i<=T_JOINT; i++)
     {
```

```
    dp3[i] = dp1[i] -
((h1[i]*norm_k[1])+(h2[i]*norm_k[2])+(h3[i]*norm_k[3])+(h4[i]*norm_k[4])+(h5[i]*norm_k[5])+(h6[i]*norm
_k[6]));
    }
for ( i=1; i<=7; i++)
{
d2[i][1] = h1[i];
d2[i][2] = h2[i];
d2[i][3] = h3[i];
d2[i][4] = h4[i];
d2[i][5] = h5[i];
d2[i][6] = h6[i];
}
/*If using max_joint_v_limit method
      j_v_limit1(dp1, d2, norm_k, dp3);*/

   new_theta(theta_rd, dp3);
   difference(theta_rd, h_transf, origin, er);
  jacobian(theta_rd, h, jc,1);

  if ( jj == 1)
   least_norm1(jc, er, d1);
  else if ( jj == 2)
   least_norm2(jc, er, d1);
  else if ( jj == 3)
   least_norm3(jc, er, d1);
  else if ( jj == 4)
   least_norm4(jc, er, d1);
  else if ( jj == 5)
   least_norm5(jc, er, d1);


  for (i=1; i<=7; i++)
  dp1[i] = d1[i][7];
   new_theta(theta_rd, dp1);
   new_theta(dp3, dp1);

  /* ---------------------------------- */
   /*  If collision avoidance is active, check    */
   /*  for a collision before sending next angles */
  /* ---------------------------------- */

   if ( collision.is_active )
   {
      check_for_collision( theta_rd, robot_index, &okay_to_move );
    if ( !okay_to_move )
     {
       /* In trouble, restore orginal angles */
       memcpy( &theta_rd[0], &copy_of_orginal_angles[0],  32);
     }
    }

  if (record_jt == 1)
    {
     for (i=1; i<=7; i++)
       {
        temp[0] = ((float) steps)/20;
          fprintf(joint_vel[i], "%f\t%f\n",temp[0], 20*dp3[i]/0.017453293);
          fprintf(joint_th[i], "%f\t%f\n", temp[0], theta_rd[i]/0.017453293 );
       }
    }

  if (record_criterion == 1)
```

```
    {
     norm1 = 0;
     for (i=1; i<=T_JOINT; i++)
         norm1 += dp3[i]*dp3[i];
      norm1 = sqrt(norm1)*1145.9156;
     total_cost += norm1;
    }
   else if (record_criterion == 2)
    {
     norm1 = 0;
     for (i=1; i<=7; i++)
         if (fabs(norm1) < fabs(dp1[i]))
             norm1 = fabs(dp1[i])*1145.9156*3;
    }
   control_path[steps][2] = norm1;
  if (draw == 1)
      draw_robot(theta_rd);
  if (win_state==1)
    {
     steps1=steps;
     draw_curv();
    }
  return;
}
}
```

c) least_normsr11( ) and least_normsr( ) subroutines added to file tools.c

```
least_normsr11(jacob, dx,grds,grd, dtheta)
float jacob[][MAT_DIM], *dx,grds[MAT_DIM], *dtheta,grd;
{
    extern float joint_weight[MAT_DIM];
   int i, j, max_w;
   float temp1[MAT_DIM],temp2[MAT_DIM];
   float tp, dtheta_h[MAT_DIM], jaco[MAT_DIM][MAT_DIM];
    extern int j_ind,jj;
  float w[MAT_DIM][MAT_DIM];
  float dy[MAT_DIM];
   extern float scheme_gain;
  float ans2[MAT_DIM][MAT_DIM];
  float ans4[MAT_DIM][MAT_DIM];
  float temp[MAT_DIM],temp4[MAT_DIM];

  for (i=1; i<=6; i++)
      memcpy(jaco[i], jacob[i], 32);
   cal_kjh(jj,7,jaco,grds,temp2);
   cal_jjtki(jj,7,grd,jaco,ans4);
  for (i=1; i<=6; i++)
  ans4[i][0] = temp2[i];
  if (-1==equit(jj, ans4, 6y))
    {
     ringbell();
     return(-1);
    }

  for (i=1; i<=7; i++)
 {
```

```
    temp1[i] = 0.0;
     for (j=1; j<=jj; j++)
      temp1[i] +=jaco[j][i]*dy[j];
   }

    cal_jjt(jj,7,jaco,ans2);
   for (i=1; i<=jj; i++)
    {
      ans2[i][0] = dx[i];
    }
   if (-1==equit(jj, ans2, temp))
      {
       ringbell();
       return(-1);
      }
    for (i=1; i<=7; i++)
   {
    temp4[i] = 0.0;
     for (j=1; j<=jj; j++)
      temp4[i] +=jaco[j][i]*temp[j];
   }

   for (i=1; i<=7; i++)
     dtheta[i] = (temp1[i]+temp4[i]-(scheme_gain*grds[i]));
    dtheta[8] = 0.0;
    return(0);
}


least_normsr(jacob, dx,grds,grd, dtheta)
float jacob[][MAT_DIM], *dx,grds[MAT_DIM], *dtheta,grd;
{
    extern float joint_weight[MAT_DIM];
   int i, j, max_w;
   float temp1[MAT_DIM], temp2[MAT_DIM];
   float tp, dtheta_h[MAT_DIM], jaco[MAT_DIM][MAT_DIM];
    extern int j_ind,jj;
   float w[MAT_DIM][MAT_DIM];
   float dy[MAT_DIM];
    extern float scheme_gain;
   float ans2[MAT_DIM][MAT_DIM];
   float ans3[MAT_DIM];
   float ans4[MAT_DIM][MAT_DIM];
   float temp[MAT_DIM];

   for (i=1; i<=6; i++)
       memcpy(jaco[i], jacob[i], 32);
    cal_kjh(jj,7,jaco,grds,temp2);
   for (i=1; i<=jj; i++)
     temp1[i] = dx[i]+temp2[i];
    cal_jjtki(jj,7,grd,jaco,ans4);
   for (i=1; i<=6; i++)
   ans4[i][0] = temp1[i];

   if (-1==equit(jj, ans4, dy))
      {
       ringbell();
       return(-1);
      }

    for (i=1; i<=7; i++)
   {
```

```
    dtheta[i] = 0.0;
     for (j=1; j<=jj; j++)
       dtheta[i] +=jaco[j][i]*dy[j];
   }

   for (i=1; i<=7; i++)
     dtheta[i] -= (scheme_gain*grds[i]);
    dtheta[8] = 0.0;
    return(0);
}
```

d) cal_j*( ) subroutines added to file tools.c

```
cal_wj(m,n,jaco,ans)
int m,n;
float jaco[MAT_DIM][MAT_DIM],ans[MAT_DIM][MAT_DIM];
{
 int i,j,k;
 extern float srk[MAT_DIM];

 for (i=1; i<=m; i++)
  {
    for (j=1; j<=7; j++)
     {
       ans[i][j] = srk[i]*jaco[i][j];
     }
  }
 return;
}


cal_wjjwtki(m,n,k1,jaco,ans)
int m,n;
float jaco[MAT_DIM][MAT_DIM], ans[MAT_DIM][MAT_DIM];
float k1;
{
 int i,j,k;
 float k0;
 float t1;

   k0 = 2.5;
   for (i=1; i<=m; i++)
    {
       for (j=i; j<=m; j++)
          {
           ans[i][j]=0;
           for (k=1; k<=n; k++)
               ans[i][j]+=jaco[i][k]*jaco[j][k];
           ans[j][i]=ans[i][j];
          }
   }

    t1 = (1.0 - (0.25/k1));
   if (t1 <= 0.0 )
     t1 = 0.0;

   k1 = k0*t1;
```

```
    for (i=1; i<=6; i++)
      ans[i][i] += k1;

    return;
}


cal_jjtki(m, n,k1, jaco, ans)
int m, n;
float jaco[MAT_DIM][MAT_DIM], ans[MAT_DIM][MAT_DIM];
float k1;
{
    int i, j, k;
    float k0;
    float t1;
    extern float tim;
    extern int stor;
    float mm[20];

    k0 = 2.5;
    for (i=1; i<=m; i++)
       for (j=i; j<=m; j++)
          {
           ans[i][j]=0;
           for (k=1; k<=n; k++)
               ans[i][j]+=jaco[i][k]*jaco[j][k];
           ans[j][i]=ans[i][j];
          }

    t1 = (1.0 - (0.25/k1));
    if (t1 <= 0.0 )
      t1 = 0.0;

    k1 = k0*t1;

    mm[17] = tim;   /* Option for storing the gain factor for the SR*/
    mm[19] = k1;
    if (stor == 1)
    store_data(mm);

    for (i=1; i<=6; i++)
      ans[i][i] += k1;
    return;
}


cal_jjt(m, n, jaco, ans)
int m, n;
float jaco[MAT_DIM][MAT_DIM], ans[MAT_DIM][MAT_DIM];
{
    int i, j, k;
    for (i=1; i<=m; i++)
     {
       for (j=i; j<=m; j++)
          {
           ans[i][j]=0;
           for (k=1; k<=n; k++)
               ans[i][j]+=jaco[i][k]*jaco[j][k];
           ans[j][i]=ans[i][j];
          }
     }
    return;
```

```
}


cal_jtwx(m, n, anss1,tmp, ans)
int m, n;
float anss1[MAT_DIM][MAT_DIM],tmp[MAT_DIM], ans[MAT_DIM];
{
   int i, j, k;

   for (i=1; i<=n; i++)
    {
      ans[i]=0.0;
      for (j=1; j<=m; j++)
      {
         ans[i] += (anss1[i][j]*tmp[j]);
      }
    }
   return;
}


cal_jtw(m, n, jaco1, ans)
int m, n;
float jaco1[MAT_DIM][MAT_DIM], ans[MAT_DIM][MAT_DIM];
{
   int i, j, k;
   extern float srk[MAT_DIM];
   for (i=1; i<=m; i++)
    {
      for (j=1; j<=n; j++)
        {
         ans[j][i] = jaco1[i][j]*srk[i];
        }
    }
   return;
}


cal_jtwj(m, n,k1,anss1, jaco1, ans)
int m, n;
float anss1[MAT_DIM][MAT_DIM], jaco1[MAT_DIM][MAT_DIM], ans[MAT_DIM][MAT_DIM];
float k1;
{
   int i, j, k;
   extern float srk1[MAT_DIM];
   for(i=1; i<=n; i++)
    {
      for(j=1; j<=n; j++)
      {
        ans[i][j] = 0.0;
         for (k=1; k<=m; k++)
         {
            ans[i][j] += anss1[i][k]*jaco1[k][j];
         }
      }
    }
  if(fabs(k1) < 0.3)
    k1=0.1;
   else if (fabs(k1) > 0.3)
    k1=1.0;
   for (i=1; i<=7; i++)
    ans[i][i] += k1;
```

```
    return;
}


cal_kjh(m,n,jacol,grs,ansl)
float jacol[MAT_DIM][MAT_DIM],grs[MAT_DIM],ansl[MAT_DIM];
int m,n;
{
  int i,j,k;
  extern float scheme_gain;

  for (i=1; i<=m; i++)
   {
     ansl[i] = 0.0;
      for (j=1; j<=n; j++)
        ansl[i] += jacol[i][j]*grs[j];
   }
  for (i=1; i<=m; i++)
    ansl[i] *= scheme_gain;
  return;
 }


cal_jjtl(m, n, jacol,wmat, ans)
int m, n;
float jacol[MAT_DIM][MAT_DIM], ans[MAT_DIM][MAT_DIM],wmat[MAT_DIM][MAT_DIM];
{
    int i, j, k;
    float ansl[MAT_DIM][MAT_DiM];

    for (i=1; i<=m; i++)
     {
      for (j=1; j<=n; j++)
         {
          ansl[i][j] =0.0;
          for(k=1; k<=n; k++)
           ansl[i][j]+=jacol[i][k]*wmat[k][j];
         }
      }
    for (i=1; i<=m; i++)
    {
      for (j=1; j<=m; j++)
      {
        ans[i][j] =0.0;
        for(k=1; k<=n; k++)
         ans[i][j] += ansl[i][k]*jacol[j][k];
      }
    }
    return;
}


cal_wjt(m,n,jacol,wmat,ans)
int m,n;
float jacol[MAT_DIM][MAT_DIM], ans[MAT_DIM][MAT_DIM],wmat[MAT_DIM][MAT_DIM];
{
 int i, j,k;

   for (i=1; i<=n; i++)
    {
      for (j=1; j<=m; j++)
      {
```

```
        ans[i][j] =0.0;
        for(k=1; k<=n; k++)
         ans[i][j] += wmat[i][k]*jaco1[j][k];
       }
    }
return;
}
```

e) least_norm*( ) subrotuines added to file tools.c

```
least_norm72(jacob, dx,grds, dtheta)
float jacob[][MAT_DIM], *dx,grds[MAT_DIM], *dtheta;
{
    extern float joint_weight[MAT_DIM];
   int i, j, max_w;
   float temp1[MAT_DIM][MAT_DIM], temp2[MAT_DIM];
   float tp, dtheta_h[MAT_DIM], jaco[MAT_DIM][MAT_DIM];
    extern int j_ind,jj;
   float w[MAT_DIM][MAT_DIM];
   float dy[MAT_DIM];
    extern float scheme_gain;
    /*tpon();
   for (i=1; i<=7; i++)
    printf("grds[%d]=%f\n ",i,grds[i]);
    printf("\n\n");*/

   for (i=1; i<=6; i++)
       memcpy(jaco[i], jacob[i], 32);
    cal_kjh(jj,7,jaco,grds,temp2);
    cal_jjt(jj,7,jaco,temp1);
   for (i=1; i<=jj; i++)
    temp1[i][0] = dx[i]+temp2[i];
  if (-1==equit(jj, temp1, dy))
     {
      ringbell();
      return(-1);
     }
  for (i=1; i<=7; i++)
  {
   dtheta[i] = 0.0;
    for (j=1; j<=jj; j++)
     dtheta[i] +=jaco[j][i]*dy[j];
  }
   for (i=1; i<=7; i++)
    dtheta[i] += (-scheme_gain*grds[i]);
   dtheta[8] = 0.0;
   return(0);
}


least_norm71(jacob, dx, dtheta)
float jacob[][MAT_DIM], *dx, *dtheta;
{
    extern float joint_weight[MAT_DIM];
   int i, j, max_w;
   float temp1[MAT_DIM][MAT_DIM], temp2[MAT_DIM][MAT_DIM];
```

