To The Graduate Council:

I am submitting herewith a thesis written by Tan Fung Chan entitled "Simulation and Real-Time Control of A Seven-Degree-of-Freedom Manipulator." 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.

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 of Science degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under 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 the 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 T. F. Chan

Date Apr. 27, 1992

SIMULATION AND REAL-TIME CONTROL OF A SEVEN-DEGREE-OF-FREEDOM MANIPULATOR

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Tan Fung Chan

May 1992

ACKNOWLEDGMENTS

I would like thank Dr. R. V. Dubey who was the major professor of this thesis. His guidance helped me to participate in this research area. Most of the new ideas come out through discussions with him. My thanks to Dr. F. H. Speckhart and Dr. G. V. Smith for being the members of my committee. I would also like thank Mr. Walter Hankins of NASA Langley Research Center. He helped me understand many practical robot control problems being addressed an NASA Langley. I would like thank Ms. Faye Muly and Mr. Winston Holmes of CCIEM for helping me with the use of computer facilities.

This research work was sponsored by the National Aeronautics And Space Administration, Langley Research Center under Grant NAG-1-1008 and National Science Foundation under Grant MSS-8910095. I thank both NASA and NSF for the financial support during the course of my research work.

Finally, I appreciate greatly the constant encouragement and help from late Dr. J. A. Euler, who could not see the finish of this work.

ABSTRACT

A simple and efficient numerical scheme for local optimization of complex performance criteria is presented to generate the joint trajectory of a seven-degree-of-freedom redundant manipulator. Several practical issues arising from the implementation of local optimization are discussed. A comparison is made of the joint trajectories obtained by optimizing different performance criteria. The seven-degree-of-freedom Robotics Research Corporation (RRC) manipulator is used as a test-bed in this research. A program for real-time control and 3-D graphics simulation of the RRC manipulator is developed on the IRIS-4D/50 GT Silicon Graphics workstation. The user-interface created for this program has several trajectory planning features. It also includes a capability to evaluate joint trajectories in real-time by using plots in 3-D space represented by the end-effector trajectory, self-motion, and the cost function or performance criterion. All the software is written in C programming language for easy portability, and can be used for other seven-degree-of-freedom manipulators. A JR^3 force/torque sensor and an ultrasonic sensor were integrated to the RRC manipulator for force control and obstacle avoidance, respectively.

TABLE OF CONTENTS

CHAPTER PAGE

1. INTRODUCTION 1

1.1 Thesis Objectives 3

1.2 Thesis Organization 4

2. BACKGROUND ON LOCALLY OPTIMIZED REDUNDANCY RESOLUTION 5

2.1 Redundancy Resolution for Optimizing Performance Criteria 5

2.2 Pseudoinverse of the Jacobian Matrix 6

2.3 An Efficient Optimization Technique for A Redundant Manipulator 8

2.4 The Implementation of the Efficient Optimization Method at a Singular Configuration 12

2.5 Computational Cost Analysis 15

3. CONTROL PROGRAM AND CARTESIAN PATH PLANNING 18

3.1 User Interface of the Program 18

3.2 A Numerical Scheme for Local Optimization 21

3.3 Circular Path Planning for Transition Between Two Linear Paths 24

3.3.1 Center of the Circular Path 24

3.3.2 Circular Path Planning Calculations 28

3.3.3 Computation of Circular Path by Finite Steps 31

3.3.4 Smooth Transition Between the Circular Path and the Linear Path 33

3.4 Path Planning of the End-effector Orientation 36

3.5 Self Motion of a Single Redundant joint Manipulator 38

4. LOCALLY OPTIMIZED REDUNDANCY RESOLUTION AND PERFORMANCE EVALUATION 40

4.1 3-D Plot of the Joint Velocity Norm 40

4.2 Pseudoinverse Control 45

4.3 Some Practical Issues in the Implementation of the Gradient Projection Optimization 48

4.4 Evaluation and Comparison of Different Performance Criteria for Minimizing Joint Velocities 50

5. CONTROL PROGRAM USER INSTRUCTIONS 65

5.1 Functions Available on the Graphics Teach Pendant 65

5.1.1 Teaching the Robot 65

5.1.2 Recording the Trajectory Points 67

5.1.3 Replaying and Editing the Recorded Trajectory . . . 68

5.1.4 Changing the View of the Graphics Model 68

5.1.5 Changing the Moving Speed of the Manipulator ... 69

5.1.6 Options of the Program ... 69

5.1.7 Calculating the Data for the 3-D Map ... 71

5.1.8 Store or Retrieve the Files ... 72

5.2 Available Functions of the Small Graphics Window ... 73

6. CONCLUSIONS AND RECOMMENDATIONS ... 76

REFERENCES ... 79

APPENDIXES ... 83

APPENDIX A
PROGRAM FUNCTION LIST ... 84

A.1 Functions in the Model File "main_pro.c" ... 84

A.2 Functions in the Model File "goalp.c" ... 85

A.3 Functions in the Model File "init_graph.c" ... 90

A.4 Functions in the Model File "tools.c" ... 93

A.5 Functions in the Model File "draw_k2107.c" ... 98

A.6 Functions in the Model File "draw_LTM.c" ... 100

APPENDIX B
COMMUNICATION WITH CONTROLLER
AND SENSORS ... 102

B.1 Communication with the RRC Robot Controller ... 102

B.2 Communication with the Force Sensor 103

B.3 Communication with the Ultrasonic Sensor 104

APPENDIX C
D-H TABLE OF RRC K-2107 MANIPULATOR 107

APPENDIX D
GRAPHICS MODEL OF RRC MANIPULATOR 108

APPENDIX E
PROGRAM SOURCE CODE 116

VITA 307

LIST OF FIGURES

FIGURE PAGE

1.1. Different RRC manipulator configurations for a fixed position and orientation of the end-effector 2

3.1. The user interface of the control program 20

3.2. A circular path connecting two linear path segments 25

3.3. Determination of the center of the circular path 27

3.4. Determination of the end-effector velocity components 29

3.5. Approximation of the circular path by a series of connected chords 32

3.6. Smooth transition between the circular and linear path 34

4.1. Different starting joint configurations for a closed end-effector path 41

4.2. 3-D cost map of the least-norm joint velocity norm 43

4.3. End-effector at different positions along the closed path 44

4.4. Least-norm cost curves for different loops 47

4.5. Cost curves using least-norm & optimized joint trajectories 49

4.6. Joint trajectories with & without joint velocity limit 51

4.7. Trajectories generated by using a very small and a very large gain factor 52

4.8. Cost curves for different performance criteria (1) 56

4.9. Converging loop for different performance criteria (1) 57

4.10. Cost curves for different performance criteria (2) 58

4.11. Converging loop for different performance criteria (2) 60

4.12. 3-D cost map of $1/\sigma_{min}$ 61

4.13. 3-D cost map of $1/\sqrt{\det(JJ^T)}$ 63

D.1. Base part of RRC manipulator model 108

D.2. First link of RRC manipulator model 109

D.3. Second link of RRC manipulator model 110

D.4. Third link of RRC manipulator model 111

D.5. Fourth link of RRC manipulator model 112

D.6. Fifth link of RRC manipulator model 113

D.7. Sixth link of RRC manipulator model 114

D.8. End-effector of RRC manipulator model 115

CHAPTER 1

INTRODUCTION

In order to move the robot end-effector to an arbitrary position in a three dimensional workspace, at least a three joint- or degree-of-freedom manipulator is required. If one wants to control the position as well as the orientation, at least six degrees of freedom are required in the manipulator. A redundant manipulator has more degrees of freedom than the number of coordinates required to completely specify the end-effector motion. Generally, a finite number of configurations exist for a nonredundant manipulator to reach a certain position and orientation. However, a redundant manipulator has infinite configurations to reach the same end-effector position.

In recent years, a few seven-degree-of-freedom redundant manipulators have been built for operating in three dimensional space. These include the CESAR manipulator [1] at the Oak Ridge National Laboratory, the Laboratory Telerobotic Manipulator (LTM) [2] at the NASA Langley Research Center, and the commercially available Robotics Research Corporation (RRC) manipulator [3]. Like a human arm which can move the elbow without moving the hand and the shoulder, these manipulators can change configuration without moving the end-effector (Fig. 1.1).

An infinite number of joint velocity vectors result in a desired end-effector velocity for manipulators with redundant joints. It has been shown that the pseudoinverse control, i.e., using the joint velocities with minimum norm, may result in joint

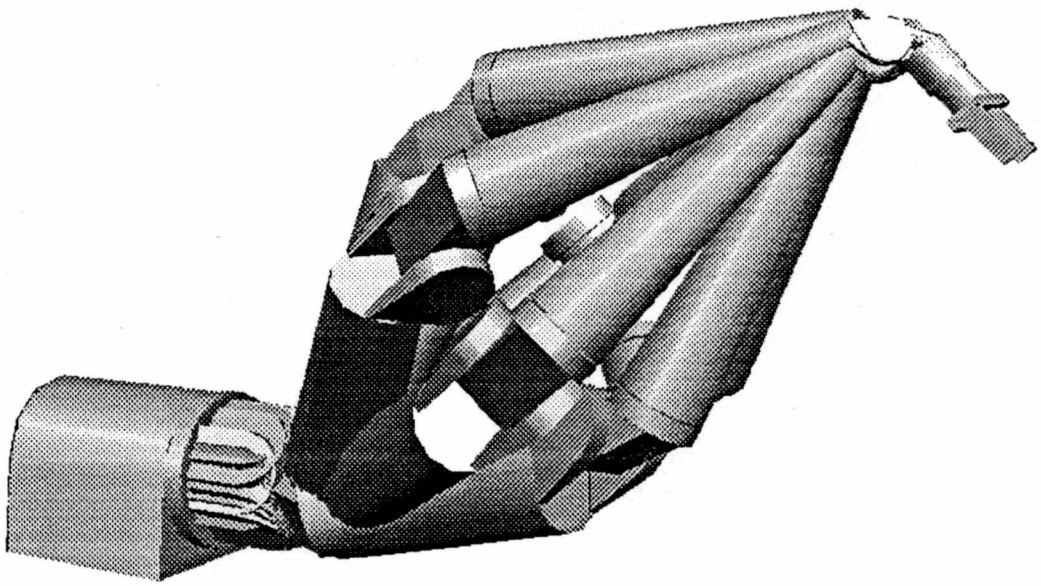


Figure 1.1

Different RRC manipulator configurations for a fixed position and orientation of the end–effector

configurations requiring extremely high joint velocities [4]. Such configurations are in the neighborhood of singularities in which the magnitude of the required joint velocities is infinite. Pseudoinverse control also results in drift in joint space when a cyclic task is performed in rectilinear space [5]. Minimization of the integral of the joint velocity vector norm over the complete trajectory would provide a joint trajectory away from singularities, if such a trajectory exists. However, such global optimization [6] can only be done off-line, and it is unsuitable for teleoperation or for robotic control in which the end-effector trajectory is modified based on sensory feedback. It may also be noted that even though the global optimization will minimize the average joint velocity norm, it does not minimize the maximum possible instantaneous joint velocity norm.

Local optimization of performance criteria such as $\sqrt{\det(JJ^T)}$ [7], where J is the Jacobian matrix, or minimum singular value [8] may be used to avoid high joint velocities. Dubey and Luh [9] proposed Manipulator Velocity Ratio (MVR), a ratio of end-effector velocity norm to the minimum joint velocity norm, as a performance criterion to be locally maximized to minimize the required joint velocities. Chan and Dubey [10] suggested defining the norm of the least-norm joint velocity vector (which, in fact, is proportional to the inverse of MVR if the weight matrices are identity matrices) as a performance criterion and minimizing it.

1.1 Thesis Objectives

The objective of this thesis is to implement optimized real-time control of the RRC K-2107 manipulator and its graphics model, and evaluate the joint trajectories

obtained through local optimization of several different performance criteria. Based on the objective, following tasks are defined for the thesis:

(1) Development of a computer program with an easy-to-use interface, that can be used to control both the RRC K-2107 manipulator and its graphics model in real-time.

(2) Development of an efficient numerical method for local optimization that does not require the calculation of the gradient in the symbolic form. This will allow us to optimize computationally complicated and multiple performance criteria.

(3) Development of software tools to analyze the results of local optimization of different performance criteria.

(4) Comparison and evaluation of several different performance criteria using the RRC K-2107 manipulator and its graphics model.

1.2 Thesis Organization

Chapter 2 discusses the back ground on locally optimized redundancy resolution. The user interface of the control program, path planning of the end-effector trajectory, self motion implementation, and the numerical local optimization scheme are discussed in Chapter 3. In Chapter 4, joint trajectories obtained by optimizing different performance criteria are evaluated using 3-D plots. Chapter 5 describes the control program user instructions. Conclusions and recommendations are presented in Chapter 6. Program listing, sensors and controller communications, D-H table, graphics model, and source codes of the computer program are provided in the Appendices.

CHAPTER 2

BACKGROUND ON LOCALLY OPTIMIZED REDUNDANCY RESOLUTION

2.1 Redundancy Resolution for Optimizing Performance Criteria

The kinematic equation describing the relationship between the end-effector velocity and joint velocities is given by:

$$\underline{\dot{x}} = J\underline{\dot{\theta}}, \tag{2.1}$$

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

For a redundant manipulator, we have m<n; therefore, an infinite number of joint velocity vectors $\underline{\dot{\theta}}$ exist for a given $\underline{\dot{x}}$ in (2.1). The joint velocity vectors can be computed as:

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

where J^{+} is the Moore-Penrose inverse [11] of the Jacobian. $J^{+}\underline{\dot{x}}$ is the least-norm solution of Eq. (2.1), i.e., it provides the $\underline{\dot{\theta}}$ with minimum Euclidian norm. Redundancy resolution using just the least-norm solution is referred to as pseudoinverse control [12]. $(I-J^{+}J)\underline{\phi} \in N(J)$, the null space of J, is the homogeneous solution of (2.1), and $\underline{\phi} \in R^{n}$ is an arbitrary vector. The homogeneous solution is referred to as the self-motion of the

manipulator and does not cause any end-effector motion. For a desired trajectory, the homogeneous solution should be appropriately selected to improve the performance of the manipulator. To improve a performance criterion $H(\underline{\theta})$ using the gradient projection method [13], the redundancy is resolved by substituting $k\nabla H(\underline{\theta})$ for $\underline{\phi}$ in (2.2):

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

where k is a real scaler coefficient. The scaler constant k is taken to be positive if $H(\underline{\theta})$ is to be maximized and negative if $H(\underline{\theta})$ is to be minimized. The maximum allowable value of k should be limited by bounds on joint velocities [14] or actuator torques [15]. $(I-J^{+}J)$ is the null space projection matrix [11] of the Jacobian, and its rank is equal to (n-m) for a full ranked Jacobian.

2.2 Pseudoinverse of the Jacobian Matrix

In order to use Eq. (2.3) directly, we must calculate the pseudoinverse of Jacobian matrix. The pseudoinverse for a full ranked Jacobian matrix J can be expressed as[16]:

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

Another way to calculate the pseudoinverse is by using the singular value decomposition (SVD) of the Jacobian matrix. It is applicable whether the matrix is full ranked or not. Any m by n matrix J can be factored as [17]:

$$J = Q_1 \Sigma Q_2^{T} , \tag{2.5}$$

where columns of mxm matrix Q_1 are the eigenvectors of JJ^T, and the columns of nxn matrix Q_2 are eigenvectors of J^TJ. The r (r = rank of J) non zero elements of the mxn

diagonal matrix Σ are singular values of J. These singular values are the square roots of the nonzero eigenvalues of JJ^T or J^TJ. Using the SVD of J, the pseudoinverse is:

$$J^+ = Q_2 \Sigma^+ Q_1^T, \tag{2.6}$$

where Σ^+ is an nxm diagonal matrix with the reciprocals of the singular values of J as its elements.

For a singular matrix J, Eq. (2.2) normally does not produce exactly the desired end-effector motion $\underline{\dot{x}}$. The solution is discontinuously switched from an exact to an approximate one at the singular points [18]. If the desired end-effector velocity vector has a motion component in the direction in which the end-effector can not move due to singularity, the pseudoinverse discards that component and determines the least-norm solution of the modified end-effector motion.

Eq. (2.2) represents a general solution which is also applicable to nonredundant manipulators for which J is a square matrix. When the square J matrix is nonsingular, J^+ is same as J^{-1}, the inverse of J. The second part of Eq. (2.2) becomes zero, since no self-motion can be produced for a nonredundant manipulator at a nonsingular configuration. If the square J matrix is singular, the manipulator loses certain degrees of freedom of the end-effector motion, but has the same number of additional degrees of freedom for self-motion. Thus, a nonredundant manipulator has the capability of self motion at its singular configurations.

Determing J^+ using Eq. (2.4) requires the calculation of JJ^T, which squares the condition number of the Jacobian matrix. This reduces the accuracy of Eq. (2.2) especially near the singular point [19]. The SVD gives better accuracy and provides other

significant information such as the condition number, the orthogonal basis of the null space of Jacobian etc.; however, it is computationally expensive.

2.3 An Efficient Optimization Technique for A Redundant Manipulator

An efficient method of finding the least-norm solution and the homogeneous solution for a redundant manipulator was proposed by Dubey et al. [20]. This method does not require the calculation of the pseudoinverse. The method was based on the fact that any particular solution $\underline{\dot{\theta}}_p$ satisfying Eq. (2.1) can be expressed as a combination of the least-norm solution and the homogeneous solution:

$$\underline{\dot{\theta}}_p = \underline{\dot{\theta}}_{min} + \underline{\dot{\theta}}_h , \tag{2.7}$$

where $\underline{\dot{\theta}}_{min}$ is the least-norm solution of Eq. (2.1), and $\underline{\dot{\theta}}_h$ is the component of the homogeneous solution corresponding to the particular solution $\underline{\dot{\theta}}_p$. The homogeneous solution $\underline{\dot{\theta}}_h$ in Eq. (2.7) can be expressed as a combination of linearly independent homogeneous solutions:

$$\underline{\dot{\theta}}_h = \sum_{i=1}^{n-r} k_i \underline{\dot{\theta}}_{h,i} , \tag{2.8}$$

where n is the number of the Jacobian matrix columns; r is the rank of the Jacobian matrix, and $\underline{\dot{\theta}}_{h,i}$ are the linearly independent homogeneous solution spanning the nullspace of the Jacobian matrix. The weight coefficients of $\underline{\dot{\theta}}_{h,i}$, k_i, in Eq. (2.8) are such that the combination can produce the homogeneous solution $\underline{\dot{\theta}}_h$ satisfying Eq. (2.7).

Combining Eq. (2.7) and (2.8) we obtain the following:

$$\underline{\dot{\theta}}_{min} = \underline{\dot{\theta}}_p - \sum_{i=1}^{n-r} k_i \underline{\dot{\theta}}_{h,i} \ . \tag{2.9}$$

For a rectangular mxn Jacobian matrix J with n>m, the end-effector velocity vector $\underline{\dot{x}} \in R^m$ and the joint velocity vector $\underline{\dot{\theta}} \in R^n$. We assume the Jacobian to be full ranked, i.e., its rank is equal to m. Thus, we may construct a nonsingular mxm matrix from any m independent columns of the Jacobian. In general, by rearranging the columns of J and the corresponding elements of $\underline{\dot{\theta}}$, we can rewrite Eq. (2.1) as:

$$[J_c \ J_d] \underline{\dot{\theta}} = \underline{\dot{x}} \ , \tag{2.10}$$

where J_c is an mxm nonsingular matrix containing the linearly independent columns of Jacobian. The remaining columns of J form an mx(n-m) matrix J_d. For finding a particular solution we assume that the remaining (n-m) elements of $\underline{\dot{\theta}}_p$ are zeros, and then the first m elements are solved from the following equation:

$$[J_c] \underline{\dot{\theta}}_{pm} = \underline{\dot{x}} \ , \tag{2.11}$$

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

The set of linearly independent homogeneous solutions $\underline{\dot{\theta}}_{h,i}$ may be obtained by setting the right side of Eq. (2.10) to zero, and assuming the last (n-m) elements of $\underline{\dot{\theta}}=\underline{\dot{\theta}}_{h,i}$ to be zeros, except the m+ith element which is assumed to be 1. The first m elements of $\underline{\dot{\theta}}_{h,i}$ can be obtained as:

$$[J_c] \underline{\dot{\theta}}_{hm,i} = -J_{d,m+i} \ , \qquad i=1,\cdots n-m \ , \tag{2.12}$$

where $\underline{\dot{\theta}}_{hm,i}$ contains the first m elements of $\underline{\dot{\theta}}_{h,i}$, and $J_{d,m+i}$ is the m+ith column of the matrix J_d.

The weight coefficients k_i can be obtained due to the fact that the least-norm solution $\underline{\dot{\theta}}_{min}$ must be orthogonal to any homogeneous solutions $\underline{\dot{\theta}}_{h,i}$. Since the dot product of the least-norm solution and $\underline{\dot{\theta}}_{h,i}$, the components of the homogeneous solution, must be zero, from Eq. (2.9) we get:

$$\underline{\dot{\theta}}_{h,i}^T \left(\underline{\dot{\theta}}_p - \sum_{j=1}^{n-m} k_j \underline{\dot{\theta}}_{h,j} \right) = 0 , \qquad i=1,\cdots n-m . \tag{2.13}$$

Which can be rewritten as:

$$\begin{bmatrix} \|\underline{\dot{\theta}}_{h,1}\|^2 & \underline{\dot{\theta}}_{h,1}^T \underline{\dot{\theta}}_{h,2} & \cdots & \underline{\dot{\theta}}_{h,1}^T \underline{\dot{\theta}}_{h,n-m} \\ \underline{\dot{\theta}}_{h,2}^T \underline{\dot{\theta}}_{h,1} & \|\underline{\dot{\theta}}_{h,2}\|^2 & \cdots & \underline{\dot{\theta}}_{h,2}^T \underline{\dot{\theta}}_{h,n-m} \\ \vdots & \vdots & \cdots & \vdots \\ \underline{\dot{\theta}}_{h,n-m}^T \underline{\dot{\theta}}_{h,1} & \underline{\dot{\theta}}_{h,n-m}^T \underline{\dot{\theta}}_{h,2} & \cdots & \|\underline{\dot{\theta}}_{h,n-m}\|^2 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_{n-m} \end{bmatrix} = \begin{bmatrix} \underline{\dot{\theta}}_{h,1}^T \underline{\dot{\theta}}_p \\ \underline{\dot{\theta}}_{h,2}^T \underline{\dot{\theta}}_p \\ \vdots \\ \underline{\dot{\theta}}_{h,n-m}^T \underline{\dot{\theta}}_p \end{bmatrix} . \tag{2.14}$$

The matrix on the left side of the equation represents the projections of the homogeneous solutions to each other, we call this matrix a homogeneous projection matrix. The right side of the equation are the projections of the particular solution onto the homogeneous solutions.

Solving k_i from Eq. (2.14), and replacing them in to Eq. (2.9) we can obtain the least-norm solution of the Eq. (2.1).

In order to improve the performance of the manipulator, we need to calculate the second part of the Eq. (2.3), This can be calculated by the projecting n dimensional gradient vector ∇H onto the null space of Jacobian matrix. Let the homogeneous solution $\underline{\dot{\theta}}_{\nabla H}$ be the projection of ∇H onto the null space. $\underline{\dot{\theta}}_{\nabla H}$ is the linear combination of the

set of homogeneous solutions $\underline{\dot{\theta}}_{h,i}$:

$$\underline{\dot{\theta}}_{\nabla H} = \sum_{i=1}^{n-m} w_i \underline{\dot{\theta}}_{h,i} , \tag{2.15}$$

where w_i are the weights which satisfy the projection relationship. The difference between ∇H and $\underline{\dot{\theta}}_{\nabla H}$ is the vector that is orthogonal to the set of the homogeneous solutions $\underline{\dot{\theta}}_{h,i}$. Therefore, we obtain the equation as:

$$\underline{\dot{\theta}}_{h,i}^T (\nabla H - \underline{\dot{\theta}}_{\nabla H}) = 0 , \qquad i=1,2,\cdots(n-m) . \tag{2.16}$$

From Eq. (2.15) and (2.16) we obtain:

$$\underline{\dot{\theta}}_{h,i}^T (\nabla H - \sum_{j=1}^{n-m} w_j \underline{\dot{\theta}}_{h,j}) = 0 , \qquad i=1,2,\cdots(n-m) . \tag{2.17}$$

Which can be rewritten as:

$$\begin{bmatrix} \|\underline{\dot{\theta}}_{h,1}\|^2 & \underline{\dot{\theta}}_{h,1}^T \underline{\dot{\theta}}_{h,2} & \cdots & \underline{\dot{\theta}}_{h,1}^T \underline{\dot{\theta}}_{h,n-m} \\ \underline{\dot{\theta}}_{h,2}^T \underline{\dot{\theta}}_{h,1} & \|\underline{\dot{\theta}}_{h,2}\|^2 & \cdots & \underline{\dot{\theta}}_{h,2}^T \underline{\dot{\theta}}_{h,n-m} \\ \vdots & \vdots & \cdots & \vdots \\ \underline{\dot{\theta}}_{h,n-m}^T \underline{\dot{\theta}}_{h,1} & \underline{\dot{\theta}}_{h,n-m}^T \underline{\dot{\theta}}_{h,2} & \cdots & \|\underline{\dot{\theta}}_{h,n-m}\|^2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{n-m} \end{bmatrix} = \begin{bmatrix} \underline{\dot{\theta}}_{h,1}^T \nabla H \\ \underline{\dot{\theta}}_{h,2}^T \nabla H \\ \vdots \\ \underline{\dot{\theta}}_{h,n-m}^T \nabla H \end{bmatrix} . \tag{2.18}$$

Solving w_i from Eq. (2.18) and substituting them into Eq. (2.15) we can obtain $\underline{\dot{\theta}}_{\nabla H}$, the null space projection of ∇H. $\underline{\dot{\theta}}_{\nabla H}$ points in the direction in which the performance criterion is maximized. This equation is very similar to Eq. (2.14). Both equations use the same homogeneous projection matrix! The difference is only on the right side of the equations. The right side of Eq. (2.18) are the projections of the vector ∇H onto $\underline{\dot{\theta}}_{h,i}$.

Finally we obtain the optimized solution (2.3) as:

$$\underline{\dot{\theta}} = \underline{\dot{\theta}}_{min} + k\underline{\dot{\theta}}_{\nabla H} . \tag{2.19}$$

2.4 The Implementation of the Efficient Optimization Method at a Singular Configuration

The efficient optimization method discussed in section 2.3 is assumed at nonsingular configurations. This method can be extended further to obtain a solution for a singular Jacobian, if the solution exists. Assume a configuration for which the rank of Jacobian is less than the number of its rows. Then the end-effector looses (m-r) degrees of freedom and the manipulator gets the same degrees of freedom for self-motion. Rearranging the rows of J matrix and corresponding elements of $\underline{\dot{x}}$, we can rewrite Eq. (2.1) as:

$$\underline{\dot{x}'} = \begin{bmatrix} J_r \\ J_d \end{bmatrix} \underline{\dot{\theta}} , \tag{2.20}$$

where J_r contains the linearly independent rows of the Jacobian matrix, J_d contains the remaining rows of the J matrix. If the solution exists for Eq. (2.20), it is the same as the solution for the first r equations:

$$\underline{\dot{x}'}_r = J_r \underline{\dot{\theta}} , \tag{2.21}$$

where $\underline{\dot{x}}'_r$ contains the first r elements of the vector $\underline{\dot{x}}'$. The matrix J_r in Eq. (2.21) is a full ranked r by n matrix. Using the method discussed in section 2.3, we get the least-norm solution which is also the least-norm solution of Eq. (2.1). In this case, the manipulator is at its singular configuration and there are certain directions in which the

end-effector cannot move. However, the end-effector velocity vector $\underline{\dot{x}}$ does not contain the velocity components which the end-effector can not move. Therefore, we still get the correct solution.

Further more, if the Jacobian matrix is inconsistent, then using the method discussed in section 2.3 we still can find the homogeneous solutions $\underline{\dot{\theta}}_{h,i}$ of Eq. (2.21). However, we cannot get the particular solution that produces the exact $\underline{\dot{x}}$. In order to obtain the particular solution which produces the end-effector velocity vector as close to the desired velocity vector $\underline{\dot{x}}$ as possible, we minimize the error between the desired velocity vector $\underline{\dot{x}}$ and the velocity vector $J\underline{\dot{\theta}}$. The error can be expressed as:

$$\|e\|^2 = \|\underline{\dot{x}} - J\underline{\dot{\theta}}\|^2 , \tag{2.22}$$

which can be written as:

$$\|e\|^2 = (\underline{\dot{x}} - J\underline{\dot{\theta}})^T(\underline{\dot{x}} - J\underline{\dot{\theta}}) . \tag{2.23}$$

Taking the partial derivative with respect to $\underline{\dot{\theta}}$ and equating it to zero:

$$2J^T(\underline{\dot{x}} - J\underline{\dot{\theta}}) = \underline{0} , \tag{2.24}$$

or:

$$J^T\underline{\dot{x}} = J^TJ\underline{\dot{\theta}} . \tag{2.25}$$

It is noticed that, the rank of the matrix J^TJ is the same as the rank of the Jacobian matrix; therefore, the inverse of J^TJ does not exist. Fortunately, since the projection of $\underline{\dot{x}}$ on the range space of J exists, the solution of Eq. (2.24) must exist; therefore, the Eq. (2.25) is consistent!

Let:

$$\underline{y} = J^T \underline{\dot{x}} \ , \tag{2.26}$$

and:

$$J^* = J^T J \ , \tag{2.27}$$

Then Eq. (2.26) can be written as:

$$J^* \underline{\dot{\theta}} = \underline{y} \ . \tag{2.28}$$

Rearranging the rows of J^* and the corresponding elements of $\underline{y}$, we obtain:

$$\begin{bmatrix} J_r^* \\ J_d^* \end{bmatrix} \underline{\dot{\theta}} = \underline{y}_r \ , \tag{2.29}$$

where J_r^* consists of r linearly independent rows of J^*. Since Eq. (2.28) is consistent, we can discard the remaining (m-r) equations of Eq. (2.29) to obtain:

$$J_r^* \underline{\dot{\theta}} = \underline{y}_r' \ , \tag{2.30}$$

where $\underline{y}_r'$ contains the first r elements of the vector $\underline{y}_r$. Solving Eq. (2.30), we can get the particular solution which will produce the projection of $\underline{\dot{x}}$.

To check whether the solution $\underline{\dot{\theta}}$ satisfying Eq. (2.25) can produce the projection of $\underline{\dot{x}}$ on the null space, we can check the orthogonality of the vector $J\underline{\dot{\theta}}$ to the error vector $(\underline{\dot{x}} - J\underline{\dot{\theta}})$:

$$
\begin{aligned}
(J\underline{\dot{\theta}})^T(\underline{\dot{x}} - J\underline{\dot{\theta}}) &= \underline{\dot{\theta}}^T J^T(\underline{\dot{x}} - J\underline{\dot{\theta}}) \\
&= \underline{\dot{\theta}}^T(J^T\underline{\dot{x}} - J^TJ\underline{\dot{\theta}}) \\
&= \underline{\dot{\theta}}^T\underline{0}\ , \\
(J\underline{\dot{\theta}})^T(\underline{\dot{x}} - J\underline{\dot{\theta}}) &= 0\ .
\end{aligned}
\tag{2.31}
$$

Therefore, the solution of Eq. (2.25) is the projection solution of $\underline{\dot{x}}$, which is also the minimum error solution. Now, using the particular solution and the set of the homogeneous solutions $\underline{\dot{\theta}}_{h,i}$, we can apply the method discussed in section 2.3 to obtain the least norm solution.

2.5 Computational Cost Analysis

The method mentioned in section 2.3 does not calculate the pseudoinverse of the Jacobian. This section analyses whether this method can actually reduce the computational cost. Since the most time consuming calculation is the floating point multiplication, we consider it as the computational cost for the methods. The Jacobian matrix discussed in this section is m by n full ranked matrix with m<n.