```
    float tp, dtheta_h[MAT_DIM], jaco[MAT_DIM][MAT_DIM];
     extern int j_ind,jj;
    float w[MAT_DIM][MAT_DIM];
    float dy[MAT_DIM];
    float w1[MAT_DIM][MAT_DIM];

    for (i=1; i<=6; i++)
        memcpy(jaco[i], jacob[i], 32);
    for (i=1; i<=7; i++)
     {
      for (j=1; j<=7; j++)
     {
       w[i][j] = w1[i][j] =0.0;
     }
     }
    for (i=1; i<=7; i++)
    {
     w[i][i] = joint_weight[i];
     if ( joint_weight[i] > 0)
      w1[i][i] = 1.0/joint_weight[i];
     else
     w1[i][i] = 1.0;
    }
     cal_jjt1(jj,7,jaco,w1,temp1);
    for (i=1; i<=6; i++)
    temp1[i][0] = dx[i];
    if (-1==equit(jj, temp1, dy))
       {
        ringbell();
        return(-1);
       }
     cal_wjt(jj,7,jaco,w1,temp2);
   for (i=1; i<=7; i++)
   {
 dtheta[i]=0.0;
 for(j=1; j<=jj; j++)
   {
     dtheta[i]+=temp2[i][j]*dy[j];
   }
 }
    dtheta[8] = 0.0;
    return(0);
}



least_norm1(jacob, dx, dt)
float jacob[][MAT_DIM], *dx, dt[MAT_DIM][MAT_DIM];
{
    extern float joint_weight[MAT_DIM];
    int i, j, max_w, kind;
    extern int j_ind,jj;
    float temp1[MAT_DIM];
    float  tp,dt_h1[MAT_DIM], jaco[MAT_DIM][MAT_DIM], h_transf[5][5];
    float dt_h2[MAT_DIM],dt_h3[MAT_DIM],dt_h4[MAT_DIM],tmp[MAT_DIM][MAT_DIM];
    float dtheta_h[MAT_DIM];
    float gh[MAT_DIM],gsum[MAT_DIM];
    float aa,bb,k;
    float tp1;
    float tmp1[MAT_DIM][MAT_DIM],tmp2[MAT_DIM][MAT_DIM];
    float dtheta[MAT_DIM];
    float dt_h5[MAT_DIM],dt_h6[MAT_DIM];
```

```
 float dirns[MAT_DIM];
 float t;

 for (i=1; i<=6; i++)
    memcpy(jaco[i], jacob[i], 32);

 for (i=1; i<=7;i++)
 {
    dt_h1[i]=dt_h2[i]=dt_h3[i]=dt_h4[i]=dt_h5[i]=dt_h6[i]=0.0;
 }
  tp = joint_weight[1];
 max_w = 1;
 for (i=2; i<=6; i++)
    {
     if (tp <= joint_weight[i])
       {
         tp = joint_weight[i];
        max_w = i;
       }
    }
/*********** Finding the particular solution ***********/

 for (i=1; i<=6; i++)
    {
     jaco[i][0] = dx[i];
      for (j=1; j<=7; j++)
        {
          jaco[i][j] = jaco[i][j]/joint_weight[j];
        }
    }
  for (i=1; i<=6; i++)
    {
     tp = jaco[i][max_w];
     jaco[i][max_w] = jaco[i][6];
     jaco[i][6] = tp;
    }
 dirns[1] = dirns[2] = 1.0;
 dirns[3] = 1.0;
/* t=0.0;
 for(i=1; i<=3; i++)
 t += jaco[i][1]*dirns[i];
 dtheta[1] = dx[1]/ t; */

if (-1==equit(jj, jaco, dtheta))
    {
     ringbell();
     return(-1);
    }
  for(i=jj+1; i<=7; i++)
 {
  dtheta[i] = 0.0;
 }

  tp = dtheta[6];
  dtheta[6] = dtheta[max_w];
  dtheta[max_w] = tp;

/********** Finding the set of homo. solutions ************/

  for (j=1; j<=7-jj; j++)
    {
        tp1 = 0.0017453293;
      for(i=1; i<=6; i++)
```

```
        jaco[i][0]=(-jaco[i][jj+j])*tp1;

    if (-1==equit(jj,jaco, dtheta_h))
      {
       ringbell();
       return(-1);
      }
    for (i=jj+1; i<=7; i++)
   {
    dtheta_h[i]=0.0;
   }
    dtheta_h[jj+j] = tp1;

    tp = dtheta_h[6];
    dtheta_h[6] = dtheta_h[max_w];
    dtheta_h[max_w] = tp;

    if (j == 1)
    {
     for ( i=1; i<=7; i++)
       dt_h1[i]=dtheta_h[i];
    }
   else if ( j == 2 )
    {
      for ( i=1; i<=7; i++)
       dt_h2[i]=dtheta_h[i];
    }
   else if ( j == 3 )
    {
      for ( i=1; i<=7; i++)
       dt_h3[i]=dtheta_h[i];
    }
   else if ( j == 4 )
    {
      for ( i=1; i<=7; i++)
       dt_h4[i]=dtheta_h[i];
    }
   else if ( j == 5 )
    {
      for ( i=1; i<=7; i++)
       dt_h5[i]=dtheta_h[i];
    }
   else if ( j == 6 )
    {
      for ( i=1; i<=7; i++)
       dt_h6[i]=dtheta_h[i];
    }
   }

/***************** Defining the matrix tmp1 *********************************/
for(i=1; i<=7; i++)
{
tmp1[i][1] = dt_h1[i];
tmp1[i][2] = dt_h2[i];
tmp1[i][3] = dt_h3[i];
tmp1[i][4] = dt_h4[i];
tmp1[i][5] = dt_h5[i];
tmp1[i][6] = dt_h6[i];
}

/********** Converting the homogenous soln. to ortho. homo. soln.**********/
```

```
get_ortho6(tmp1,tmp2);


for ( i=1; i<=7; i++)
{
dt_h1[i] = tmp2[i][1];
dt_h2[i] = tmp2[i][2];
dt_h3[i] = tmp2[i][3];
dt_h4[i] = tmp2[i][4];
dt_h5[i] = tmp2[i][5];
dt_h6[i] = tmp2[i][6];
}
/***************** Getting the scalars ************************************/
 for (i=1; i<=6; i++)
     for (j=0; j<=6; j++)
      tmp[i][j] = 0.0;

  for ( i=1; i<=7; i++)
  {
   tmp[1][0] += dtheta[i]*dt_h1[i];
   tmp[2][0] += dtheta[i]*dt_h2[i];
   tmp[3][0] += dtheta[i]*dt_h3[i];
   tmp[4][0] += dtheta[i]*dt_h4[i];
   tmp[5][0] += dtheta[i]*dt_h5[i];
   tmp[6][0] += dtheta[i]*dt_h6[i];
  }

 for (i=1; i<=7; i++)
    {
      tmp[1][1] += dt_h1[i]*dt_h1[i];
      tmp[2][2] += dt_h2[i]*dt_h2[i];
      tmp[3][3] += dt_h3[i]*dt_h3[i];
      tmp[4][4] += dt_h4[i]*dt_h4[i];
      tmp[5][5] += dt_h5[i]*dt_h5[i];
      tmp[6][6] += dt_h6[i]*dt_h6[i];
    }

for ( i=1; i<=7-jj; i++)
  temp1[i] = tmp[i][0]/tmp[i][i];

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

 for (i=1; i<=7; i++)
       {
        dtheta_h[i] =
((dt_h1[i]*temp1[1])+(dt_h2[i]*temp1[2])+(dt_h3[i]*temp1[3])+(dt_h4[i]*temp1[4])+(dt_h5[i]*temp1[5])+(dt
_h6[i]*temp1[6]));
         dtheta[i] -= dtheta_h[i];
         dtheta[i] /= joint_weight[i];

       }
dtheta[8]=0.0;
for (i=1; i<=7; i++)
{
dt[i][1] = dt_h1[i];
dt[i][2] = dt_h2[i];
dt[i][3] = dt_h3[i];
dt[i][4] = dt_h4[i];
dt[i][5] = dt_h5[i];
dt[i][6] = dt_h6[i];
dt[i][7] = dtheta[i];
dt[i][8] = dtheta_h[i];
}
```

```
return(0);
}



least_norm2(jacob, dx, dt)
float jacob[][MAT_DIM], *dx, dt[MAT_DIM][MAT_DIM];
{
    extern float joint_weight[MAT_DIM];
    int i, j, max_w, kind;
    extern int j_ind,jj;
    float temp1[MAT_DIM];
    float  tp,dt_h1[MAT_DIM], jaco[MAT_DIM][MAT_DIM], h_transf[5][5];
    float dt_h2[MAT_DIM],dt_h3[MAT_DIM],dt_h4[MAT_DIM],tmp[MAT_DIM][MAT_DIM];
    float dtheta_h[MAT_DIM];
    float gh[MAT_DIM],gsum[MAT_DIM];
    float aa,bb,k;
    float tp1;
    float tmp1[MAT_DIM][MAT_DIM],tmp2[MAT_DIM][MAT_DIM];
    float dtheta[MAT_DIM];
    float dt_h5[MAT_DIM],dt_h6[MAT_DIM];


    for (i=1; i<=6; i++)
        memcpy(jaco[i], jacob[i], 32);

    for (i=1; i<=7;i++)
    {
        dt_h1[i]=dt_h2[i]=dt_h3[i]=dt_h4[i]=dt_h5[i]=dt_h6[i]=0.0;
    }
      tp = joint_weight[1];
    max_w = 1;
    for (i=2; i<=6; i++)
       {
        if (tp <= joint_weight[i])
          {
            tp = joint_weight[i];
           max_w = i;
          }
       }

  /************ Finding the particular solution ************/

    for (i=1; i<=6; i++)
       {
        jaco[i][0] = dx[i];
         for (j=1; j<=7; j++)
           {
             jaco[i][j] = jaco[i][j]/joint_weight[j];
           }
       }

     for (i=1; i<=6; i++)
       {
        tp = jaco[i][max_w];
        jaco[i][max_w] = jaco[i][6];
        jaco[i][6] = tp;
       }

     if (-1==equit(2, jaco, dtheta))
       {
        ringbell();
```

```
      return(-1);
     }
   for(i=3; i<=7; i++)
   {
   dtheta[i] = 0.0;
   }

   tp = dtheta[6];
   dtheta[6] = dtheta[max_w];
   dtheta[max_w] = tp;

/********** Finding the set of homo. solutions ************/

   for (j=1; j<=5; j++)

     {
       tp1 = 0.0017453293;
      for(i=1; i<=6; i++)

       jaco[i][0]=(-jaco[i][jj+j])*tp1;

   if (-1==equit(2,jaco, dtheta_h))
     {
      ringbell();
     return(-1);
     }
   for (i=3; i<=7; i++)
{
   dtheta_h[i]=0.0;
}
    dtheta_h[jj+j] =tp1;
   tp = dtheta_h[6];
   dtheta_h[6] = dtheta_h[max_w];
   dtheta_h[max_w] = tp;

   if (j == 1)
    {
     for ( i=1; i<=7; i++)
       dt_h1[i]=dtheta_h[i];
    }
   else if ( j == 2 )
    {
     for ( i=1; i<=7; i++)
       dt_h2[i]=dtheta_h[i];
    }
   else if ( j == 3 )
    {
     for ( i=1; i<=7; i++)
       dt_h3[i]=dtheta_h[i];
    }
   else if ( j == 4 )
    {
     for ( i=1; i<=7; i++)
       dt_h4[i]=dtheta_h[i];
    }
   else if ( j == 5 )
    {
     for ( i=1; i<=7; i++)
       dt_h5[i]=dtheta_h[i];
    }
   }

/***************** Defining the matrix tmp1 *********************************/
```

```
for(i=1; i<=7; i++)
{
tmp1[i][1] = dt_h1[i];
tmp1[i][2] = dt_h2[i];
tmp1[i][3] = dt_h3[i];
tmp1[i][4] = dt_h4[i];
tmp1[i][5] = dt_h5[i];
}

/********** Converting the homogenous soln. to ortho. homo. soln.**********/
get_ortho5(tmp1,tmp2);
for ( i=1; i<=7; i++)
{
dt_h1[i] = tmp2[i][1];
dt_h2[i] = tmp2[i][2];
dt_h3[i] = tmp2[i][3];
dt_h4[i] = tmp2[i][4];
dt_h5[i] = tmp2[i][5];
}
/***************** Getting the scalars ************************************/
 for (i=1; i<=4; i++)
    for (j=0; j<=4; j++)
     tmp[i][j] = 0.0;

  for ( i=1; i<=7; i++)
  {
   tmp[1][0] += dtheta[i]*dt_h1[i];
   tmp[2][0] += dtheta[i]*dt_h2[i];
   tmp[3][0] += dtheta[i]*dt_h3[i];
   tmp[4][0] += dtheta[i]*dt_h4[i];
   tmp[5][0] += dtheta[i]*dt_h5[i];
  }

 for (i=1; i<=7; i++)
   {
     tmp[1][1] += dt_h1[i]*dt_h1[i];
     tmp[2][2] += dt_h2[i]*dt_h2[i];
     tmp[3][3] += dt_h3[i]*dt_h3[i];
     tmp[4][4] += dt_h4[i]*dt_h4[i];
     tmp[5][5] += dt_h5[i]*dt_h5[i];
   }

for ( i=1; i<=5; i++)
  temp1[i] = tmp[i][0]/tmp[i][i];


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

 for (i=1; i<=7; i++)
      {
       dtheta_h[i] =
((dt_h1[i]*temp1[1])+(dt_h2[i]*temp1[2])+(dt_h3[i]*temp1[3])+(dt_h4[i]*temp1[4])+(dt_h5[i]*temp1[5]));
        dtheta[i] -= dtheta_h[i];
        dtheta[i] /= joint_weight[i];

      }
dtheta[8]=0.0;
for (i=1; i<=7; i++)
{
dt[i][1] = dt_h1[i];
dt[i][2] = dt_h2[i];
dt[i][3] = dt_h3[i];
dt[i][4] = dt_h4[i];
```