First, let us calculate the cost of pseudoinverse of Eq. (2.4). To avoid the inverse of the JJ^T in Eq. (2.4), we define the middle variable $\underline{v}$. The vector $\underline{v}$ is defined as:

$$
\underline{v} = (JJ^T)^{-1}\ \underline{\dot{x}}\ . \tag{2.32}
$$

This equation can be rewritten as:

$$
(JJ^T)\,\underline{v} = \underline{\dot{x}}\ . \tag{2.33}
$$

Eq. (2.33) can be solved by Gaussian elimination to get the value $\underline{v}$. The least-norm

solution can be obtained as:

$$\underline{\dot{\theta}}_{min} = J^T \underline{v} \ . \tag{2.34}$$

For calculating JJ^T, we need $n \cdot m(m+1)/2$ number of multiplications. Using Gaussian elimination to solve Eq. (2.33) requires $m(m-1)(2m+5)/6+m(m+1)/2$ multiplications. And finally Eq. (2.34) requires $n \cdot m$ multiplications. Therefore, the total number of multiplications to obtain the least-norm solution is:

$$C_1 = \frac{n \cdot m}{2}(m + 3) + \frac{m}{3}(m^2 + 3m - 1) \ . \tag{2.35}$$

When using the efficient optimization method presented in section 2.3, we need to calculate the particular solution, the set of homogeneous solutions, the entries of the homogeneous projection matrix, the projections of the particular solution onto the set of homogeneous solutions, the weights k_i, and finally the multiplication of $k_i \underline{\dot{\theta}}_{h,i}$. For calculating the particular solution and the homogeneous solution, we notice that the matrix J_c in Eq. (2.11) and Eq. (2.12) are exactly the same. This implies that the forward Gaussian elimination is needed to be done only once. Calculating $\underline{\dot{\theta}}_p$ and $\underline{\dot{\theta}}_{h,i}$ requires $m(3m \cdot n-2m^2+3m-1)/3$ multiplications. The entries of the homogeneous projection matrix, and the dot products of the particular solution and the homogeneous solutions $\underline{\dot{\theta}}_{h,i}$ require $m(n^2-(2m-3)n+m^2-3m)/2$ multiplications. Using Gaussian elimination to solve k_i needs $(n^3-(3m-3)n^2+(3m^2-6m-1)n-m^3+3m^2+m)/3$ multiplications. And the final step requires $m(n-m)$ multiplications. Thus, total computing cost is:

$$C_2 = \frac{1}{6}\left[2n^3 - (3m - 6)n^2 + (6m^2 + 3m - 2)n - 3m^2(m + 1)\right] . \quad (2.36)$$

Notice, we get a third power polynomial of n here. This means that with an increase in n, the cost C_2 will increases much faster than C_1, the linear function of n. Substituting a value of m=6, in Eq. (2.35) we get:

$$C_1 = 27n + 106 . \quad (2.37)$$

From Eq. (2.36) we obtain:

$$C_2 = \frac{1}{6}(2n^3 - 12n^2 + 232n - 756) . \quad (2.38)$$

From the above two equations we obtain the table showing C_1 and C_2 for different values of n as:

n = numbers of joints	C_1	C_2
7	295	161
8	322	226
9	349	303
10	376	394
11	403	501

From this table we can see that for calculating the least-norm solution with m=6, the efficient optimization method in section 2.3 requires more calculations when n>9.

CHAPTER 3

CONTROL PROGRAM AND CARTESIAN PATH PLANNING

The computer program for controlling the RRC manipulator was developed on the IRIS-4D Silicon Graphics workstation. An interactive and easy-to-use user interface was developed for the control program. The user interface includes a 3-D graphics model of the RRC manipulator. This interface allows the user to manipulate the robot, test different control schemes and optimize different performance criteria. A circular path planning routine was developed to smoothly connect two linear path segments of the end-effector. This routine allows the end-effector speed to be constant during the change of direction from one linear path segment to another. The orientation of the end-effector is specified by Euler angles, and the rotation trajectory between two end points is obtained such that minimum rotation is required to reach the desired end-effector orientation. The descriptions of the control program functions and their listing are provided in Chapter 5 and Appendix A, respectively. This chapter discusses some of the original and important contributions in the development of the control program for the RRC manipulator.

3.1 User Interface of the Program

An interactive, mouse driven user interface was developed for the robot control program. The user interface consists of a 3-D graphics model of the RRC manipulator

along with an on screen teach-pendant (Fig. 3.1). The animated graphics model can be used for real-time simulation of the programmed trajectory before it is used to run the robot. The user interface allows the user to manipulate the robot, program different robot trajectories, and specify performance criteria for optimized redundancy resolution. The user can use the mouse of the computer to call the various service of the user-interface. This interface has three different levels of service functions.

The first level service functions are teaching functions. At this level the functions let the user change the configuration of the manipulator by simply placing the cursor on the appropriate "button" on the teach pendant, and then pressing the left or the middle mouse button. These include moving individual joints, translating and rotating the end-effector in a particular direction, self motion, and moving the manipulator to its "home" configuration.

The second level are used to control the parameters that are required to be changed frequently. These functions do not change the configuration of the manipulator. They are used to change the joint or end-effector velocity, change the view of the graphics model, record or delete the trajectory points, replay the recorded trajectory points, and interrupt the executing program.

The third level require the key board to input some parameters. These can be activated by using the mouse to choose the function, and then using the key board to change the parameters. These functions include storing or retrieving files, generating a 3-D map of performance criteria, sending commands to the serial port, and reading the feed back, changing the performance criteria, displaying the manipulability ellipsoid etc.

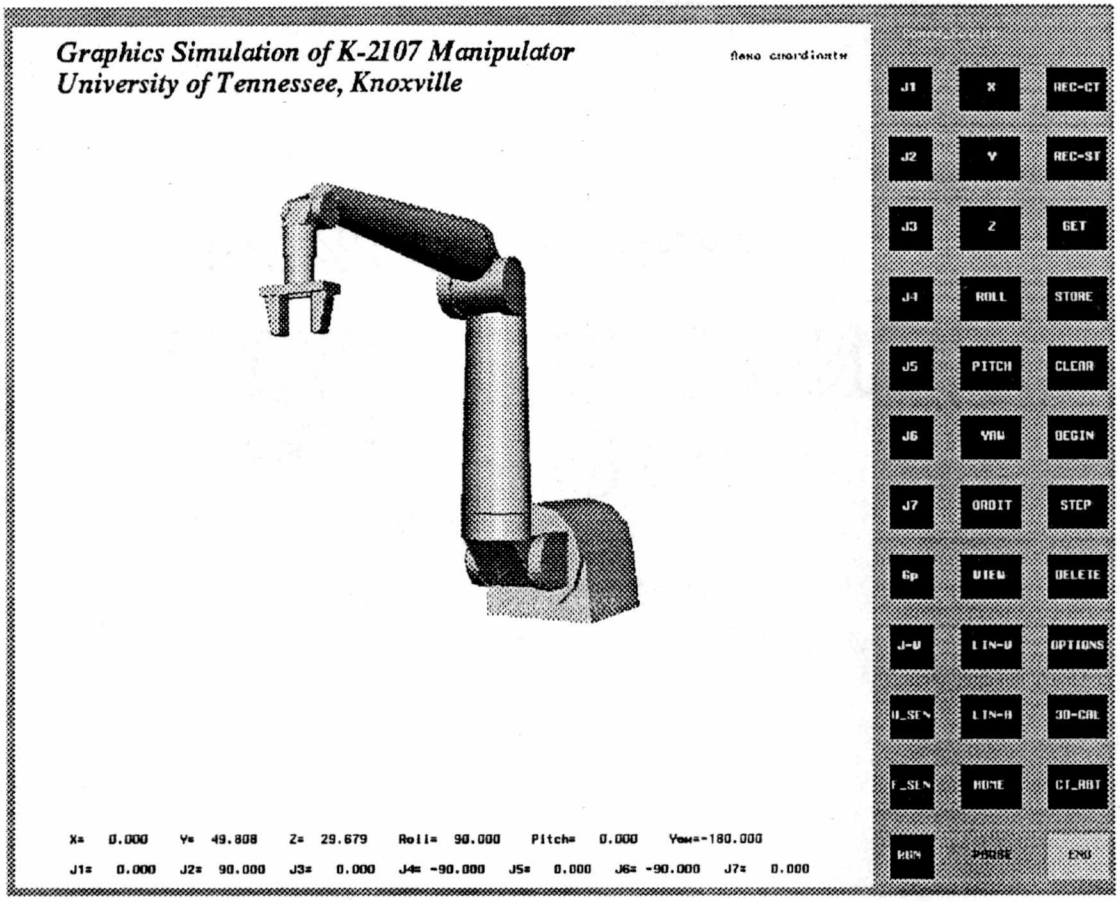


Figure 3.1

The user interface of the control program

The RRC controller program requires about 5 position commands per second during the jogging mode [21]. If the controller program does not receive the new command within specified time window, it will stop the jogging mode. For this reason, in the robot jogging mode, the first and the second level service are designed such that they will maintain the robot jogging mode. If the third level are used, the robot jogging mode will be stopped.

The detailed instructions for the user-interface are provided in Chapter 5.

3.2 A Numerical Scheme for Local Optimization

The right hand side of the Eq. (2.2) in Chapter 2 consists of the least-norm solution term $J^{+}\underline{\dot{x}}$ and the homogeneous solution term $(I\text{-}J^{+}J)\underline{\phi}$. The matrix $(I\text{-}J^{+}J)$ is the null space projection matrix [11] of the Jacobian, and its rank is equal to (n-m) for a full ranked Jacobian. If (n-m), which is the degree of redundancy, is equal to 1, then the null space of the Jacobian is one-dimensional. If $\underline{n}_J \in R^n$ is a unit vector from the one-dimensional null space of the Jacobian, then the projection of ∇H onto the null space of the Jacobian may be written as $\underline{n}_J^T \nabla H$, i.e.,

$$(I - J^{+}J)\nabla H = (\underline{n}_J^T \nabla H)\,\underline{n}_J . \tag{3.1}$$

Let $\underline{\dot{\theta}}_h \in R^n$ represent the homogeneous solution of Eq. (2.1) of Chapter 2, i.e.,

$$J\underline{\dot{\theta}}_h = \underline{0}, \tag{3.2}$$

where $\underline{0} \in R^n$ is a null vector. Thus $\underline{n}_J$, the unit vector in the direction of the null space of Jacobian, may be represented as:

$$n_J = \frac{\dot{\underline{\theta}}_h}{\|\dot{\underline{\theta}}_h\|} . \tag{3.3}$$

It may be noted that the components of ∇H represent change in H per unit change in the corresponding joint angle. Thus the projection of ∇H onto $\underline{n}_J$, represents change in H per unit change in the joint motion in the direction of $\underline{n}_J$. Thus, projection of ∇H onto the one dimensional null space of J can be represented as:

$$(I - J^{+}J)\nabla H = \frac{\partial H}{\partial \theta_h}\underline{n}_J , \tag{3.4}$$

where θ_h is the joint displacement in the direction of $\underline{n}_J$. This may also be shown by defining the component of the joint angle vector $\dot{\underline{\theta}} \epsilon R^7$ along the homogeneous solution as one of its generalized coordinates.

We may now rewrite Eq. (2.2) of Chapter 2 to determine the joint velocities to optimize $H(\underline{\theta})$ as:

$$\dot{\underline{\theta}} = J^{+}\dot{\underline{x}} + k\frac{\partial H}{\partial \theta_h}\underline{n}_J , \tag{3.5}$$

which can be written as:

$$\dot{\underline{\theta}} = \dot{\underline{\theta}}_{min} + k\frac{\partial H}{\partial \theta_h}\underline{n}_J . \tag{3.6}$$

In the numerical implementation of Eq. (3.6), we determine $\Delta\underline{\theta}$, change in the joint angle vector for the time period Δt as:

$$\Delta\underline{\theta} = \Delta\underline{\theta}_{min} + k\frac{\Delta H_h}{\|\Delta\underline{\theta}_h\|}\underline{n}_J, \tag{3.7}$$

where the time derivative terms of Eq. (3.6) are replaced by the corresponding incremental Δ terms. In Eq. (3.7), $\Delta\theta_{min}$ may be computed using an efficient method such as proposed by Dubey et al., [20], and ΔH_h is obtained as follows:

$$\Delta H_h = H(\underline{\theta} + \Delta\underline{\theta}_h) - H(\underline{\theta}), \tag{3.8}$$

where $\underline{\theta}$ is the current joint angle vector. Thus ΔH_h represents change in H due to the self motion only. The scaler coefficient k in Eq. (3.7) is taken to be a negative value to minimize several performance criteria in the implementation presented in this thesis. This numerical resolution method allows us to optimize a performance criterion without knowing the symbolic form of its gradient. This method can be extended to multi-degree-of redundancy manipulators. However, the calculation costs will increase with the increase in the degree of redundancy. For n degree-of-redundancy system, numerical method will require n+1 evaluations of the performance criterion.

The implementation of the above numerical scheme in the control program is done through the following steps:

(1) Calculate the Jacobian matrix for the current configuration.

(2) Find the least-norm solution $\Delta\underline{\theta}_{min}$ and the homogeneous solution $\Delta\underline{\theta}_h$.

(3) Calculate the performance criterion $H(\underline{\theta})$ at the present configuration.

(4) Calculate the performance criterion $H(\underline{\theta}+\Delta\underline{\theta}_h)$.

(5) Calculate the change in the performance criteria $\Delta H_h=H(\underline{\theta}+\Delta\underline{\theta}_h)-H(\underline{\theta})$.

(6) Use Eq. (3.7) to get the joint revolution increment $\Delta\underline{\theta}$.

(7) Use forward kinematics to calculate the end-effector position and orientation at $(\underline{\theta}+\Delta\underline{\theta})$.

(8) Calculate the error between the end-effector position and orientation at the desired location and at $(\underline{\theta}+\Delta\underline{\theta})$.

(9) Find the error correction joint increments $\Delta\underline{\theta}_{EC}$ by finding the least-norm solution based on the cartesian error in step (8).

(10) The joint configuration can now be obtained as:

$$\underline{\theta}_{NEW} = \underline{\theta} + \Delta\underline{\theta} + \Delta\underline{\theta}_{EC} \,. \tag{3.9}$$

3.3 Circular Path Planning for Transition Between Two Linear Paths

A circular motion path planning routine is developed to connect two linear paths that a constant speed of end-effector is maintained during the transition. The radius of the circular path may be chosen based on the trajectory requirement and acceleration constraints.

3.3.1 Center of the Circular Path

Assuming the end-effector is required to move from point A to point B, and then to point C. As shown in Fig. 3.2, we can use two straight lines to connect these 3 points. If these 3 points are not on the same straight line, the end-effector must stop at point B

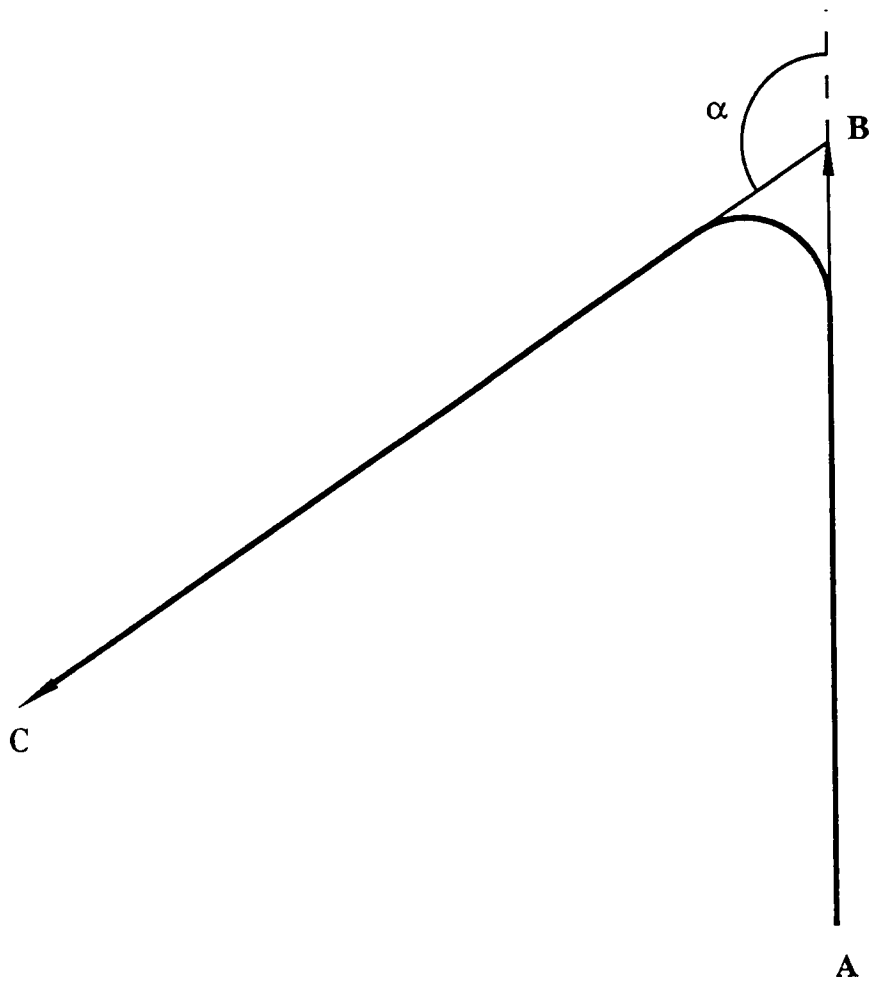


Figure 3.2

A circular path connecting two linear path segments

before going to C. To avoid stopping B and maintain constant speed, we can use a circular path which is tangential to straight lines $\underline{AB}$ and $\underline{BC}$. Since circular path and both the straight lines are in the same plane, the direction of $\underline{n}_p$, the normal to this plane, can be given by:

$$\underline{n}_p = \underline{AB} \times \underline{BC} \ . \tag{3.10}$$

The angle between the two lines can be calculated by the law of cosines as:

$$\alpha = \cos^{-1}\left[\frac{\underline{AB} \cdot \underline{BC}}{|\underline{AB}| \cdot |\underline{BC}|}\right] . \tag{3.11}$$

We can determine a circular path of a radius R, such that it is tangential to $\underline{AB}$ and $\underline{BC}$ at points P1 and P2, respectively as shown in Fig. 3.3. The distance between P1 and point B in Fig. 3.3 can be obtained by:

$$D_{P1 \to B} = R \cdot \tan(\frac{\alpha}{2}) \ . \tag{3.12}$$

The value of $D_{P1 \to B}$ is also the same as $D_{P2 \to B}$, the distance between P2 and point B. Points A, B and C can be represented by their coordinates as (X_A, Y_A, Z_A), (X_B, Y_B, Z_B) and (X_C, Y_C, Z_C) respectively. The coordinates of the tangent point P1 are:

$$\begin{aligned} X_{P1} &= X_B - \frac{D_{P1 \to B}}{|\underline{AB}|} \cdot (X_B - X_A) \\ Y_{P1} &= Y_B - \frac{D_{P1 \to B}}{|\underline{AB}|} \cdot (Y_B - Y_A) \\ Z_{P1} &= Z_B - \frac{D_{P1 \to B}}{|\underline{AB}|} \cdot (Z_B - Z_A) \ . \end{aligned} \tag{3.13}$$

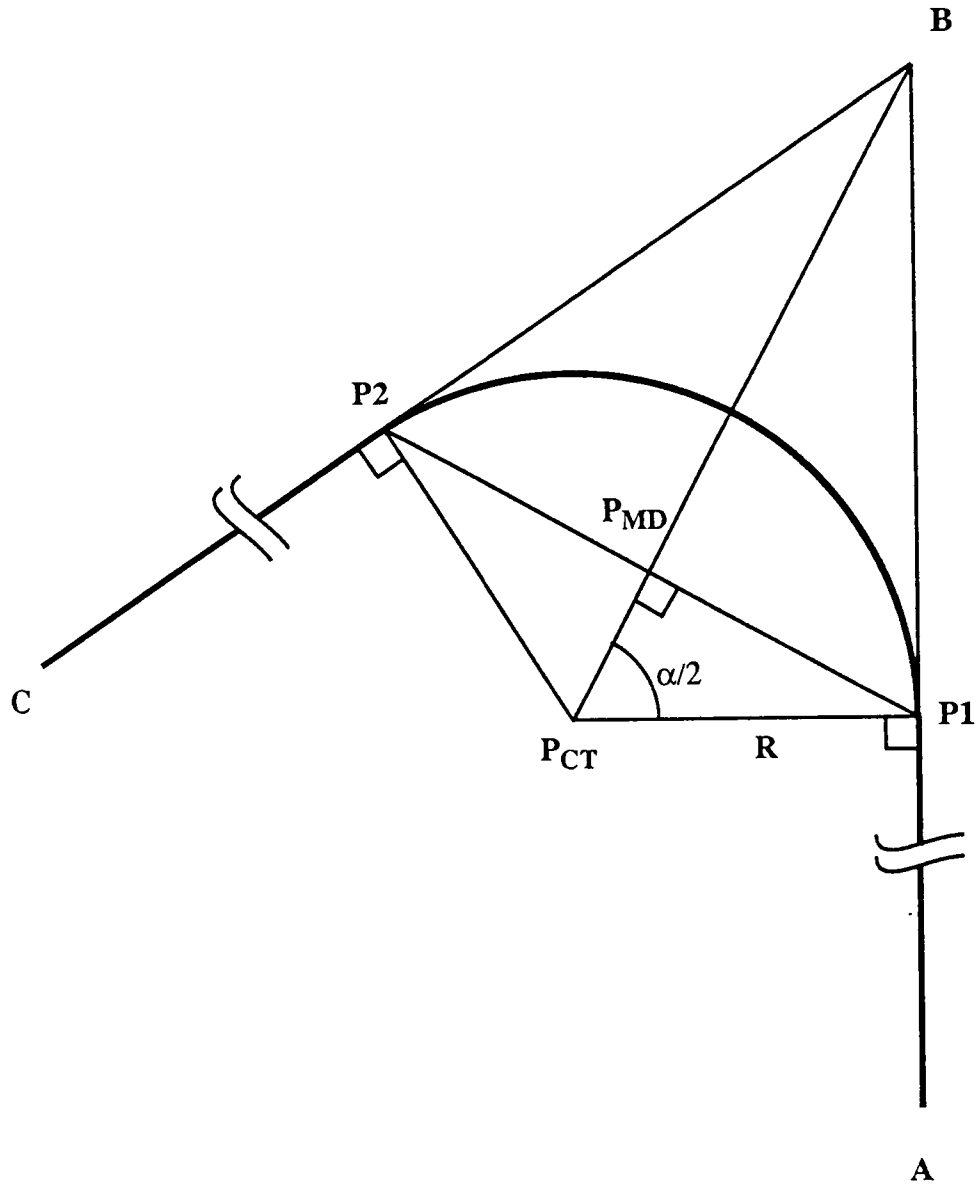


Figure 3.3

Determination of the center of the circular path

In the same way we can get the coordinates of the tangent point P2 as:

$$\begin{aligned} X_{P2} &= X_B - \frac{D_{P1\to B}}{|\underline{BC}|} \cdot (X_B - X_C) \\ Y_{P2} &= Y_B - \frac{D_{P1\to B}}{|\underline{BC}|} \cdot (Y_B - Y_C) \\ Z_{P2} &= Z_B - \frac{D_{P1\to B}}{|\underline{BC}|} \cdot (Z_B - Z_C) \,. \end{aligned} \tag{3.14}$$

From the triangular relationships we can easily show that:

$$\frac{|P_{CT}\, B|}{|P_{MD}\, B|} = \frac{1}{\sin^2(\frac{\alpha}{2})} \,, \tag{3.15}$$

where $|P_{CT}B|$ is the distance between the circle center P_{CT} and point B. $|P_{MD}\, B|$ is the distance between the point P_{MD}, the midpoint between the two tangent point, and point B. Finally, we get the circular path center coordinates as:

$$\begin{aligned} X_{CT} &= X_B + \left[\frac{1}{2}(X_{P1} + X_{P2}) - X_B\right] \cdot \frac{1}{\sin^2(\frac{\alpha}{2})} \\ Y_{CT} &= Y_B + \left[\frac{1}{2}(Y_{P1} + Y_{P2}) - Y_B\right] \cdot \frac{1}{\sin^2(\frac{\alpha}{2})} \\ Z_{CT} &= Z_B + \left[\frac{1}{2}(Z_{p1} + Z_{P2}) - Z_B\right] \cdot \frac{1}{\sin^2(\frac{\alpha}{2})} \,. \end{aligned} \tag{3.16}$$

3.3.2 Circular Path Planning Calculations

Now we build a coordinate frame $\underline{x}_P$, $\underline{y}_P$, $\underline{z}_P$, that has its origin at the circle center as shown in Fig. 3.4. The $\underline{z}_P$ axis is in the same direction as $\underline{n}_p$, the $\underline{x}_P$ axis goes through

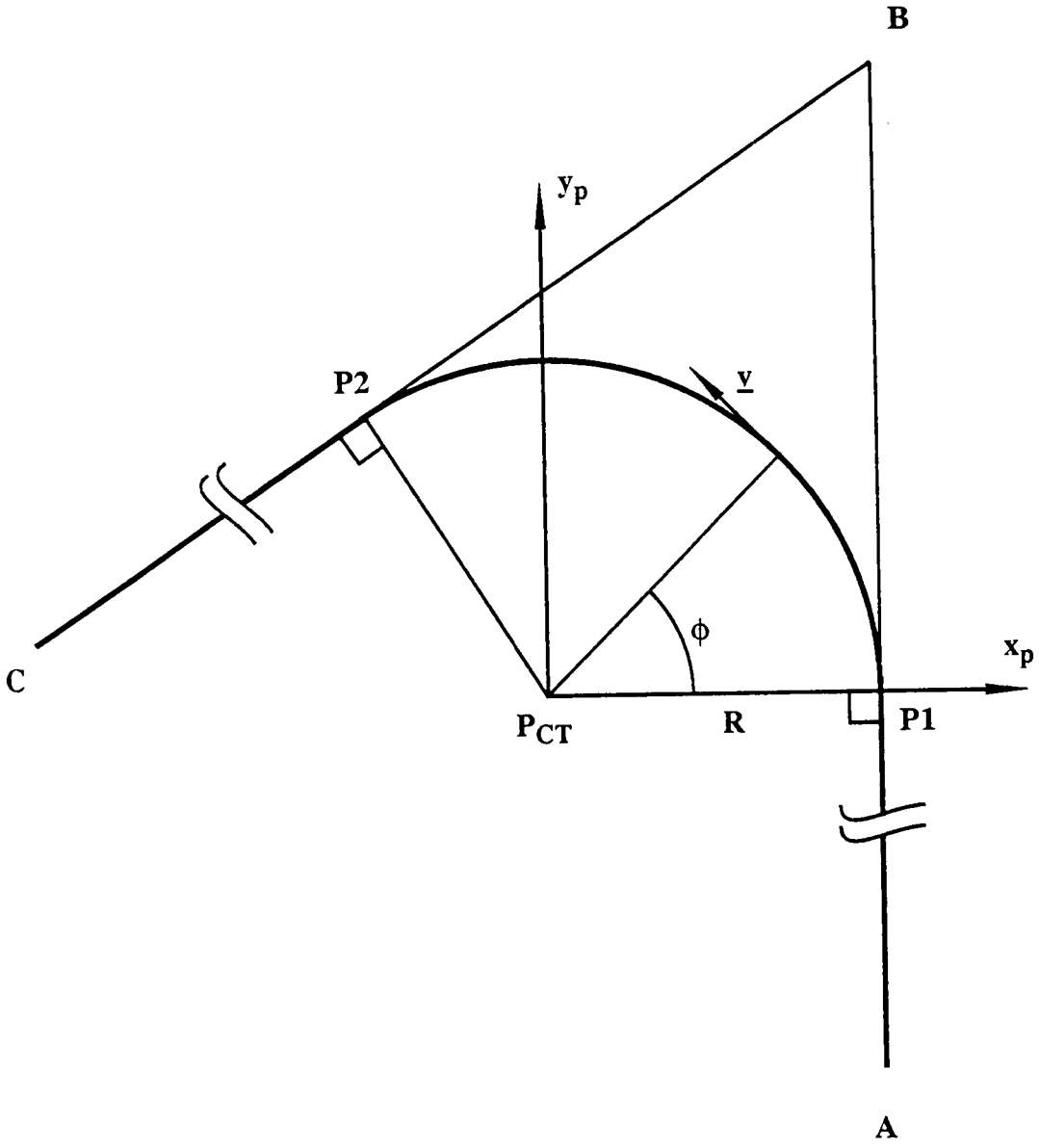


Figure 3.4

Determination of the end-effector velocity components

point P1, and the $\underline{y}_P$ axis is obtained by:

$$\underline{y}_P = \underline{z}_P \times \underline{x}_P \ , \tag{3.17}$$

where $\underline{x}_P$, $\underline{y}_P$ and $\underline{z}_P$ are unit vectors. Moving along the circular path, the end-effector velocity has no component in the $\underline{z}_P$ direction. These velocity components can be calculated by:

$$\begin{aligned} v_{xp} &= -|\underline{v}| \cdot \sin\phi \\ v_{yp} &= |\underline{v}| \cdot \cos\phi \\ v_{zp} &= 0 \ . \end{aligned} \tag{3.18}$$

where v_{xp}, v_{yp} and v_{zp} are the velocity components along $\underline{x}_P$, $\underline{y}_P$ and $\underline{z}_P$ directions, and ϕ is the angle between $\underline{x}_P$ and the line connecting the circle center to the end-effector. Assuming the time variable t is zero when the end-effector begins its circular motion, we have:

$$\begin{aligned} \phi &= \frac{|\underline{v}|}{R} \cdot t \\ 0 \le t &\le \frac{R \cdot \alpha}{|\underline{v}|} \ . \end{aligned} \tag{3.19}$$

The end-effector velocity vector with respect to the base coordinates can be calculated by:

$$\underline{v} = v_{xp} \cdot \underline{x}_P + v_{yp} \cdot \underline{y}_P + v_{zp} \cdot \underline{z}_P \ . \tag{3.20}$$

From Eq. (3.18), (3.19) and (3.20) we obtain:

$$\underline{v} = |\underline{v}| \cdot \left(\underline{y}_P \cdot \cos(\frac{|\underline{v}|}{R} \cdot t) - \underline{x}_P \cdot \sin(\frac{|\underline{v}|}{R} \cdot t) \right) . \tag{3.21}$$

If we specify the speed of end-effector as $|\underline{v}|$, the norm of its velocity vector, we

can use Eq. (3.21) to calculate the velocity vector $\underline{v}$ on the circular path.

3.3.3 Computation of Circular Path by Finite Steps

Since the end-effector motion is computed numerically by small increments, the end-effector moves away from the original circular path in each step. Therefore, implementation of Eq. (3.21) produces numerical errors. These errors will be zero if the steps become infinitesimally small or if one of the joint axis coincides with $\underline{z}_P$. Using very small increments to approach the circular motion can reduce the error, but it increases the calculation cost and requires very high communication speed.

A numerical way to reduce the error is to approximate the circular path by a series of connected chords as shown in Fig. 3.5. The lengths of the chords are chosen according to the distance of the end-effector moving in each step of increment. We can calculate $\Delta\phi$ as in Fig 3.5 as:

$$\Delta\phi = 2 \cdot \sin^{-1}\left(\frac{|\underline{v}|}{2R}\right) . \tag{3.22}$$

In the $\underline{x}_P$, $\underline{y}_P$, $\underline{z}_P$ coordinate frame, we can use the present position of end-effector coordinate X_{Ci} and Y_{Ci} to calculate the next step end-effector coordinate X_{Ci+1}, Y_{Ci+1} by:

$$\begin{aligned} X_{Ci+1} &= X_{Ci} \cdot \cos(\Delta\phi) - Y_{Ci} \cdot \sin(\Delta\phi) \\ Y_{Ci+1} &= Y_{Ci} \cdot \cos(\Delta\phi) + X_{Ci} \cdot \sin(\Delta\phi) \end{aligned} \tag{3.23}$$

The difference between the two consecutive points in $\underline{x}_P$, $\underline{y}_P$, $\underline{z}_P$ coordinates can

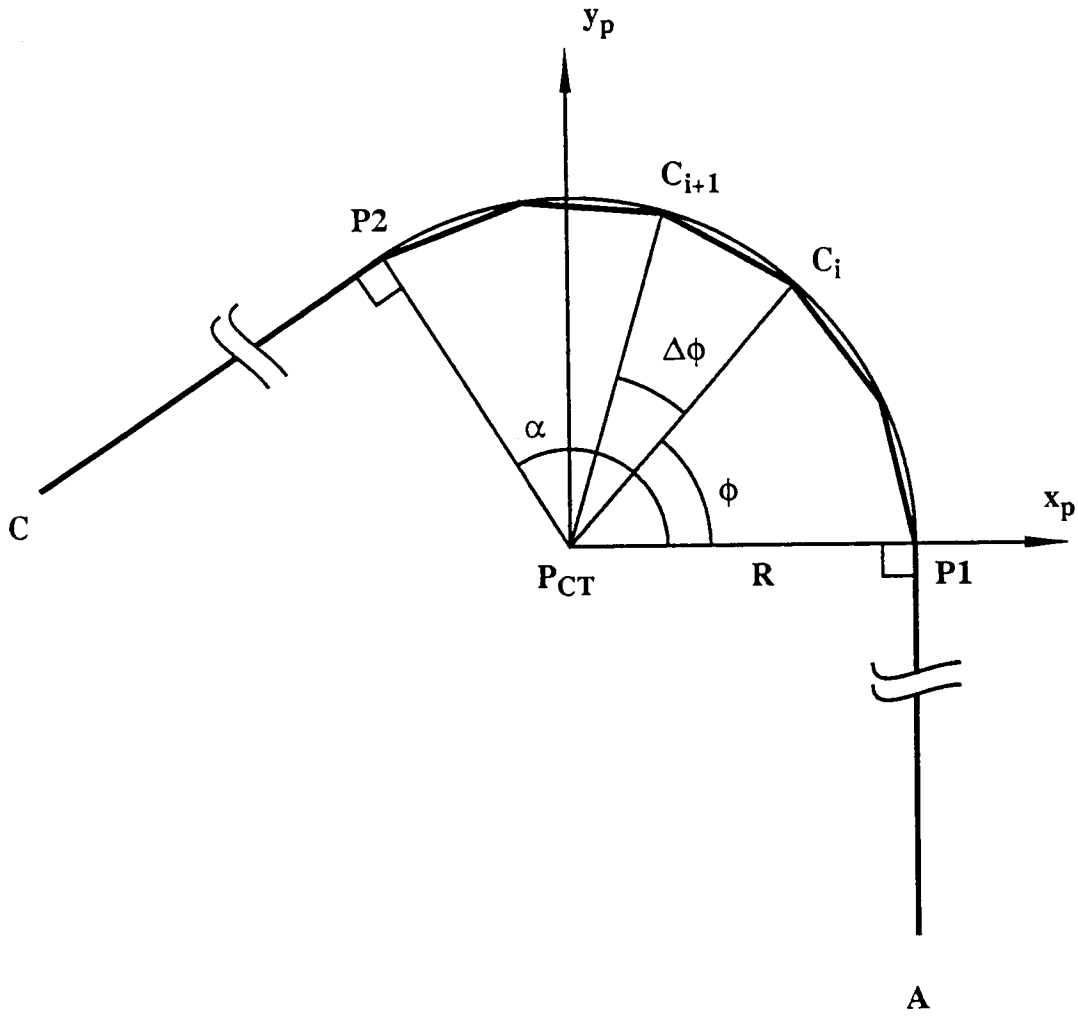


Figure 3.5

Approximation of the circular path by a series of connected chords

be transformed to the base coordinates by:

$$\begin{bmatrix} \Delta X \\ \Delta Y \\ \Delta Z \end{bmatrix} = (X_{Ci+1} - X_{Ci}) \cdot \underline{x}_P + (Y_{Ci+1} - Y_{Ci}) \cdot \underline{y}_P \tag{3.24}$$

To maintain a constant speed during the circular motion, $\Delta\phi$ will be incremental steps. Since $\sin(\Delta\phi)$ and $\cos(\Delta\phi)$ are calculated only once, computational cost are minimal.