```
dt[i][5] = dt_h5[i];
dt[i][6] = 0.0;
dt[i][7] = dtheta[i];
dt[i][8] = dtheta_h[i];
}
return(0);
}



least_norm3(jacob, dx, dt)
float jacob[][MAT_DIM], *dx, dt[MAT_DIM][MAT_DIM] ;
{
    extern float joint_weight[MAT_DIM];
    int i, j, max_w, kind;
    extern int j_ind,jj;
    float temp1[MAT_DIM];
    float  tp,dt_h1[MAT_DIM], jaco[MAT_DIM][MAT_DIM], h_transf[5][5];
    float dt_h2[MAT_DIM],dt_h3[MAT_DIM],dt_h4[MAT_DIM],tmp[MAT_DIM][MAT_DIM];
    float dtheta_h[MAT_DIM];
    float gh[MAT_DIM],gsum[MAT_DIM];
    float aa,bb,k;
    float tp1;
    float tmp1[MAT_DIM][MAT_DIM],tmp2[MAT_DIM][MAT_DIM];
    float dtheta[MAT_DIM];
    float dt_h5[MAT_DIM],dt_h6[MAT_DIM];
    extern int scheme_index;
    float dtt[MAT_DIM][MAT_DIM];

    for (i=1; i<=6; i++)
        memcpy(jaco[i], jacob[i], 32);



    tp = joint_weight[1];
    max_w = 1;
    for (i=2; i<=6; i++)
      {
       if (tp <= joint_weight[i])
         {
          tp = joint_weight[i];
          max_w = i;
         }
      }
  for (i=1; i<=6; i++)
      {
       jaco[i][0] = dx[i];
        for (j=1; j<=7; j++)
          {
            jaco[i][j] = jaco[i][j]/joint_weight[j];
          }
      }
    for (i=1; i<=6; i++)
      {
       tp = jaco[i][max_w];
       jaco[i][max_w] = jaco[i][6];
       jaco[i][6] = tp;
      }

    if (-1==equit(3, jaco, dtheta))
      {
       ringbell();
```

```
      return(-1);
     }

    dtheta[4]=dtheta[5]=dtheta[6]=dtheta[7] =0.0;
   tp = dtheta[6];
   dtheta[6] = dtheta[max_w];
   dtheta[max_w] = tp;

/********** Finding the set of homo. solutions ************/

   for (j=1; j<=4; j++)
     {
       tp1 = 0.0017453293;
      for(i=1; i<=6; i++)

       jaco[i][0]=(-jaco[i][jj+j])*tp1;

    if (-1==equit(3,jaco, dtheta_h))
     {
      ringbell();
      return(-1);
     }
    for (i=4; i<=7; i++)
   {
    dtheta_h[i]=0.0;
   }
     dtheta_h[jj+j] =tp1;
    tp = dtheta_h[6];
    dtheta_h[6] = dtheta_h[max_w];
    dtheta_h[max_w] = tp;
   if (j == 1)
    {
     for ( i=1; i<=7; i++)
       dt_h1[i]=dtheta_h[i];
    }
   else if ( j == 2 )
    {
     for ( i=1; i<=7; i++)
       dt_h2[i]=dtheta_h[i];
    }
   else if ( j == 3 )
    {
     for ( i=1; i<=7; i++)
       dt_h3[i]=dtheta_h[i];
    }
   else
    {
     for ( i=1; i<=7; i++)
       dt_h4[i]=dtheta_h[i];
    }
   }

/***************** Defining the matrix tmp1 *********************************/
for(i=1; i<=7; i++)
{
tmp1[i][1] = dt_h1[i];
tmp1[i][2] = dt_h2[i];
tmp1[i][3] = dt_h3[i];
tmp1[i][4] = dt_h4[i];
}

/********** Converting the homogenous soln. to ortho. homo. soln.**********/
get_ortho4(tmp1,tmp2);
```

```
for ( i=1; i<=7; i++)
{
dt_h1[i] = tmp2[i][1];
dt_h2[i] = tmp2[i][2];
dt_h3[i] = tmp2[i][3];
dt_h4[i] = tmp2[i][4];
}
/***************** Getting the scalars ************************************/
 for (i=1; i<=4; i++)
     for (j=0; j<=4; j++)
      tmp[i][j] = 0.0;

  for ( i=1; i<=7; i++)
  {
   tmp[1][0] += dtheta[i]*dt_h1[i];
   tmp[2][0] += dtheta[i]*dt_h2[i];
   tmp[3][0] += dtheta[i]*dt_h3[i];
   tmp[4][0] += dtheta[i]*dt_h4[i];
  }

 for (i=1; i<=7; i++)
    {
      tmp[1][1] += dt_h1[i]*dt_h1[i];
      tmp[2][2] += dt_h2[i]*dt_h2[i];
      tmp[3][3] += dt_h3[i]*dt_h3[i];
      tmp[4][4] += dt_h4[i]*dt_h4[i];
    }

for ( i=1; i<=4; i++)
  temp1[i] = tmp[i][0]/tmp[i][i];


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

 for (i=1; i<=7; i++)
       {
          dtheta_h[i] = ((dt_h1[i]*temp1[1])+(dt_h2[i]*temp1[2])+(dt_h3[i]*temp1[3])+(dt_h4[i]*temp1[4]));
         dtheta[i] -= dtheta_h[i];
         dtheta[i] /= joint_weight[i];

       }
dtheta[8]=0.0;

for (i=1; i<=7; i++)
{
dt[i][1] = dt_h1[i];
dt[i][2] = dt_h2[i];
dt[i][3] = dt_h3[i];
dt[i][4] = dt_h4[i];
dt[i][5] = 0.0;
dt[i][6] = 0.0;
dt[i][7] = dtheta[i];
dt[i][8] = dtheta_h[i];
}
return(0);
}



least_norm4(jacob, dx, dt)
```

```
float jacob[][MAT_DIM], *dx, dt[MAT_DIM][MAT_DIM];
{
    extern float joint_weight[MAT_DIM];
    int i, j, max_w, kind;
    extern int j_ind,jj;
    float temp1[MAT_DIM];
    float  tp,dt_h1[MAT_DIM], jaco[MAT_DIM][MAT_DIM], h_transf[5][5];
    float dt_h2[MAT_DIM],dt_h3[MAT_DIM],dt_h4[MAT_DIM],tmp[MAT_DIM][MAT_DIM];
    float dtheta_h[MAT_DIM];
    float gh[MAT_DIM],gsum[MAT_DIM];
    float aa,bb,k;
    float tp1;
    float tmp1[MAT_DIM][MAT_DIM],tmp2[MAT_DIM][MAT_DIM];
    float dtheta[MAT_DIM];
    float dt_h5[MAT_DIM],dt_h6[MAT_DIM];


    for (i=1; i<=6; i++)
        memcpy(jaco[i], jacob[i], 32);

    for (i=1; i<=7;i++)
    {
        dt_h1[i]=dt_h2[i]=dt_h3[i]=dt_h4[i]=dt_h5[i]=dt_h6[i]=0.0;
    }
     tp = joint_weight[1];
    max_w = 1;
    for (i=2; i<=6; i++)
       {
        if (tp <= joint_weight[i])
          {
            tp = joint_weight[i];
           max_w = i;
          }
       }

   /************ Finding the particular solution ************/

    for (i=1; i<=6; i++)
       {
        jaco[i][0] = dx[i];
          for (j=1; j<=7; j++)
           {
             jaco[i][j] = jaco[i][j]/joint_weight[j];
           }
       }

     for (i=1; i<=6; i++)
       {
        tp = jaco[i][max_w];
        jaco[i][max_w] = jaco[i][6];
        jaco[i][6] = tp;
       }

     if (-1==equit(4, jaco, dtheta))
       {
        ringbell();
       return(-1);
       }
     for(i=5; i<=7; i++)
    {
     dtheta[i] = 0.0;
    }
```

```
    tp = dtheta[6];
    dtheta[6] = dtheta[max_w];
    dtheta[max_w] = tp;


 /********** Finding the set of homo. solutions ************/

    for (j=1; j<=3; j++)

      {
        tp1 = 0.0017453293;
       for(i=1; i<=6; i++)

        jaco[i][0]=(-jaco[i][jj+j])*tp1;

     if (-1==equit(4,jaco, dtheta_h))
       {
        ringbell();
       return(-1);
       }
     for (i=5; i<=7; i++)
    {
     dtheta_h[i]=0.0;
    }
      dtheta_h[jj+j] = tp1;
      tp = dtheta_h[6];
      dtheta_h[6] = dtheta_h[max_w];
      dtheta_h[max_w] = tp;

    if (j == 1)
     {
      for ( i=1; i<=7; i++)
        dt_h1[i]=dtheta_h[i];
     }
    else if ( j == 2 )
     {
      for ( i=1; i<=7; i++)
        dt_h2[i]=dtheta_h[i];
     }
    else if ( j == 3 )
     {
      for ( i=1; i<=7; i++)
        dt_h3[i]=dtheta_h[i];
     }
    }

/***************** Defining the matrix tmp1 *********************************/
for(i=1; i<=7; i++)
{
tmp1[i][1] = dt_h1[i];
tmp1[i][2] = dt_h2[i];
tmp1[i][3] = dt_h3[i];
}

/********** Converting the homogenous soln. to ortho. homo. soln.**********/
get_ortho3(tmp1,tmp2);
for ( i=1; i<=7; i++)
{
dt_h1[i] = tmp2[i][1];
dt_h2[i] = tmp2[i][2];
dt_h3[i] = tmp2[i][3];
}
/***************** Getting the scalars ************************************/
```

```
for (i=1; i<=4; i++)
    for (j=0; j<=4; j++)
     tmp[i][j] = 0.0;

  for ( i=1; i<=7; i++)
  {
   tmp[1][0] += dtheta[i]*dt_h1[i];
   tmp[2][0] += dtheta[i]*dt_h2[i];
   tmp[3][0] += dtheta[i]*dt_h3[i];
  }

 for (i=1; i<=7; i++)
    {
      tmp[1][1] += dt_h1[i]*dt_h1[i];
      tmp[2][2] += dt_h2[i]*dt_h2[i];
      tmp[3][3] += dt_h3[i]*dt_h3[i];
    }

for ( i=1; i<=3; i++)
  temp1[i] = tmp[i][0]/tmp[i][i];

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

 for (i=1; i<=7; i++)
       {
          dtheta_h[i] = ((dt_h1[i]*temp1[1])+(dt_h2[i]*temp1[2])+(dt_h3[i]*temp1[3]));
         dtheta[i] -= dtheta_h[i];
         dtheta[i] /= joint_weight[i];

       }
dtheta[8]=0.0;
for (i=1; i<=7; i++)
{
dt[i][1] = dt_h1[i];
dt[i][2] = dt_h2[i];
dt[i][3] = dt_h3[i];
dt[i][4] = 0.0;
dt[i][5] = 0.0;
dt[i][6] = 0.0;
dt[i][7] = dtheta[i];
dt[i][8] = dtheta_h[i];
}
return(0);
}



least_norm5(jacob, dx, dt)
float jacob[][MAT_DIM], *dx, dt[MAT_DIM][MAT_DIM];
{
    extern float joint_weight[MAT_DIM];
    int i, j, max_w, kind;
    extern int j_ind,jj;
    float temp1[MAT_DIM];
    float  tp,dt_h1[MAT_DIM], jaco[MAT_DIM][MAT_DIM], h_transf[5][5];
    float dt_h2[MAT_DIM],dt_h3[MAT_DIM],dt_h4[MAT_DIM],tmp[MAT_DIM][MAT_DIM];
    float dtheta_h[MAT_DIM];
    float gh[MAT_DIM],gsum[MAT_DIM];
    float aa,bb,k;
    float tp1;
    float tmp1[MAT_DIM][MAT_DIM],tmp2[MAT_DIM][MAT_DIM];
    float dtheta[MAT_DIM];
```

```
  float dt_h5[MAT_DIM],dt_h6[MAT_DIM];


  for (i=1; i<=6; i++)
      memcpy(jaco[i], jacob[i], 32);
  for (i=1; i<=7;i++)
  {
     dt_h1[i]=dt_h2[i]=dt_h3[i]=dt_h4[i]=dt_h5[i]=dt_h6[i]=0.0;
  }
  tp = joint_weight[1];
  max_w = 1;
  for (i=2; i<=6; i++)
     {
      if (tp <= joint_weight[i])
        {
          tp = joint_weight[i];
         max_w = i;
        }
     }

 /************ Finding the particular solution ************/

  for (i=1; i<=6; i++)
     {
      jaco[i][0] = dx[i];
       for (j=1; j<=7; j++)
         {
           jaco[i][j] = jaco[i][j]/joint_weight[j];
         }
     }

   for (i=1; i<=6; i++)
     {
      tp = jaco[i][max_w];
      jaco[i][max_w] = jaco[i][6];
      jaco[i][6] = tp;
     }

   if (-1==equit(5, jaco, dtheta))
     {
      ringbell();
      return(-1);
     }
  for(i=6; i<=7; i++)
  {
  dtheta[i] = 0.0;
  }

   tp = dtheta[6];
   dtheta[6] = dtheta[max_w];
   dtheta[max_w] = tp;

 /********** Finding the set of homo. solutions ************/

   for (j=1; j<=2; j++)

     {
       tp1 = 0.0017453293;
      for(i=1; i<=6; i++)

      jaco[i][0]=(-jaco[i][jj+j])*tp1;

   if (-1==equit(5,jaco, dtheta_h))
```

```
      {
       ringbell();
       return(-1);
      }
    for (i=6; i<=7; i++)
   {
    dtheta_h[i]=0.0;
   }
    dtheta_h[jj+j] = tp1;
    tp = dtheta_h[6];
    dtheta_h[6] = dtheta_h[max_w];
    dtheta_h[max_w] = tp;

   if (j == 1)
    {
     for ( i=1; i<=7; i++)
       dt_h1[i]=dtheta_h[i];
    }
   else if ( j == 2 )
    {
     for ( i=1; i<=7; i++)
      dt_h2[i]=dtheta_h[i];
    }
   }

/***************** Defining the matrix tmp1 *********************************/
for(i=1; i<=7; i++)
{
tmp1[i][1] = dt_h1[i];
tmp1[i][2] = dt_h2[i];
}

/********** Converting the homogenous soln. to ortho. homo. soln.**********/
get_ortho2(tmp1,tmp2);
for ( i=1; i<=7; i++)
{
dt_h1[i] = tmp2[i][1];
dt_h2[i] = tmp2[i][2];
}
/***************** Getting the scalars ***************************************/
 for (i=1; i<=4; i++)
     for (j=0; j<=4; j++)
      tmp[i][j] = 0.0;

  for ( i=1; i<=7; i++)
  {
   tmp[1][0] += dtheta[i]*dt_h1[i];
   tmp[2][0] += dtheta[i]*dt_h2[i];
  }

 for (i=1; i<=7; i++)
    {
      tmp[1][1] += dt_h1[i]*dt_h1[i];
      tmp[2][2] += dt_h2[i]*dt_h2[i];
    }

for ( i=1; i<=2; i++)
  temp1[i] = tmp[i][0]/tmp[i][i];