3.3.4 Smooth Transition Between the Circular Path and the Linear Path

When the end-effector moves from point A to the first tangent point P1 using the same length of the moving distance $|\Delta\underline{r}|$ for each incremental step, it does not necessarily arrive at point P1. In that case, a linear step different from $|\Delta\underline{r}|$ will be required before the circular motion, and would result in a change of speed. Therefore, we need to get the step $\Delta\underline{r}_{con1}$ to connect the linear path and the circular path, such that $|\Delta\underline{r}_{con1}|$ is equal to $|\Delta\underline{r}|$. As shown in Fig 3.6, the linear motion ends at point P_{LE1} before the tangent point P1, and the distance $|P_{LE1} \rightarrow P1|$ between P_{LE1} and P1 is smaller than or equal to $|\Delta\underline{r}|$. From Figure 3.6, we can calculate the angle between $P_{CT} \rightarrow P_{LE1}$ and $\underline{x}_P$ as:

$$\gamma = \tan^{-1}\left[\frac{|P_{LE1} \rightarrow P1|}{R}\right] \tag{3.25}$$

The connection line of the first point on the circular path to the circular path

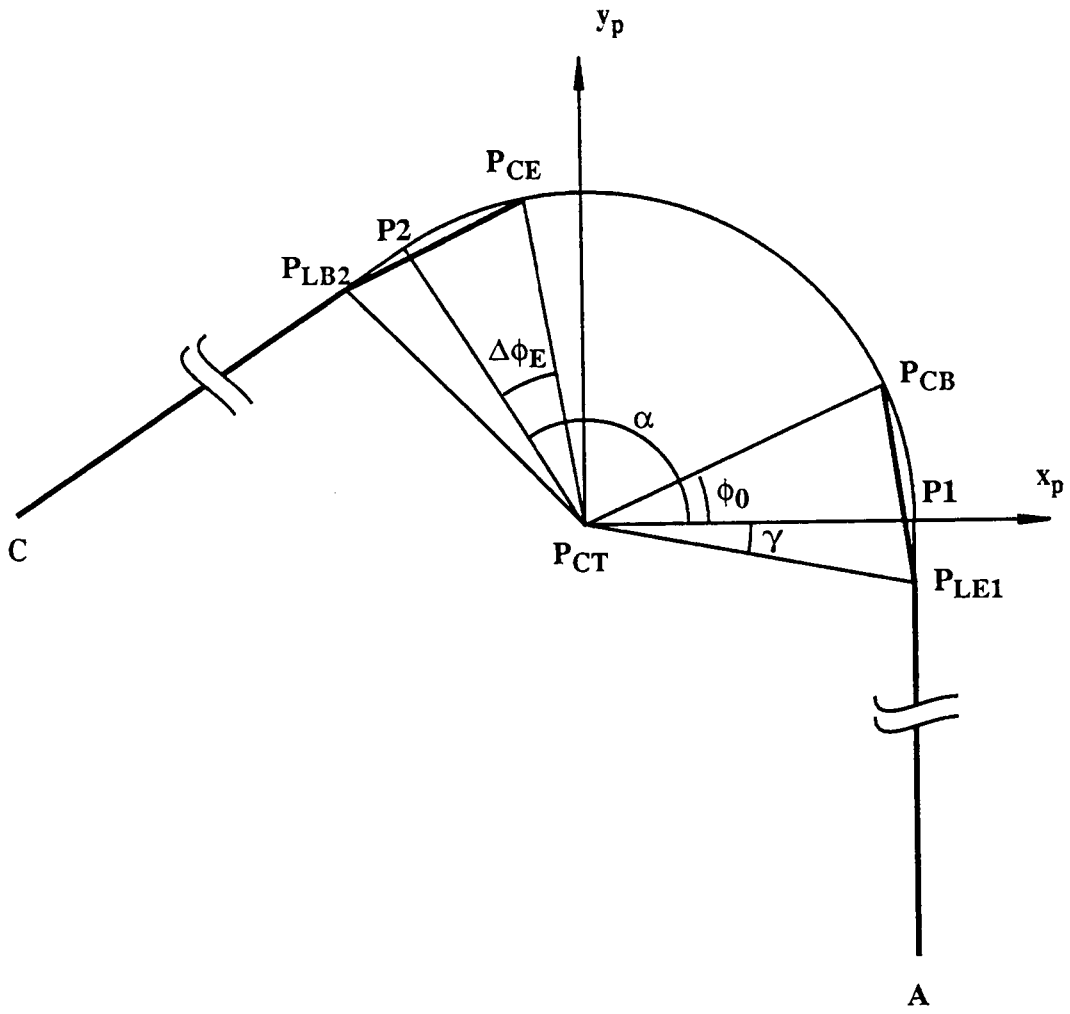


Figure 3.6

Smooth transition between the circular and linear path

center has the initial ϕ_0, which can be gotten by:

$$\phi_0 = \cos^{-1}\left[\frac{2R^2 + |P_{LE1}\rightarrow P_1|^2 - |\underline{\Delta r}|^2}{2 \cdot R \cdot \sqrt{R^2 + |P_{LE1}\rightarrow P_1|^2}}\right] - \gamma \tag{3.26}$$

The difference between the first point on the circle P_{CB} and P_{LE1} in $\underline{x}_P$, $\underline{y}_P$ and $\underline{z}_P$ coordinate are:

$$\begin{aligned} \Delta X_{CB} &= R \cdot \cos(\phi_0) - R \\ \Delta Y_{CB} &= R \cdot \sin(\phi_0) + |P_{LE1}\rightarrow P1| \\ \Delta Z_{CB} &= 0 \end{aligned} \tag{3.27}$$

Then the displacement vector with respect to the base coordinate is:

$$\underline{\Delta r}_{con1} = \Delta X_{CB} \cdot \underline{x}_P + \Delta Y_{CB} \cdot \underline{y}_P \tag{3.28}$$

Similarly, we can calculate the step which connects the circular path to the second linear path. When the circular motion end before it reaches the tangent point P2, and the angle $\Delta\phi_E$ is smaller than or equal to $\Delta\phi$. then $\Delta\phi_E$ can be obtained by:

$$\Delta\phi_E = \alpha - N \cdot \Delta\phi - \phi_0 \tag{3.29}$$

Here N is the total steps of the circular motion.

At this time, the end-effector position is at P_{CE}. Its components in the $\underline{x}_p$, $\underline{y}_p$ and $\underline{z}_p$ coordinates are:

$$\begin{aligned} X_{CE} &= R \cdot \cos(N \cdot \Delta\phi + \phi_0) \\ Y_{CE} &= R \cdot \sin(N \cdot \Delta\phi + \phi_0) \\ Z_{CE} &= 0 \end{aligned} \tag{3.30}$$

The distance between P2 and the point P_{LB2} on the line $\underline{BC}$ is:

$$|P2 \rightarrow P_{LB2}| = \sqrt{|\underline{\Delta r}|^2 - R^2 \cdot [1 - \cos(\Delta\phi_E)]^2} - R \cdot \sin(\Delta\phi_E) \tag{3.31}$$

The $\underline{x}_P$, $\underline{y}_P$ and $\underline{z}_P$ coordinates of the connection point P_{LB2} are:

$$\begin{aligned} X_{LB2} &= R \cdot \cos(\alpha) - |P2 \rightarrow P_{LB2}| \cdot \sin(\alpha) \\ Y_{LB2} &= R \cdot \sin(\alpha) + |P2 \rightarrow P_{LB2}| \cdot \cos(\alpha) \\ Z_{LB2} &= 0 \end{aligned} \tag{3.32}$$

The difference $\Delta\underline{r}_{con2}$ in the base coordinate can be calculated by:

$$\Delta\underline{r}_{con2} = (X_{LB2} - X_{CE}) \cdot \underline{x}_P + (Y_{LB2} - Y_{CE}) \cdot \underline{y}_P \tag{3.33}$$

Using this method, we can let the end-effector keep constant $|\underline{\Delta r}|$ in each step during the linear and circular paths, except the last step of the complete path.

3.4 Path Planning of the End-effector Orientation

The orientation of the end-effector is usually defined by the Euler angles. In the control program of this thesis, the roll angle α represents a rotation about the Z axis, while the pitch angle β represents the rotation of the new y axis after the first rotation about Z axis; and the yaw angle γ represents the rotation about the new x axis after roll and pitch rotations.

For small rotations, the angular velocity about the base coordinates can be written as the combination of the time derivatives of the roll, pitch and yaw angles as:

$$\underline{\omega} = \underline{\dot{\alpha}} + \underline{\dot{\beta}} + \underline{\dot{\gamma}} \tag{3.34}$$

By summing the orthogonal projections of $\underline{\alpha}$, $\underline{\beta}$ and $\underline{\gamma}$ onto each of the axes of the

X, Y and Z coordinates [22], we obtain:

$$\begin{aligned} \omega_x &= \dot{\gamma} - \dot{\alpha} \cdot \sin\beta \\ \omega_y &= \dot{\beta} \cdot \cos\gamma + \dot{\alpha} \cdot \cos\beta \cdot \sin\gamma \\ \omega_z &= \dot{\alpha} \cdot \cos\beta \cdot \cos\gamma - \dot{\beta} \cdot \sin\gamma \end{aligned} \tag{3.35}$$

Interpolating the Euler angles between the current and the desired end-effector orientation, and calculating the angular velocity using Eq. (3.35) does not provide the minimum rotation of the end-effector. In order to move the end-effector from one orientation to another desired orientation with smallest end-effector rotation, a single rotation is used. Let $[H_1]$ and $[H_2]$ be the rotation matrices corresponding to two different configurations of the end-effector. The rotation matrix for the second position relative to the first position is:

$$\begin{bmatrix} l_{1,1} & l_{1,2} & l_{1,3} \\ l_{2,1} & l_{2,2} & l_{2,3} \\ l_{3,1} & l_{3,2} & l_{3,3} \end{bmatrix} = [H_2]^T[H_1] \tag{3.36}$$

We can obtain the magnitude and direction of the single rotation as follows [22]:

$$\begin{aligned} d &= \sqrt{(l_{2,3} - l_{3,2})^2 + (l_{3,1} - l_{1,3})^2 + (l_{1,2} - l_{2,1})^2} \\ a_x &= \frac{1}{d}(l_{2,3} - l_{3,2}) \\ a_y &= \frac{1}{d}(l_{3,1} - l_{1,3}) \\ a_z &= \frac{1}{d}(l_{1,2} - l_{2,1}) \\ \phi &= \sin^{-1}(\frac{d}{2}) \end{aligned} \tag{3.37}$$

where a_x, a_y and a_z are the direction cosines of the rotation axis with respect to the

coordinate fixed on the first position of the end-effector, and ϕ is the rotation angles about this axis. If the value of d in Eq. (3.37) is zero, the two orientations of the end-effector are the same, and no rotation is required.

3.5 Self Motion of a Single Redundant Joint Manipulator

The RRC manipulator K-2107 and its controlling program has the capability to use the teach pendant to let the manipulator change its configuration without moving the end-effector, and this is referred to as the self motion or "Obit motion". In the RRC control program to determine the orbit motion, the magnitude and the direction of joint 1 motion is specified, and rest of the joints are moved such that the end-effector position is unchanged. The problem arises with this method when the first joint velocity must be a zero for the self motion. The combination of joint velocities to achieve the self motion are the elements of the null space vector of the Jacobian matrix. When the manipulator changes the joint configuration by adding the null space vector, the end-effector will not change the position and orientation. If the first element of the null space vector is not supposed to be zero, we assign some finite value to it, and calculate the rest of the elements of this vector according to the ratio of the elements of the null space vector. The following equations is used for this calculation:

$$\left[J_1, J_2, \cdots J_7\right] \cdot \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \vdots \\ \dot{\theta}_7 \end{bmatrix} = \underline{0}^{7\times 1} \ , \tag{3.38}$$

which can be rewritten as:

$$\left[J_2, J_3, \cdots J_7\right] \cdot \begin{bmatrix} \dot{\theta}_2 \\ \dot{\theta}_3 \\ \vdots \\ \dot{\theta}_7 \end{bmatrix} = J_1 \cdot \dot{\theta}_1 \ , \tag{3.39}$$

where J_i is the ith column of the Jacobian matrix. If at some configuration, the first element of the null space vector is zero but we still give a non zero value to the joint 1 velocity, the rest of the joint velocities will go to infinity. On the other hand, even if the configuration does not require the velocity of joint 1 to be exactly zero for self motion, it may be necessary to change its sign and move joint 1 in the opposite direction. In this case if we do not change the sign of joint 1 velocity, the manipulator will oscillate at the current configuration, without moving the elbow further.

This problem can be easily solved by using the previous information of the "Orbit motion". From the previous "Orbit motion", we find out which joint has the maximum joint velocity in that configuration, for that joint we specify the joint velocity joint to calculate the current "Orbit motion" by Eq. (3.38). This method was used in our control program. It allows the manipulator to continue the self motion without stopping or oscillating as happens with the original controller.

CHAPTER 4

LOCALLY OPTIMIZED REDUNDANCY RESOLUTION AND PERFORMANCE EVALUATION

This chapter presents evaluation of locally optimized joint trajectories for a 7-DOF Robotics Research Corporation (RRC) redundant manipulator following a closed end-effector path. Section 4.1 develops a tool of 3-dimensional plots to evaluate the joint trajectories. Section 4.2 discusses the nature of joint trajectories resulting from the pseudoinverse control. Some practical issues important in the implementation of the gradient projection method are discussed in section 4.3. Section 4.4 presents a comparison of pseudoinverse control joint trajectories with the joint trajectories obtained by optimizing the norm of least-norm joint velocities, minimum singular value, and $\sqrt{\det(JJ^T)}$.

4.1 3-D Plot of the Joint Velocity Norm

Fig. 4.1 shows the 7-DOF RRC manipulator in different joint configurations for a fixed starting position of the end-effector on a rectangular path without considering the joint limits. We can use the self motion or the homogeneous solution to change the joint configurations without moving the end-effector. The joint velocity cost required to move along the shown rectangular trajectory would be different for different starting configurations. This cost can be represented by the 3-D plot of the joint velocity norm

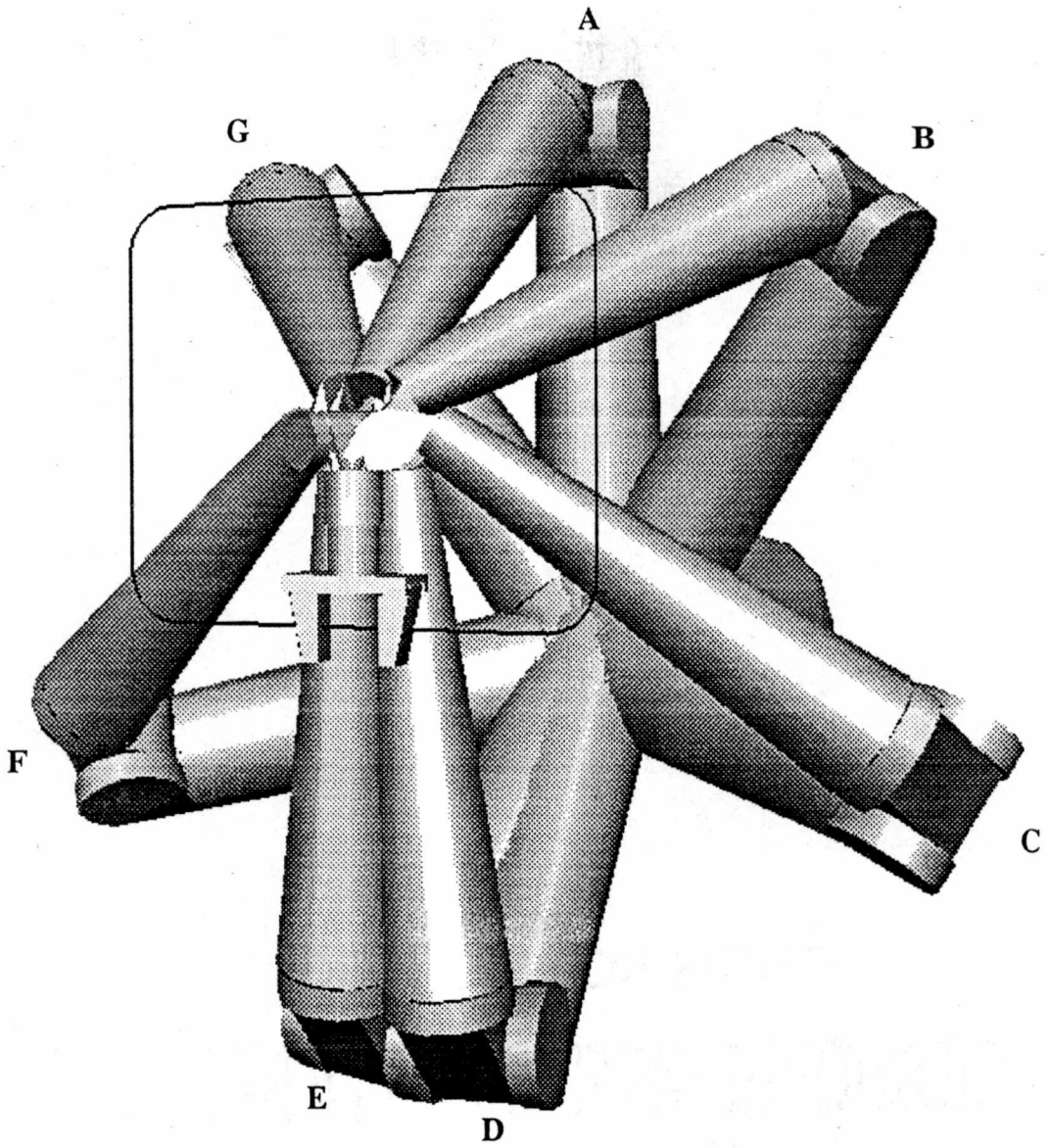


Figure 4.1

Different starting joint configurations for a closed end–effector path

shown in Fig. 4.2. We define the three axes in the 3-D plot to represent the end-effector moving distance, self motion or the configuration changes without moving the end-effector, and the norm of the joint velocity. Since it is a one degree redundancy manipulator, if the configuration of the manipulator is not at a singularity, we can use one particular joint revolution to represent the self motion. This joint revolution should have a one-to-one correspondence with the self motion. In Fig. 4.2, we choose the revolution of joint 5 to represent the self motion because if we fix the end-effector at a specified position and orientation along the rectangular trajectory in Fig. 4.1, and specify the revolution of joint 5, a unique solution is obtained for the remaining six joints. The configuration of the manipulator is completely defined if we specify the position of the end-effector along its trajectory and the revolution of the joint 5. In case, a one-to-one correspondence between a joint motion and self motion does not exist for the complete end-effector path, we can divide the path or the self motion to get the 3-D plot in smaller parts. In the joint trajectory calculations, we keep the end-effector orientation fixed and move it with a constant speed along the rectangular trajectory. Circular path planning is used to generate circular paths connecting two linear segments of the rectangular path, and maintaining the linear speed.

The curves A, B, C, D, E, F and G plotted in the Fig. 4.2 were obtained by using the least-norm solution to follow the rectangular path at different starting configurations of the manipulator. The end-effector moves through positions 1, 2, 3, and 4 in that order and back to the starting position 1 as shown in Fig. 4.3. The end-effector positions 1 through 4 can be represented by the curves 1-1, 2-2, 3-3 and 4-4 in Fig. 4.2. These

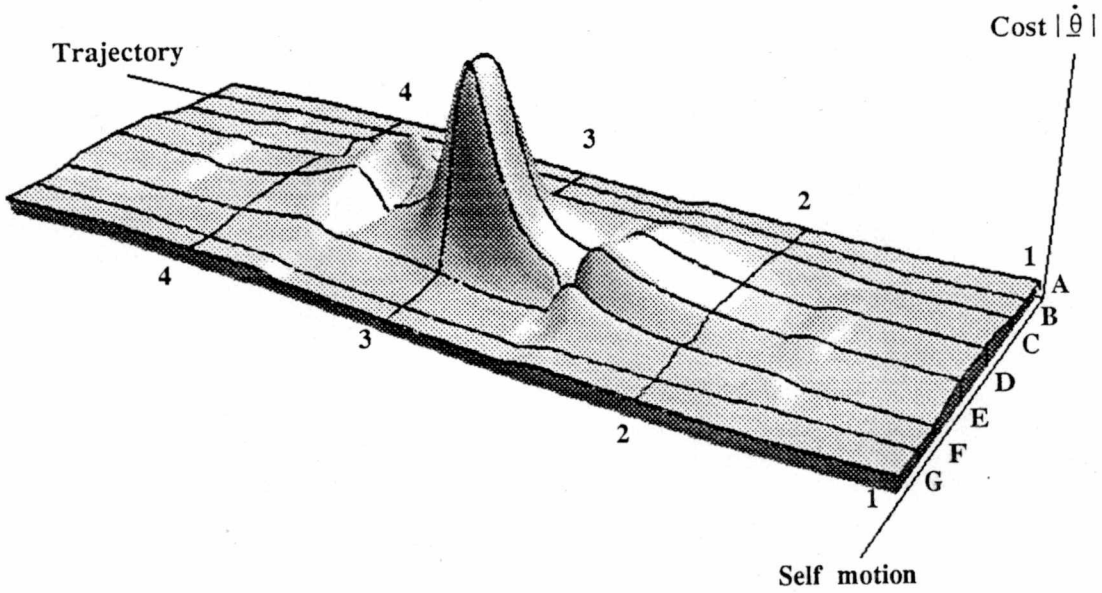


Figure 4.2

3–D cost map of the least–norm joint velocity norm

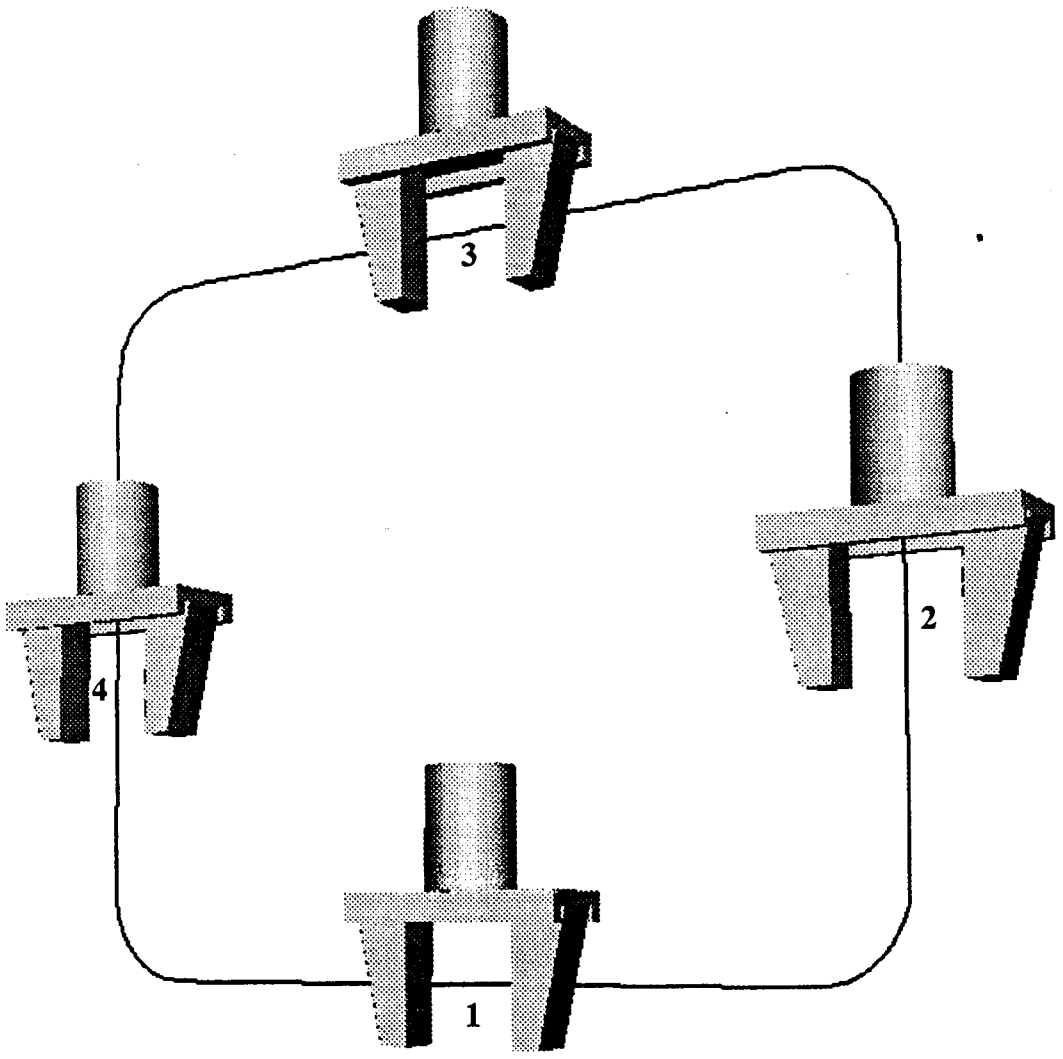


Figure 4.3

End–effector at different positions along the closed path

curves represent the self motion of the configuration corresponding to the fixed end-effector positions. From the plot we can see that if the starting configuration is at A, B, C or G, the joint velocity norm is lower while following the given rectangular trajectory. However, starting at D or E will result in very large joint velocity norms when the end-effector is near or at position 3. By generating several curves either from the self motion at a number of locations along the end-effector trajectory, or by moving the manipulator using the least-norm solution from several different starting configurations we obtain a unique surface in the defined 3-D space. The surface shown in Fig. 4.2 contains the norm of the least-norm solutions along the end-effector trajectory for all possible configurations of the manipulator. We call it the "least norm surface" or the "3-D map of the least-norm". When we use the least-norm solution to move the end-effector following the trajectory, we get a curve on the least-norm surface. If we use the solution other than the least-norm solution to follow the end-effector trajectory, the resulting cost in terms of the joint velocity norm, is above the least-norm surface. If we change the representation of the cost in the 3-D plots by some other performance criterion such as the inverse of the $\sqrt{\det(JJ^T)}$ or the inverse of the minimum singular value of Jacobian matrix, we can get the 3-D map of $1/\sqrt{\det(JJ^T)}$ or the 3-D map of $1/\sigma_{min}$.

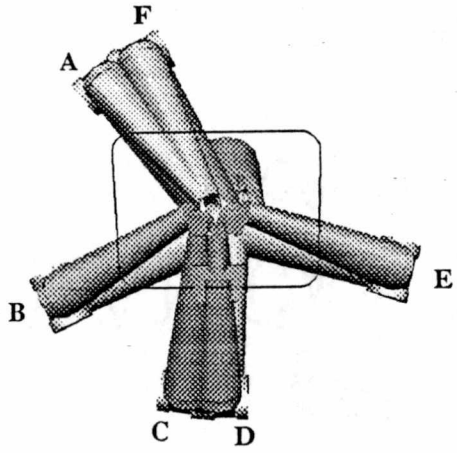
4.2 Pseudoinverse Control

It may be noted from Fig. 4.2 that the cost curves for the pseudoinverse or least-norm control may vary considerably depending on the starting configuration. As discussed before, for the starting configuration A, the joint velocity norm does not change

much along the end-effector trajectory. However, for the starting configuration D, the velocity norm becomes very large near position 3 of the same end-effector trajectory.

As shown in earlier works [12] [23], the pseudoinverse control is generally not conservative, i.e., a closed end-effector trajectory dose not result in a closed path joint trajectory. If the closed end-effector trajectory is repeated, there is a continuous drift in the joint trajectory, which may or may not reach a limit or a steady state. After the end-effector goes back to the starting point, the joint configuration is different from the previous starting configuration. If for each successive loop, the starting configuration changes continuously in the same direction, it may not reach the limit.

For example, the end-effector trajectory of Fig. 4.3 has no limit for the pseudoinverse solution. As shown in Fig. 4.4, if we let the starting configuration be A, after 3 complete cycles, the starting configuration changes to B. After 7 complete cycles, the starting configuration changes to C. The starting configuration is D after 8 cycles. Both starting configurations C & D give high cost curves for the pseudoinverse control. After 20 cycles of the pseudoinverse control, the starting configuration F is reached, which is very near A, and represents joint 5 rotation of about 360 degrees. Thereafter, the same pattern is repeated. Such a behavior was noted for most closed trajectories for which simulations were performed. In a few cases a limit or a steady state was reached for the joint trajectory, and it continued to be conservative from there on. Such instances were observed only for the end-effector trajectories with nearly flat least-norm surfaces. As seen from this example, the pseudoinverse solution does not avoid the high velocity configurations or singularities.



A---starting configuration of the first loop
B---starting configuration of the third loop
C---starting configuration of the seventh loop
D---starting configuration of the eighth loop
E---starting configuration of the twelfth loop
F---starting configuration of the twenty-first loop

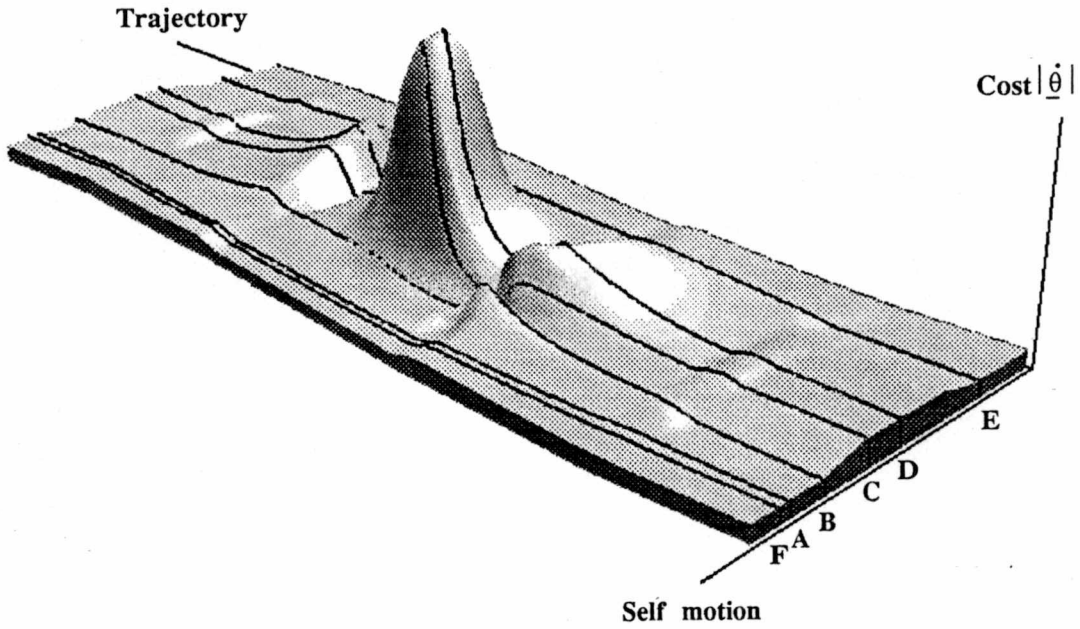
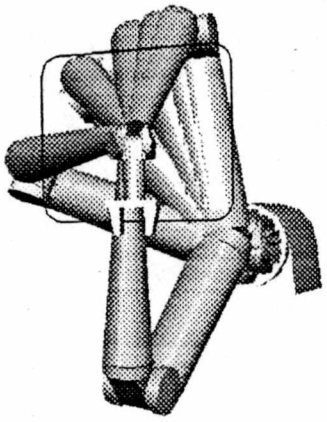


Figure 4.4

Least-norm cost curves for different loops

4.3 Some Practical Issues in the Implementation of the Gradient Projection Optimization

The gradient projection method provides only the direction of the null space vector to be used as self motion for optimization. This direction is obtained by projecting the gradient of the performance criterion onto the null space of the Jacobian matrix. The magnitude of the self motion may be varied using the scaler coefficient k in Eq. (2.3) of Chapter 2. Efficient computation for the real-time implementation of the gradient projection method has been discussed by Dubey et al. [20] and Zghal et al. [24]. Determination of k based on joint velocity limits and actuator torque bounds was discussed by Euler et al [14] [15]. One example of the gradient projection optimization to minimize the least-norm solution is shown in Fig. 4.5, which shows the 3-D plot of the curves generated by minimizing the norm of the least-norm solution along the end-effector trajectory [10]. Since we want to minimize the norm of the least-norm solution, the self motion should change the configuration toward the low cost direction. As seen in the Fig. 4.5, the curves are above the least-norm surface because we add the self motion to let configuration toward the low cost area. The configuration changes away from the high cost area in the first loop. After several repeated loops using the gradient projection optimization method, the joint trajectory path is near its optimum and the null space projection vector is nearly zero in magnitude. The optimum joint trajectory cost is much lower compared to the first cycle joint trajectory cost, and is very close to the least-norm solution cost in that region. At the optimum, the joint trajectory reaches a limit or a steady-state, and becomes conservative, thus, repeating itself.



p---pseudoinverse control scheme
l---minimize the norm of the least-norm solution

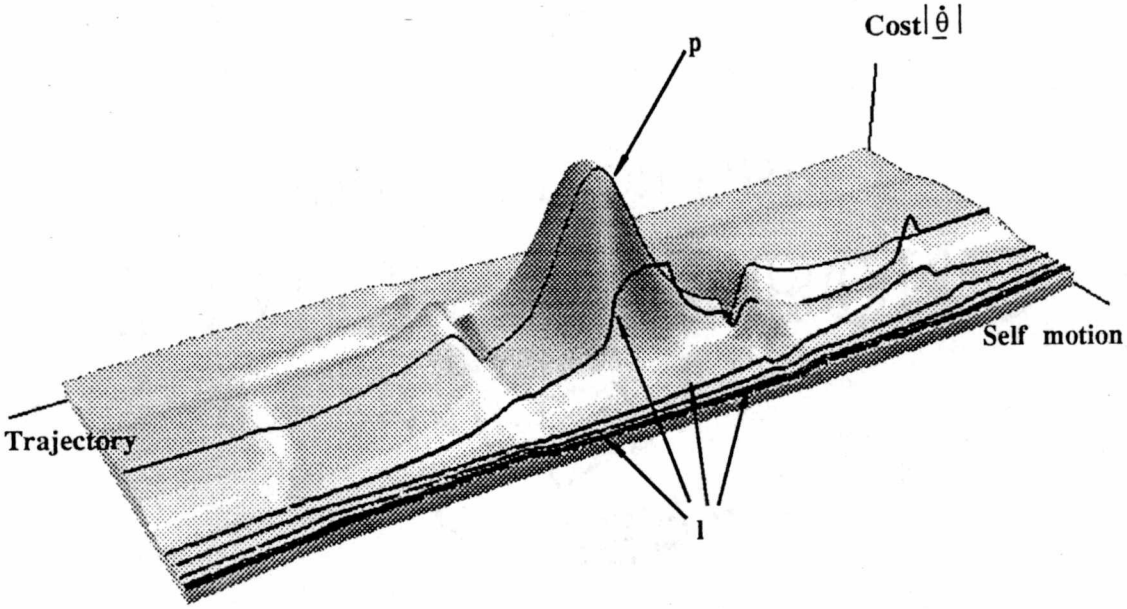


Figure 4.5

Cost curves using least-norm & optimized joint trajectories

There are problems associated with using Eq. (2.3) directly with a constant gain k. Due to large values of $\partial H/\partial\theta_h$ in certain manipulator configurations, the joint velocities may get very large. This is shown in Fig. 4.6 by the trajectory without joint velocity limits. This problem may be solved by reducing k so as to keep the joint velocities within bounds [14]. A joint trajectory obtained by limiting the joint velocities is also shown in Fig. 4.6.

Selection of an appropriate gain factor k is also important in the implementation of the gradient projection method in Eq. (2.3). Fig. 4.7 shows joint velocity cost curves for different values of k used in gradient projection optimization of the norm of the least-norm solution. If the gain factor is too small, the resulting trajectory will be similar to the least-norm trajectory. As shown in Fig. 4.7, a smaller than required k can lead to a high cost joint trajectory limit different from the low cost optimum limit expected in the gradient projection method. A very large gain brings the manipulator near the optimum joint trajectory limit in only one cycle. However, the resulting trajectory oscillates about the optimum trajectory [25]. High gains are also more likely to result in joint velocities above the joint velocity limits.

4.4 Evaluation and Comparison of Different Performance Criteria for Minimizing Joint Velocities

Utilization of redundancy to minimize joint velocities results in avoiding singularity regions and allowing faster end-effector motions within the hardware constraints on joint velocities. Lower joint velocities also result in reduced energy consumption which may be important in some space related and other applications.

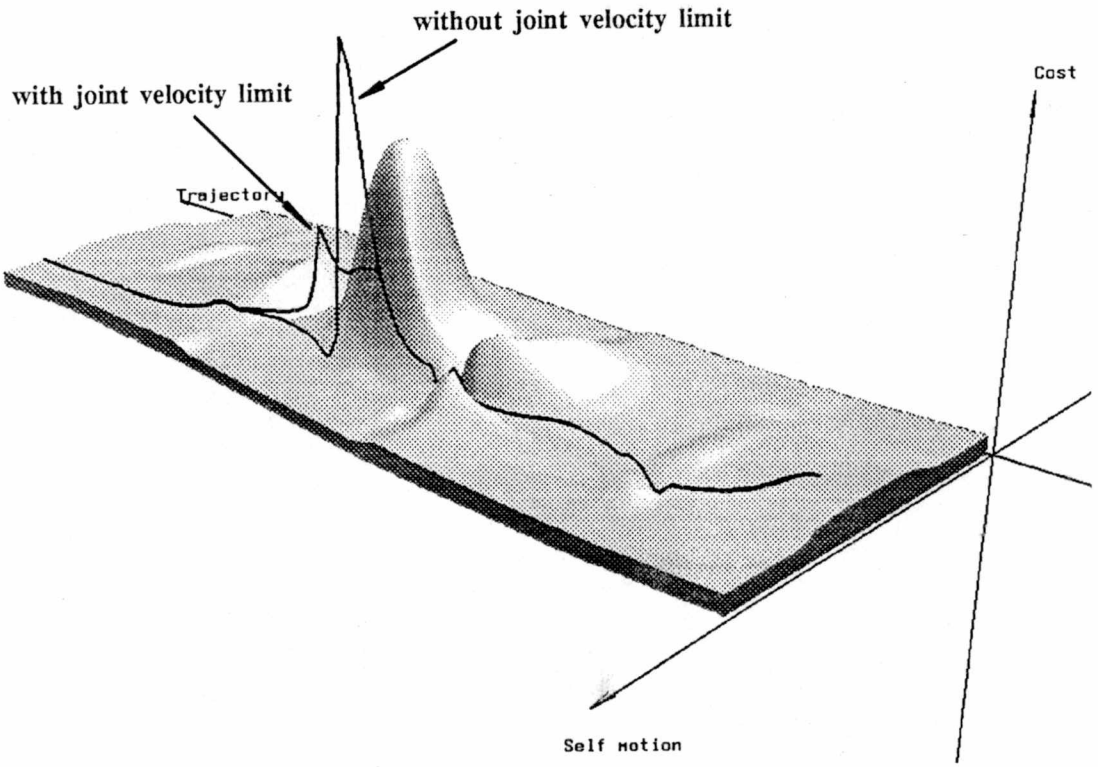


Figure 4.6

Joint trajectories with & without joint velocity limit

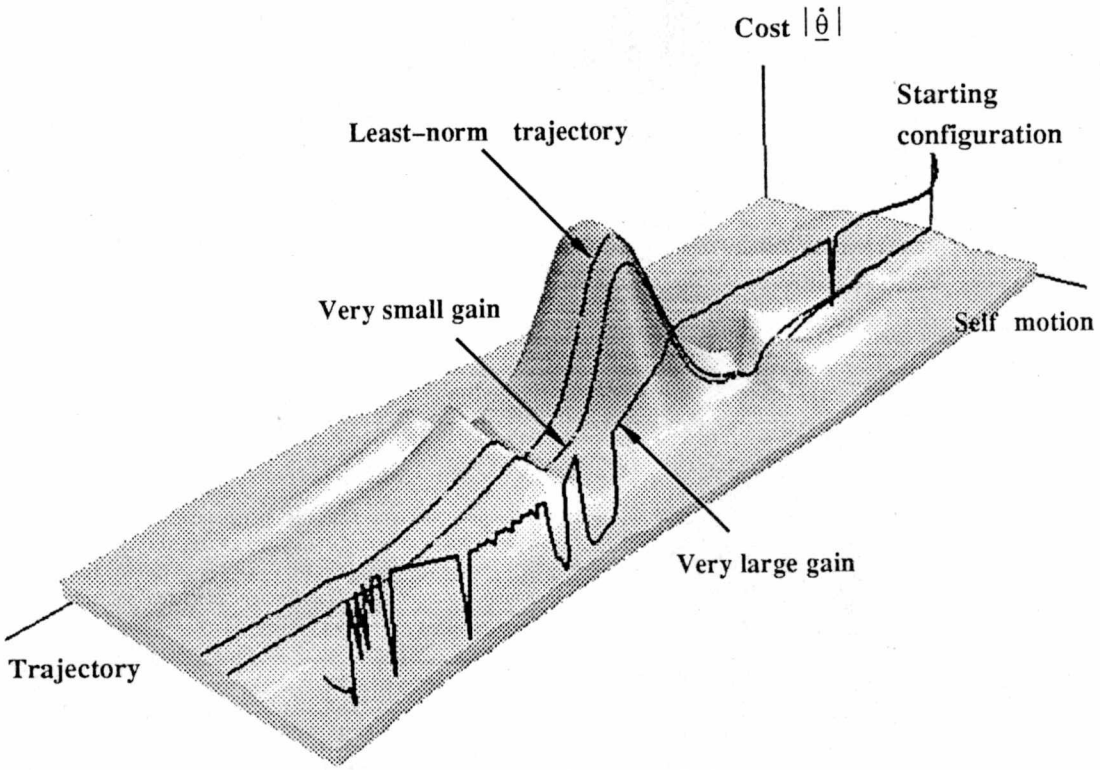


Figure 4.7

Trajectories generated by using a very small and a very large gain factor

Manipulability measure $\sqrt{\det(JJ^T)}$ [7], minimum singular value σ_{min} [8], and Manipulator Velocity Ratio [9] have been used to reduce joint velocities. Chan and Dubey [10] proposed to minimize the norm of the instantaneous least-norm joint velocity to reduce the joint velocity cost. In this section we evaluate the joint trajectories obtained by minimizing a) the inverse of minimum singular value, $1/\sigma_{min}$, b) $1/\sqrt{\det(JJ^T)}$ and c) the norm of the least-norm or pseudoinverse solution.

The joint velocities required to move the manipulator in a given configuration depend on the direction of motion. The eigenvector of JJ^T corresponding to its minimum eigenvalue σ^2_{min}, represents the direction of the end-effector motion requiring maximum joint velocity [9], and σ_{min} is the inverse of the minimum joint velocity norm for a unit end-effector velocity in that direction. Therefore, σ_{min} represents the upper limit on the required joint velocity for the motion of the end-effector. At a singular configuration, σ_{min} is zero implying infinite joint velocity required in the corresponding eigen vector direction. Maximization of σ_{min} results in reducing the maximum joint velocity requirement, and it was proposed by Dubey and Luh [9] for improving a manipulator's flexibility or its ability to move in a direction in which it is worst.

The manipulability measure $\sqrt{\det(JJ^T)}$ [7] is the multiplication of all the singular values, i.e., $\sigma_1 \cdot \sigma_2 \cdots \sigma_m$, for an mxn Jacobian matrix. As it has been shown, each of the singular values σ_i, i=1 to m, is the minimum norm of the joint velocity vector for moving the end-effector with a unit velocity in the direction of the corresponding eigenvector of JJ^T [9]. Thus $\sqrt{\det(JJ^T)}$ represents the combination of the manipulator's ability to move the end-effector in m different directions spanning the m-dimensional

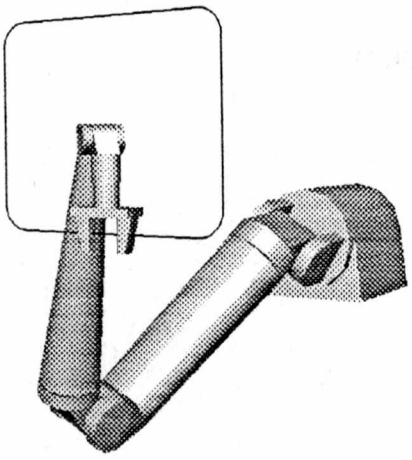
space. Maximization of $\sqrt{\det(JJ^T)}$ results in improving its ability to move in all directions. When the Jacobian matrix is singular, at least one of its singular values is zero, resulting in zero value of $\sqrt{\det(JJ^T)}$. From the determinant value we know whether the manipulator is at a singularity or not. However, if the Jacobian matrix is nonsingular, we may not know whether the configuration is changing toward or away from the singular configuration. This is because when one of the singular values is decreasing, the other singular values may be increasing, resulting in a higher value of $\sqrt{\det(JJ^T)}$ even closer to a singularity. In maximizing $\sqrt{\det(JJ^T)}$, we do not insure the manipulator's ability to move in a particular direction, but it is a combination of its ability to move in different directions.

MVR [9] and the norm of the least-norm joint velocity [10], on the other hand, represent the ability of the manipulator to move in the direction of motion. Thus, by minimizing the required norm of least-norm joint velocity or by maximizing MVR, we utilize the redundancy to improve the manipulator's ability to move in the direction of motion. Therefore, it is task dependent, and does not try to avoid singularities unless the end-effector trajectory contains the components in the direction of the eigenvector of JJ^T corresponding to σ_{min}.

We now evaluate various optimized joint trajectories for the end-effector trajectory in Fig. 4.3. The joint trajectories were optimized by minimizing a) $1/\sigma_{min}$ and b) $1/\sqrt{\det(JJ^T)}$, and c) the norm of the least-norm joint velocity $J^+\underline{\dot{x}}$. By minimizing the $1/\sigma_{min}$ and $1/\sqrt{\det(JJ^T)}$, the performance criterion automatically gets a higher weight near singularity, and thus results in a larger self motion.

It may be noted that in many cases, optimization of $\sqrt{\det(JJ^T)}$, σ_{min}, and the norm of the least-norm solution have similar effect on joint configuration changes. For example, if the desired end-effector velocity has a component along the eigenvector of JJ^T corresponding to σ_{min}, and if the manipulator is near a singularity, then minimizing the norm of the least-norm will result in configurations that also avoid singularities and increase $\sqrt{\det(JJ^T)}$ or σ_{min}. This phenomena may be observed in Fig. 4.8, which shows the cost curves using the three performance criteria for the end-effector trajectory in Fig. 4.3. When the starting manipulator configuration is as shown in Fig. 4.8, minimizing any of the three criteria, i.e., the norm of the least-norm, $1/\sigma_{min}$, or $1/\sqrt{\det(JJ^T)}$, the joint configuration moves toward the low cost area. In this particular case, the curve generated by minimizing the norm of the least-norm has higher cost than the other two curves because there exists a local optimum of the least-norm solution earlier in the path as indicated by two smaller peaks of the 3-D least-norm velocity map. The trajectory gets trapped in the valley between these two peaks due to local optimization. Once the valley is over, the trajectory gets out of the high cost region and into the low cost flat area. It may be noted that the curves shown in Fig. 4.8 shows only the first optimized cycle and not the limit or steady-state cycle. After about 5-6 cycles all three joint trajectories reach a limit in the low cost area as shown in Fig. 4.9. Thereafter, these joint trajectories are conservative and have approximately the same cumulative joint velocity norm cost per cycle.

Cost curves using different starting position for the same end-effector trajectory of Fig. 4.3 are shown in Fig. 4.10. In this case, minimizing the norm of the least-norm



Starting configuration

p---pseudoinverse control scheme
l---minimize the norm of the
least-norm solution
d---minimize $1/\sqrt{\det(JJ^T)}$
m---minimize $1/\sigma_{min}$

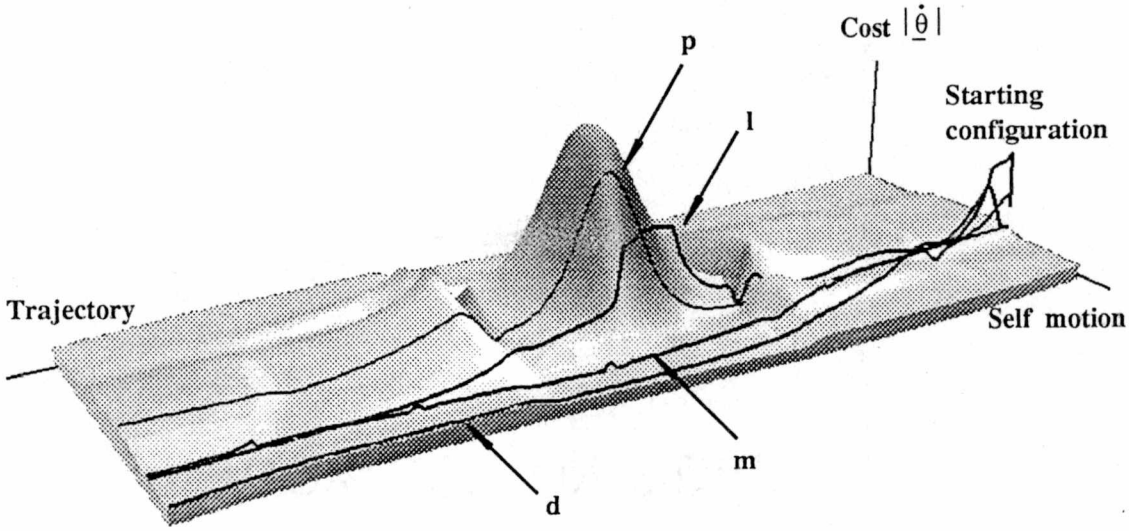
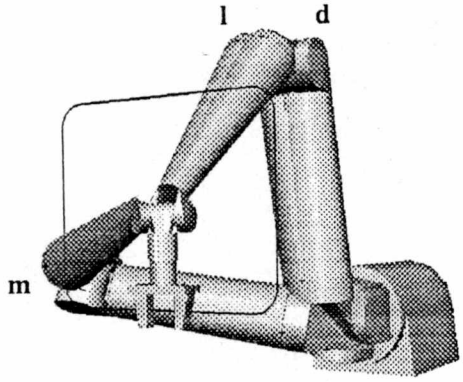


Figure 4.8

Cost curves for different performance criteria (1)



l---minimize the norm of the least-norm solution

d---minimize $1/\sqrt{\det(JJ^T)}$

m---minimize $1/\sigma_{min}$

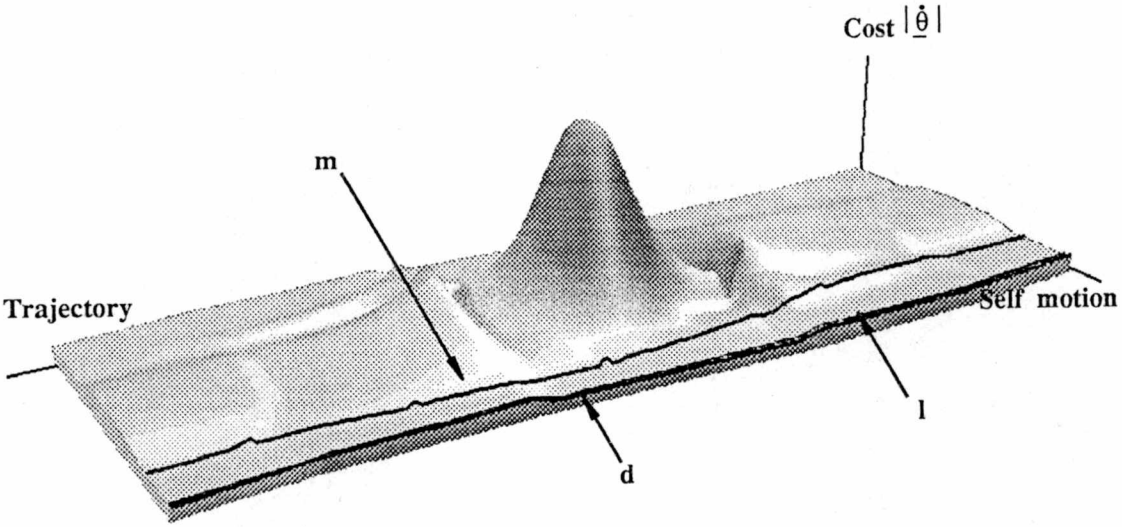
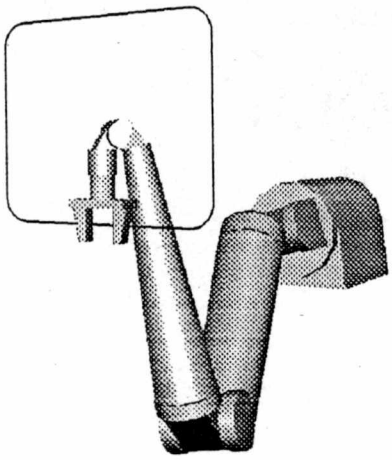


Figure 4.9

Converging loop for different performance criteria (1)



Starting configuration

p---pseudoinverse control scheme
l---minimize the norm of the least
-norm solution
d---minimize $1/\sqrt{\det(JJ^T)}$
m---minimize $1/\sigma_{min}$

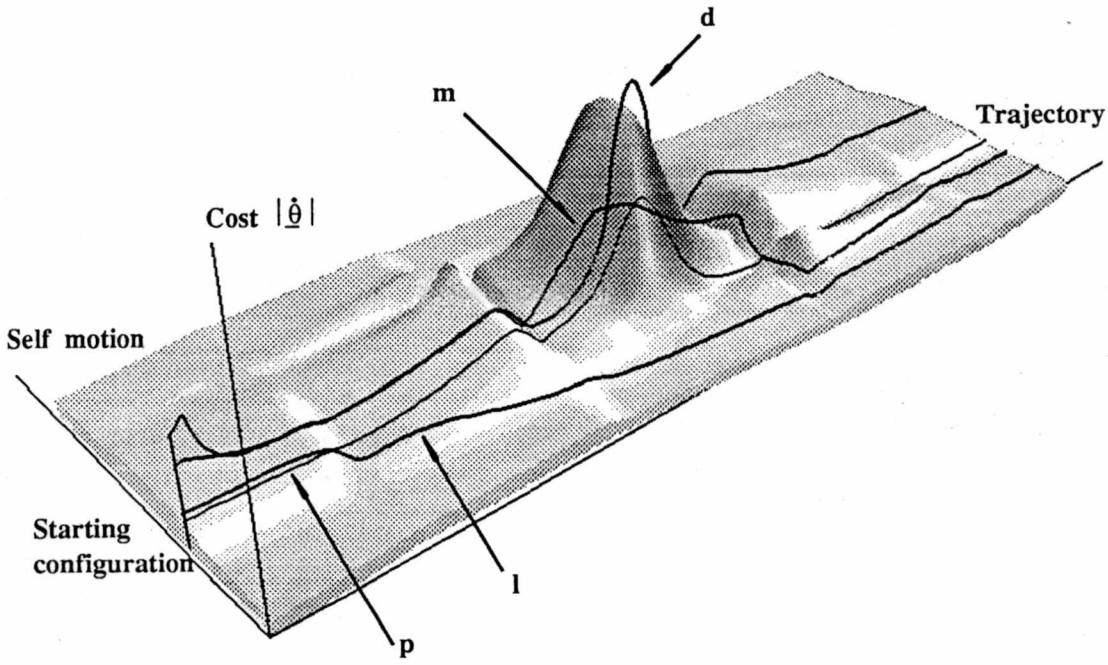
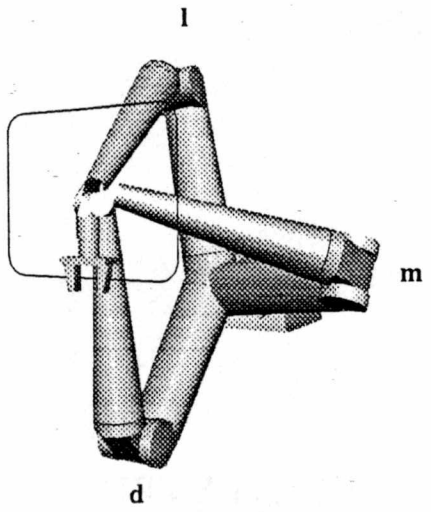


Figure 4.10

Cost curves for different performance criteria (2)

gives the best result. From the 3-D map of the norm of the least-norm, we see that the low cost direction of this trajectory is in the direction in which joint 5 position value decreases. But minimizing $1/\sqrt{\det(JJ^T)}$ or $1/\sigma_{min}$ results in a trajectory toward a high cost area with increased joint 5 value in the beginning. Toward the end of the trajectory, minimizing $1/\sigma_{min}$ moved the trajectory away from the high cost area, while minimizing $1/\sqrt{\det(JJ^T)}$ did not do that. Fig. 4.11 shows the converging loop for different performance criteria.

To understand the behavior of the cost curves in Fig. 4.10 and 4.11, we use the 3-D maps of $1/\sigma_{min}$ and $1/\sqrt{\det(JJ^T)}$. Fig. 4.12 shows the 3-D map of $1/\sigma_{min}$; the cost axis in this map represents $1/\sigma_{min}$. The location of the high cost area in this map is in the similar region in which the costs are high in the 3-D map of the least-norm. This is probably because the end-effector velocity vector in the region has a component in the direction of the eigenvector of JJ^T corresponding to the minimum singular value. The curves a-a, b-b, c-c, and d-d on the map show how the performance criterion $1/\sigma_{min}$ changes with self motion at four different end-effector positions a, b, c and d. These curves lie on the surface of the 3-D map. If $1/\sigma_{min}$ is minimized, the manipulator configuration should change toward the lower value direction of the self motion. It may be seen from the shape of the map that if the starting configuration is between A and B, the configuration will change toward the local optimum located between A and B. Such a local optimum vanishes after the end-effector is near position d as shown by curve d-d. Thereafter, the manipulator does not have a tendency to get trapped at a local optimum. Since the starting configuration in Fig. 4.10 is between B and a local optimum, the



l---minimize the norm of the least-norm solution

d---minimize $1/\sqrt{\det(JJ^T)}$

m---minimize $1/\sigma_{min}$

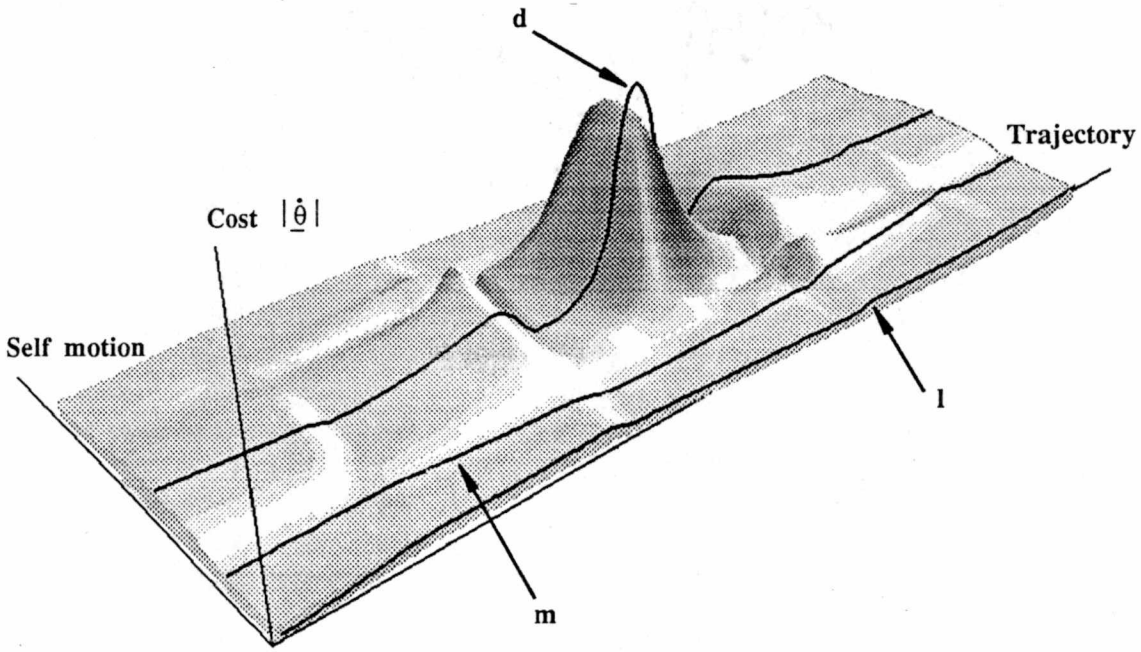


Figure 4.11

Converging loop for different performance criteria (2)

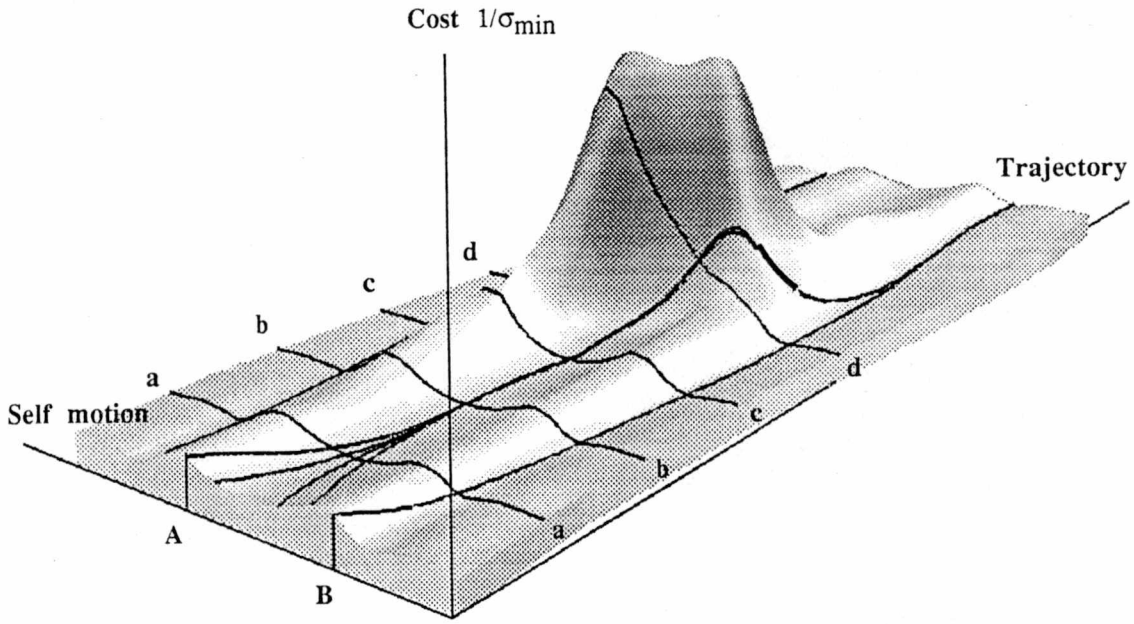


Figure 4.12

3–D cost map of $1/\sigma_{min}$

manipulator stays at this local optimum corresponding to a relatively high joint velocity norm cost until position d is reached. Due to the disappearance of a local optimum after position d, the manipulator moves toward a low cost region similar to the curve obtained by minimizing the norm of the least-norm solution.

Fig. 4.13 shows the 3-D map of $1/\sqrt{\det(JJ^T)}$ along with several cost curves on this map for the end-effector trajectory of Fig. 4.3. Curves in the direction of the self motion represent the variation of $1/\sqrt{\det(JJ^T)}$ as a function of self motion for several end-effector positions. From the shape of the curves along the self motion direction, it may be noted that a local optimum is present throughout the end-effector path, and not just in the first half as was the case in Fig. 4.12 where $1/\sigma_{min}$ was used as a performance criterion or cost function. Therefore, starting configurations between A and B result in trajectories following this optimum throughout the path as they get trapped in the valley in Fig. 4.13. In these cases, trajectories reach a limit in the valley, which corresponds to a high cost area. The high cost area in Fig. 4.13 also corresponds to high cost in terms of joint velocity norm in Fig. 4.10 and high cost in terms of $1/\sigma_{min}$ in Fig. 4.12. Thus minimization of $1/\sqrt{\det(JJ^T)}$ at the starting configuration between A and B in Fig. 4.10, does not avoid high joint velocities for the given trajectory, and it also does not move the manipulator away from singularity as indicated by the value of $1/\sigma_{min}$. From the 3-D map of $1/\sqrt{\det(JJ^T)}$ we can see that if the starting configuration is not between A and B, minimizing this criterion also leads to the configurations of low cost area in this map, which corresponds to the low cost area of the velocity norm map of Fig. 4.10.