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

 for (i=1; i<=7; i++)
```

```
        {
          dtheta_h[i] = ((dt_h1[i]*temp1[1])+(dt_h2[i]*temp1[2]));
         dtheta[i] -= dtheta_h[i];
          dtheta[i] /= joint_weight[i];

        }
dtheta[8]=0.0;
for (i=1; i<=7; i++)
{
dt[i][1] = dt_h1[i];
dt[i][2] = dt_h2[i];
dt[i][3] = 0.0;
dt[i][4] = 0.0;
dt[i][5] = 0.0;
dt[i][6] = 0.0;
dt[i][7] = dtheta[i];
dt[i][8] = dtheta_h[i];
}
return(0);
}
```

f) get_ortho* ( ) subroutines added to file tools.c

```
get_ortho2(mat1,mat2)
float mat1[MAT_DIM][MAT_DIM],mat2[MAT_DIM][MAT_DIM];
{
int i;
float h1[MAT_DIM],h2[MAT_DIM],h3[MAT_DIM],h4[MAT_DIM];
float a12;
float t1,t2;

for ( i=1; i<=7; i++)
{
h1[i] = mat1[i][1];
h2[i] = mat1[i][2];
}

t1 = t2 = 0.0;
for (i=1; i<=7; i++)
{
t1+= h1[i]*h2[i];
}
t2=get_inp(h1);
a12 = t1/t2;

for ( i=1; i<=7; i++)
h2[i] = h2[i] - (a12*h1[i]);

/** Writing all in matrix **/

for ( i=1; i<=7; i++)
{
 mat2[i][1] = h1[i];
 mat2[i][2] = h2[i];
}
return(0);
```

```
}


get_ortho3(mat1,mat2)
float mat1[MAT_DIM][MAT_DIM],mat2[MAT_DIM][MAT_DIM];
{
int i;
float h1[MAT_DIM],h2[MAT_DIM],h3[MAT_DIM],h4[MAT_DIM];
float a12,a13,a23;
float t1,t2;

for ( i=1; i<=7; i++)
{
h1[i] = mat1[i][1];
h2[i] = mat1[i][2];
h3[i] = mat1[i][3];
}

t1 = t2 = 0.0;
for (i=1; i<=7; i++)
{
t1+= h1[i]*h2[i];
}
t2=get_inp(h1);
a12 = t1/t2;

for ( i=1; i<=7; i++)
h2[i] = h2[i] - (a12*h1[i]);

/** ortho. third **/

t1 = t2 =  0.0;
for (i=1; i<=7; i++)
{
t1+= h1[i]*h3[i];
}
t2=get_inp(h1);
a13 = t1/t2;

t1 = t2 =  0.0;
for (i=1; i<=7; i++)
{
t1+= h2[i]*h3[i];
}
t2=get_inp(h2);
a23 = t1/t2;

for ( i=1; i<=7; i++)
h3[i] = h3[i] - (a13*h1[i]) - (a23*h2[i]);

/** Writing all in matrix **/

for ( i=1; i<=7; i++)
{
 mat2[i][1] = h1[i];
 mat2[i][2] = h2[i];
 mat2[i][3] = h3[i];
}
return(0);
}
```

```
get_ortho4(mat1,mat2)
float mat1[MAT_DIM][MAT_DIM],mat2[MAT_DIM][MAT_DIM];
{
int i;
float h1[MAT_DIM],h2[MAT_DIM],h3[MAT_DIM],h4[MAT_DIM];
float a12,a13,a23,a14,a24,a34;
float t1,t2;

for ( i=1; i<=7; i++)
{
h1[i] = mat1[i][1];
h2[i] = mat1[i][2];
h3[i] = mat1[i][3];
h4[i] = mat1[i][4];
}

t1 = t2 = 0.0;
for (i=1; i<=7; i++)
{
t1+= h1[i]*h2[i];
}
t2=get_inp(h1);
a12 = t1/t2;

for ( i=1; i<=7; i++)
h2[i] = h2[i] - (a12*h1[i]);

/** ortho. third **/

t1 = t2 = 0.0;
for (i=1; i<=7; i++)
{
t1+= h1[i]*h3[i];
}
t2=get_inp(h1);
a13 = t1/t2;

t1 = t2 = 0.0;
for (i=1; i<=7; i++)
{
t1+= h2[i]*h3[i];
}
t2=get_inp(h2);
a23 = t1/t2;

for ( i=1; i<=7; i++)
h3[i] = h3[i] - (a13*h1[i]) - (a23*h2[i]);

/** ortho. fourth **/

t1 = t2 = 0.0;
for (i=1; i<=7; i++)
{
t1+= h1[i]*h4[i];
}
t2=get_inp(h1);
a14 = t1/t2;

t1 = t2 = 0.0;
for (i=1; i<=7; i++)
```

```
{
t1+= h2[i]*h4[i];
}
t2=get_inp(h2);
a24 = t1/t2;

t1 = t2 = 0.0;
for (i=1; i<=7; i++)
{
t1+= h3[i]*h4[i];
}
t2=get_inp(h3);
a34 = t1/t2;

for ( i=1; i<=7; i++)
h4[i] = h4[i] - (a14*h1[i]) - (a24*h2[i]) - (a34*h3[i]);

/** Writing all in matrix **/

for ( i=1; i<=7; i++)
{
 mat2[i][1] = h1[i];
 mat2[i][2] = h2[i];
 mat2[i][3] = h3[i];
 mat2[i][4] = h4[i];
}
return(0);
}


get_ortho5(mat1,mat2)
float mat1[MAT_DIM][MAT_DIM],mat2[MAT_DIM][MAT_DIM];
{
int i;
float h1[MAT_DIM],h2[MAT_DIM],h3[MAT_DIM],h4[MAT_DIM],h5[MAT_DIM];
float a12,a13,a23,a14,a24,a34,a15,a25,a35,a45;
float t1,t2;

for ( i=1; i<=7; i++)
{
h1[i] = mat1[i][1];
h2[i] = mat1[i][2];
h3[i] = mat1[i][3];
h4[i] = mat1[i][4];
h5[i] = mat1[i][5];
}

t1 = t2 = 0.0;
for (i=1; i<=7; i++)
{
t1+= h1[i]*h2[i];
}
t2=get_inp(h1);
a12 = t1/t2;

for ( i=1; i<=7; i++)
h2[i] = h2[i] - (a12*h1[i]);

/** ortho. third **/

t1 = t2 = 0.0;
for (i=1; i<=7; i++)
```

```
{
t1+= h1[i]*h3[i];
}
t2=get_inp(h1);
a13 = t1/t2;

t1 = t2 = 0.0;
for (i=1; i<=7; i++)
{
t1+= h2[i]*h3[i];
}
t2=get_inp(h2);
a23 = t1/t2;

for ( i=1; i<=7; i++)
h3[i] = h3[i] - (a13*h1[ij) - (a23*h2[i]);

/** ortho. fourth **/

t1 = t2 = 0.0;
for (i=1; i<=7; i++)
{
t1+= h1[i]*h4[i];
}
t2=get_inp(h1);
a14 = t1/t2;

t1 = t2 = 0.0;
for (i=1; i<=7; i++)
{
t1+= h2[i]*h4[i];
}
t2=get_inp(h2);
a24 = t1/t2;

t1 = t2 = 0.0;
for (i=1; i<=7; i++)
{
t1+= h3[i]*h4[i];
}
t2=get_inp(h3);
a34 = t1/t2;

for ( i=1; i<=7; i++)
h4[i] = h4[i] - (a14*h1[i]) - (a24*h2[i]) - (a34*h3[i]);

/** ortho. fifth **/

t1 = t2 = 0.0;

for (i=1; i<=7; i++)
{
t1+= h1[i]*h5[i];
}
t2=get_inp(h1);
a15 = t1/t2;

t1 = t2 = 0.0;
for (i=1; i<=7; i++)
{
t1+= h2[i]*h5[i];
}
t2=get_inp(h2);
```

```
a25 = t1/t2;

t1 = t2 =  0.0;
for (i=1; i<=7; i++)
{
t1+= h3[i]*h5[i];
}
t2=get_inp(h3);
a35 = t1/t2;

t1 = t2 =  0.0;
for (i=1; i<=7; i++)
{
t1+= h4[i]*h5[i];
}
t2=get_inp(h4);
a45 = t1/t2;

for ( i=1; i<=7; i++)
h5[i] = h5[i] - (a15*h1[i]) - (a25*h2[i]) - (a35*h3[i]) - (a45*h4[i]);

/** Writing all in matrix **/

for ( i=1; i<=7; i++)
{
 mat2[i][1] = h1[i];
 mat2[i][2] = h2[i];
 mat2[i][3] = h3[i];
 mat2[i][4] = h4[i];
 mat2[i][5] = h5[i];
}
return(0);
}



get_ortho6(mat1,mat2)
float mat1[MAT_DIM][MAT_DIM],mat2[MAT_DIM][MAT_DIM];
{
int i;
float h1[MAT_DIM],h2[MAT_DIM],h3[MAT_DIM],h4[MAT_DIM],h5[MAT_DIM],h6[MAT_DIM];
float a12,a13,a23,a14,a24,a34,a15,a25,a35,a45,a16,a26,a36,a46,a56;
float t1,t2;

for ( i=1; i<=7; i++)
{
h1[i] = mat1[i][1];
h2[i] = mat1[i][2];
h3[i] = mat1[i][3];
h4[i] = mat1[i][4];
h5[i] = mat1[i][5];
h6[i] = mat1[i][6];
}
t1 = t2 = 0.0;
for (i=1; i<=7; i++)
{
t1+= h1[i]*h2[i];
}
t2=get_inp(h1);
a12 = t1/t2;

for ( i=1; i<=7; i++)
```

```
h2[i] = h2[i] - (a12*h1[i]);

/** ortho. third **/

t1 = t2 =  0.0;
for (i=1; i<=7; i++)
{
t1+= h1[i]*h3[i];
}
t2=get_inp(h1);
a13 = t1/t2;

t1 = t2 =  0.0;
for (i=1; i<=7; i++)
{
t1+= h2[i]*h3[i];
}
t2=get_inp(h2);
a23 = t1/t2;

for ( i=1; i<=7; i++)
h3[i] = h3[i] - (a13*h1[i]) - (a23*h2[i]);

/** ortho. fourth **/

t1 = t2 =  0.0;
for (i=1; i<=7; i++)
{
t1+= h1[i]*h4[i];
}
t2=get_inp(h1);
a14 = t1/t2;

t1 = t2 =  0.0;
for (i=1; i<=7; i++)
{
t1+= h2[i]*h4[i];
}
t2=get_inp(h2);
a24 = t1/t2;

t1 = t2 =  0.0;
for (i=1; i<=7; i++)
{
t1+= h3[i]*h4[i];
}
t2=get_inp(h3);
a34 = t1/t2;

for ( i=1; i<=7; i++)
h4[i] = h4[i] - (a14*h1[i]) - (a24*h2[i]) - (a34*h3[i]);

/** ortho. fifth **/

t1 = t2 = 0.0;

for (i=1; i<=7; i++)
{
t1+= h1[i]*h5[i];
}
t2=get_inp(h1);
a15 = t1/t2;
```

```
t1 = t2 =  0.0;
for (i=1; i<=7; i++)
{
t1+= h2[i]*h5[i];
}
t2=get_inp(h2);
a25 = t1/t2;

t1 = t2 =  0.0;
for (i=1; i<=7; i++)
{
t1+= h3[i]*h5[i];
}
t2=get_inp(h3);
a35 = t1/t2;

t1 = t2 =  0.0;
for (i=1; i<=7; i++)
{
t1+= h4[i]*h5[i];
}
t2=get_inp(h4);
a45 = t1/t2;

for ( i=1; i<=7; i++)
h5[i] = h5[i] - (a15*h1[i]) - (a25*h2[i]) - (a35*h3[i]) - (a45*h4[i]);

/**  ortho. sixth  **/
t1 = t2 = 0.0;

for (i=1; i<=7; i++)
{
t1+= h1[i]*h6[i];
}
t2=get_inp(h1);
a16 = t1/t2;

t1 = t2 =  0.0;
for (i=1; i<=7; i++)
{
t1+= h2[i]*h6[i];
}
t2=get_inp(h2);
a26 = t1/t2;

t1 = t2 =  0.0;
for (i=1; i<=7; i++)
{
t1+= h3[i]*h6[i];
}
t2=get_inp(h3);
a36 = t1/t2;

t1 = t2 =  0.0;
for (i=1; i<=7; i++)
{
t1+= h4[i]*h6[i];
}
t2=get_inp(h4);
a46 = t1/t2;

t1 = t2 =  0.0;
for (i=1; i<=7; i++)
```

```
{
t1+= h5[i]*h6[i];
}
t2=get_inp(h5);
a56 = t1/t2;
for ( i=1; i<=7; i++)
h6[i] = h6[i] - (a16*h1[i]) - (a26*h2[i]) - (a36*h3[i]) - (a46*h4[i]) - (a56*h5[i]);

/** Writing all in matrix **/

for ( i=1; i<=7; i++)
{
 mat2[i][1] = h1[i];
 mat2[i][2] = h2[i];
 mat2[i][3] = h3[i];
 mat2[i][4] = h4[i];
 mat2[i][5] = h5[i];
 mat2[i][6] = h6[i];
}
return(0);
}
```

g) Subroutines added to file init_graph.c

```
view_butn1()
{
    int i;
    cpack(0x9e91cd);
    for (i=0; i<=44; i += 4)
      {
       bgnpolygon();
        v3f(view_bt[i]); v3f(view_bt[(i+1)]); v3f(view_bt[(i+2)]); v3f(view_bt[(i+3)]);
       endpolygon();
      }
    cpack(0x0);
     cmov(0.029,0.0345,-data_distance);
    charstr("Translate 1");
     cmov(0.029,0.0295,-data_distance);
    charstr("Translate 2");
     cmov(0.029,0.0245,-data_distance);
    charstr("Translate 3");
     cmov(0.029,0.0195,-data_distance);
    charstr("Rotation ");
     cmov(0.029,0.0145,-data_distance);
    charstr("Diag 1");
     cmov(0.029,0.0095,-data_distance);
    charstr("Diag 2 ");
     cmov(0.029,0.0045,-data_distance);
    charstr("Diag 3");
     cmov(0.029,-0.0005,-data_distance);
    charstr("Size");
     cmov(0.029,-0.0055,-data_distance);
     charstr("Initiation");
     cmov(0.029,-0.0105,-data_distance);
    charstr("Retrieve ");
     cmov(0.029,-0.0155,-data_distance);
```