From the above analysis we note that depending on the shape of the 3-D map of

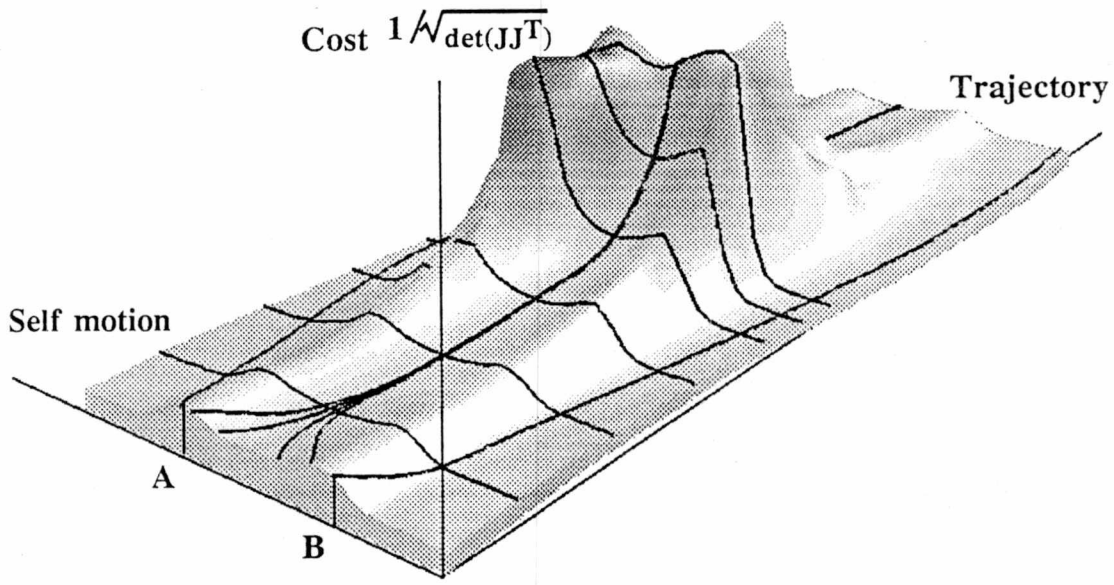


Figure 4.13

3–D cost map of $1/\sqrt{\det(JJ^T)}$

the performance criterion, a local optimization may either lead the joint trajectory to a low cost area, or it may get trapped in a high cost area. In practical applications, it may therefore be important to create and study the 3-D maps of the performance criteria being used in local optimization. These maps may be created for the parts of the workspace to be utilized for a given application. From the shapes of the 3-D maps, one may be able to compare different performance criteria and also determine the effectiveness of the local optimization of these criteria. It may be noted that local optimization is desired in teleoperation, and in many robotics applications in which the end-effector trajectory is modified based on sensory feedback.

CHAPTER 5

CONTROL PROGRAM USER INSTRUCTIONS

The control program was developed in C language on a Silicon Graphics Workstation 4D/50 GT. The user interface includes a 3-D graphics model of the RRC manipulator. Since the graphics model is complicated, the program can draw 30 pictures per second. A "Graphics Teach Pendant" interface was developed to allow the user to control the graphics model, as well as the RRC K-2107 manipulator. The "mouse" and the key board of the computer are used to change the configuration of the model and/or the manipulator and various available parameters. 3-D map plotting and SVD ellipsoid display on a "Small Graphics Window" are also available to evaluate the performance of the manipulator in real-time.

Users instructions for the functions available on the Graphics Teach Pendant and the Small Graphics Window are discussed in this chapter.

5.1 Functions Available on the Graphics Teach Pendant

5.1.1 Teaching the Robot

Joint motion: A user can use the mouse to manipulate the manipulator in the joint mode motion by placing the cursor on the corresponding "button" named J1, J2, ... J7, and pressing left or middle mouse button. Pressing of the left mouse button will decrease the angle of that joint revolution, and pressing the middle button will increase

it.

Linear motion: Placing the cursor on the X, Y or Z button and pressing the left or middle mouse button, a user can move the end-effector in X, Y or Z direction, respectively. Pressing the left mouse button will decrease the corresponding coordinate; and pressing the middle mouse button will increase it. The X, Y and Z can be referenced to the global or the hand coordinate system.

Roll, Pitch and Yaw motion: Placing the cursor on the Roll, Pitch or Yaw and pressing the left or the middle mouse button, a user can change the orientation of the end-effector. Pressing the left mouse button will decrease the corresponding value; and pressing the middle mouse button will increase it. If the base coordinate is used, the rotation will be in the direction of roll, pitch or yaw with respect to the base coordinates. If the hand coordinates are used, the rotation will be about X, Y or Z axis of the hand coordinate.

Self motion: Placing the cursor on the Orbit button and pressing left or middle mouse button, a user can allow the manipulator to change its configuration in two different and opposite directions without moving the end-effector.

Changing the Reference Coordinate: A user can switch the reference coordinates by putting the cursor on the "Coords_Switch" and pressing the left or middle mouse button. Left mouse button is used to choose the base coordinates; and the right mouse button is used to choose the hand coordinates system. The selected coordinate system is displayed on the up right corner of the graphic window.

Open or Close the Gripper: A user can have the gripper opened or closed by

placing the cursor on Gp button, and pressing the left or middle mouse button.

5.1.2 Recording the Trajectory Points

After moving the manipulator to the desired configuration, the user can record this configuration by placing the cursor on the REC-CT or REC-ST button then pressing left or middle mouse button. If no joint motion button or ORBIT button is used between two trajectory points, it will be recorded as a linear path control for the end-effector moving from the last point to the current point. If the joint motion button, ORBIT button or HOME button is used before recording, joint motion will be used to move the manipulator. If only ORBIT button is used in this period, the program will record it as self motion.

REC-ST: If a manipulator configuration is recorded using this option, the manipulator goes through this point and has to stop, then goes to the following trajectory point if there is a change of end-effector motion direction.

REC-CT: If the trajectory point to be recorded is of linear motion type, and if the following trajectory is of linear motion type, the recorded point will be a continuous point. A circular path control will be used to connect the linear path from first point to the second point and from the second point to the third point. In this case the manipulator does not necessarily go through this point, but is close to it. If the following path is not a linear path, the program will not use a circular path control. If the trajectory point to be recorded is not a linear motion type, the function of this button is the same as REC-ST.

5.1.3 Replaying and Editing the Recorded Trajectory

The trajectory can be replayed by choosing RUN or STEP options on the "teach pendant".

STEP: Placing the cursor on this button then pressing the left or middle mouse button, the program will allow the manipulator to move from the current configuration to the next trajectory point according to the move type recorded.

RUN: If the user uses this button instead of the STEP button, the program will move the manipulator from the present configuration through every following trajectory points until the end of the trajectory.

BEGIN: The replay of a trajectory can be commenced from the first recorded point by using this option and pressing the left or middle mouse button. STEP or RUN function preceded by this step will let the trajectory begin from the first trajectory point.

PAUSE: During the replay of a trajectory, the user can stop the replay by placing the cursor on PAUSE button then clicking left or middle mouse button.

DELETE: The trajectory points can be erased by the DELETE options. A STEP option preceding the DELETE function will delete that particular step. If the DELETE option is used more than one time, the following trajectory points will be erased too.

CLEAR: A user can clear all recorded trajectory points by placing the cursor on the CLEAR button and then clicking the left or middle mouse button.

5.1.4 Changing the View of the Graphics Model

VIEW: Clicking on the VIEW button will pop up 7 buttons in the graphic

window. These buttons allow the user to change the direction of viewing. The left and middle mouse button will change the view in opposite directions. The END VIEW button allows the user to exit the changing view option.

5.1.5 Changing the Moving Speed of the Manipulator

The moving speed of the manipulator can be changed by changing the absolute value of the increment of each step. This change can be done as follows:

L-V: Placing the cursor on the L-V button then pressing the left or middle mouse button, one can easily change the linear motion speed. The middle mouse button increases the linear motion speed; and the left mouse button decreases it. The corresponding value of the speed will be printed on the screen.

J-V: The operating procedure is the same as for the L-V button. It will increase/decrease the speed of joint motion, self motion and roll-pitch-yaw motion.

The L-V and J-V options changes the motion speed both in the teach and replay mode.

5.1.6 Options of the Program

OPTION: Clicking on this button, pops up a text window on the screen which has 10 options. The user can type in the number of the option to activate that particular function. The details of which are as follow:

(1) Change the Force Gains: When user chooses this option, the program will print out the gain factors currently used. Then, it will ask user to enter the new values

of the gain factors. The values kf[1] through kf[6] represent the gains of force and moment components along the X, Y and Z directions of the hand coordinate system.

(2) Select a Control Scheme: Typing '2' and then a return, the program will print out the available optimization. A user can chose one of them and then enter the gain factors used by the optimization. Normally a large gain will produce more self motion during optimization.

(3) Change the Turning Radius: This option allows the user to specify a radius for the circular path control.

(4) Change the Parameter for 3-D Plot: The user can choose this option to change the reference joint used by the 3-D plot and change the cost to be recorded as the norm of joint velocity vector or as the value of the performance criterion.

(5) Display End-Effector Path: This option allows the path of the end-effector trajectory to be plotted while the robot is moving.

(6) Allow Repeat of Trajectory: If the user answers affirmatively at this option and when the RUN button is activated, the manipulator will repeat the trajectory continuously from present trajectory point through the complete path. Only PAUSE button can stop it.

(7) Change the Color Defaults: This option allows a user to change the color of the back ground, buttons, data and characters displayed on the screen etc. The colors are represented in hex numbers (See the Silicon Graphics manual about the graphics library).

(8) Select a Manipulator: There are two graphic models available, the RRC K-2107 and the LTM. A user can select one of the two manipulators.

(9) Change Swap Graphics Time: The default speed for drawing the graphic model is 20 pictures per second. If this program is running on a different model of the Silicon Graphics computer, the maximum speed may be faster or slower than 20 pictures per second. The user can use this option to change the frames per second and have correct speed for the graphic model.

(10) Exit the 'OPTIONS': This option is used to exit the "OPTION" function.

5.1.7 Calculating the Data for the 3-D Map

3D-CAL: Before choosing this option, a user should let the graphic model reach the desired starting configuration. When the user places the cursor on this button and presses the left or middle mouse button, a small text window will pop up. Within the window, 3 parameters will have to be entered for generating the 3-D map. They are:

(1) The Resolution of the 3-D Map in the Self Motion Direction: This is represented by the increment of the reference joint of the self motion at the starting point of the end-effector.

(2) The Total Paths for the 3-D Map: For generating a 3-D map, the program uses the linear path control for the end-effector and least norm solution to move it. Along the trajectory, the program calculates and records the data of the performance criterion. After the path is completed, the program will change the starting configuration according to the resolution in the self motion direction. Then the program will calculate and record the performance criterion for the same end-effector trajectory and the whole sequence is repeated.

(3) The Performance Criterion to be Calculated: A user can choose the performance criterion to form the 3-D map.

Upon answering these questions, the program starts the calculations. It assumes that the end-effector goes from the present position and goes through the complete trajectory. This function does not support joint motion mode. A beep is sounded to inform the user the completion of the calculations. Sometimes the program may hang up if the reference joint can not reach the desired starting configuration by self motion. If this happens, the user can use the PAUSE button to interrupt the program, and then use OPTION to change the reference joint. The user may use self motion at the starting configuration to watch the display of the joint revolutions on the screen and then pick up the correct reference joint for the 3-D map.

5.1.8 Store or Retrieve the Files

The trajectory point and the 3-D map data can be stored onto and retrieved from the hard disk of the computer. These can be done by choosing STORE or GET functions.

GET: A user can retrieve a data file of the trajectory points or a 3-D map by putting the cursor on the GET then pressing the left or middle mouse button. A small text window will pop up on the screen. Within the window, one can choose to retrieve a trajectory data file or retrieve the 3-D map data file. After retrieving the trajectory data file or the 3-D data file, one can run the manipulator or plot the 3-D map by using these data sets.

STORE: The trajectory points data file, the 3-D map data file and the 3-D plot

file for single trajectory can be stored by this option. Before storing the files, one must generate the 3-D map data, the trajectory points data or the single path 3-D plot data. On placing the cursor on the STORE button and pressing the left or middle mouse button, a small text window will pop up and the user can choose the file type he wants to store it in, and type in the file name for the data file.

5.2 Available Functions of the Small Graphics Window

The small graphics window is used to display the 3-D map or the ellipsoid. The following options are available for the small graphics window.

(1) Popping or Hiding the Small Graphics Window: The user can pop up the small graphics window by placing the cursor in the upper left corner of the screen. For hiding the window, the user should place the cursor out of this window, then press the left and middle mouse buttons simultaneously.

(2) Change the Size of the Small Graphics Window: The size of the window can be changed by placing the cursor within the window, depressing and releasing the space bar of the key board, moving the cursor to the desired position for the window lower right corner and clicking the left or middle mouse button.

(3) Switching between the 3-D Map and the Ellipsoid: "F4" key will select to plot the 3-D map. "F5" key will plot the singular value decomposition ellipsoid for the linear motion components.

(4) Changing the Views of the Small Graphics Window: The following keys can change the view of the small graphics window:

R : Moves the graphics to the right.

L : Moves the graphics to the left.

N : Zooms in.

F : Zooms out.

U : Moves the graphics up.

D : Moves the graphics down.

I : Resetting the position of the graphics.

1, 2 or 3 : When these keys are used in conjunction with the left or middle mouse button, the graphic image can be rotated in various directions.

(5) Plotting Options for the 3-D Map: If there is a 3-D map in the small graphics window, the user can choose the 3-D map to be formed by the lines or a solid surface, or just plot the 3-D curves generated by the least-norm solutions. The above can be performed by placing the cursor inside the window, and pressing the left and middle mouse buttons.

(6) Highlighting the Specified Curves on a 3-D Map: The "left" and "right" arrow keys can highlight the curves generated by the least-norm solution on the 3-D map. The "up" and "down" arrow keys can highlight the curves that are formed by self motion. When both highlighted curves are on the 3-D map, three numbers are printed on the top of the window. The numbers represent the 3-D coordinates of the intersection point of the two curves. The first number is the step number of the end-effector's motion from the beginning to the current position. The second number is the reference joint

revolution. The third number is the value of the performance criterion.

(7) **Plotting Parts of a 3-D Map:** A part of a 3-D map can be plotted by pressing "P" key. Then the plotting part will be from the first "least-norm solution" curve to the highlighted "least-norm solution" curves; and from the beginning "step" curve to the highlighted "step" curve.

CHAPTER 6

CONCLUSIONS AND RECOMMENDATIONS

Conclusions

Real-time optimized control software for the 7-DOF RRC manipulator was developed and implemented. A simple and efficient numerical scheme was developed and used in the control program for local optimization of complex performance criteria in real-time. A new user-interface with 3-D graphics capability was developed. Several practical issues arising from the implementation of the pseudoinverse control and gradient-projection optimization of redundant-joint manipulators were discussed. A force/torque sensor and an ultrasonic sensor were integrated to the RRC manipulator.

The control program was written in "C" programming language for speed and portability. It can be easily modified and used for other 7-DOF manipulators by changing the manipulator parameters. The new user-interface allows the user to manipulate in real-time the RRC robot and its 3-D graphics model, by using a mouse. This capability can be easily extended to teleoperation with input from the hand controller instead of the mouse. The user interface has several important features such as ability to record or edit trajectory points, and display continually updated end-effector cartesian coordinates and joint coordinates, while the manipulator is in motion. On-line 3-D plotting capability and display of SVD ellipsoid allows the user to observe the performance of the manipulator in real-time. The control program is also able to read and use the data from the JR^3

force/torque sensor and an ultrasonic sensor integrated to the RRC manipulator for force control and obstacle avoidance, respectively.

The control program used a numerical method to optimize multiple complicated performance criteria in real-time for the RRC manipulator. Error correction was done at each step to maintain the desired trajectory. The circular path planning routine can maintain the end-effector speed while smooth transition of the end-effector motion between two consecutive linear segments takes place. The orientation of the end-effector between two end points is obtained such that rotation is minimized. A new self-motion function was developed to avoid algorithmic singularities inherent in the original controller.

Locally optimized joint trajectories were evaluated for the 7-DOF RRC manipulator following a closed path. Pseudoinverse control and gradient projection optimization of different performance criteria were used to generate the joint trajectories. The performance criteria used were the norm of the least-norm joint velocity, minimum singular value σ_{min}, and the manipulability measure $\sqrt{\det(JJ^T)}$. 3-D maps of the above performance criteria as functions of the end-effector position and self motion, were developed for a closed rectangular path. Effects of different gains and joint velocity limits in the gradient projection method implementation were discussed. The joint trajectories obtained from the minimization of the norm of the least-norm joint velocity, $1/\sigma_{min}$, and $1/\sqrt{\det(JJ^T)}$ were evaluated and compared. Behavior of the joint trajectories obtained from local optimization of the three performance criteria was analyzed with the help of 3-D maps. Trapping of an optimized trajectory at a local optimum was shown

to be due to the shape of the 3-D map of the corresponding criterion.

Recommendations

A few of the program functions could be improved for further portability by reducing the use of the global variables. For teleoperation, the program will require an interface with the master hand controller. To minimize the time delay during real-time robotic and telerobotic control, parallel communication or even bus to bus communication is recommended.

The present 3-D map of the norm of the least-norm assumes constant end-effector speed. For more generalized cases, the 3-D map of the inverse of MVR ($\underline{\dot{\theta}}^T_{min}\underline{\dot{\theta}}_{min}/\underline{\dot{x}}^T\underline{\dot{x}}$) could be used. In this case, the "cost" in this plot could be defined as $\underline{\dot{\theta}}^T\underline{\dot{\theta}}/\underline{\dot{x}}^T\underline{\dot{x}}$ for comparing different control schemes.

The basis of selection of performance criteria and their gain value 'k' in Eq. (2.3) for local optimization, requires further research. Using statistical techniques for the selection of 'k' within the work space, could provide useful results. Since local optimum could lead the manipulator to a high cost area, parallel trajectory calculations using different starting configurations is recommended. While one path is used for generating the joint trajectory of the manipulator, the other paths may be used to determine whether the manipulator is likely to be trapped at a local optimum. If the cost of the local optimum trajectory is higher than the cost for other trajectories, the trajectory needs to be modified to avoid trapping at a high cost local optimum. Use of the 3-D plot for comparison between local and global optimization techniques is recommended for future study.

LIST OF REFERENCES

REFERENCES

[1] S. M. Babcock, W. R. Hamel, and S. M. Killough, "Advanced Manipulation for autonomous mobile robots," in Proc. ANS Int. Topical Meeting Remote Syst. Robotics Hostile Environments, Mar. 1987, pp. 290-297.

[2] R. B. Magness, "Robotic control of the seven-degree-of-freedom NASA laboratory telerobotic manipulator," M.S. thesis, Univ. of Tennessee, Knoxville, Aug. 1989.

[3] Robotics Research Corporation "User Manual," of K-series robot arms type 1 robot controllers, Pub. No. UM-100-87, 1986.

[4] J. M. Hollerbach and K. C. Suh, "Redundancy Resolution of Manipulators through Torque Optimization," Proc. IEEE Int. Conf. Robotics and Automation, 1016-21, March 1985.

[5] C. A. Klein and K. B. Kee "The Nature of drift in Pseudoinverse Control of Kinematically Redundant Manipulators" IEEE Transactions on Robotics and Automation, Vol. 5, No. 2, April 1989.

[6] Nakamura, Y. and Hanafusa, H., "Optimal Redundancy Control of Robot Manipulators," The International Journal of Robotics Research., Vol. 6, No. 1, 1987, pp 32-42.

[7] Yoshikawa, T., "Dynamic Manipulability of Robot Manipulators," Proc. 1985 IEEE Int. Conf. on Robotics and Automation, 1033-1038.

[8] R. V. Dubey and J. Y. S. Luh (1987), "Redundant Robot Control for Higher Flexibility," Proceedings of the IEEE International Conference on Robotics and Automation, Raleigh, North Carolina, March, pp. 1066-1072.

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

[10] T. F. Chan and R. V. Dubey "Generation of Joint Trajectory Requiring Lower Joint Velocities for A Redundant Manipulator," Proc. of the 5th International Conference on Advanced Robotics, Pisa, Italy, June 1991.

[11] Albert, A., Regression and the Moore-Penrose Pseudo-inverse, Academic

Press, 1972.

[12] C. Klein and C. Huang, "Review of pseudoinverse control for use with kinematically redundant manipulators," IEEE Trans. Syst., Man, Cybern., vol. SMC-13, no. 3, pp. 245-250, Mar. 1983.

[13] A. Liegeois, "Automatic Supervisory Control of the Configuration and Behavior of Multibody Mechanisms," IEEE Trans. Systems, Man, Cybern., SMC-7, No. 12, 1977.

[14] J. A. Euler, and R. V. Dubey, and S. M. Babcock, "Self Motion Determination Based on Actuator Velocity Bounds for Redundant Manipulators", Journal of Robotics systems, 6(4), 417-425, August 1989.

[15] J. A. Euler and R. V. Dubey, "Self Motion Determination Based on Actuator Torque Bound for Redundant Manipulators," Proc. on Robotics Research, DSC-Vol. 14, edited by K. Youcef-Toumi and H. Kazerooni, ASME Winter Annual Meeting, San Francisco, California, December 10-15, pp. 221-226. 1989.

[16] B. Noble and J. W. Daniel, Applied Linear Algebra. Englewood Cliffs, NJ: Prentice-Hall, 1977.

[17] G. Strang, "Linear Algebra and Its Applications," Third Edition, Harcourt Brace Jovanovich, Inc., San Diego, 1988.

[18] Y. Nakamura and H. Hanafusa, "Inverse Kinematic Solutions With Singularity Robustness for Robot Manipulator Control," Journal of Dynamic Systems, Measurement, and Control, Vol. 108/163, Sep. 1986.

[19] A. A. Maciejewski and C. A. Klein, "The Singular Value Decomposition: Computation and Applications to Robotics," Int. J. Robotics Res., Vol. 8, No. 6, December 1989.

[20] R. V. Dubey, J. A. Euler, and Scott M. Babcock, "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.

[21] Robotics Research Corporation, "Host Interface User's Guide", REV. 890823, June, 1989.

[22] D. T. Greenwood, "Principles of Dynamics", Second Edition, 1988.

[23] C. Wampler, "Inverse kinematic functions for redundant manipulators," in 1987 IEEE Int. Conf. on Robotics and Automation, (Raleigh, NC Mar. 1987), pp. 610-617.

[24] H. Zghal, R. V. Dubey and J. A. Euler "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.

[25] J. A. Euler, R. V. Dubey, S. M. Babcock, and W. R. Hamel, "A Comparison of Two Real-Time Control Schemes for Redundant Manipulators with Bounded Joint Velocities," Proc. of the IEEE International Conference on Robotics and Automation, Scottsdale, Arizona, May 1989, pp. 106-112.

APPENDIXES

APPENDIX A

PROGRAM FUNCTION LIST

A.1 Functions in the Model File "main_pro.c"

The functions in this model may call the functions in the model file "goalp.c", "tools.c" and "init_graph.c".

main(): This is the main program. First it initiates the graphics window, the serial ports. Then it reads the cursor position on the screen and finds out which mouse button has been activated. It selects the corresponding function and responds accordingly. The teaching robot job is also done by this function.

talk_to_k2107_controller(): This function does the communication with the RRC K-2107 controller through the serial port 2. The user can issue commands by typing in the key board and read the robot controller feed back on the screen. This function also does the initiation work for real-time control.

talk_to_force_sensor(): This function is the communication function to send commands and read the force sensor controller feed back through the serial port 3.

talk_to_ult_sensor(): This function is the communication function to send the commands to the ultrasonic sensor controller and read the feed back through serial port 4.

A.2 Functions in the Model File "goalp.c"

The functions in this model file are the end-effector path planning functions and the 3-D map data calculation functions. They may call the functions from model file "tools.c" and "init_graph.c".

goal_lin(scheme_gain, dst): This function will allow the end-effector to move from the present position to the position specified by the array "dst". The linear path planning is used to control the end-effector motion. "scheme_gain" is the value used by the performance optimization. "dst[1]" to "dst[6]" contain the end-effector position components of X, Y, Z, Roll, Pitch and Yaw.

goal_rpy(scheme_gain, dst): This function will enable the end-effector to rotate from the present position to the position specified by the array "dst" when the major end-effector motion is indicated to be roll-pitch-yaw motion. "scheme_gain" is the value used by the performance optimization. "dst[1]" to "dst[6]" contain the end-effector position components of X, Y, Z, Roll, Pitch and Yaw.

goal_jnt(n, dst): This function lets the manipulator move from present configuration to the configuration defined by the array "dst". No path control is used in this function. n is the total joint number of joints of the manipulator. "dst[1]" to "dst[n]" contain the final joint revolutions.

goal_gp(theta_rd, gp): This is the gripper moving function. It allows the gripper to open or close depending on the value of "gp". "theta[1]" to "theta[7]" contain the joint revolution for the present configuration. "theta[0]" is used by the program to control the gripper.

goal_obt(n, dst): This function lets the manipulator move from the current configuration to the configuration defined by the array "dst" without moving the end-effector. "dst[1]" to "dst[n]" contain the desired configuration for the manipulator, where "n" happens to be total number of joints of the manipulator.

linear_cont(scheme_gain, r, x2, x3, stop): This is a "linear-continue" function. It lets the end-effector move from the present position to the position specified by the array "x2"; then it move the end-effector to the position specified by the array "x3". Two linear path planning and one circular path planning is used to control the end-effector motion. The array "x2" and "x3" contain the end-effector position X, Y, Z, Roll, Pitch and Yaw through the elements, "x2[1]" and "x3[1]", to the seventh element, "x2[6]" and "x3[6]". "r" is the radius for the circular path planning. "stop" indicates the continuity of the next step. The "scheme_gain" is similar to the one mentioned earlier.

circle(scheme_gain, r, coords, center, omega, turn): This is the circular path control function. "coords" is an array containing the present end-effector position X, Y and Z in "coords[1]", "coords[2]" and "coords[3]" separately. "center" is an array containing the circle center coordinates X, Y and Z at "center[1]", "center[2]" and "center[3]". "omega" is the total radian of the desired circular path. "turn" is the increment of the circular motion in radiance. "scheme_gain" and "r" are the same as indicated earlier.

k_inverse(draw, scheme_gain, theta_rd, vels, origin): This function calculates the kinematic inverse and the optimization for the given configuration. "scheme_gain" is the same as earlier. "theta_rd[1]" to "theta_rd[7]" are the present joint revolutions.

"vels[1]" to "vels[6]" contain the end-effector position increments ΔX, ΔY, ΔZ and the rotation increments about X, Y and Z axes. The array "origin" contains the end-effector desired position vectors which are used to correct the errors. "draw" indicates whether the manipulator configuration is changed or not.

grad_data(ref_joint, self_inc, total_path, r, s_count, goal_steps): This is the major function for generating the 3-D map. It calculates the 3-D map data from the present trajectory point to the end of the trajectory. "ref_joint" is the reference joint number used by the 3-D map to represent the self motion. "self_inc" indicates the change in the starting configuration according to the reference joint revolutions. "total_path" is the number of paths along the same end-effector trajectory at different starting configurations to generate 3-D map. "r" is the radius for the circular path planning. "s_count" indicates the present trajectory point. "goal_steps" is the total trajectory points for the complete trajectory.

self_motion(ref_joint, self_inc, goal_rd, origin, theta_rd): This function is used by "grad_data()" to change the starting configurations. "ref_joint" is the reference joint number used to indicate self motion. "self_inc" is the calculation increment of the reference joint. "goal_rd" is the final revolution of the reference joint. "origin[1]" to "origin[6]" contain the present end-effector position X, Y, Z, Roll, Pitch and Yaw which are used to correct the error. "theta[1]" to "theta[7]" contain the present joint revolution from joint 1 to joint 7. These joint revolutions will be updated once the function is called.

linear_cont_grad(ref_joint, path, r, x2, x3, stop): This function is similar to the

function "linear_cont()". It is used by "grad_data()" to calculate the 3-D map data along the trajectory from the present end-effector position to the point indicated by "x2", and then to the point indicated by "x3". One circular and two linear paths planning are used. "ref_joint" is the same as mentioned before. "r", "x2", "x3" and "stop" are the same as in the function "linear_cont()". "path" indicates the index of the calculation path along the trajectory.

grad_lin(ref_joint, path, dst): This function is similar to the function "goal_lin". It is used to calculate the 3-D map data along a linear path trajectory. "ref_joint" and "path" are the same as mentioned before. The array "dst" is the same as in the function "goal_lin()".

grad_circle(ref_joint, path, r, coords, center, omega, turn): This is the circular path planning function for generating 3-D map data. It is similar to the function "circle()". "ref_joint" and "path" are the same as mentioned before. "r", "coords", "center", "omega" and "turn" are the same as in the function "circle()".

get_grad_dat(ref_joint, path, steps, vels, origin): This function selects the functions for calculating the performance criterion of the 3-D map according to the "scheme_index" value. "ref_joint" and "path" are the same as mentioned before. "steps" is the movement counter of the manipulator from the starting configuration to present configuration. The array "vels" contains the end-effector velocity vector. The array "origin" is the desired end-effector position vector used to correct the error.

gcost_data(ref_joint, path, steps, vels, origin): This function calculate the norm of the least-norm solution as a performance criterion for the 3-D map. All calling

variables are the same as in the function "get_grad_dat()".

gdet_data(ref_joint, path, steps, vels, origin): This function calculate the inverse of the manipulability, $1/\sqrt{\det(JJ^T)}$, as a performance criterion for the 3-D map. All calling variables are the same as in the function "get_grad_dat()".

geig_data(ref_joint, path, steps, vels, origin): This function calculates the inverse of the minimum singular value of the Jacobian, $1/\sigma_{min}$, as a performance criterion for the 3-D map. All calling variables are the same as in the function "get_grad_dat()".

gdet6_data(ref_joint, path, steps, vels, origin): This function calculates the inverse of the minimum singular value of the Jacobian, $1/\sigma_{min}$, as a performance criterion for the 3-D map. This calculation does not perform singular value decomposition; therefore, the calculation speed is faster than the function "geig_dat()". However, the accuracy may be less than the result calculated by SVD. All calling variables are the same as in the function "get_grad_dat()".

get_jlim_data(ref_joint, path, steps, vels, origin): This function calculates the joint limit performance criterion for the 3-D map. All calling variables are the same as in the function "get_grad_dat()".

get_jlim_data2(ref_joint, path, steps, vels, origin): This function calculates the joint limit performance criterion in a different method for the 3-D map. All calling variables are the same as in the function "get_grad_dat()".

home_robot(): This function moves the manipulator back to its home position, and initiates the plotting of the end-effector path.

A.3 Functions in the Model File "init_graph.c"

This model file contains the function to initiate the graphic windows, print data in the window, 3-D map and ellipsoid plotting, end-effector trajectory plotting and change of views. The functions in this model may call the functions in the model "tools.c", "draw_k2107.c" and "draw_LTM.c".

def_light(): This function defines the lighting model used by the main graphic window to draw the robots[].

use_light(): This function uses the defined lights when drawing the robots.

init_rob_win(): This function initiates the main graphic window and prints the "teach pendant" and "buttons".

fm_print(): This function prints out the title of the manipulator being used.

view_butn(): This function displays the view buttons in the main graphic window.

change_view(): This function changes the view of the robot window.

c_view(view_point): This function displays the view buttons in the main graphic window and changes the array "view_point". "view_point[1]" to "view_point[3]" contain the translation of the object in X, Y and Z directions. "view_point[4]" to "view_point[6]" are the rotations in X, Y and Z directions.

c_view_cont(view_point): This function resets the hardware lighting model calculation. "view_point" is the same as in the function "c_view()".

def_light2(): This function defines the lights used by the small graphic window.

use_light2(): This function binds the defined lights to draw the objector in the

small graphic window.

init_curv_win(): This function initiates the small graphic window.

grad_plot(total_steps, total_paths): This is the major function for manipulating the small graphic window. "total_steps" is the counter for the changes of the manipulator from the starting configuration to the present configuration. "total_paths" is the number of the curves used to plot a 3-D map.

change_view2(view_point2): This function is similar to "change_view()". It changes the view of the small graphic window. The array structure of "view_point2" has the same structure of array "view_point".

drawaxis(): This function draws the X, Y and Z axes in the small graphic window.

draw_grad(total_steps, total_paths): This function uses solid lines to draw a 3-D map. "total_steps" is the total number of points in the trajectory direction on a 3-D map. "total_paths" is the number of the total curves used to generate a 3-D map.

grad_surf(part_plot, total_steps, total_paths, t_path1): This function uses a solid surface to draw a 3-D map. "part_plot" is the flag to indicate part or complete plotting of a 3-D map. "t_path1" has the value of the total curves plus one. This value is used to draw the side of the solid surface. "total_steps" and "total_paths" is the same as in the function "draw_grad()".

n_cal(total_steps, total_paths, t_path1): This function calculates the norms of the 3-D map surface for 3-D plotting. "total_steps" and "total_paths" is the same as in the function "grad_plot". "t_path1" is the same as in the function "grad_surf()".

least_curv(total_steps, total_paths): This function plots the curves that are used to generate a 3-D map. "total_steps" and "total_paths" are the same as in the function "draw_grad".

c_path(steps1, control_path): This function plots a 3-D curve. "steps1" is the number of the points for a 3-D curve. The two dimensional array "control_path" contains the 3-D curve data.

show_paths(total_steps, path): This function plots one of the curves which are used to generates a 3-D map. "total_steps" is the same as in "draw_grad()". "path" indicates the plotting curve.

show_steps(total_paths, step): This function plots the curve that is formed by the points which have the same position along the end-effector trajectory on a 3-D map. "total_paths" is the same as in "draw_grad()". "step" indicates the position of the curve on a 3-D map.

show_data(path, step): This function prints out the 3-D coordinates for the point on a 3-D map. This point is defined by the "path" and "step". These two variables are the same as in the functions "show_paths()" and "show_steps()".

draw_curv(): This function is used to plot a 3-D curve or the ellipsoid in real-time.

fourth_ellipse(): This function draws part of the ellipsoid.

ellipse(x_scaler, y_scaler, z_scaler): This function draws a ellipsoid. "x_scaler", "y_scaler" and "z_scaler" are scaler factors in X, Y and Z axes directions. If these three variable are the same, the function will plot a sphere.

draw_robot(theta_rd): This function prints the required data and plots the end-effector path on the screen. It also selects the robot to be drawn. "theta_rd[1]" to "theta_rd[7]" contain the joint revolutions. "theta_rd[0]" is the value to indicate the gripper status.