```
    charstr("Save");
     cmov(0.029,-0.0205,-data_distance);
    charstr("End");
    return;
}



move_coll_obj(distan)
float distan[10];
{
 int i;
 long mx, my;
 extern float theta_rd[MAT_DIM];
 extern int o_move;

 while (TRUE)
      {
       mx=getvaluator(MOUSEX);
       my=getvaluator(MOUSEY);
       if ((mx>=840) && (mx<=960))
         {
          if ((901<=my) && (my<=949) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
           {
            if (getbutton(MOUSE2))
               distan[1] = 2.0;
           else
                distan[1] = -2.0;
            o_move =1;
               distan[2]= distan[3]=distan[4]=distan[5]= distan[6]=0.0;
              distan[7]=distan[8]= distan[9]=0.0;
            angg=0.0;
            r_on = 0;
            size_on = 0;
            return;
           }
          else if ((841<=my) && (my<=889) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
           {
            if (getbutton(MOUSE2))
                distan[2] = 2.0;
           else
               distan[2] = -2.0;
            o_move =1;
               distan[1]= distan[3]=distan[4]=distan[5]= distan[6]=0.0;
              distan[7]=distan[8]= distan[9]=0.0;
            angg=0.0;
            r_on = 0;
            size_on = 0;
            return;
           }
          else if ((781<=my) && (my<=829) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
           {
            if (getbutton(MOUSE2))
               distan[3] = 2.0;
           else
               distan[3] = -2.0;
            o_move =1;
              distan[1]= distan[2]=distan[4]=distan[5]= distan[6]=0.0;
             distan[7]=distan[8]= distan[9]=0.0;
            angg=0.0;
            r_on = 0;
            size_on = 0;
```

```
 return;
 }
else if ((721<=my) && (my<=769) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
 {
  if (getbutton(MOUSE2))
    angg= 1.0;
 else
    angg= -1.0;
  o_move =1;
     distan[1]= distan[2]=distan[3]=distan[4]=distan[5]=distan[6]=0.0;
   distan[7]=distan[8]= distan[9]=0.0;
 r_on = 1;
 size_on = 0;
 return;
 }
else if ((661<=my) && (my<=709) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
 {
  if (getbutton(MOUSE2))
       distan[1]= distan[2]= 2.0;
 else
       distan[1]= distan[2]= -2.0;
  o_move =1;
    distan[3]=distan[4]=distan[5]=distan[6]=0.0;
   distan[7]=distan[8]=distan[9]=0.0;
 angg=0.0;
 r_on = 0;
 size_on = 0;
 return;
 }
else if ((601<=my) && (my<=649) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
 {
   if (getbutton(MOUSE2))
       distan[2]= distan[3]= 2.0;
 else
       distan[2]= distan[3]= -2.0;
  o_move =1;
    distan[1]=distan[4]=distan[5]=distan[6]=0.0;
   distan[7]=distan[8]=distan[9]=0.0;
 angg=0.0;
 r_on = 0;
 size_on = 0;
 return;
 }
else if ((541<=my) && (my<=589) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
 {
   if (getbutton(MOUSE2))
       distan[1]= distan[3]= 2.0;
 else
       distan[1]= distan[3]= -2.0;
 o_move =1;
    distan[2]=distan[4]=distan[5]=distan[6]=0.0;
   distan[7]=distan[8]=distan[9]=0.0;
 angg=0.0;
 r_on = 0;
 size_on = 0;
 return;
 }
 else if ((481<=my) && (my<=529) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
 {
     distan[1]= distan[2]=distan[3]=distan[4]=distan[5]=distan[6]=0.0;
   distan[7]= distan[8]= distan[9]=0.0;
 angg=0.0;
 r_on = 3;
```

```
            size_on = 1;
            return;
           }
          else if ((421<=my) && (my<=469) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
           {
           o_move =1;
              distan[1]= distan[2]=distan[3]=distan[4]=distan[5]=distan[6]=0.0;
            distan[7]=distan[8]= distan[9]=0.0;
           angg=0.0;
           r_on = 2;
           size_on = 0;
           return;
           }
          else if ((361<=my) && (my<=409) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
           {
            tpon();
              retrv_collision_environment(1);
            sleep(1);
            ringbell();
            tpoff();
            o_move =1;
               distan[1]= distan[2]=distan[3]=distan[4]=distan[5]=distan[6]=0.0;
              distan[7]=distan[8]=distan[9]=0.0;
            angg=0.0;
            r_on = 0;
            size_on = 0;
            return;
           }
          else if ((301<=my) && (my<=349) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
           {
           o_move =1;
            save_current_obj();
           r_on = 0;
           size_on = 0;
           return;
           }
          else if ((241<=my) && (my<=289) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
           {
            o_move =0;
               distan[1]= distan[2]=distan[3]=distan[4]=distan[5]=distan[6]=0.0;
              distan[7]=distan[8]=distan[9]=0.0;
            angg=0.0;
            r_on = 0;
            size_on = 0;
            return;
           }

        }

      }

}


view_btn2()
{
 int i;
 extern float theta_rd[];

    cpack(0xfff68f);
    for (i=48; i<=63; i += 4)
```

```
      {
       bgnpolygon();
        v3f(view_bt[i]); v3f(view_bt[(i+1)]); v3f(view_bt[(i+2)]); v3f(view_bt[(i+3)]);
       endpolygon();
      }
    cpack(0x0);
     cmov(0.018,0.0345,-data_distance);
    charstr("Size X");
     cmov(0.018,0.0295,-data_distance);
    charstr("Size Y");
     cmov(0.018,0.0245,-data_distance);
    charstr("Size Z");
     cmov(0.018,0.0195,-data_distance);
    charstr("End size ");
draw_robot(theta_rd);
return;
}



get_size_obj(distan)
float distan[10];
{
 int i;
 long mx, my;
 extern float theta_rd[MAT_DIM];
 extern int o_move;

 while (TRUE)
      {
      mx=getvaluator(MOUSEX);
      my=getvaluator(MOUSEY);
      if ((mx>=710) && (mx<=830))
        {
        if ((901<=my) && (my<=949) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
         {
          if (getbutton(MOUSE2))
             distan[7] = 2.0;
         else
             distan[7] = -2.0;
         o_move =1;
            distan[1]= distan[2]=distan[3]=distan[4]= distan[5]=0.0;
           distan[6]=distan[8]= distan[9]=0.0;
         size_dirn = 1;
         angg=0.0;
         size_on = 1;
         return;
         }
        else if ((841<=my) && (my<=889) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
         {
         if (getbutton(MOUSE2))
             distan[7] = 2.0;
        else
             distan[7] = -2.0;
         o_move =1;
            distan[1]= distan[2]=distan[3]=distan[4]= distan[5]=0.0;
           distan[6]=distan[8]= distan[9]=0.0;
         angg=0.0;
         size_dirn = 2;
         size_on = 1;
         return;
         }
```

```
        else if ((781<=my) && (my<=829) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
         {
          if (getbutton(MOUSE2))
             distan[7] = 2.0;
          else
             distan[7] = -2.0;
          o_move =1;
            distan[1]= distan[2]=distan[3]=distan[4]= distan[5]=0.0;
           distan[6]=distan[8]= distan[9]=0.0;
          angg=0.0;
          size_dirn = 3;
          size_on = 1;
          return;
         }
        else if ((721<=my) && (my<=769) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
         {
          o_move =1;
          o_move = 1;
          r_on =0;
          size_dirn = 0;
          size_on = 0;
             distan[1]= distan[2]=distan[3]=distan[4]=distan[5]=distan[6]=0.0;
           distan[7]=distan[8]=distan[9]=0.0;
          angg = 0.0;
          return;
         }
       }
     }
}



view_butn3()
{
 int i;
 extern float theta_rd[];
    cpack(0xe6e0b0);
    for (i=0; i<=16; i += 4)
      {
       bgnpolygon();
        v3f(view_bt[i]); v3f(view_bt[(i+1)]); v3f(view_bt[(i+2)]); v3f(view_bt[(i+3)]);
       endpolygon();
      }
    cpack(0x0);
     cmov(0.029,0.0345,-data_distance);
    charstr("1x7");
     cmov(0.029,0.0295,-data_distance);
    charstr("2x7");
     cmov(0.029,0.0245,-data_distance);
    charstr("3x7");
     cmov(0.029,0.0195,-data_distance);
    charstr("4x7");
     cmov(0.029,0.0145,-data_distance);
    charstr("5x7");
draw_robot(theta_rd);
get_jacob();
return;
}



get_jacob()
```

```
{
int i,j;
extern int j_ind,jj,kk;
long mx,my;
while (TRUE)
      {
      mx=getvaluator(MOUSEX);
      my=getvaluator(MOUSEY);
      if ((mx>=840) && (mx<=960))
        {
         if ((901<=my) && (my<=949) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
          {
           j_ind = 1;
           jj = 1;
           kk =7;
           ringbell();
           return;
          }
         else if ((841<=my) && (my<=889) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
          {
           j_ind = 1;
           jj = 2;
           kk =7;
           ringbell();
           return;
          }
         else if ((781<=my) && (my<=829) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
          {
           j_ind = 1;
           jj = 3;
           kk =7;
           ringbell();
           return;
          }
         else if ((721<=my) && (my<=769) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
          {
           j_ind = 1;
           jj = 4;
           kk =7;
           ringbell();
           return;
          }
         else if ((661<=my) && (my<=709) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
          {
           j_ind = 1;
           jj = 5;
           kk =7;
           ringbell();
           return;
          }
        }
    }
}



view_butn4()
{
   int i;

    cpack(0x9e91cd);
   for (i=0; i<=44; i += 4)
```

```
      {
       bgnpolygon();
        v3f(view_bt[i]); v3f(view_bt[(i+1)]); v3f(view_bt[(i+2)]); v3f(view_bt[(i+3)]);
       endpolygon();
      }
    cpack(0x0);
     cmov(0.029,0.0345,-data_distance);
    charstr("Gain");
     cmov(0.029,0.0295,-data_distance);
    charstr("Scheme");
     cmov(0.029,0.0245,-data_distance);
    charstr("Radius");
     cmov(0.029,0.0195,-data_distance);
   charstr("E.E. Path ");
     cmov(0.029,0.0145,-data_distance);
    charstr("Trajectory");
     cmov(0.029,0.0095,-data_distance);
    charstr("Color");
     cmov(0.029,0.0045,-data_distance);
    charstr("Store");
     cmov(0.029,-0.0005,-data_distance);
    charstr("S.G.time");
     cmov(0.029,-0.0055,-data_distance);
    charstr("Limit");
     cmov(0.029,-0.0105,-data_distance);
    charstr("Teleop");
     cmov(0.029,-0.0155,-data_distance);
    charstr("Coll.Av.");
     cmov(0.029,-0.0205,-data_distance);
    charstr("Exit");
   return;
}



get_case(chose)
int chose[2][2];
{
long mx, my;
while (TRUE)
      {
       mx=getvaluator(MOUSEX);
       my=getvaluator(MOUSEY);
       if ((mx>=840) && (mx<=960))
         {
          if ((901<=my) && (my<=949) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
           {
            chose[0][0] = 1;
            chose[1][1] = 0;
           return;
           }
          else if ((841<=my) && (my<=889) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
           {
            chose[0][0] = 2;
            chose[1][1] = 0;
           return;
           }
          else if ((781<=my) && (my<=829) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
           {
            chose[0][0] = 3;
            chose[1][1] = 0;
           return;
```

```
        }
       else if ((721<=my) && (my<=769) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
        {
          chose[0][0] = 5;
          chose[1][1] = 0;
         return;
        }
       else if ((661<=my) && (my<=709) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
        {
          chose[0][0] = 6;
          chose[1][1] = 0;
         return;
        }
       else if ((601<=my) && (my<=649) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
        {
          chose[0][0] = 7;
          chose[1][1] = 0;
         return;
        }
       else if ((541<=my) && (my<=589) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
        {
          chose[0][0] = 8;
          chose[1][1] = 0;
         return;
        }
        else if ((481<=my) && (my<=529) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
        {
          chose[0][0] = 9;
          chose[1][1] = 0;
         return;
        }
       else if ((421<=my) && (my<=469) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
        {
          chose[0][0] = 10;
          chose[1][1] = 0;
         return;
        }
       else if ((361<=my) && (my<=409) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
        {
          chose[0][0] = 11;
          chose[1][1] = 0;
         return;
        }
       else if ((301<=my) && (my<=349) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
        {
          chose[0][0] = 12;
          chose[1][1] = 0;
         return;
        }
       else if ((241<=my) && (my<=289) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
        {
          chose[0][0] = 14;
          chose[1][1] = 1;
         return;
        }
     }
   }
}


draw_all()
```

```
{
 extern int camera_index, theta_count, e_traj_count;
 extern long color_back;
 extern float axis0[3], x_axis[3], y_axis[3], z_axis[3];
 extern int show_path,robot_index;
 extern float theta_all[][MAT_DIM], view_mat[][4][4], h_transf[5][5], trajectory[TSTEPS][3];
 extern float view_mat[][4][4];
 float theta[MAT_DIM], h_b[5][5];
 int i, j;