A.4 Functions in the Model File "tools.c"

This model file contains the mathematic tools, computer hardware initiation tools and the functions to transfer the parameters between the program and the two different robot graphic models. The functions in this file may call the functions in the model "draw_k2107.c" and "draw_LTM.c".

determ(n, d): This function calculates the determinant of a matrix. "n" is the dimension of the matrix. "d[1][1]" to "d[n][n]" are the elements of the matrix. The value of the determinant will be in "d[0][0]".

equit(n, d, x): This function solves n equations. "n" is the number of equations. "d[1][1]" to "d[n][n]" are the entries of a matrix. "d[1][0], d[2][0],...d[n][0]" contain the values of the right side of the equations. The solution will be stored in "x[1]" to "x[n]".

cal_jjt(m, n, jaco, ans): This function calculates a matrix multiple and its transpose. "m" and "n" are the rows and columns of the matrix. "jaco[1][1]" to "jaco[m][n]" contain the entries of the matrix. The resultant matrix will be in "ans[1][1]" to "ans[m][m]".

least_norm(theta_rd, dx, dtheta): This function calculates the least-norm solution for a desired end-effector velocity vector. "theta_rd[1]" to "theta_rd[7]" are the

joint revolutions. "dx[1]" to "dx[6]" are the end-effector difference components of translation ΔX, ΔY, ΔZ and rotation about X, Y and Z axes. The least-norm solution will be in "dtheta[1]" to "dtheta[7]".

o_motion(theta_rd, dtheta): This function calculates the homogeneous solution specified by the structure "obtctl". The array "theta_rd" is the same as in "least_norm()". "dtheta[1]" to "dtheta[7]" will contain the homogeneous solution of joint increments.

elbow_direction(theta_rd): This function initiates the "obtctl" structure of the RRC manipulator. It calculate the moving direction of the elbow for self-motion. This direction is defined by the variable "obtctl.sign". "theta_rd[1]" to "theta_rd[7]" are the present joint revolutions.

svd(n, a, eigenvalues): This function does the singular value decomposition of a matrix. "n" is the dimension of the matrix. "a[1][1]" to "a[n][n]" are the entries of the matrix. The singular values will be stored in "eigenvalues[1]" to "eigenvalues[n]".

new_theta(theta, dtheta): This function adds the values in "dtheta[1]" through "dtheta[7]" to the values "theta[1]" through "theta[7]".

rpy_angles(h_transf): This function calculates the roll, pitch and yaw in Z-Y-X rotation sequence for a rotation matrix stored at "h_transf[1][1]" to "h_transf[3][3]". The roll, pitch and yaw will be in "h_transf[0][1]", "h_transf[0][2]" and "h_transf[0][3]".

rec_position(index, state, steps, theta_rd, h_transf): This function records the present manipulator configuration or the end-effector position to the memory. "index" indicates the recording trajectory point status-continue or stop. "state" indicates the motion forward this trajectory point is cartesian motion, joint motion or the self motion.

"steps" is the counter of the trajectory points from the beginning of the trajectory up till the current trajectory point. "theta_rd" has the present joint revolutions. "h_transf" contains the homogeneous transformation matrix of the end-effector form "h_transf[1][1]" to "h_transf[4][4]".

get_position(origin, h_transf): This function passes the present end-effector position vector to the array "origin". The values in "origin[1]" to "origin[6]" will be the end-effector position X, Y, Z, Roll, Pitch and Yaw". "h_transf" is the same as in the function "rec_position()".

j_v_limit(dtheta_m, dtheta_h, gain, dtheta): This function checks the joint velocities according to the least-norm solution, "dtheta_m[1]" to "dtheta_m[7]", and the homogeneous solution, "dtheta_h[1]" to "dtheta_h[7]", with the gain factor "gain". If there is no joint velocity beyond its limit, "dtheta[1]" to "dtheta[7]" will contain the sum of them. Otherwise, the function will try to reduce the value of "gain" of the homogeneous solution to allow all the joint velocities in "dtheta[1]" to "dtheta[7]" are within their limits. If this fails, the array "dtheta" will be the same as the least-norm solution.

difference(theta_rd, h_transf, goal, diff): This function calculates the difference of the present end-effector position to its desired position. "theta_rd[1]" to "theta_rd[7]" are the present joint revolutions. "goal[1]" to "goal[6]" contain the desired end-effector position components X, Y, Z, Roll Pitch and Yaw. The difference of the end-effector from the present position to the desired position, ΔX, ΔY, ΔZ, $\Delta\theta_x$, $\Delta\theta_y$ and $\Delta\theta_z$, will be in "diff[1]" to "diff[6]". After calling the function, "h_transf" will contain the

homogeneous transformation matrix from "h_transf[1][1]" to "h_transf[4][4]".

minimum_rot(h_transf, h, diff): This function calculates the minimum rotation between two coordinates represented by two rotation matrices "h_transf" and "h". The rotation vector will be in the array "diff". This vector uses the first rotation matrix for the reference.

jacobian(theta, h_transf, jaco): This function calculates the homogeneous transformation and the Jacobian matrix. "theta[1]" to "theta[7]" are the present joint revolutions. The homogenous transformation will be in "h_transf[1][1]" to "h_transf[4][4]"; and the Jacobian matrix will be in "jaco[1][1]" to "jaco[6][7]".

transf(theta, h_transf): This function only calculates the homogeneous transformation. "theta[1]" to "theta[7]" are the present joint revolutions. The homogenous transformation will be in "h_transf[1][1]" to "h_transf[4][4]".

select_robot(robot_index): This function selects the graphic model to be drawn. "robot_index" indicates the robot to be selected.

save_parameters(robot_index): This function save the parameters of the present graphic model. "robot_index" is the same as in the function "select_robot()".

init_parameters(): This function initiates the parameters of the graphic models.

init_obt_struct(robot_index): This function initiates the structure "obtctl" which is used to control self motion. "robot_index" is the same as mentioned before.

get_j_lim(j_lim_h, theta_rd): This function calculates the performance criterion used to represent the joint limits. "theta_rd[1]" to "theta_rd[7]" are the present joint revolutions. The performance criterion will be in the "j_lim_h" [ref. Zghal theses].

get_j_lim2(j_lim_h, theta_rd): This function uses different method to calculate the performance criterion of the joint limits. [ref. Zghal theses]. All variable are the same as in the function "get_j_lim()".

robot_jog(): This function is used in real-time control of the RRC K-2107 manipulator. When the robot controller requires the motion command for the next loop, the function reads the feed back information, and send the present configuration of the graphic model to the robot controller.

send_command(ser_port, command): This function sends a character string to the serial port specified by "ser_port". The array "command" contains the characters to be sent.

feed_back(time_delay, buffer): This function reads the second serial port. "time_delay" indicates number of times the program tries to read the information if there is no information available. The array "buffer" will contain the readings if the function reads them from the port.

init_comm2(): This function initiates the second serial port of the computer. This port is used to connect the robot controller.

init_comm3(): This function initiates the third serial port of the computer. This port is used for the JR3 force sensor controller.

init_comm4(): This function initiates the fourth serial port of the computer. The port is used to communicate with the ultrasonic sensor controller.

ult_sensor(buffer): This function calculates the distance of ultrasonic sensor detector to the object, and corresponding self-motion of the manipulator. "buffer"

contains the value of time delay between sending and receiving signals of the ultrasonic waves.

force_sensor(force_buffer): The function calculates the forces acting on the force sensor, and corresponding self-motion. "force_buffer" contains the information read from force sensor.

poll_manager(control, read_index): This function checks the third and the fourth serial communication port. If any one of the port has the incoming data, the function will call the corresponding service function and give the new command to the serial port. "read_index" contains reading status.

pop_cursor(): This function sets the cursor to the position X=800, Y=250.

A.5 Functions in the Model File "draw_k2107.c"

The functions in this model file draw the k-2107 graphic model, store or pass the parameters and calculate the Jacobian and homogeneous transformation. The functions of this model do not call the functions of other model files.

drawlink0(): This function draws the base part of the K-2107 manipulator.

drawlink1(): This function draws the first link of the K-2107 manipulator.

drawlink2(): This function draws the second link of the K-2107 manipulator.

drawlink3(): This function draws the third link of the K-2107 manipulator.

drawlink4(): This function draws the fourth link of the K-2107 manipulator.

drawlink5(): This function draws the fifth link of the K-2107 manipulator.

drawlink6(): This function draws the sixth link of the K-2107 manipulator.

drawgripper(): This function draws the gripper of the k-2107 manipulator.

drawfig(): This function draws the moving part of the gripper.

draw_k2107_robot(theta): This function draws the complete RRC K-2107 robot. "theta[1]" to "theta[7]" are the joint revolutions of seven joint. "theta[0]" indicates the opening of the gripper.

init_k2107_parameters(): This function reads the default parameters of the RRC K-2107 graphics model to the memory.

get_k2107_parameters(): This function passes the present parameters of the RRC K-2107 graphic model into the main program.

save_k2107_parameters(): This function saves the present parameters of the RRC K-2107 graphic model into the memory.

init_k2107_obt_struct(): This function initiates the structure "obtctl" for the RRC K-2107 manipulator. This function should be used only after the robot is in the home position.

k2107_jacobian(theta, t, jc): This function calculates the homogeneous transformation and Jacobian matrix of the RRC K-2107 manipulator. "theta[1]" to "theta[7]" are the joint revolutions. "t[1][1]" to "t[4][4]" will be the homogeneous transformation matrix; and "jc[1][1]" to "jc[6][7]" will be the Jacobian matrix.

k2107_transf(theta, t): This function just calculates the homogeneous transformation of RRC K-2107 manipulator. "theta[1]" to "theta[7]" are the joint revolutions. "t[1][1]" to "t[4][4]" will be the homogeneous transformation matrix.

A.6 Functions in the Model File "draw_LTM.c"

The functions in this model file draw the LTM graphic model, store or pass the parameters and calculate the Jacobian and homogeneous transformation. The functions of this model do not call the functions of other model files.

cylinder(array): This function draws a cylinder. "array[0]", "array[1]" and "array[2]" are translations of the cylinder in X, Y and Z directions. "array[3]" and "array[4]" are rotations along X and new Y axes. "array[5]" has the radius of the cylinder. "array[6]" contains the length of the cylinder.

draw_LTM_axis(): This function draws the joint connection part of the LTM manipulator graphic model.

draw_LTM_stand(): This function draws the base of the LTM manipulator.

draw_LTM_part(): This function draws the part of the base, which is also used as the part of the upper arm of the LTM manipulator.

draw_LTM_uparm(): This function draws the upper arm of the LTM manipulator.

draw_LTM_lowarm(): This function draws the lower arm of the LTM manipulator.

draw_LTM_rst(): This function draws the wrist of the LTM manipulator.

draw_LTM_grip(): This function draws the gripper of the LTM manipulator.

draw_LTM_fg(): This function draws the moving part of the gripper.

draw_LTM_robot(theta): This function draws the complete LTM robot. "theta[1]" to "theta[7]" are the joint revolutions of seven joint. "theta[0]" indicates the

opening of the gripper.

init_LTM_parameters(): This function reads the default parameters of the LTM graphics model to memory.

get_LTM_parameters(): This function passes the present parameters of the LTM graphic model to the main program.

save_LTM_parameters(): This function saves the present parameters of the LTM graphic model to the memory.

init_LTM_obt_struct(): This function initiates the structure "obtctl" for the LTM manipulator. This function should be used only after the robot is in home position.

LTM_jacobian(theta, t, jc): This function calculates the homogeneous transformation and the Jacobian matrix of the LTM manipulator. "theta[1]" to "theta[7]" are the joint revolutions. "t[1][1]" to "t[4][4]" will be the homogeneous transformation matrix; and "jc[1][1]" to "jc[6][7]" will be Jacobian matrix.

LTM_transf(theta, t): This function only calculates the homogeneous transformation of LTM manipulator. "theta[1]" to "theta[7]" are the joint revolutions. "t[1][1]" to "t[4][4]" will be the homogeneous transformation matrix.

APPENDIX B

COMMUNICATION WITH CONTROLLER AND SENSORS

B.1 Communication with the RRC Robot Controller

RRC manipulator has the Serial Jog Mode which allows a host computer to control the robot arm in real time with a serial communications cable (RS232). The second serial port of the Silicon Graphics computer has been connected for robot control. The Serial Jog Mode is a synchronous mode of operation and communicates at regular intervals with the host computer[21]. On a preset cycle or loop time, the robot interrupts the host and passes to it the status and the robot configuration. The robot begins executing the incoming command at the beginning of the next cycle. The commands used by the program in this thesis are:

S30	Change the communication speed of the serial linking. This command is followed by the Baud Factor (Baud Factor = 9.216MHz/(baud*16)).
S70W	Set the loop time synchronous to the motion controller.
S72U	Character interrupt turned off.
S71U0	Position report not sent automatically.
S40	Followed by a U0 disables the leading zero suppression Followed by a UFF enables the leading zero suppression
m57	Ask for the current joint positions.

G20	Absolute joint jogging command. This command allows the manipulator to reach the desired joint configuration within the loop time.
m42	Reset the data buffers of the robot controller.
m95	Error clear.
m99	Shut down the robot.
S22	Change to the Serial Jog Mode.

To communicate with the robot controller, one can choose the RBT-CT option. Once the small text window pops up on the screen, the user can type the 'Ctl' key, and then type in the command. The feed back can be read from the window. To end this function, the user can type 'Alt' key, and then make a choice between controlling the robot in real-time or not. If control robot in real-time option is chosen, and the communication is established between the computer and the robot controller, the user can use the first and second level functions; the RRC K-2107 manipulator will follow the configuration of its graphics model.

During real-time control mode, or the robot jogging mode, only 'G20' command is used for sending the absolute joint position in each loop by the program. The details are available in the Robotics Research reference manual.

B.2 Communication with the Force Sensor

JR^3 Force-Moment Sensor System has the capability of measuring the forces and moments acting on it. This sensor, mounted on the end of the manipulator, communicates

with the third serial port of the Silicon Graphics computer for data transfer. The program uses the data sent from the sensor to keep a "zero force" at the end of the arm. This is implemented by changing the position and orientation of the end of the manipulator to move in the direction of the applied force and moment. Choosing the SEN2 option, the user can read the sensor feed back from a text window. To send a command, the user should type the "Ctl" key, and then type in the command. The "Alt" key exits this option and choice for using it is succeeded. If using the force sensor option is chosen, the program will send the "dp s" command to the sensor, and will read the feedback. To avoid unnecessary data transformation on serial communication, the program sends command "COM A EC OFF EL OFF" to shut down the echo generated by the force sensor controller.

B.3 Communication with the Ultrasonic Sensor

The ultrasonic sensor is fixed on the elbow for object detection. In calculating the distance between the object and the elbow, the M86HC11 single board computer is used to control the triggering of the ultrasonic signal and calculate the time delay in receiving an echo. The time delay is sent to the Silicon Graphics computer so that this data can be used to calculate the distance between the object and the elbow.

The program used to control the ultrasonic signal and calculate the time delay was developed in assembly language. The program was located at address B600. At every reset, a "g b600" command has to be used to activate the program. Once this program reads an "A" from the serial port, it enables the sensor operation. The time delay will be

sent back to the Silicon Graphics computer after receiving the echo. This program can be easily modified to control two or more ultrasonic sensors. The program code is listed as follow:

```
************ TRANSFER THE DATA FROM ULTRASONIC TO SERIAL ********

********** The constants used by this program **********************

PORTB     EQU   $1004          Port B segment address
PORTC     EQU   $1003          Port C segment address

********** The buffalo subroutines used by the program *************

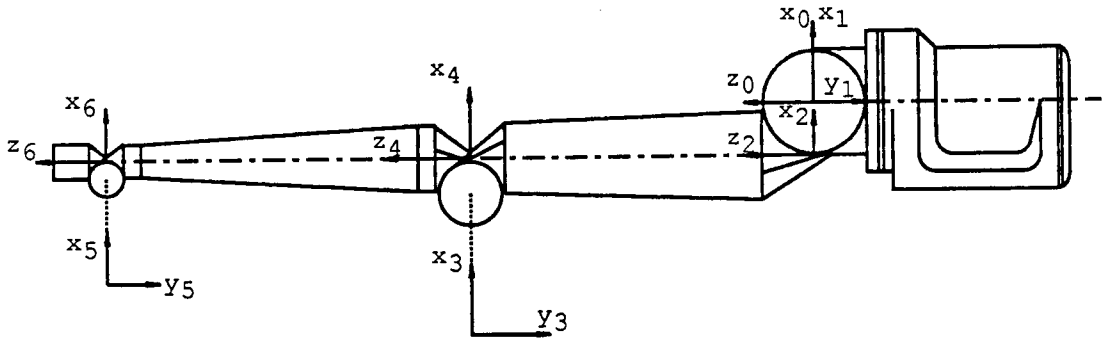
INPUT     EQU   $FFAC
OUT1BYT   EQU   $FFBB          Conv. bi. to 2 ASCII char. and output
OUTA      EQU   $FFB8          Output one ASCII cord

********************** Main program  ****************************
          ORG   $C000          Program located address
BEGIN     LDAA  #$00
          STAA  PORTB          Init. the VSW
READ      JSR   INPUT          Read the host command
          CMPA  #$41
          BNE   READ           If it is not a 'A',
*                              read another one.
          LDX   #$FFFF
          LDAA  #$80
          STAA  PORTB          Give the VSW signal
AGAIN1    LDAA  PORTC
          ANDA  #$40           Check the XLG signal;
          BEQ   AGAIN1         if not receive, goto AGAIN
AGAIN2    INX                  Increase the Time counter
          LDAA  PORTC
          ANDA  #$80           Check the FLG signal;
          BEQ   AGAIN2         if not received, increase time
*                              counter and wait until received.
          LDAA  #$00
          STAA  PORTB          Init. the VSW
          STX   BUFFER         Save timer to buffer
          LDX   #BUFFER        Get the address of buffer to X
          JSR   OUT1BYT        Output the time to host
          JSR   OUT1BYT
```

```
            LDAA    #$0D
            JSR     OUTA            Output LF
            BRA     READ
********************** BUFFER ************************************
BUFFER      FCB     $00, $00
```

APPENDIX C

DH TABLE OF RRC K-2107 MANIPULATOR



α_i	a_i	d_i	θ_i
-90	0	0	θ_1
90	-5.625	0	θ_2
-90	-4.25	37.985	θ_3
90	4.25	0	θ_4
-90	-1.937	37.996	θ_5
90	1.937	0	θ_6
0	0	10.619	θ_7