 /*czclear(0x000000,0x0);*/
czclear(0xffffff,0x0);
 for (i=1; i<=theta_count; i++)
  {
   transf(theta_all[i],h_transf);
  for (j=1; j<=7; j++)
       theta[j]=theta_all[i][j]/0.017453293;
   theta[0] = theta_all[i][0];
   theta[8] = theta_all[i][8];
    theta[9] = theta_all[i][9]/0.017453293;
   theta[10] = theta_all[i][10];

   pushmatrix();
    loadmatrix(view_mat[camera_index]);
    use_camera(camera_index, theta, theta_all[i][8], theta_all[i][9], theta_all[i][10], h_transf, h_b);
    draw_k2107_robot(theta);
   popmatrix();
  }
pushmatrix();
 if (e_traj_count > 0)
  {
   loadmatrix(view_mat[camera_index]);
  cpack(0x0);
   for (i=0; i<e_traj_count; i++)
   {
   j=i+1;
    bgnline();
     v3f(trajectory[i]); v3f(trajectory[j]);
    endline();
   }
  }
 if ( show_path == 1 )
     {
      cpack(0x0);
      bgnline();
       v3f(axis0); v3f(x_axis);
      endline();
      bgnline();
       v3f(axis0); v3f(y_axis);
      endline();
      bgnline();
       v3f(axis0); v3f(z_axis);
      endline();
        cmov(x_axis[0]+1,x_axis[1],x_axis[2]);
      charstr("X");
        cmov(y_axis[0],y_axis[1]+1,y_axis[2]);
      charstr("Y");
        cmov(z_axis[0],z_axis[1],z_axis[2]+1);
      charstr("Z");
     }
 popmatrix();
 draw_collision_objects( view_mat[camera_index] );
swapbuffers();
```

```
  czclear(color_back, 0x70000);
  return;
}



store_data(temp)
float temp[20];
{
int i,j;
extern FILE *fil_name[5];

fprintf(fil_name[1],"%6.3f\t ",temp[17]);
for(i=1; i<=7; i++)
{
fprintf(fil_name[1],"%6.3f\t",temp[i]);
}
for (i=11; i<=16; i++)
{
fprintf(fil_name[1],"%6.3f\t ",temp[i]);
}
fprintf(fil_name[1],"%6.3f\t ",temp[18]);
fprintf(fil_name[1],"%6.3f ",temp[19]);
fprintf(fil_name[1],"\n");
return;
}
```

h) File objenvs.c added to the robot code

```
#include <stdio.h>
#include <math.h>
#include <gl/gl.h>

#define RADtoDEG 57.2?577951
#define TPATHS 370              /* Total paths for 3-D Map plotting      */
#define TSTEPS 2500             /* The max. steps of trajectory          */
#define MAT_DIM 10
#define pi 3.1415927

float ogm[50][9],omm[50][6],oang;
int ocl;
float x[25],y[25],z[25];
float ox[25],oy[25],oz[25];
 extern struct d_minimum
        { float d_min;     float Sd_pnt_edge[3];      float Sd_pnt_link[3];   };

  extern struct math_models
        { float math[10];
     float graphic[10];      };

  extern struct edge_proto
        { float rel_weight;
          struct d_minimum edge[100]; };
```

```
  extern struct link_proto
        { struct edge_proto link[10]; };

  extern struct collision_info
        {  int is_active;
           int show_sd;
           int transparent;
           int robot_transparent;
           int show_safety_dist;
           int total_links;
           int total_edges;
           int max_links;
      int max_edges;
           int which_is_worst;
           float absolute_min_d;
      int monitor;
           int moving_away;
           float Dstop;
           struct math_models object_model[100];
           struct math_models arm_model[10];
           struct link_proto feedback;
           int record_to_file;
           FILE *collision_data_file;
       char collision_file_name[150];
      int rec_win_dat;
       int out_graphics;
         };

   extern struct collision_info collision;

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

ret_coll_env()
{
 extern int o_move;
 float ds1[12];
int i=0,j=0;
extern int ll;
extern int r_on;
extern float angg;
extern float angs;
extern int size_on;

 for( i=0; i<=12; i++)
   ds1[i] = 0.0;
   if (o_move == 1)
    move_coll_obj( ds1 );
  for( i=8; i<=12; i++)
   ds1[i] = 0.0;

/********* Update new object position  *********/

 if( r_on == 0 )
   {
    move_obj(ds1);
   }

/********* Update new object orientation *********/

 else if ( r_on == 1)
    {
     angs += angg;
      angs = angs*(pi/180.0);
```

```
      collision.total_edges = 11;
      rot_obj(angs,11);
      angs = angs/(pi/180.0);
     }

/********** Get object to initial position ****************/

 else if ( r_on == 2 )
    {
      init_obj();
     angs = 0.0;
    }

/************ Size the object ******************************/

else if ( r_on == 3 )
    {
    while( size_on == 1)
    {
     view_btn2();
      get_size_obj(ds1);
     size_obj(ds1);
    }
    }
   return;
}

/************************ Routine to save current object *****************/

save_current_obj()
{
   extern FILE *collision_data_file;
  FILE *file_name;
  int i=0, j=0;
  extern int 11;
  extern float angs;
  char buffer[20];

   tpon();
   system("clear");
   printf(" Give name of the file you want to store the data in :\n");
   pop_cursor();
   gets(buffer);
   if (0==(file_name=fopen(buffer,"w")))
    {
      printf("Can't save the data to the file!\n");
     sleep(1);
     ringbell();
     tpoff();
      return;
     }

/************* Save the number of edges ***********************/

  fprintf(file_name,"%d",11);
  fprintf(file_name,"\n");

/************* Save the math model  ***************************/

for ( i=1; i<=11; i++ )
            {
             for ( j=1; j<=6; j++ )
               {
```

```
                fprintf(file_name,"%f ",  collision.object_model[i].math[j] );
              }
          fprintf(file_name,"\n");
            }

/************ Save the graphical model **********************/

 for ( i=1; i<=11; i++ )
            {
             for ( j=1; j<=9; j++ )
               {
                 fprintf(file_name,"%f ",  collision.object_model[i].graphic[j] );
               }
         fprintf(file_name,"\n");
            }
     for ( j=1; j<=25; j++ )
               {
                fprintf(file_name,"%f     %f     %f ",x[j],y[j],z[j]);
           fprintf(file_name,"\n");
               }
     fprintf(file_name,"%f ", angs);
     fprintf(file_name,"\n");

/************** Close file ********************/

fclose(file_name);
printf("FILE SAVED\n");
sleep(1);
ringbell();
tpoff();
return;
}

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


/********************* Routine to rotate objects ******************************/

rot_obj(angle,edges)
float angle;
int edges;
{
 float temp[3];
float c2,s2,ang;
ang = angle;
c2 = cos(angle);
s2 = sin(angle);

if ( edges == 4 ) /* rotation for frame */
{
temp[1] = ((collision.object_model[2].graphic[1] + collision.object_model[4].graphic[1])/2.0);
temp[2] = ((collision.object_model[2].graphic[2] + collision.object_model[4].graphic[2])/2.0);
temp[3] = ((collision.object_model[2].graphic[3] + collision.object_model[4].graphic[3])/2.0);


collision.object_model[1].graphic[1] = ((s2*z[1]) + temp[1]) ;
collision.object_model[1].graphic[2] = (temp[2]) ;
collision.object_model[1].graphic[3] = ((c2*z[1]) + temp[3]) ;
collision.object_model[1].graphic[7] = collision.object_model[1].graphic[7];
collision.object_model[1].graphic[8] = collision.object_model[1].graphic[8];
collision.object_model[1].graphic[9] = collision.object_model[1].graphic[9];
collision.object_model[2].graphic[1] = collision.object_model[2].graphic[1];
collision.object_model[2].graphic[2] = collision.object_model[2].graphic[2];
```

```
collision.object_model[2].graphic[3] = collision.object_model[2].graphic[3];
collision.object_model[2].graphic[7] = collision.object_model[2].graphic[7];
collision.object_model[2].graphic[8] = collision.object_model[2].graphic[8];
collision.object_model[2].graphic[9] = collision.object_model[2].graphic[9];
collision.object_model[3].graphic[1] = ((s2*z[3]) + temp[1]) ;
collision.object_model[3].graphic[2] = ( temp[2]) ;
collision.object_model[3].graphic[3] = ((c2*z[3]) + temp[3]) ;
collision.object_model[3].graphic[7] = collision.object_model[3].graphic[7];
collision.object_model[3].graphic[8] = collision.object_model[3].graphic[8];
collision.object_model[3].graphic[9] = collision.object_model[3].graphic[9];
collision.object_model[4].graphic[1] = collision.object_model[4].graphic[1];
collision.object_model[4].graphic[2] = collision.object_model[4].graphic[2];
collision.object_model[4].graphic[3] = collision.object_model[4].graphic[3];
collision.object_model[4].graphic[7] = collision.object_model[4].graphic[7];
collision.object_model[4].graphic[8] = collision.object_model[4].graphic[8];
collision.object_model[4].graphic[9] = collision.object_model[4].graphic[9];

 collision.object_model[1].graphic[4] = (ogm[1][4])+(ang * (180.0/pi));
 collision.object_model[1].graphic[5] = collision.object_model[1].graphic[5];
 collision.object_model[1].graphic[6] = collision.object_model[1].graphic[6];
 collision.object_model[2].graphic[4] = collision.object_model[2].graphic[4];
 collision.object_model[2].graphic[5] = (90.0+(ang * (180.0/pi)));
 collision.object_model[2].graphic[6] = collision.object_model[2].graphic[6];
 collision.object_model[3].graphic[4] = (ogm[3][4])+(ang * (180.0/pi));
 collision.object_model[3].graphic[5] = collision.object_model[3].graphic[5];
 collision.object_model[3].graphic[6] = collision.object_model[3].graphic[6];
 collision.object_model[4].graphic[4] = collision.object_model[4].graphic[4];
 collision.object_model[4].graphic[5] = (-90.0+ (ang * (180.0/pi)));
 collision.object_model[4].graphic[6] = collision.object_model[4].graphic[6];

 collision.object_model[1].math[1] = ((s2*z[1]) + temp[1]) ;
 collision.object_model[1].math[2] = collision.object_model[1].math[2];
 collision.object_model[1].math[3] = ((c2*z[1]) + temp[3]);
 collision.object_model[2].math[1] = ((s2*z[1]) + temp[1]);
 collision.object_model[2].math[2] = collision.object_model[2].math[2];
 collision.object_model[2].math[3] = ((c2*z[1]) + temp[3]);
 collision.object_model[3].math[1] = ((s2*z[3]) + temp[1]);
 collision.object_model[3].math[2] = collision.object_model[3].math[2];
 collision.object_model[3].math[3] = ((c2*z[3]) + temp[3]);
 collision.object_model[4].math[1] = ((s2*z[3]) + temp[1]);
 collision.object_model[4].math[2] = collision.object_model[4].math[2];
 collision.object_model[4].math[3] = ((c2*z[3]) + temp[3]);
 collision.object_model[1].math[4] = collision.object_model[2].math[1] - collision.object_model[1].math[1];
 collision.object_model[1].math[5] = collision.object_model[2].math[2] - collision.object_model[1].math[2];
 collision.object_model[1].math[6] = collision.object_model[2].math[3] - collision.object_model[1].math[3];
 collision.object_model[2].math[4] = collision.object_model[3].math[1] - collision.object_model[2].math[1];
 collision.object_model[2].math[5] = collision.object_model[3].math[2] - collision.object_model[2].math[2];
 collision.object_model[2].math[6] = collision.object_model[3].math[3] - collision.object_model[2].math[3];
 collision.object_model[3].math[4] = collision.object_model[4].math[1] - collision.object_model[3].math[1];
 collision.object_model[3].math[5] = collision.object_model[4].math[2] - collision.object_model[3].math[2];
 collision.object_model[3].math[6] = collision.object_model[4].math[3] - collision.object_model[3].math[3];
 collision.object_model[4].math[4] = collision.object_model[1].math[1] - collision.object_model[4].math[1];
 collision.object_model[4].math[5] = collision.object_model[1].math[2] - collision.object_model[4].math[2];
 collision.object_model[4].math[6] = collision.object_model[1].math[3] - collision.object_model[4].math[3];
}
else if ( edges == 12 ) /* rotation for cube */
{
temp[1] = (((collision.object_model[9].graphic[1] + collision.object_model[10].graphic[1])/2.0) +
((collision.object_model[11].graphic[1] + collision.object_model[12].graphic[1])/2.0))/2.0;
temp[2] = (((collision.object_model[9].graphic[2] + collision.object_model[10].graphic[2])/2.0) +
((collision.object_model[11].graphic[2] + collision.object_model[12].graphic[2])/2.0))/2.0;
temp[3] = (((collision.object_model[9].graphic[3] + collision.object_model[10].graphic[3])/2.0) +
((collision.object_model[11].graphic[3] + collision.object_model[12].graphic[3])/2.0))/2.0;
```

```
/* Graphic model calculations */

collision.object_model[1].graphic[1] = ((c2*x[13]) + temp[1]);
collision.object_model[1].graphic[2] = (y[13] + temp[2]);
collision.object_model[1].graphic[3] = ((-s2*x[13]) + temp[3]);
collision.object_model[1].graphic[4] = collision.object_model[1].graphic[4];
collision.object_model[1].graphic[5] = (90.0 + (ang * (180.0/pi)));
collision.object_model[1].graphic[6] = collision.object_model[1].graphic[6];
collision.object_model[1].graphic[7] = collision.object_model[1].graphic[7];
collision.object_model[1].graphic[8] = collision.object_model[1].graphic[8];
collision.object_model[1].graphic[9] = collision.object_model[1].graphic[9];

collision.object_model[2].graphic[1] = ((c2*x[14]) + (s2*z[14]) + temp[1]);
collision.object_model[2].graphic[2] = (temp[2]);
collision.object_model[2].graphic[3] = ((-s2*x[14]) + (c2*z[14]) + temp[3]);
collision.object_model[2].graphic[4] = (ogm[2][4])+(ang * (180.0/pi));
collision.object_model[2].graphic[5] = collision.object_model[2].graphic[5];
collision.object_model[2].graphic[6] = collision.object_model[2].graphic[6];
collision.object_model[2].graphic[7] = collision.object_model[2].graphic[7];
collision.object_model[2].graphic[8] = collision.object_model[2].graphic[8];
collision.object_model[2].graphic[9] = collision.object_model[2].graphic[9];

collision.object_model[3].graphic[1] = ((c2*x[15]) + temp[1]);
collision.object_model[3].graphic[2] = (y[15] + temp[2]);
collision.object_model[3].graphic[3] = ((-s2*x[15]) + temp[3]);
collision.object_model[3].graphic[4] = collision.object_model[3].graphic[4];
collision.object_model[3].graphic[5] = (-90.0 + (ang * (180.0/pi)));
collision.object_model[3].graphic[6] = collision.object_model[3].graphic[6];
collision.object_model[3].graphic[7] = collision.object_model[3].graphic[7];
collision.object_model[3].graphic[8] = collision.object_model[3].graphic[8];
collision.object_model[3].graphic[9] = collision.object_model[3].graphic[9];

collision.object_model[4].graphic[1] = ((c2*x[16]) + (s2*z[16]) + temp[1]);
collision.object_model[4].graphic[2] = (temp[2]);
collision.object_model[4].graphic[3] = ((-s2*x[16]) + (c2*z[16]) + temp[3]);
collision.object_model[4].graphic[4] = (ogm[4][4])+(ang * (180.0/pi));
collision.object_model[4].graphic[5] = collision.object_model[4].graphic[5];
collision.object_model[4].graphic[6] = collision.object_model[4].graphic[6];
collision.object_model[4].graphic[7] = collision.object_model[4].graphic[7];
collision.object_model[4].graphic[8] = collision.object_model[4].graphic[8];
collision.object_model[4].graphic[9] = collision.object_model[4].graphic[9];

collision.object_model[5].graphic[1] = ((c2*x[17]) + temp[1]);
collision.object_model[5].graphic[2] = (y[17] + temp[2]);
collision.object_model[5].graphic[3] = ((-s2*x[17]) + temp[3]);
collision.object_model[5].graphic[4] = collision.object_model[5].graphic[4];
collision.object_model[5].graphic[5] = (90.0 + (ang * (180.0/pi)));
collision.object_model[5].graphic[6] = collision.object_model[5].graphic[6];
collision.object_model[5].graphic[7] = collision.object_model[5].graphic[7];
collision.object_model[5].graphic[8] = collision.object_model[5].graphic[8];
collision.object_model[5].graphic[9] = collision.object_model[5].graphic[9];

collision.object_model[6].graphic[1] = ((c2*x[18]) + (s2*z[18]) + temp[1]);
collision.object_model[6].graphic[2] = (temp[2]);
collision.object_model[6].graphic[3] = ((-s2*x[18]) + (c2*z[18]) + temp[3]);
collision.object_model[6].graphic[4] = (ogm[6][4])+(ang * (180.0/pi));
collision.object_model[6].graphic[5] = collision.object_model[6].graphic[5];
collision.object_model[6].graphic[6] = collision.object_model[6].graphic[6];
collision.object_model[6].graphic[7] = collision.object_model[6].graphic[7];
collision.object_model[6].graphic[8] = collision.object_model[6].graphic[8];
collision.object_model[6].graphic[9] = collision.object_model[6].graphic[9];
```

```
collision.object_model[7].graphic[1] = ((c2*x[19]) + temp[1]);
collision.object_model[7].graphic[2] = (y[19] + temp[2]);
collision.object_model[7].graphic[3] = ((-s2*x[19]) + temp[3]);
collision.object_model[7].graphic[4] = collision.object_model[7].graphic[4];
collision.object_model[7].graphic[5] = (-90.0 + (ang * (180.0/pi)));
collision.object_model[7].graphic[6] = collision.object_model[7].graphic[6];
collision.object_model[7].graphic[7] = collision.object_model[7].graphic[7];
collision.object_model[7].graphic[8] = collision.object_model[7].graphic[8];
collision.object_model[7].graphic[9] = collision.object_model[7].graphic[9];

collision.object_model[8].graphic[1] = ((c2*x[20]) + (s2*z[20]) + temp[1]);
collision.object_model[8].graphic[2] = (temp[2]);
collision.object_model[8].graphic[3] = ((-s2*x[20]) + (c2*z[20]) + temp[3]);
collision.object_model[8].graphic[4] = (ogm[8][4])+(ang * (180.0/pi));
collision.object_model[8].graphic[5] = collision.object_model[8].graphic[5];
collision.object_model[8].graphic[6] = collision.object_model[8].graphic[6];
collision.object_model[8].graphic[7] = collision.object_model[8].graphic[7];
collision.object_model[8].graphic[8] = collision.object_model[8].graphic[8];
collision.object_model[8].graphic[9] = collision.object_model[8].graphic[9];

collision.object_model[9].graphic[1] = ((s2*z[21]) + temp[1]);
collision.object_model[9].graphic[2] = (y[21] + temp[2]);
collision.object_model[9].graphic[3] = ((c2*z[21]) + temp[3]);
collision.object_model[9].graphic[4] = collision.object_model[9].graphic[4];
collision.object_model[9].graphic[5] = (0.0) - (ang * (180.0/pi));
collision.object_model[9].graphic[6] = collision.object_model[9].graphic[6];
collision.object_model[9].graphic[7] = collision.object_model[9].graphic[7];
collision.object_model[9].graphic[8] = collision.object_model[9].graphic[8];
collision.object_model[9].graphic[9] = collision.object_model[9].graphic[9];

collision.object_model[10].graphic[1] = ((s2*z[22]) + temp[1]);
collision.object_model[10].graphic[2] = (y[22] + temp[2]);
collision.object_model[10].graphic[3] = ((c2*z[22]) + temp[3]);
collision.object_model[10].graphic[4] = collision.object_model[10].graphic[4];
collision.object_model[10].graphic[5] = (0.0)+(ang * (180.0/pi));
collision.object_model[10].graphic[6] = collision.object_model[10].graphic[6];
collision.object_model[10].graphic[7] = collision.object_model[10].graphic[7];
collision.object_model[10].graphic[8] = collision.object_model[10].graphic[8];
collision.object_model[10].graphic[9] = collision.object_model[10].graphic[9];

collision.object_model[11].graphic[1] = ((s2*z[23]) + temp[1]);
collision.object_model[11].graphic[2] = (y[23] + temp[2]);
collision.object_model[11].graphic[3] = ((c2*z[23]) + temp[3]);
collision.object_model[11].graphic[4] = collision.object_model[11].graphic[4];
collision.object_model[11].graphic[5] = (0.0)+(ang * (180.0/pi));
collision.object_model[11].graphic[6] = collision.object_model[11].graphic[6];
collision.object_model[11].graphic[7] = collision.object_model[11].graphic[7];
collision.object_model[11].graphic[8] = collision.object_model[11].graphic[8];
collision.object_model[11].graphic[9] = collision.object_model[11].graphic[9];

collision.object_model[12].graphic[1] = ((s2*z[24]) + temp[1]);
collision.object_model[12].graphic[2] = (y[24] + temp[2]);
collision.object_model[12].graphic[3] = ((c2*z[24]) + temp[3]);
collision.object_model[12].graphic[4] = collision.object_model[12].graphic[4];
collision.object_model[12].graphic[5] = (0.0)+(ang * (180.0/pi));
collision.object_model[12].graphic[6] = collision.object_model[12].graphic[6];
collision.object_model[12].graphic[7] = collision.object_model[12].graphic[7];
collision.object_model[12].graphic[8] = collision.object_model[12].graphic[8];
collision.object_model[12].graphic[9] = collision.object_model[12].graphic[9];

/* Math model calculations */

collision.object_model[1].math[1] =collision.object_model[10].math[1] = ((c2*x[1]) + (s2*z[1]) + temp[1]);
```

```
collision.object_model[1].math[2] =collision.object_model[10].math[2] = (y[1] + temp[2]);
collision.object_model[1].math[3] =collision.object_model[10].math[3] = ((-s2*x[1]) + (c2*z[1]) + temp[3]);
collision.object_model[2].math[1] =collision.object_model[11].math[1] = ((c2*x[2]) + (s2*z[2]) + temp[1]);
collision.object_model[2].math[2] =collision.object_model[11].math[2] = (y[2] + temp[2]);
collision.object_model[2].math[3] =collision.object_model[11].math[3] = ((-s2*x[2]) + (c2*z[2]) + temp[3]);
collision.object_model[3].math[1] =collision.object_model[12].math[1] = ((c2*x[3]) + (s2*z[3]) + temp[1]);
collision.object_model[3].math[2] =collision.object_model[12].math[2] = (y[3] + temp[2]);
collision.object_model[3].math[3] =collision.object_model[12].math[3] = ((-s2*x[3]) + (c2*z[3]) + temp[3]);
collision.object_model[4].math[1] = ((c2*x[4]) + (s2*z[4]) + temp[1]);
collision.object_model[4].math[2] = (y[4] + temp[2]);
collision.object_model[4].math[3] = ((-s2*x[4]) + (c2*z[4]) + temp[3]);
collision.object_model[5].math[1] = ((c2*x[5]) + (s2*z[5]) + temp[1]);
collision.object_model[5].math[2] = (y[5] + temp[2]);
collision.object_model[5].math[3] = ((-s2*x[5]) + (c2*z[5]) + temp[3]);
collision.object_model[6].math[1] = ((c2*x[6]) + (s2*z[6]) + temp[1]);
collision.object_model[6].math[2] = (y[6] + temp[2]);
collision.object_model[6].math[3] = ((-s2*x[6]) + (c2*z[6]) + temp[3]);
collision.object_model[7].math[1] = ((c2*x[7]) + (s2*z[7]) + temp[1]);
collision.object_model[7].math[2] = (y[7] + temp[2]);
collision.object_model[7].math[3] = ((-s2*x[7]) + (c2*z[7]) + temp[3]);
collision.object_model[8].math[1] =collision.object_model[9].math[1] = ((c2*x[8]) + (s2*z[8]) + temp[1]);
collision.object_model[8].math[2] =collision.object_model[9].math[2] = (y[8] + temp[2]);
collision.object_model[8].math[3] =collision.object_model[9].math[3] = ((-s2*x[8]) + (c2*z[8]) + temp[3]);

collision.object_model[1].math[4] = collision.object_model[2].math[1] - collision.object_model[1].math[1];
collision.object_model[1].math[5] = collision.object_model[2].math[2] - collision.object_model[1].math[2];
collision.object_model[1].math[6] = collision.object_model[2].math[3] - collision.object_model[1].math[3];
collision.object_model[2].math[4] = collision.object_model[3].math[1] - collision.object_model[2].math[1];
collision.object_model[2].math[5] = collision.object_model[3].math[2] - collision.object_model[2].math[2];
collision.object_model[2].math[6] = collision.object_model[3].math[3] - collision.object_model[2].math[3];
collision.object_model[3].math[4] = collision.object_model[4].math[1] - collision.object_model[3].math[1];
collision.object_model[3].math[5] = collision.object_model[4].math[2] - collision.object_model[3].math[2];
collision.object_model[3].math[6] = collision.object_model[4].math[3] - collision.object_model[3].math[3];
collision.object_model[4].math[4] = collision.object_model[1].math[1] - collision.object_model[4].math[1];
collision.object_model[4].math[5] = collision.object_model[1].math[2] - collision.object_model[4].math[2];
collision.object_model[4].math[6] = collision.object_model[1].math[3] - collision.object_model[4].math[3];
collision.object_model[5].math[4] = collision.object_model[6].math[1] - collision.object_model[5].math[1];
collision.object_model[5].math[5] = collision.object_model[6].math[2] - collision.object_model[5].math[2];
collision.object_model[5].math[6] = collision.object_model[6].math[3] - collision.object_model[5].math[3];
collision.object_model[6].math[4] = collision.object_model[7].math[1] - collision.object_model[6].math[1];
collision.object_model[6].math[5] = collision.object_model[7].math[2] - collision.object_model[6].math[2];
collision.object_model[6].math[6] = collision.object_model[7].math[3] - collision.object_model[6].math[3];
collision.object_model[7].math[4] = collision.object_model[8].math[1] - collision.object_model[7].math[1];
collision.object_model[7].math[5] = collision.object_model[8].math[2] - collision.object_model[7].math[2];
collision.object_model[7].math[6] = collision.object_model[8].math[3] - collision.object_model[7].math[3];
collision.object_model[8].math[4] = collision.object_model[5].math[1] - collision.object_model[8].math[1];
collision.object_model[8].math[5] = collision.object_model[5].math[2] - collision.object_model[8].math[2];
collision.object_model[8].math[6] = collision.object_model[5].math[3] - collision.object_model[8].math[3];
collision.object_model[9].math[4] = collision.object_model[4].math[1] - collision.object_model[8].math[1];
collision.object_model[9].math[5] = collision.object_model[4].math[2] - collision.object_model[8].math[2];
collision.object_model[9].math[6] = collision.object_model[4].math[3] - collision.object_model[8].math[3];
collision.object_model[10].math[4] = collision.object_model[5].math[1] - collision.object_model[1].math[1];
collision.object_model[10].math[5] = collision.object_model[5].math[2] - collision.object_model[1].math[2];
collision.object_model[10].math[6] = collision.object_model[5].math[3] - collision.object_model[1].math[3];
collision.object_model[11].math[4] = collision.object_model[6].math[1] - collision.object_model[2].math[1];
collision.object_model[11].math[5] = collision.object_model[6].math[2] - collision.object_model[2].math[2];
collision.object_model[11].math[6] = collision.object_model[6].math[3] - collision.object_model[2].math[3];
collision.object_model[12].math[4] = collision.object_model[7].math[1] - collision.object_model[3].math[1];
collision.object_model[12].math[5] = collision.object_model[7].math[2] - collision.object_model[3].math[2];
collision.object_model[12].math[6] = collision.object_model[7].math[3] - collision.object_model[3].math[3];
}

else        /* if other objects neglect rotation */
```

```
{
ang = 0.0;
}
return;
}
/*****************************************************************************/


/***************************** Initialize objects.*****************************/

init_obj()
{
int i=0,j=0;
extern int l1;
extern float angs;

 collision.total_edges = l1;
      for ( i=1; i<=l1; i++ )
              {
              for ( j=1; j<=6; j++ )
                {
                  collision.object_model[i].math[j] = omm[i][j];
                }
              }
   for ( i=1; i<=l1; i++ )
              {
              for ( j=1; j<=9; j++ )
                {
                      collision.object_model[i].graphic[j] = ogm[i][j];
                }
              }
      for ( j=1; j<=25; j++ )
                {
                  x[j] = ox[j];
             y[j] = oy[j];
             z[j] = oz[j];
                }
   angs = oang;
return;
}
/*****************************************************************************/


/************************ Move the object *********************************/

move_obj(ds)
float ds[12];
{
int i=0,j=0;
extern int l1;

collision.total_edges = l1;
      for ( i=1; i<=l1; i++ )
              {
              for ( j=1; j<=6; j++ )
                {
                  collision.object_model[i].math[j] += ds[j];
                }
              }

   for ( i=1; i<=l1; i++ )
              {
              for ( j=1; j<=9; j++ )
```

```
            {
                collision.object_model[i].graphic[j] += ds[j];
            }
        }
return;
}
/*****************************************************************************/

/************************* Sizing the object ********************************/

size_obj( ds)
float ds[12];
{
int i=0,j=0;
extern int ll;
extern int size_dirn;

collision.total_edges = ll;