APPENDIX D

GRAPHICS MODEL OF RRC MANIPULATOR

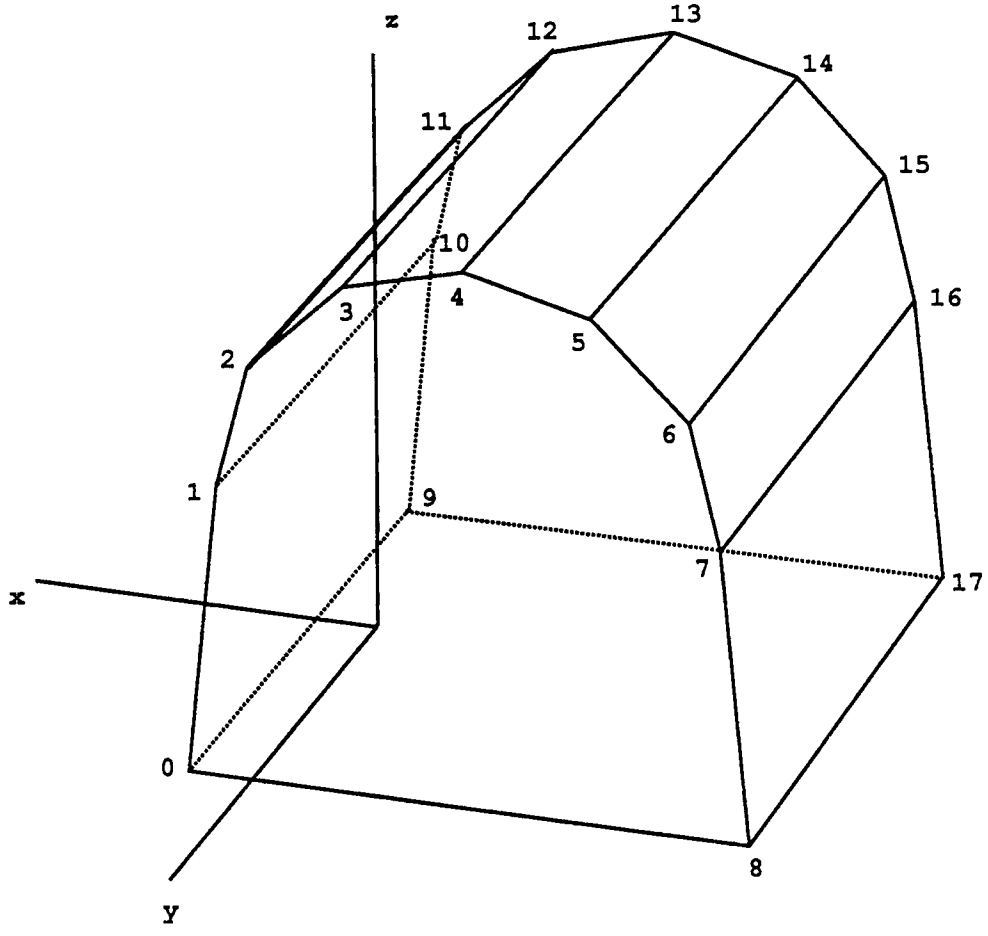


Figure D.1

Base part of RRC manipulator model

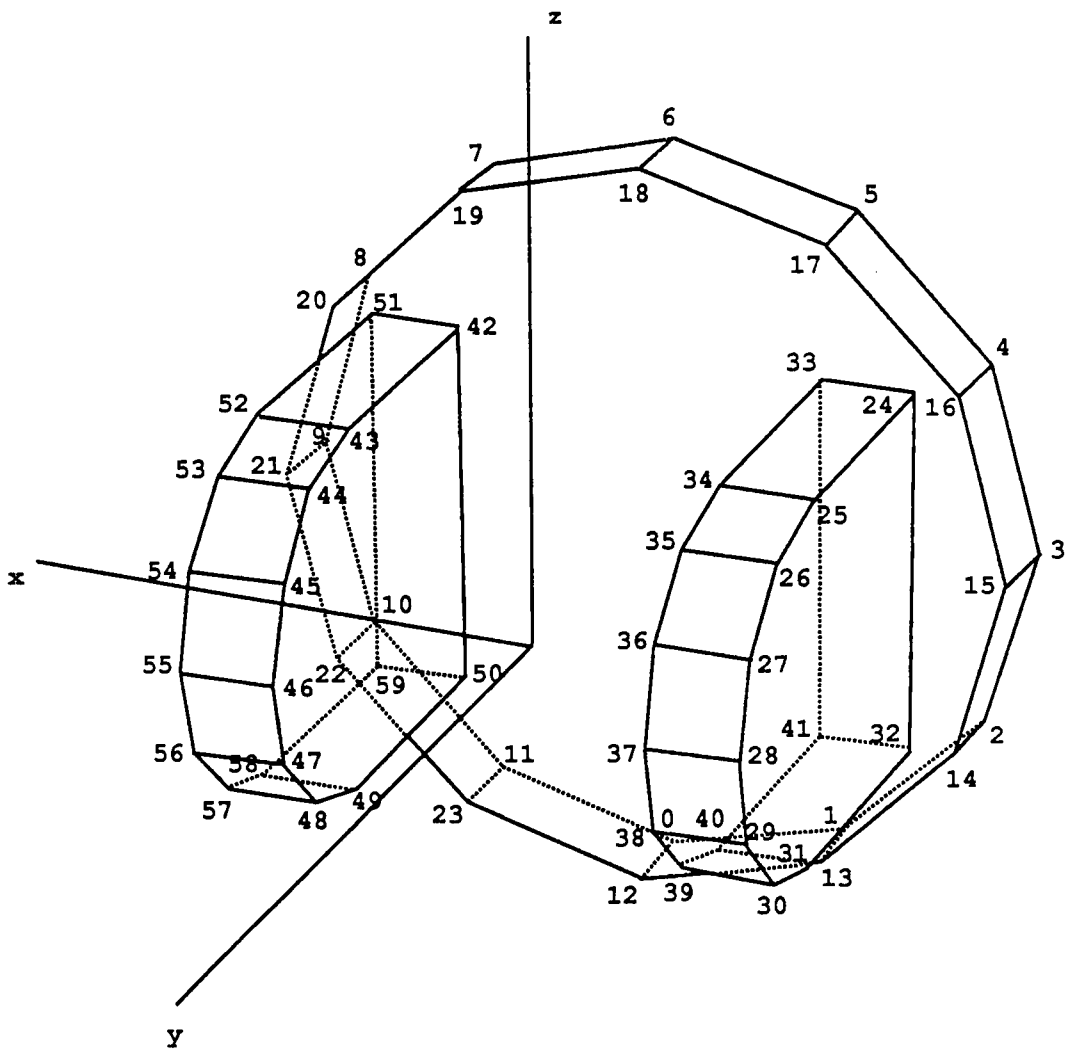


Figure D.2

First link of RRC manipulator model

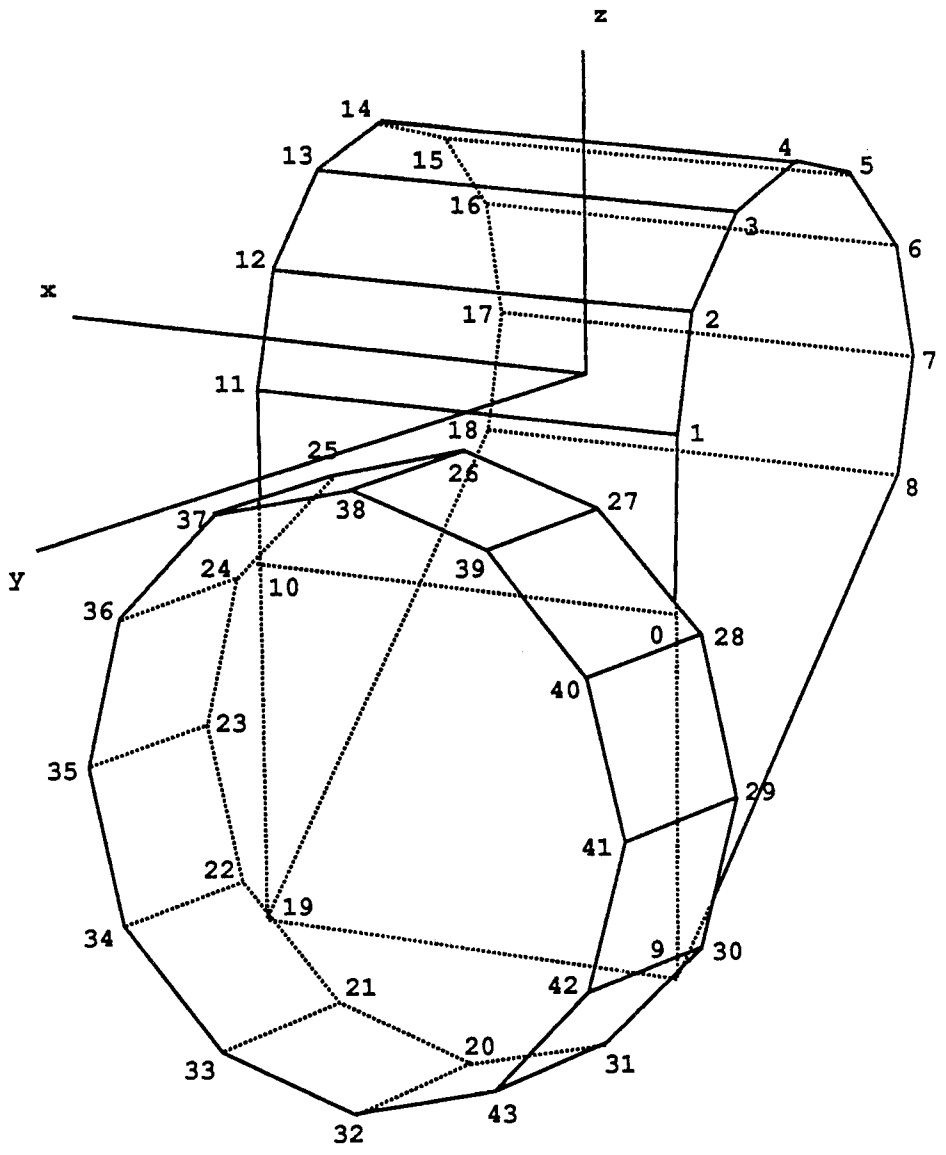


Figure D.3

Second link of RRC manipulator model

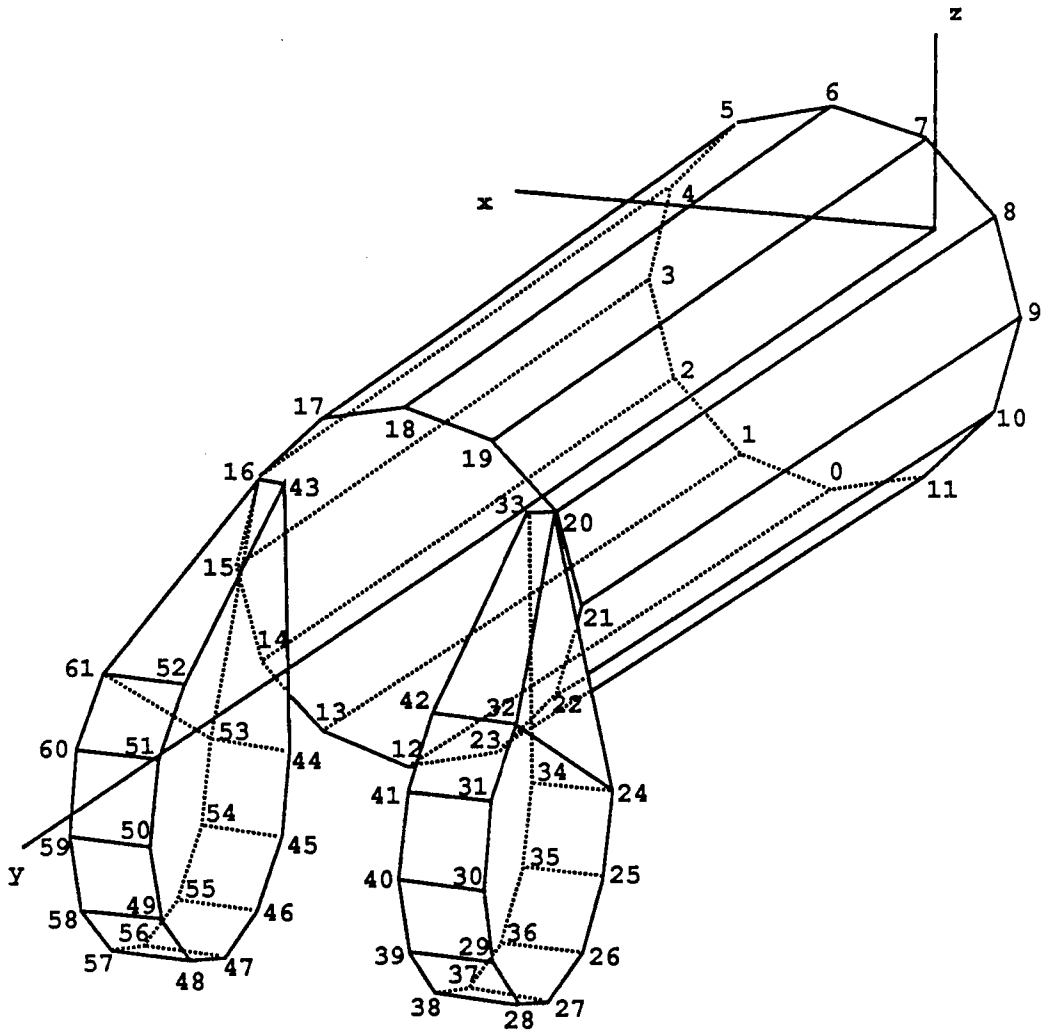


Figure D.4

Third link of RRC manipulator model

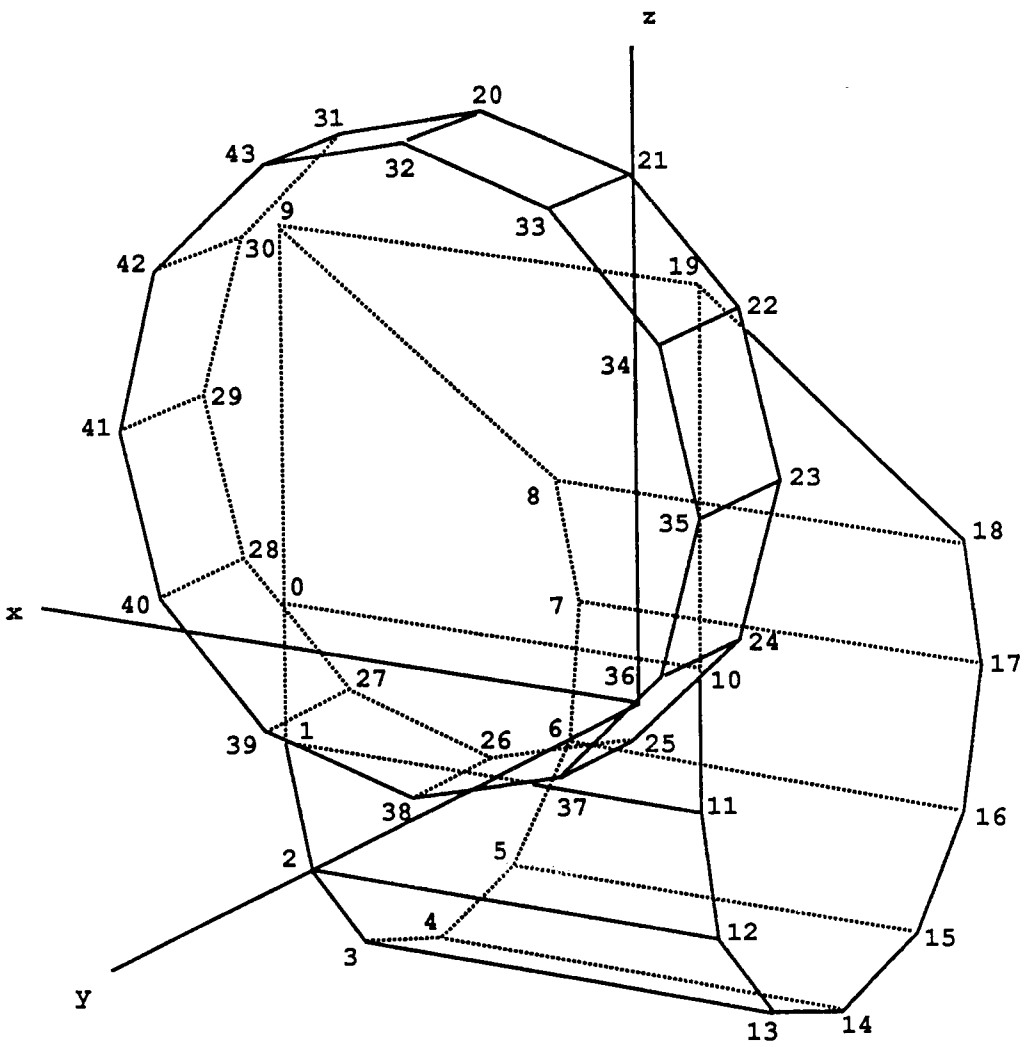


Figure D.5

Fourth link of RRC manipulator model

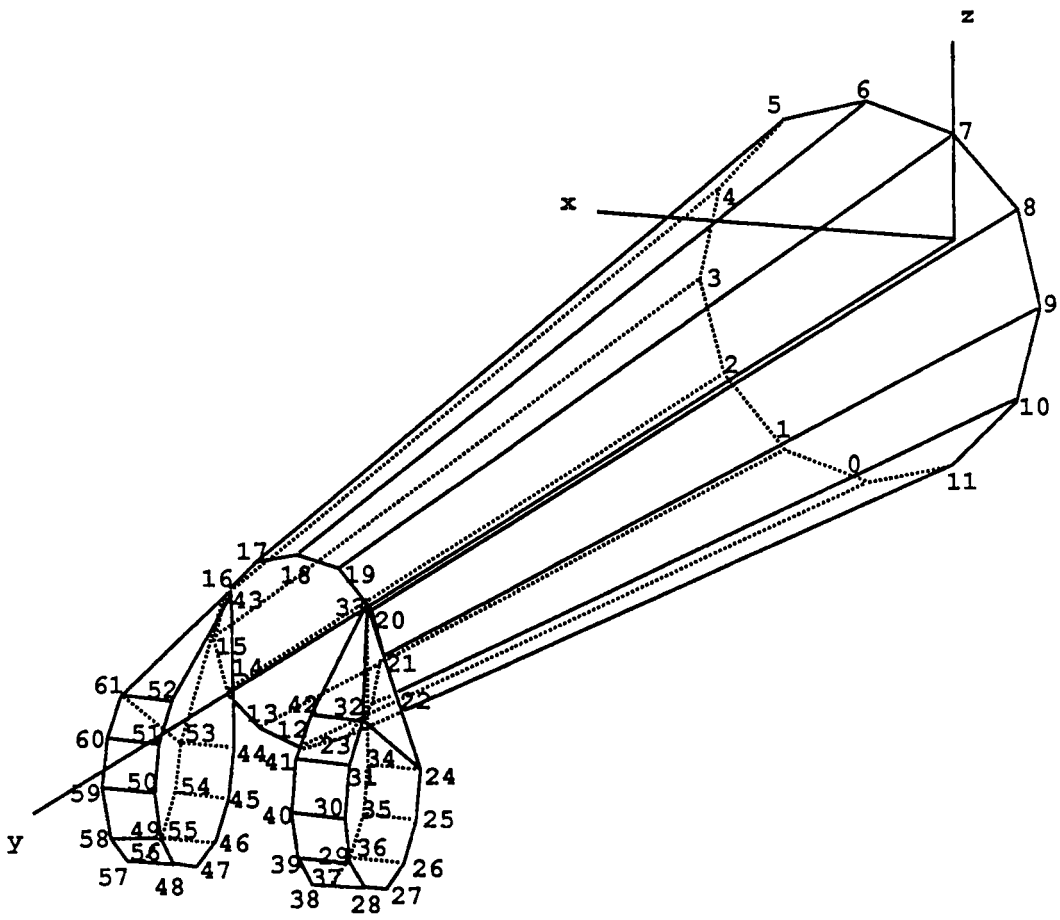


Figure D.6

Fifth link of RRC manipulator model

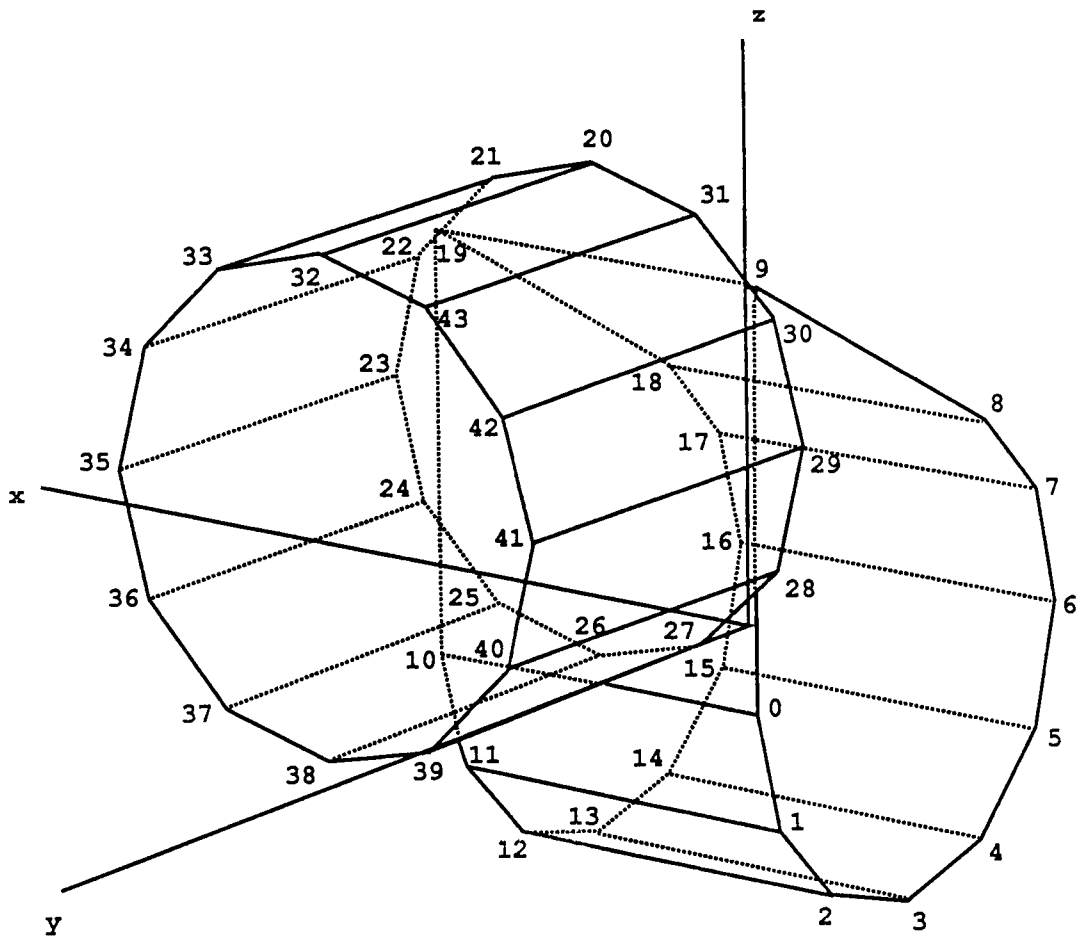


Figure D.7

Sixth link of RRC manipulator model

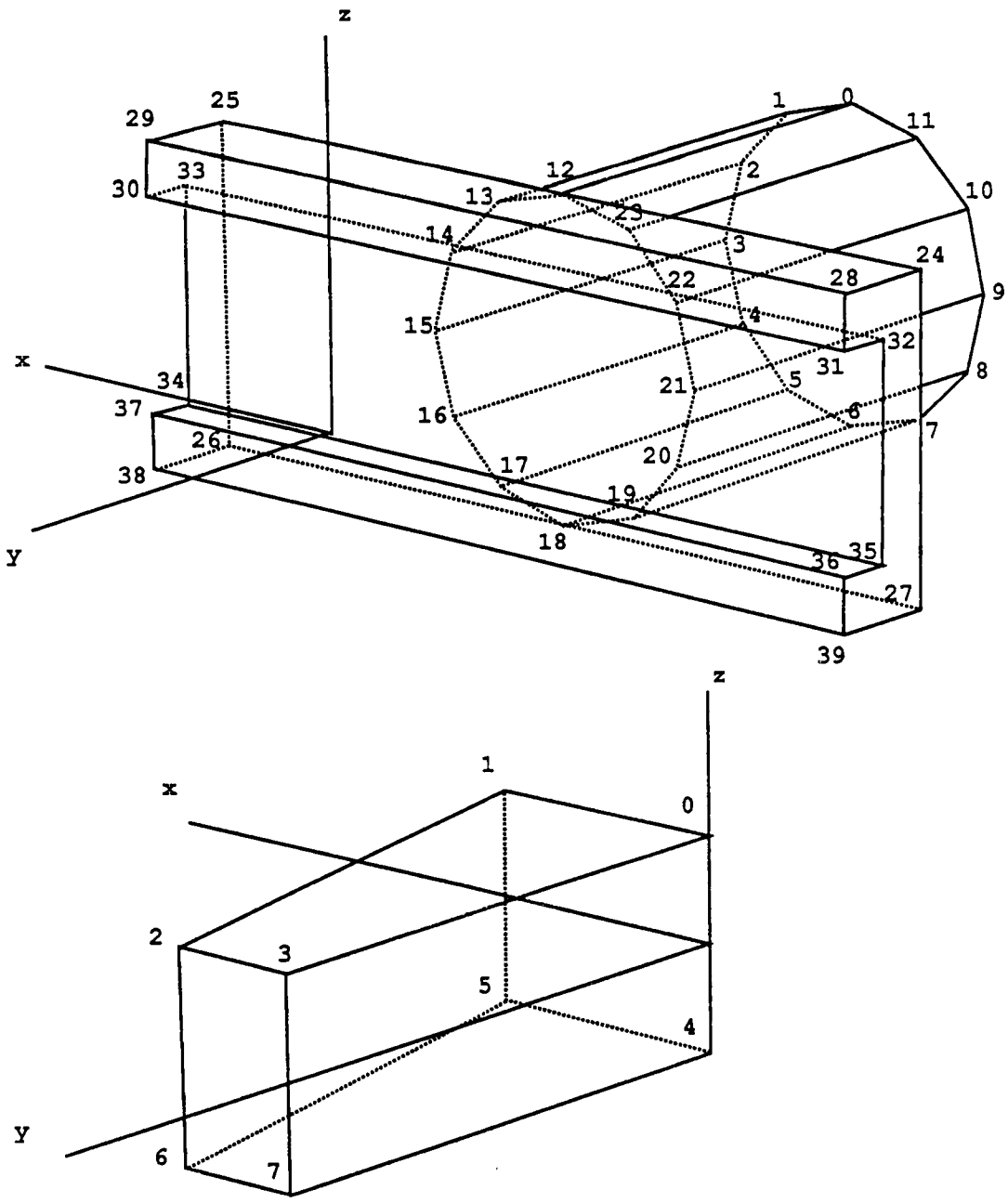


Figure D.8

End-effector of RRC manipulator model

APPENDIX E

PROGRAM SOURCE CODE

```
/*                    Source code file "main_pro.c"                    */

#include <math.h>
#include <stdio.h>
#include <gl/gl.h>
#include <gl/device.h>
#include <termio.h>
#include <fcntl.h>
#include <signal.h>
#include <poll.h>
#include <stropts.h>
#include <errno.h>
#include <string.h>

#define TPATHS 40                   /* Total paths for 3-D Map plotting   */
#define TSTEPS 10000                /* The max. steps of trajectory       */
#define MAT_DIM 10                  /* The dimension of arrays or matrices*/
#define T_JOINT 7                   /* Total joints of the manipulator    */


/*          Graphics and color variables:                                 */

int pic_delay = 3;                  /* Swap buffer delay controller       */
int total_steps=0;                  /* Total steps for a 3-D map plotting */
int total_paths=0;                  /* Total paths for a 3-D map plotting */
int steps1=0;                       /* The step counter for plotting curves*/
int win_state=0;                    /* Small gra[hics window indicator    */
                                    /*   0: The small window is not activated*/
                                    /*   1: The small window is activated */
int show_path=0;                    /* Switch for plot the end-effector path*/
                                    /*   0: Do not plot the end-effector path*/
                                    /*   1: Plot the end-effector path    */
int ellip=0;                        /* Switch of plotting ellipsoid or 3-D map*/
                                    /*   0: 3-D plotting                  */
                                    /*   1: Drawing MVR ellipsoid         */
long color_back=0x0;                /* The background color of robot window*/
long color_data=0x002fff;           /* The data color inside the robot window*/
long color_button=0x8f2f00;         /* The teach pendant button color     */
long color_title=0x0000ff;          /* The Title color                    */
long color_bt_char=0x00ffff;        /* The color of the button charactor  */
long color_back2=0x404040;          /* The the small window background color*/
long color_frame=0xaaaaaa;          /* The window fram color              */
long rob_win;                       /* The id of the robot window         */
long sub_win;                       /* The id of the small graphics window*/
long *font_big;                     /* The font used by title             */
long sub_winx;                      /* The uper border of sub. wind. in x */
long sub_winy;                      /* The uper border of sub. wind. in y */

float view_point[MAT_DIM];          /* The present view position          */
```

```
float init_view_point[MAT_DIM];          /* The initial view position                */
float view_mat[4][4];                    /* View matrix for changing the view point  */
float norm_dat[TSTEPS][TPATHS][3];       /* Data of 3-D map                          */
float norm_light[TSTEPS][TPATHS][3];     /* The norms for the 3-D map surface            */
float control_path[TSTEPS][3];           /* Data of one trajectory for 3-D plot      */
float trajectory[TSTEPS][3];             /* Data for drawing end-effector path       */
float eig_vect[4][4];                    /* Eigen vectors of MVR ellipsoid           */
float axis0[]={0.0, 0.0, 0.0},           /* data for drawing axes                    */
    x_axis[]={50.0, 0.0, 0.0},
    y_axis[]={0.0, 50.0, 0.0},
    z_axis[]={0.0, 0.0, 50.0};

/*              Robot hardware limit                                                */

float th_lim1[MAT_DIM];                  /* Joints limit lower boundary              */
float th_lim2[MAT_DIM];                  /* Joints limit higher boundary             */
float dth_lim[MAT_DIM];                  /* Joints limit rangs                       */
float theta_rd[MAT_DIM];                 /* Joint revolutions in radians             */
float jv_lim[MAT_DIM];                   /* Joint velocity limits    (Rad*5/sec)     */


/*              Control variables                                                   */

char force_buffer[200];                  /* Readings of the force sensor             */
char r_feed_back[200];                   /* Readings of the robot controller         */
int goal_steps=0;                        /* Trajectory points counter                */
int state=0;                             /* Motion type index                        */
int steps=0;                             /* Movement step counter                    */
int s_count = 0;                         /* Runing step counter                      */
int e_traj_count;                        /* End-effector trajectory plot counter     */
int ref_joint = 5;                       /* The reference joint of the 3-D plot      */
int scheme_index=0;                      /* The index of control scheme              */
int begin_step;                          /* The starting step when press 'RUN'       */
int ser_port2;                           /* The stream of serial port 2              */
int ser_port3;                           /* The stream of serial port 3              */
int ser_port4;                           /* The stream of serial port 4              */
int control_robot;                       /* The swich of real time control           */
int end_prog=0;                          /* The switch of endding the program        */
                                         /*   0:  Do not stop the main program       */
                                         /*   1:  Stop the main program              */
int repeat=0;                            /* The switch for repeat motion             */
                                         /*   0:  Do not repeat trajectory           */
                                         /*   1:  Repeat trajectory                  */
int local_coord = 0;                     /* The switch for changing the coordinates  */
                                         /*   0:  Use base coordinate                */
                                         /*   1:  Use hand coordinate                */
int ult_switch = 0;                      /* The switch for ult_sensor                */
                                         /*   0:  Do not use ultrasonic sensor       */
                                         /*   1:  Use ultrasonic sensor              */
int force_switch = 0;                    /* The switch for force_sensor              */
```

```
                                    /*   0:  Do not use force sensor          */
                                    /*   1:  Use force sensor                 */
int ult_sensor_fb = 0;              /* The state of ult_sensor feed back       */
int read_force = 0;                 /* The times of trying to read force data  */
int record_criterion = 1;           /* The type of recording cost of 3D plot   */
                                    /*   0:  Record the performance criterion */
                                    /*   1:  Record the joint velocity norm   */
int config_change = 1;              /* The state of the changing of config.    */
int changing_view = 0;              /* The state of the changing of view.      */
                                    /*   0:  The view is not changing         */
                                    /*   1:  The view is changing             */
int robot_index = 1;                /* The index for selecting the robot       */
                                    /*   1:  Use RRC K-2107 manipulator model */
                                    /*   2:  Use LTM manipulator model        */
int drawing;                        /* Indicator the drawing of the robot      */
                                    /*   0:  The robot has not been drawn     */
                                    /*   1:  The robot has been drawn         */
int teach_orbit = 0;                /* The previous motion type                */
                                    /*   0:  It is not a teach orbit motion   */
                                    /*   1:  It is a teach orbit motion       */
int interrupt;                      /* See if the program ended normally       */
                                    /*   0:  It is normal ending              */
                                    /*   1:  It is paused                     */
int joint_out[MAT_DIM];             /* The joint position of RRC manipulater   */

long mx;                            /* The position of mouse in x direction    */
long my;                            /* The position of mouse in y direction    */
long time_delay;                    /* The time delay used by communication    */

float self_inc;                     /* Increments of self motion at starting   */
                                    /* configuration with respect to ref. joint */
float linear_v=0.5;                 /* Linear velocity (10 in/sec)             */
float joint_v=0.0087266463;         /* Joint velocity (10 degree/sec)          */
float jaco[MAT_DIM][MAT_DIM];       /* Jacobian matrix                         */
float h_transf[5][5];               /* Homogeneous transformation              */
float dst[12];                      /* The end-effector desired position       */
float scheme_gain=0;                /* Gain of optimization using self motion  */
float r=3.0;                        /* Circular path radius                    */
float path_distance=0;              /* E-F moving distance along the trajectory */
float total_cost;                   /* The total joint vel. norm for one path  */
float obst_dis;                     /* Object distance from the elbow          */
float robot_home[MAT_DIM];          /* The home position of the robot          */
float dlinear_v = 0.005;            /* The change of the linear velocity       */
float djoint_v = 0.000087266463;    /* The change of the joints velocity       */
float force_data[MAT_DIM];          /* The buffer for store the force data     */
float force_dtheta[MAT_DIM];        /* The joints increas calculated by force  */
float ult_dtheta[MAT_DIM];          /* The joints increas calculated by ult_sensor */
float kf[] = { 0, 0.2, 0.2, 0.1, 0.00001, 0.00001, 0.00001, 0.0};
                                    /* The gain of the force sensor data       */

struct termio port;                 /* The structure used by control commu.    */
```

```
struct obt                              /* Self motion control structure          */
    {int index;                         /* The max. joint velocity joint number   */

    int sign;                           /* Self motion signal                     */
                                        /*   1 --- forward motion                 */
                                        /*   -1 --- back motion                   */

    float obt_v;                        /* The max. joint velocity for self motion */
    };
struct obt obtctl;

struct gt                               /* The structure for recording the        */
                                        /* trajectory data and motion type        */
    {char st_con[2];                    /* Motion type recorder                   */
                                        /* st_con[0]: motion type                 */
                                        /* st_con[1]: continue or stop signal     */
                                        /*   0: Do not use circular path planning */
                                        /*   1: Use circular path planning        */

    float goal_position[MAT_DIM];       /* Trajectory data of joint revolutions or */
                                        /* the end-effector position              */

    };
struct gt g_state[TSTEPS];

FILE *file_name;                        /* The stream of the file                 */

extern robot_jog();                     /* Robot real time control function       */

main()
{
    int i, j, k, l, tp, okay, read_index[4];
    char buffer[200], buffer1[200];
    float temp1, temp2, temp3;
    float dtheta[MAT_DIM], dtheta_h[MAT_DIM], origin[MAT_DIM], vels[MAT_DIM];


    init_comm2();           /* Initialize serial port #2    */
    init_comm3();           /* Initialize serial port #3    */
    init_comm4();           /* Initialize serial port #4    */
    poll_manager(1,read_index);

    /* Initialize Robotics Research Arm to be Drawn and Load Parameters */
    init_parameters();
    robot_index = 1;
    select_robot(robot_index);
    obtctl.obt_v = joint_v;

    /* Set up the plotting window in upper left corner */
    sub_winx=300;
    sub_winy=700;
    init_curv_win();
```

```
init_rob_win();          /* Initialize the Main Window */

setbell(1);
e_traj_count=(-1);

draw_robot(theta_rd);
get_position(origin, h_transf);
qdevice(MOUSE2);
qdevice(MOUSE3);
blanktime(0);

/* All the Main options for the teach pendant begin here */
while (end_prog==0)
    {
    mx=getvaluator(MOUSEX);
    my=getvaluator(MOUSEY);

    /* Option for global or hand coordinate useage. */
    if ((1030<=mx) && (mx<=1165) && (my>=980) && (my<=1000) &&
        (getbutton(MOUSE2) || getbutton(MOUSE3)))
        {
        if (getbutton(MOUSE2))
            local_coord = 1;
        else
            local_coord = 0;
        draw_robot(theta_rd);
        drawing = 1;
        }


    else if ((1020<=mx) && (mx<=1070))
        {
        if ((my>=900) && (my<=950) &&
            (getbutton(MOUSE2) || getbutton(MOUSE3)))
            {
            if (getbutton(MOUSE2))
                {
                theta_rd[1]+=joint_v;
                if (theta_rd[1]>th_lim2[1])
                    {
                    theta_rd[1]=th_lim2[1];
                    ringbell();
                    }
                }
            else
                {
                theta_rd[1]-=joint_v;
                if (theta_rd[1]<th_lim1[1])
                    {
                    theta_rd[1]=th_lim1[1];
```

```
                ringbell();
                }
            }
        draw_robot(theta_rd);
        drawing = 1;
        state=(state | 16);
        config_change = 1;
        teach_orbit = 0;
        }
    else if ((820<=my) && (my<=870) &&
        (getbutton(MOUSE2) || getbutton(MOUSE3)))
        {
        if (getbutton(MOUSE2))
            {
            theta_rd[2]+=joint_v;
            if (theta_rd[2]>th_lim2[2])
                {
                theta_rd[2]=th_lim2[2];
                ringbell();
                }
            }
        else
            {
            theta_rd[2]-=joint_v;
            if (theta_rd[2]<th_lim1[2])
                {
                theta_rd[2]=th_lim1[2];
                ringbell();
                }
            }
        state=(state | 16);
        draw_robot(theta_rd);
        drawing = 1;
        config_change = 1;
        teach_orbit = 0;
        }
    else if ((740<=my) && (my<=790) &&
        (getbutton(MOUSE2) || getbutton(MOUSE3)))
        {
        if (getbutton(MOUSE2))
            {
            theta_rd[3]+=joint_v;
            if (theta_rd[3]>th_lim2[3])
                {
                theta_rd[3]=th_lim2[3];
                ringbell();
                }
            }
        else
            {
            theta_rd[3]-=joint_v;
```

```
        if (theta_rd[3]<th_lim1[3])
            {
            theta_rd[3]=th_lim1[3];
            ringbell();
            }
        }
    state=(state | 16);
    draw_robot(theta_rd);
    drawing = 1;
    config_change = 1;
    teach_orbit = 0;
    }
else if ((660<=my) && (my<=710) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    if (getbutton(MOUSE2))
        {
        theta_rd[4]+=joint_v;
        if (theta_rd[4]>th_lim2[4])
            {
            theta_rd[4]=th_lim2[4];
            ringbell();
            }
        }
    else
        {
        theta_rd[4]-=joint_v;
        if (theta_rd[4]<th_lim1[4])
            {
            theta_rd[4]=th_lim1[4];
            ringbell();
            }
        }
    state=(state | 16);
    draw_robot(theta_rd);
    drawing = 1;
    config_change = 1;
    teach_orbit = 0;
    }
else if ((580<=my) && (my<=630) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    if (getbutton(MOUSE2))
        {
        theta_rd[5]+=joint_v;
        if (theta_rd[5]>th_lim2[5])
            {
            theta_rd[5]=th_lim2[5];
            ringbell();
            }
        }
```

```
    else
        {
        theta_rd[5]-=joint_v;
        if (theta_rd[5]<th_lim1[5])
            {
            theta_rd[5]=th_lim1[5];
            ringbell();
            }
        }
    state=(state | 16);
    draw_robot(theta_rd);
    drawing = 1;
    config_change = 1;
    teach_orbit = 0;
    }
else if ((500<=my) && (my<=550) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    if (getbutton(MOUSE2))
        {
        theta_rd[6]+=joint_v;
        if (theta_rd[6]>th_lim2[6])
            {
            theta_rd[6]=th_lim2[6];
            ringbell();
            }
        }
    else
        {
        theta_rd[6]-=joint_v;
        if (theta_rd[6]<th_lim1[6])
            {
            theta_rd[6]=th_lim1[6];
            ringbell();
            }
        }
    state=(state | 16);
    draw_robot(theta_rd);
    drawing = 1;
    config_change = 1;
    teach_orbit = 0;
    }
else if ((420<=my) && (my<=470) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    if (getbutton(MOUSE2))
        {
        theta_rd[7]+=joint_v;
        if (theta_rd[7]>th_lim2[7])
            theta_rd[7]=th_lim2[7];
        }
```

```
    else
        {
        theta_rd[7]-=joint_v;
        if (theta_rd[7]<th_lim1[7])
            theta_rd[7]=th_lim1[7];
        }
    state=(state | 16);
    draw_robot(theta_rd);
    drawing = 1;
    config_change = 1;
    teach_orbit = 0;
    }
else if ((340<=my) && (my<=390) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    if (getbutton(MOUSE2))
        {
        theta_rd[0]+=(joint_v*20);
        if (theta_rd[0]>th_lim2[0])
            theta_rd[0]=th_lim2[0];
        }
    else
        {
        theta_rd[0]-=(joint_v*20);
        if (theta_rd[0]<th_lim1[0])
            theta_rd[0]=th_lim1[0];
        }
    state=(state | 1);
    draw_robot(theta_rd);
    drawing = 1;
    config_change = 1;
    }
else if ((260<=my) && (my<=310) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    if (getbutton(MOUSE2))
        {
        joint_v+= djoint_v;
        }
    else if (getbutton(MOUSE3))
        {
        joint_v-= djoint_v;
        if (joint_v<djoint_v)
            joint_v= djoint_v;
        }
    cpack(0x00ffff);
    sprintf(buffer,"Joint velocity=%6.4f deg/sec",
        joint_v*3437.7468/pic_delay);
    cmov(0.001, -0.0033, -0.01);
    charstr(buffer);
    draw_robot(theta_rd);
```

```
    drawing = 1;
    }
else if ((180<=my) && (my<=230) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    control_robot == 0;
    sigignore(SIGPOLL);
    tpon();
    printf("Option for using ULTRA-SONIC SENSOR.\n");
    pop_cursor();

    if ( ser_port4 <= 0 )
        {
        ringbell();
        printf("\n Sensor is not available. \n");
        sleep(3);
        }
    else
        {
        talk_to_ult_sensor();
        }
    tpoff();
    draw_robot(theta_rd);
    }
 else if ((100<=my) && (my<=150) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    control_robot == 0;
    sigignore(SIGPOLL);
    system("clear");
    tpon();
    printf("Option for using FORCE SENSOR.\n");
    pop_cursor();

    if ( ser_port3 <= 0 )
        {
        ringbell();
        printf("\n Sensor is not available. \n");
        sleep(3);
        }
    else
        {
        talk_to_force_sensor();
        }
    tpoff();
    draw_robot(theta_rd);
    }
 else if ((20<=my) && (my<=70) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    total_cost = 0;
```

```
repeatloop:  if (s_count>=goal_steps)
                 s_count=begin_step;
             else if (goal_steps > 2)
                 {
                 s_count++;
                 begin_step=s_count;
                 }
             steps=e_traj_count=0;
             trajectory[0][0]=h_transf[1][4];
             trajectory[0][1]=h_transf[2][4];
             trajectory[0][2]=h_transf[3][4];
             path_distance=0;
             for (s_count; s_count<=goal_steps; s_count++)
                 {
                 if (g_state[s_count].st_con[0]==1)
                     {
                     interrupt = goal_gp(theta_rd,
                         g_state[s_count].goal_position[0]);
                     if (interrupt == 1)
                         break;
                     }
                 else if (g_state[s_count].st_con[0]>=10)
                     {
                     interrupt =goal_jnt(T_JOINT,
                         g_state[s_count].goal_position);
                     if (interrupt == 1)
                         break;
                     }
                 else if (g_state[s_count].st_con[0]>=8)
                     {
                     interrupt = goal_obt(T_JOINT,
                         g_state[s_count].goal_position);
                     if (interrupt ==1)
                         break;
                     }
                 else if (0!=(g_state[s_count].st_con[0] & 0x4))
                     {
                     i=s_count+1;
                     if ((g_state[s_count].st_con[1]==0) ||
                         (s_count==goal_steps)     ||
                         (g_state[i].st_con[0]>=8) ||
                         (g_state[i].st_con[0] <= 2) ||
                         (s_count == goal_steps))
                         {
                         jacobian(theta_rd, h_transf, jaco);
                         interrupt =goal_lin(scheme_gain,
                             g_state[s_count].goal_position);
                         if (1==interrupt)
                             break;
                         else if (win_state != 1)
                             {
```

```
                    difference(theta_rd, h_transf,
                            g_state[s_count].goal_position, vels);
                    temp1=sqrt(vels[1]*vels[1] + vels[2]*vels[2]
                            + vels[3]*vels[3]);
                    path_distance+=temp1;
                    k_inverse(1, scheme_gain, theta_rd, vels,
                            g_state[s_count].goal_position);
                    drawing = 1;
                    }
                }
            else
                {
                if ((g_state[i].st_con[1]==0) || (i==goal_steps)
                    || (g_state[i].st_con[0]>=8) || (g_state[i].st_con[0] <= 2))
                    j = 1;
                else
                    j = 0;
                interrupt =linear_cont(scheme_gain, r,
                        g_state[s_count].goal_position,
                        g_state[i].goal_position, j);
                if (1==interrupt)
                    break;
                }
            }
        else if (0!=(g_state[s_count].st_con[0] & 0x2))
            {
            interrupt=goal_rpy(scheme_gain,
                g_state[s_count].goal_position);
            if ( interrupt==1)
                break;
            }
        }
    if (interrupt==1)
        s_count--;
    if ((repeat==1) && (interrupt==0) && (goal_steps>=2))
        goto repeatloop;
    state=0;
    ringbell();
    draw_robot(theta_rd);
    teach_orbit = 0;
    }

}
else if ((1100<=mx) && (mx<=1170))
{
if ((my>=900) && (my<=950) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    get_position(origin, h_transf);
    if (getbutton(MOUSE2))
        temp1 = linear_v;
```

```
    else
        temp1 = -linear_v;
    if (local_coord == 0)
        {
        vels[1] = temp1;
        origin[1] += vels[1];
        vels[2]=vels[3]=vels[4]=vels[5]=vels[6]=0;
        }
    else
        {
        for (i=1; i<=3; i++)
            {
            vels[i] = temp1*h_transf[i][1];
            origin[i] += vels[i];
            }
        vels[4]=vels[5]=vels[6]=0;
        }
    k_inverse(1, scheme_gain, theta_rd, vels, origin);
    drawing = 1;
    state=(state | 4);
    config_change = 1;
    teach_orbit = 0;
    }
else if ((my>=820) && (my<=870) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    get_position(origin, h_transf);
    if (getbutton(MOUSE2))
        temp1 = linear_v;
    else
        temp1 = -linear_v;
    if (local_coord == 0)
        {
        vels[2] = temp1;
        origin[2]+=vels[2];
        vels[1]=vels[3]=vels[4]=vels[5]=vels[6]=0;
        }
    else
        {
        for (i=1; i<=3; i++)
            {
            vels[i] = temp1*h_transf[i][2];
            origin[i] += vels[i];
            }
        vels[4]=vels[5]=vels[6]=0;
        }
    k_inverse(1, scheme_gain, theta_rd, vels, origin);
    drawing = 1;
    state=(state | 4);
    config_change = 1;
    teach_orbit = 0;
```

```
    }
else if ((my>=740) && (my<=790) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    get_position(origin, h_transf);
    if (getbutton(MOUSE2))
        temp1 = linear_v;
    else
        temp1 = -linear_v;
    if (local_coord == 0)
        {
        vels[3] = temp1;
        origin[3]+=vels[3];
        vels[1]=vels[2]=vels[4]=vels[5]=vels[6]=0;
        }
    else
        {
        for (i=1; i<=3; i++)
            {
            vels[i] = temp1*h_transf[i][3];
            origin[i] += vels[i];
            }
        vels[4]=vels[5]=vels[6]=0;
        }
    k_inverse(1, scheme_gain, theta_rd, vels, origin);
    drawing = 1;
    state=(state | 4);
    config_change = 1;
    teach_orbit = 0;
    }
else if ((my>=660) && (my<=710) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    get_position(origin, h_transf);
    if (getbutton(MOUSE2))
        temp1 = joint_v;
    else
        temp1 = -joint_v;
    if (local_coord == 0)
        {
        origin[4] += temp1;
        if (origin[4]>3.1415926)
            origin[4] -= 6.2831852;
        else if (origin[4] < -3.1415926)
            origin[4] += 6.2831852;
        vels[6] = temp1;
        vels[1]=vels[2]=vels[3]=vels[4]=vels[5]=0;
        }
    else
        {
        for (i=1; i<=3; i++)
```

```
                {
                j = i + 3;
                vels[j] = h_transf[i][1]*temp1;
                }
            temp1 = cos(origin[4]);
            temp2 = sin(origin[4]);
            temp3 = cos(origin[5]);
            origin[5] += vels[5]*temp1 - vels[4]*temp2;
            if (fabs(temp3) > 1e-10)
                {
                temp3 = (vels[4]*temp1 + vels[5]*temp2)/temp3;
                origin[6] += temp3;
                origin[4] += (vels[6] + temp3*sin(origin[5]));
                }
            else
                {
                origin[6] += vels[4];
                }
            }
        vels[1]=vels[2]=vels[3]=0;
        k_inverse(1, scheme_gain, theta_rd, vels, origin);
        drawing = 1;
        state=(state | 2);
        config_change = 1;
        teach_orbit = 0;
        }
    else if ((my>=580) && (my<=630) &&
        (getbutton(MOUSE2) || getbutton(MOUSE3)))
        {
        get_position(origin, h_transf):
        if (getbutton(MOUSE2))
            temp1 = joint_v;
        else
            temp1 = -joint_v;
        if (local_coord == 0)
            {
            origin[5] += temp1;
            vels[4] = -temp1*sin(h_transf[0][1]);
            vels[5] = temp1*cos(h_transf[0][1]);