/************** Size increase for frame *********/

if ( ll == 4)
 {
  if ( size_dirn == 1) /*   X-dirn size increase  */
   {
    ringbell();
   }
  else if ( size_dirn == 2) /*   Y-dirn size increase  */
   {
     collision.object_model[1].math[2] -= (ds[7]/2.0);
     collision.object_model[2].math[2] += (ds[7]/2.0);
     collision.object_model[3].math[2] += (ds[7]/2.0);
     collision.object_model[4].math[2] -= (ds[7]/2.0);
     collision.object_model[1].math[5] += (ds[7]);
     collision.object_model[3].math[5] -= (ds[7]);

     collision.object_model[2].graphic[2] += (ds[7]/2.0);
     collision.object_model[4].graphic[2] -= (ds[7]/2.0);
     collision.object_model[1].graphic[7] += ds[7];
     collision.object_model[3].graphic[7] += ds[7];
   }
  else if ( size_dirn == 3) /*   Z-dirn size increase  */
   {
     collision.object_model[1].math[3] -= (ds[7]/2.0);
     collision.object_model[2].math[3] -= (ds[7]/2.0);
     collision.object_model[3].math[3] += (ds[7]/2.0);
     collision.object_model[4].math[3] += (ds[7]/2.0);
     collision.object_model[2].math[6] += (ds[7]);
     collision.object_model[4].math[6] -= (ds[7]);

     collision.object_model[1].graphic[3] -= (ds[7]/2.0);
     collision.object_model[3].graphic[3] += (ds[7]/2.0);
     collision.object_model[2].graphic[7] += ds[7];
     collision.object_model[4].graphic[7] += ds[7];
   }

    x[1] = (collision.object_model[1].graphic[1])-((collision.object_model[2].graphic[1] +
collision.object_model[4].graphic[1])/2.0);
    y[1] = (collision.object_model[1].graphic[2])-((collision.object_model[2].graphic[2] +
collision.object_model[4].graphic[2])/2.0);
    z[1] = (collision.object_model[1].graphic[3])-((collision.object_model[2].graphic[3] +
collision.object_model[4].graphic[3])/2.0);
```

```
    x[2] = (collision.object_model[2].graphic[1])-((collision.object_model[2].graphic[1] +
collision.object_model[4].graphic[1])/2.0);
    y[2] = (collision.object_model[2].graphic[2])-((collision.object_model[2].graphic[2] +
collision.object_model[4].graphic[2])/2.0);
    z[2] = (collision.object_model[2].graphic[3])-((collision.object_model[2].graphic[3] +
collision.object_model[4].graphic[3])/2.0);


    x[3] = (collision.object_model[3].graphic[1])-((collision.object_model[2].graphic[1] +
collision.object_model[4].graphic[1])/2.0);
    y[3] = (collision.object_model[3].graphic[2])-((collision.object_model[2].graphic[2] +
collision.object_model[4].graphic[2])/2.0);
    z[3] = (collision.object_model[3].graphic[3])-((collision.object_model[2].graphic[3] +
collision.object_model[4].graphic[3])/2.0);

    x[4] = (collision.object_model[4].graphic[1])-((collision.object_model[2].graphic[1] +
collision.object_model[4].graphic[1])/2.0);
    y[4] = (collision.object_model[4].graphic[2])-((collision.object_model[2].graphic[2] +
collision.object_model[4].graphic[2])/2.0);
    z[4] = (collision.object_model[4].graphic[3])-((collision.object_model[2].graphic[3] +
collision.object_model[4].graphic[3])/2.0);

  for ( j = 5; j<=25; j++)
  {
   x[j] = 0.0;
   y[j] = 0.0;
   z[j] = 0.0;
  }
}

/************************ Size increase for cube ******************************/

else if (l1 == 12)
{
  if ( size_dirn == 1) /*   X-dirn size increase  */
  {
     collision.object_model[1].math[1]  -= (ds[7]/2.0);
     collision.object_model[2].math[1]  -= (ds[7]/2.0);
     collision.object_model[3].math[1]  -= (ds[7]/2.0);
     collision.object_model[4].math[1]  -= (ds[7]/2.0);
     collision.object_model[5].math[1]  += (ds[7]/2.0);
     collision.object_model[6].math[1]  += (ds[7]/2.0);
     collision.object_model[7].math[1]  += (ds[7]/2.0);
     collision.object_model[8].math[1]  += (ds[7]/2.0);
     collision.object_model[9].math[1]  += (ds[7]/2.0);
     collision.object_model[10].math[1] -= (ds[7]/2.0);
     collision.object_model[11].math[1] -= (ds[7]/2.0);
     collision.object_model[12].math[1] -= (ds[7]/2.0);

    collision.object_model[9].math[4]  -=  (ds[7]);
     collision.object_model[10].math[4] += (ds[7]);
     collision.object_model[11].math[4] += (ds[7]);
     collision.object_model[12].math[4] += (ds[7]);

     collision.object_model[1].graphic[1] -= (ds[7]/2.0);
     collision.object_model[2].graphic[1] -= (ds[7]/2.0);
     collision.object_model[3].graphic[1] -= (ds[7]/2.0);
     collision.object_model[4].graphic[1] -= (ds[7]/2.0);
     collision.object_model[5].graphic[1] += (ds[7]/2.0);
     collision.object_model[6].graphic[1] += (ds[7]/2.0);
     collision.object_model[7].graphic[1] += (ds[7]/2.0);
     collision.object_model[8].graphic[1] += (ds[7]/2.0);
```

```
    collision.object_model[9].graphic[7]  += (ds[7]);
    collision.object_model[10].graphic[7] += (ds[7]);
    collision.object_model[11].graphic[7] += (ds[7]);
    collision.object_model[12].graphic[7] += (ds[7]);
}
else if ( size_dirn == 2) /*   Y-dirn size increase  */
{
    collision.object_model[1].math[2]  -= (ds[7]/2.0);
    collision.object_model[2].math[2]  -= (ds[7]/2.0);
    collision.object_model[3].math[2]  += (ds[7]/2.0);
    collision.object_model[4].math[2]  += (ds[7]/2.0);
    collision.object_model[5].math[2]  -= (ds[7]/2.0);
    collision.object_model[6].math[2]  -= (ds[7]/2.0);
    collision.object_model[7].math[2]  += (ds[7]/2.0);
    collision.object_model[8].math[2]  += (ds[7]/2.0);
    collision.object_model[9].math[2]  += (ds[7]/2.0);
    collision.object_model[10].math[2] -= (ds[7]/2.0);
    collision.object_model[11].math[2] -= (ds[7]/2.0);
    collision.object_model[12].math[2] += (ds[7]/2.0);

    collision.object_model[2].math[5]  += (ds[7]);
    collision.object_model[4].math[5]  -= (ds[7]);
    collision.object_model[6].math[5]  += (ds[7]);
    collision.object_model[8].math[5]  -= (ds[7]);

    collision.object_model[1].graphic[2]  -= (ds[7]/2.0);
    collision.object_model[3].graphic[2]  += (ds[7]/2.0);
    collision.object_model[5].graphic[2]  -= (ds[7]/2.0);
    collision.object_model[7].graphic[2]  += (ds[7]/2.0);
    collision.object_model[9].graphic[2]  += (ds[7]/2.0);
    collision.object_model[10].graphic[2] -= (ds[7]/2.0);
    collision.object_model[11].graphic[2] -= (ds[7]/2.0);
    collision.object_model[12].graphic[2] += (ds[7]/2.0);

    collision.object_model[2].graphic[7] += (ds[7]);
    collision.object_model[4].graphic[7] += (ds[7]);
    collision.object_model[6].graphic[7] += (ds[7]);
    collision.object_model[8].graphic[7] += (ds[7]);
}
else if ( size_dirn == 3) /*   Z-dirn size increase  */
{
    collision.object_model[1].math[3]  -= (ds[7]/2.0);
    collision.object_model[2].math[3]  += (ds[7]/2.0);
    collision.object_model[3].math[3]  += (ds[7]/2.0);
    collision.object_model[4].math[3]  -= (ds[7]/2.0);
    collision.object_model[5].math[3]  -= (ds[7]/2.0);
    collision.object_model[6].math[3]  += (ds[7]/2.0);
    collision.object_model[7].math[3]  += (ds[7]/2.0);
    collision.object_model[8].math[3]  -= (ds[7]/2.0);
    collision.object_model[9].math[3]  -= (ds[7]/2.0);
    collision.object_model[10].math[3] -= (ds[7]/2.0);
    collision.object_model[11].math[3] += (ds[7]/2.0);
    collision.object_model[12].math[3] += (ds[7]/2.0);

    collision.object_model[1].math[6]  += (ds[7]);
    collision.object_model[3].math[6]  -= (ds[7]);
    collision.object_model[5].math[6]  += (ds[7]);
    collision.object_model[7].math[6]  -= (ds[7]);

    collision.object_model[2].graphic[3]  += (ds[7]/2.0);
    collision.object_model[4].graphic[3]  -= (ds[7]/2.0);
    collision.object_model[6].graphic[3]  += (ds[7]/2.0);
    collision.object_model[8].graphic[3]  -= (ds[7]/2.0);
```

```
    collision.object_model[9].graphic[3]  -= (ds[7]/2.0);
    collision.object_model[10].graphic[3] -= (ds[7]/2.0);
    collision.object_model[11].graphic[3] += (ds[7]/2.0);
    collision.object_model[12].graphic[3] += (ds[7]/2.0);

    collision.object_model[1].graphic[7] += (ds[7]);
    collision.object_model[3].graphic[7] += (ds[7]);
    collision.object_model[5].graphic[7] += (ds[7]);
    collision.object_model[7].graphic[7] += (ds[7]);
}
   x[1] = x[10] = (collision.object_model[1].math[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);
   x[2] = x[11] = (collision.object_model[2].math[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);
   x[3] = x[12] = (collision.object_model[3].math[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);
   x[4] = (collision.object_model[4].math[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);
   x[5] = (collision.object_model[5].math[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);
   x[6] = (collision.object_model[6].math[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);
   x[7] = (collision.object_model[7].math[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);
   x[8] = x[9] = (collision.object_model[8].math[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);

   y[1] = y[10] = ( collision.object_model[1].math[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);
   y[2] = y[11] = ( collision.object_model[2].math[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);
   y[3] = y[12] = ( collision.object_model[3].math[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);
   y[4] = ( collision.object_model[4].math[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);
   y[5] = ( collision.object_model[5].math[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);
   y[6] = ( collision.object_model[6].math[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);
   y[7] = ( collision.object_model[7].math[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);
   y[8] = y[9]  = ( collision.object_model[8].math[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);

   z[1] = z[10] = ( collision.object_model[1].math[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);
```

```
    z[2] = z[11] = ( collision.object_model[2].math[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);
    z[3] = z[12] =( collision.object_model[3].math[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);
    z[4] = ( collision.object_model[4].math[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);
    z[5] = ( collision.object_model[5].math[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);
    z[6] = ( collision.object_model[6].math[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);
    z[7] = ( collision.object_model[7].math[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);
    z[8] = z[9] = ( collision.object_model[8].math[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);

    x[13] = ( collision.object_model[1].graphic[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);
    x[14] = ( collision.object_model[2].graphic[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);
    x[15] = ( collision.object_model[3].graphic[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);
    x[16] = ( collision.object_model[4].graphic[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);
    x[17] = ( collision.object_model[5].graphic[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);
    x[18] = ( collision.object_model[6].graphic[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);
    x[19] = ( collision.object_model[7].graphic[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);
    x[20] = ( collision.object_model[8].graphic[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);
    x[21] = ( collision.object_model[9].graphic[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);
    x[22] = ( collision.object_model[10].graphic[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);
    x[23] = ( collision.object_model[11].graphic[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);
    x[24] = ( collision.object_model[12].graphic[1] - (((collision.object_model[9].graphic[1] +
collision.object_model[10].graphic[1])/2.0) + ((collision.object_model[11].graphic[1] +
collision.object_model[12].graphic[1])/2.0))/2.0);

    y[13] = ( collision.object_model[1].graphic[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);
```

```
    y[14] = ( collision.object_model[2].graphic[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);
    y[15] = ( collision.object_model[3].graphic[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);
    y[16] = ( collision.object_model[4].graphic[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);
    y[17] = ( collision.object_model[5].graphic[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);
    y[18] = ( collision.object_model[6].graphic[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);
    y[19] = ( collision.object_model[7].graphic[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);
    y[20] = ( collision.object_model[8].graphic[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);
    y[21] = ( collision.object_model[9].graphic[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);
    y[22] = ( collision.object_model[10].graphic[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);
    y[23] = ( collision.object_model[11].graphic[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);
    y[24] = ( collision.object_model[12].graphic[2] - (((collision.object_model[9].graphic[2] +
collision.object_model[10].graphic[2])/2.0) + ((collision.object_model[11].graphic[2] +
collision.object_model[12].graphic[2])/2.0))/2.0);

    z[13] = ( collision.object_model[1].graphic[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);
    z[14] = ( collision.object_model[2].graphic[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);
    z[15] = ( collision.object_model[3].graphic[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);
    z[16] = ( collision.object_model[4].graphic[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);
    z[17] = ( collision.object_model[5].graphic[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);
    z[18] = ( collision.object_model[6].graphic[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);
    z[19] = ( collision.object_model[7].graphic[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);
    z[20] = ( collision.object_model[8].graphic[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);
    z[21] = ( collision.object_model[9].graphic[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);
```

```
    z[22] = ( collision.object_model[10].graphic[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);
    z[23] = ( collision.object_model[11].graphic[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);
    z[24] = ( collision.object_model[12].graphic[3] - (((collision.object_model[9].graphic[3] +
collision.object_model[10].graphic[3])/2.0) + ((collision.object_model[11].graphic[3] +
collision.object_model[12].graphic[3])/2.0))/2.0);
 }
 return;
}
/*****************************************************************************************/
```

Vita

Mr. Amrish S. Shah was born on April 18, 1969 in Pune City in India. He completed his primary, secondary and higher secondary studies at the St. Vincent's High School and Junior College in Pune. Graduating the 12th grade with distinction enabled him to join the Government College of Engineering, Pune for a career in Engineering. He received his Bachelor of Engineering degree in Mechanical Engineering in the Summer of 1991. After being awarded an assistantship for higher studies Mr. Shah joined the Mechanical and Aerospace Engineering Department at the University of Tennessee, Knoxville. After graduation Mr. Shah hopes to start a career in computer progamming and software development.