            }
        else
            {
            for (i=1; i<=3; i++)
                {
                j = i + 3;
                vels[j] = h_transf[i][2]*temp1;
                }
            temp1 = cos(origin[4]);
            temp2 = sin(origin[4]);
            temp3 = cos(origin[5]);
            origin[5] += vels[5]*temp1 - vels[4]*temp2;
```

```
            if (fabs(temp3) > 1e-10)
                {
                temp3 = (vels[4]*temp1 + vels[5]*temp2)/temp3;
                origin[6] += temp3;
                origin[4] += (vels[6] + temp3*sin(origin[5]));
                }
            else
                origin[6] += vels[4];
            }
        vels[1]=vels[2]=vels[3]=0;
        k_inverse(1, scheme_gain, theta_rd, vels, origin);
        drawing = 1;
        state=(state | 2);
        config_change = 1;
        teach_orbit = 0;
        }
    else if ((my>=500) && (my<=550) &&
        (getbutton(MOUSE2) || getbutton(MOUSE3)))
        {
        get_position(origin, h_transf);
        if (getbutton(MOUSE2))
            temp1 = joint_v;
        else
            temp1 = -joint_v;
        if (local_coord == 0)
            {
            origin[6]+=temp1;
            if (origin[6]>3.1415926)
                origin[6] -= 6.2831852;
            else if (origin[6] < -3.1415926)
                origin[6] += 6.2831852;
            temp2=temp1*cos(h_transf[0][2]);
            vels[5]=temp2*sin(h_transf[0][1]);
            vels[4]=temp2*cos(h_transf[0][1]);
            vels[6]=(-temp1)*sin(h_transf[0][2]);
            }
        else
            {
            for (i=1; i<=3; i++)
                {
                j = i + 3;
                vels[j] = h_transf[i][3]*temp1;
                }
            temp1 = cos(origin[4]);
            temp2 = sin(origin[4]);
            temp3 = cos(origin[5]);
            origin[5] += vels[5]*temp1 - vels[4]*temp2;
            if (fabs(temp3) > 1e-10)
                {
                temp3 = (vels[4]*temp1 + vels[5]*temp2)/temp3;
                origin[6] += temp3;
```

```
            origin[4] += (vels[6] + temp3*sin(origin[5]));
            }
        else
            origin[6] += vels[4];
        }
    vels[1]=vels[2]=vels[3]=0;
    k_inverse(1, scheme_gain, theta_rd, vels, origin);
    drawing = 1;
    state=(state | 2);
    config_change = 1;
    teach_orbit = 0;
    }
else if ((my>=420) && (my<=470) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    get_position(origin, h_transf);
    if (teach_orbit == 0)
        elbow_direction(theta_rd);
    if (obtctl.obt_v>0)
        obtctl.obt_v=joint_v;
    else
        obtctl.obt_v=(-joint_v);
    if (getbutton(MOUSE2))
        obtctl.sign = 1;
    else if (getbutton(MOUSE3))
        obtctl.sign= -1;
    o_motion(theta_rd, dtheta_h);
    new_theta(theta_rd, dtheta_h);
    difference(theta_rd, h_transf, origin, vels);
    least_norm(theta_rd, vels, dtheta, dtheta_h);
    new_theta(theta_rd, dtheta);
    e_traj_count--;
    draw_robot(theta_rd);
    state=(state | 8);
    drawing = 1;
    config_change = 1;
    teach_orbit = 1;
    }
else if ((my>=340) && (my<=390) &&
     (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    view_butn();
    draw_robot(theta_rd);
    changing_view = 1;
    change_view();
    changing_view = 0;
    }
else if ((my>=260) && (my<=310) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    if (getbutton(MOUSE2))
```

```
            {
            linear_v += dlinear_v;
            }
        else if (getbutton(MOUSE3))
            {
            linear_v -= dlinear_v;
            if (linear_v < dlinear_v)
                linear_v = dlinear_v;
            }
        cpack(0x00ffff);
        sprintf(buffer,"Linear velocity=%8.2f in/sec",linear_v*60/pic_delay);
        cmov(0.001, -0.0033, -0.01);
        charstr(buffer);
        draw_robot(theta_rd);
        drawing = 1;
        }
    else if ((my>=100) && (my<=150) &&
        (getbutton(MOUSE2) || getbutton(MOUSE3)))
        {
        state=16 | state;
        home_robot();
        config_change = 1;
        drawing = 1;
        teach_orbit = 0;
        }
    }
else if ((1200<=mx) && (mx<=1270))
    {
    if ((900<=my) && (my<=950) &&
        (getbutton(MOUSE2) || getbutton(MOUSE3)))
        {
        if (config_change == 1)
            {
            s_count++;
            for (i=goal_steps; i>=s_count; i--)
                {
                j=i+1;
                for (k=0; k<=9; k++)
                    g_state[j].goal_position[k] =
                        g_state[i].goal_position[k];

                g_state[j].st_con[0]=g_state[i].st_con[0];
                g_state[j].st_con[1]=g_state[i].st_con[1];
                }
            goal_steps++;
            rec_position(1, state, s_count, theta_rd, h_transf);
            state=0;
            ringbell();
            config_change = 0;
            }
        }
```

```
else if ((820<=my) && (my<=870) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    if (config_change == 1)
        {
        s_count++;
        for (i=goal_steps; i>=s_count; i--)
            {
            j=i+1;
            for (k=0; k<=9; k++)
                g_state[j].goal_position[k] = g_state[i].goal_position[k];
            g_state[j].st_con[0] = g_state[i].st_con[0];
            g_state[j].st_con[1] = g_state[i].st_con[1];
            }
        goal_steps++;
        rec_position(0, state, s_count, theta_rd, h_transf);
        state=0;
        ringbell();
        config_change = 0;
        }
    }
else if ((my>=740) && (my<=790) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    control_robot == 0;
    sigignore(SIGPOLL);
    tpon();
    system("clear");
    printf("\n[ 1 ] Get the path data: \n");
    printf("[ 2 ] Get the gradient data: \n");
    pop_cursor();
    tp=atoi(fgets(buffer, 10, stdin));
    printf("Enter the name of your data file: \n");
    gets(buffer);
    if (0==(file_name=fopen(buffer,"r")))
        {
        printf("Can't open or find that file!\n");
        ringbell();
        sleep(1);
        }
    else
        {
        if (tp==1)
            {
            i=j=0;
            while (j==0)
                {
                i++;
                fscanf(file_name,"%d",&k);
                g_state[i].st_con[0]=k;
                fscanf(file_name,"%d",&k);
```

```
                    g_state[i].st_con[1]=k;
                    for (j=0; j<=9; j++)
                        {
                        fscanf(file_name,"%f",&temp1);
                        g_state[i].goal_position[j]=temp1;
                        }
                    j=feof(file_name);
                    }
                goal_steps=i-1;
                }
            else if (tp==2)
                {
                fscanf(file_name,"%d",&total_paths);
                fscanf(file_name,"%f",&temp1);
                i=j=0;
                while (j==0)
                    {
                    i++;
                    for (k=0; k<=total_paths; k++)
                        {
                        norm_dat[i][k][0]=temp1;
                        fscanf(file_name,"%f",&temp1);
                        norm_dat[i][k][1]=temp1;
                        fscanf(file_name,"%f",&temp1);
                        norm_dat[i][k][2]=temp1;
                        fscanf(file_name,"%f",&temp1);
                        }
                    j=feof(file_name);
                    }
                total_steps=i;
                steps1=0;
                k=total_paths+1;
                n_cal(total_steps, total_paths, k);
                }
            fclose(file_name);
            setbell(2);
            ringbell();
            setbell(1);
            }
        tpoff();
        sleep(1);
        draw_robot(theta_rd);
        }

    else if ((660<=my) && (my<=710) &&
        (getbutton(MOUSE2) || getbutton(MOUSE3)))
        {
        control_robot == 0;
        sigignore(SIGPOLL);
        tpon();
        system("clear");
```

```
    printf("\n(1) Store the path data: \n");
    printf("(2) Store one trajectory data: \n");
    printf("(3) Store the gradient data: \n");
    pop_cursor();
    tp=atoi(fgets(buffer, 10, stdin));
    printf("Enter the name you want to store the data: \n");
    gets(buffer);
    if (0==(file_name=fopen(buffer,"w")))
        printf("Can't open the file!\n");
    else if (tp==1)
        for (i=1; i<=goal_steps; i++)
            {
            fprintf(file_name,"%d %d\n",g_state[i].st_con[0],
                    g_state[i].st_con[1]);

            fprintf(file_name, "%e %e %e %e %e %e %e %e %e %e\n",
                    g_state[i].goal_position[0], g_state[i].goal_position[1],
                    g_state[i].goal_position[2], g_state[i].goal_position[3],
                    g_state[i].goal_position[4], g_state[i].goal_position[5],
                    g_state[i].goal_position[6], g_state[i].goal_position[7],
                    g_state[i].goal_position[8], g_state[i].goal_position[9]);
            }
    else if (tp==2)
        for (i=1; i<=steps; i++)
            fprintf(file_name,"%e %e %e\n",control_path[i][0],
                    control_path[i][1], control_path[i][2]);
    else if (tp==3)
        {
        fprintf(file_name,"%d\n",total_paths);
        for (i=1; i<=total_steps; i++)
            for (j=0; j<=total_paths; j++)
                fprintf(file_name,"%e %e %e\n", norm_dat[i][j][0],
                        norm_dat[i][j][1], norm_dat[i][j][2]);
        }
    fclose(file_name);
    tpoff();
    sleep(1);
    draw_robot(theta_rd);
    }
else if ((580<=my) && (my<=630) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    goal_steps=0;
    s_count=0;
    state=0;
    setbell(2);
    ringbell();
    setbell(1);
    }
else if ((500<=my) && (my<=550) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
```

```
    {
    s_count=0;
    ringbell();
    draw_robot(theta_rd);
    drawing = 1;
    }
else if ((420<=my) && (my<=470) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    if (s_count>=goal_steps)
        {
        s_count=0;
        e_traj_count=0;
        trajectory[0][0]=h_transf[1][4];
        trajectory[0][1]=h_transf[2][4];
        trajectory[0][2]=h_transf[3][4];
        }
    if (goal_steps!=0)
        {
        s_count++;
        if (g_state[s_count].st_con[0]==1)
            {
            interrupt=goal_gp(theta_rd,
                g_state[s_count].goal_position[0]);
            }
        else if (g_state[s_count].st_con[0]>=10)
            {
            interrupt=goal_jnt(T_JOINT,
                g_state[s_count].goal_position);
            }
        else if (g_state[s_count].st_con[0]>=8)
            {
            interrupt=goal_obt(T_JOINT,
                g_state[s_count].goal_position);
            }
        else if (0!=(g_state[s_count].st_con[0] & 0x4))
            {
            interrupt=goal_lin(scheme_gain,
                g_state[s_count].goal_position);
            if (interrupt == 0)
                {
                difference(theta_rd, h_transf,
                        g_state[s_count].goal_position, vels);
                least_norm(theta_rd, vels, dtheta, dtheta_h);
                new_theta(theta_rd, dtheta);
                draw_robot(theta_rd);
                drawing = 1;
                }
            else
                s_count--;
            }
```

```
            else if (0!=(g_state[s_count].st_con[0] & 0x2))
                {
                interrupt=goal_rpy(scheme_gain,
                    g_state[s_count].goal_position);
                }
            if ((s_count==goal_steps) && (state!=1))
                {
                cpack(0x009f05);
                cmov(0.001, 0.0037, -0.01);
                charstr("The last step executed");
                draw_robot(theta_rd);
                drawing = 1;
                }
            state=0;
            }
        teach_orbit = 0;
        }
    else if ((340<=my) && (my<=390) &&
        (getbutton(MOUSE2) || getbutton(MOUSE3)))
        {
        for (i=s_count; i<goal_steps; i++)
            {
            j=i+1;
            for (k=0; k<=9; k++)
                g_state[i].goal_position[k]=g_state[j].goal_position[k];
            g_state[i].st_con[0]=g_state[j].st_con[0];
            g_state[i].st_con[1]=g_state[j].st_con[1];
            }
        s_count--;
        if (s_count <=0 )
            s_count = 0;
        goal_steps--;
        if (goal_steps <= 0)
            goal_steps = 0;
        ringbell();
        sleep(1);
        }
/* Option button Section */
    else if ((260<=my) && (my<=310) &&
        (getbutton(MOUSE2) || getbutton(MOUSE3)))
        {
        control_robot == 0;
        sigignore(SIGPOLL);
        /* Main Menu of Options */
        okay=0;
        while ( okay==0)
            {
            system("clear");
            tpon();
            printf("AVAILABLE OPTIONS: \n\n");
            printf("[ 1 ] Change the force gains\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 ] Exit the 'OPTIONS'.\n");
    printf("\n YOUR SELECTION: ");
    pop_cursor();
    fgets(buffer, 10, stdin);    /* Get user input */
    i=atoi(buffer);

switch( i )
{
case 1:

    okay=0;
    system("clear");
    printf("The current gain factors are:\n");
    for (i=1; i<=6; i++)
        printf("kf(%d)=%8e\n",i,kf[i]);
    printf("Enter the new gain factors : \n");
    for (i=1; i<=6; i++)
        {
        printf("kf(%d)= ",i);
        fgets(buffer, 20, stdin);
        kf[i]=atof(buffer);
        }
    break;

case 2:

    okay=0;
    system("clear");
    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("All other keys give least norm & no optimization \n\n");
    printf(" Your choice: ");
    scheme_index = atoi(fgets(buffer, 20, stdin));
    if ((scheme_index < 1) || (scheme_index > 6))
        scheme_index = 0;
    else
        {
        printf("\n\n Gain factor: ");
```

```
            fgets(buffer, 20, stdin);
            scheme_gain = atof(buffer);
            }
        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;
        break;

case 4:

        okay=0;
        system("clear");
        printf("[ 1 ] Change the reference joint.\n");
        printf("[ 2 ] Record the joint velocity norm\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;
            }
        break;

case 5:

        okay=0;
        system("clear");
        printf("Show end-effector path (y/n) ? ");
        gets(buffer);
        if ((buffer[0]=='y') || (buffer[0]=='Y'))
            {
            e_traj_count= -1;
```

```
            show_path=1;
            }
        else
            show_path = 0;
        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;
        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,buffer[i], 16);
        printf("Sub_window background color=%x\n", color_back2);
        gets(buffer);
        i = strlen(buffer);
        if (i > 0)
            color_back2=strtol(buffer,buffer[i], 16);
        printf("output data color=%x\n", color_data);
        gets(buffer);
        i = strlen(buffer);
        if (i > 0)
            color_data=strtol(buffer,buffer[i], 16);
        printf("teach pendant button color=%x\n", color_button);
        gets(buffer);
        i = strlen(buffer);
        if (i > 0)
            color_button=strtol(buffer,buffer[i], 16);
        printf("title color=%x\n", color_title);
        gets(buffer);
        i = strlen(buffer);
        if (i > 0)
            color_title=strtol(buffer,buffer[i], 16);
```

```
    printf("color_bt_char=%x\n", color_bt_char);
    gets(buffer);
    i = strlen(buffer);
    if (i > 0)
        color_bt_char=strtol(buffer,buffer[i], 16);
    printf("frame color=%x\n", color_frame);
    gets(buffer);
    i = strlen(buffer);
    if (i > 0)
        color_frame=strtol(buffer,buffer[i], 16);
    fullscrn();
    init_rob_win();
    draw_robot(theta_rd);
    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<=2))
        {
        save_parameters(robot_index);
        robot_index = i;
        select_robot(robot_index);
        c_view_cont(view_point);
        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)
            {
            z_axis[2] = y_axis[1] = x_axis[0] = 50;
            }
        else if (robot_index == 2)
            z_axis[2] = y_axis[1] = x_axis[0] = 25;
        }
    fm_print();
    draw_robot(theta_rd);
    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 *= temp2;
                djoint_v *= temp2;
                dlinear_v *= temp2;
                pic_delay = j;
                swapinterval(pic_delay);
                }
            }
        break;

    case 10:
        okay = 1;
    }
        }

/* Finishing Options */
    tpoff();
    ringbell();
    sleep(1);
    draw_robot(theta_rd);
    draw_robot(theta_rd);
    }

else if ((180<=my) && (my<=230) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    control_robot == 0;
    sigignore(SIGPOLL);
    tpon();
    system("clear");
    printf("This section is for computational options regarding\n");
    printf("the cost map.\n\n\n");
    printf("Degrees (angle) of self-motion between paths:");
    pop_cursor();
    self_inc=0.017453293*atof(fgets(buffer, 20, stdin));
    printf("Total number of path:");
    total_paths=atoi(fgets(buffer, 20, stdin));
    printf("\n\nGenerating the 3-D map of:\n");
    printf("(1) Norm of the least-norm solution.\n");
    printf("(2) Inverse of the det(J*J^T) gradiance.\n");
    printf("(3) Inverse of the min. singular value of J.\n");
    scheme_index = atoi(fgets(buffer, 20, stdin));
    tpoff();
```

```
        control_robot=0;
        sleep(1);
        draw_robot(theta_rd);
        if (s_count>=goal_steps)
            s_count=0;
        s_count++;
        steps=0;
        i=0;
        theta_rd[0]=g_state[s_count].goal_position[0];
        if (-1!=grad_data(ref_joint,self_inc,total_paths,r,s_count,goal_steps))
            {
            total_steps=steps-1;
            i=total_paths+1;
            n_cal(total_steps, total_paths, i);
            steps=0;
            state=0;
            s_count--;
            ringbell();
            }
        teach_orbit = 0;
        }
    else if ((my>=100) && (my<=150) &&
        (getbutton(MOUSE2) || getbutton(MOUSE3))
        && (robot_index == 1))
        {
        system("clear");
        control_robot == 0;
        sigignore(SIGPOLL);
        tpon();
        printf("This option is for REAL TIME control of the k2107.\n");
        if ( ser_port2 <= 0 )
            {
            ringbell();
            printf("\n\nOption not enabled with selected robot.\n");
            printf("Returning to teach pendant.");
            sleep(4);
            }
        else
            {
            pop_cursor();
            talk_to_k2107_controller();
            }
        tpoff();
        }
    else if ((20<=my) && (my<=70) &&
        (getbutton(MOUSE2) || getbutton(MOUSE3)))
        end_prog=1;
    }
else if ((mx>=10) && (mx<=sub_winx) && (my>=sub_winy) && (my<=1000))
    {
    winset(sub_win);
```

```
            winpop();
            grad_plot(total_steps, total_paths);
            win_state=1;
            }
        else if (getbutton(MOUSE2) && getbutton(MOUSE3))
            {
            winset(rob_win);
            winpop();
            draw_robot(theta_rd);
            drawing = 1;
            win_state=0;
            }
        if ((win_state==1) && (ellip==1))
            draw_curv();
        final_check(theta_rd);
        drawing = 0;
        }
    if (control_robot == 1)
        sigignore(SIGPOLL);
    blanktime(20000);
    textinit();
    tpon();
    close(ser_port2);
    close(ser_port3);
    close(ser_port4);
    gexit();
}


/************************ Talk to the robot controller ************************/

talk_to_k2107_controller()
{
    extern int control_robot, ser_port2;
    extern float theta_rd[MAT_DIM];
    extern int joint_out[MAT_DIM];
    long time_delay;
    int i, j, tp, okay;
    char buffer[200], buffer1[200];


    if (ser_port2 < 0)
        return;
    printf("\n\n Press the 'Ctrl' key for sending a command\n");
    printf("Press the 'Alt' key to exit\n");
    while (1)
        {
        i=feed_back(0, buffer);
        if (i>0)
            printf("%s",buffer);
        else if (getbutton(CTRLKEY))
```

```
        {
        printf("Sending command: ");
        fgets(buffer, 190, stdin);
        send_command(ser_port2, buffer);
        }
    else if (getbutton(BUT144))
        {
        printf("Changing the communication speed to:\n");
        printf("(1) 9600 baud even parity\n");
        printf("(2) 19200 baud even parity\n");
        printf("(3) 38400 baud even parity\n");
        fgets(buffer, 20, stdin);
        i=atoi(buffer);
        if ((i>=1) && (i<=3))
            {
            if (i==1)
                {
                port.c_cflag = 3517;
                sprintf(buffer, "NS30W3C");
                }
            else if (i==2)
                {
                port.c_cflag = 3518;
                sprintf(buffer, "NS30W1E");
                }
            else if (i==3)
                {
                port.c_cflag = 3519;
                sprintf(buffer, "NS30WF");
                }
            send_command(ser_port2, buffer);
            sleep(1);
            ringbell();
            tp=ioctl(ser_port2, TCSETA, &port);
            if (tp== -1)
                {
                printf("Fail to change the port baud rate!\n");
                printf("errno=%d\n", errno);
                }
            else
                printf("The communication speed be changed!\n", buffer);
            }
        }
    else if (getbutton(BUT142))
        break;
    }
printf("Controlling the robot in real time (y/n)? ");
fgets(buffer, 20, stdin);
if ((buffer[0]=='y') || (buffer[0]=='Y'))
    {
    time_delay=1000;
```

```
send_command(ser_port2, "N40S40U0");
i=feed_back(time_delay, buffer);
send_command(ser_port2, "N57m57");
time_delay=10000;
i=feed_back(time_delay, buffer);
if (i == 0)
    {
    control_robot=0;
    printf("Fail to read the feed back (time out)\n");
    return;
    }
printf("%s\n", buffer);
time_delay=1000;
j=0;
while (buffer[++j] != 'a');
j++;
for (i=1; i<=7; i++)
    {
    strncpy(buffer1, &buffer[j], 8);
    joint_out[i]= strtol (buffer1, &buffer1[8], 16);
    j+=9;
    }
for (i=1; i<=7; i++)
    theta_rd[i] = ((float) joint_out[i])/10000;
control_robot=0;
draw_robot(theta_rd);
send_command(ser_port2, "NS40UFF");
i=feed_back(time_delay, buffer);
if (i>0)
    printf("%s\n",buffer);
printf("Is the configuration correct?\n");
fgets(buffer, 10, stdin);
if ((buffer[0]=='y') || (buffer[0]=='Y'))
    {
    send_command(ser_port2, "NS24");          /* Set to Teach model     */
    i=feed_back(time_delay, buffer);
    if (i>0)
        printf("[1] %s\n",buffer);
    send_command(ser_port2, "N72S72U");      /* turn of char. interrupt */
    i=feed_back(time_delay, buffer);
    if (i>0)
        printf("[2] %s\n",buffer);
    send_command(ser_port2, "N70S70W");
    i=feed_back(time_delay, buffer);
    if (i>0)
        printf("[3] %s\n",buffer);
    send_command(ser_port2, "N71S71U");
    i=feed_back(time_delay, buffer);
    if (i>0)
        printf("[4] %s\n",buffer);
    time_delay = 5000;
```

```
i = 0;
okay = 0;
while (i++ < 10)
    {
    ioctl(ser_port2, TCFLSH, 2);
    send_command(ser_port2, "Nm80");
    j=feed_back(time_delay, buffer);
    if (j>0)
        {
        printf("(1) %s\n",buffer);
        if (0 != strncmp(buffer, "NHA0", 4))
            {
            send_command(ser_port2, "Nm42");
            j=feed_back(time_delay, buffer);
            if (j>0)
                printf("(2) %s\n",buffer);
            ioctl(ser_port2, TCFLSH, 0);
            send_command(ser_port2, "Nm95");
            j=feed_back(time_delay, buffer);
            if (j>0)
                {
                printf("(3) %s\n",buffer);
                if (0 == strncmp(buffer, "nHA0", 4))
                    okay = 1;
                }
            }
        else
            okay = 1;
        }
    else
        {
        send_command(ser_port2, "Nm42");
        j=feed_back(time_delay, buffer);
        if (j>0)
            printf("(4) %s\n",buffer);
        ioctl(ser_port2, TCFLSH, 0);
        send_command(ser_port2, "Nm95");
        j=feed_back(time_delay, buffer);
        if (j>0)
            {
            printf("(5) %s\n",buffer);
            if (0 == strncmp(buffer, "nHA0", 4))
                okay = 1;
            }
        }
    printf("ok = %d: Tried %d times to get teach model.\n", okay, i);
    if (okay == 1)
        break;
    }
send_command(ser_port2, "N42m42");
i=feed_back(time_delay, buffer);
```

```
    if (i>0)
        printf("%s\n",buffer);
    ioctl(ser_port2, TCFLSH, 0);
    signal(SIGPOLL, robot_jog);
    send_command(ser_port2, "N22S22");
    pause();
    printf("%s\n", r_feed_back);
    if (0 != strncmp(r_feed_back, "n22H90", 6))
        {
        okay = 0;
        sigignore(SIGPOLL);
        j = 0;
        while (j++ < 10)
            {
            send_command(ser_port2, "Nm42");
            time_delay = 5000;
            i=feed_back(time_delay, buffer);
            ioctl(ser_port2, TCFLSH, 0);
            send_command(ser_port2, "Nm95");
            if (i== 0)
                {
                send_command(ser_port2, "Nm80");
                i=feed_back(time_delay, buffer);
                }
            if (i>0)
                printf("%s\n",buffer);
            if ((0 == strncmp(buffer, "nHA0", 4))
                || (0 == strncmp(buffer, "NHA0", 4)))
                {
                send_command(ser_port2, "Nm42");
                i=feed_back(time_delay, buffer);
                ioctl(ser_port2, TCFLSH, 0);
                signal(SIGPOLL, robot_jog);
                send_command(ser_port2, "NS22");
                pause();
                printf("%s\n", r_feed_back);
                if ((0 == strncmp(r_feed_back, "nH90", 4))
                    || (0 == strncmp(r_feed_back, "nH890", 5)))
                    okay = 1;
                else
                    sigignore(SIGPOLL);
                }
            if (okay == 1)
                break;
            }
        }
    ringbell();
    control_robot=1;
    }
else
    sigignore(SIGPOLL);
```

```
        }
    else
        control_robot=0;
    return;
}

/****************** Talk to the ultrasonic sensor controller *****************/
talk_to_ult_sensor()
{
    extern int ser_port4, ult_switch;
    extern float obst_dis;
    char buffer[200];
    int i;

    if (ser_port4 < 0)
        return;
    printf("\n\nPress the 'Ctrl' key for sending a command\n");
    printf("Press the 'Alt' key to exit\n");
    while (1)
        {
        i = read(ser_port4, buffer, 100);
        if (i > 0)
            {
            buffer[i] = '\0';
            printf("%s",buffer);
            }
        if (getbutton(CTRLKEY))
            {
            printf("Sending command: ");
            fgets(buffer, 190, stdin);
            i=strlen(buffer);
            if (i!=0)
                i--;
            buffer[i]=0x0d;
            i++;
            write(ser_port4, buffer, i);
            }
        if (getbutton(BUT142))
            break;
        }
    printf("Using the ult_sensor? (y/n) ");
    fgets(buffer, 4, stdin);
    if ((buffer[0] == 'y') || (buffer[0] == 'Y'))
        {
        ult_switch = 1;
        buffer[0] = 0x41;
        buffer[1] = '\0';
        write(ser_port4, buffer, 1);
        for (i=1; i<=T_JOINT; i++)
            ult_dtheta[i] = 0;
        }
```

```
    else
        ult_switch = 0;
    obst_dis = 1000;
    return;
}

/******************** Talk to the JR3 force sensor controller *****************/
talk_to_force_sensor()
{
    extern int ser_port3, force_switch;
    extern float force_dtheta[MAT_DIM], force_data[MAT_DIM];
    char buffer[200];
    int i, j;

    if (ser_port3 < 0)
        return;

    printf("\n\n");
    printf("Press the 'Ctrl' key for sending a command\n");
    printf("Press the 'Alt' key to exit\n");
    pop_cursor();
    i=ioctl(ser_port3, TCGETA, &port);
    port.c_cc[4] = 0;
    port.c_cc[5] = 0;
    i=ioctl(ser_port3, TCSETA, &port);
    ioctl(ser_port3, TCFLSH, 1);
    send_command(ser_port3, "COM A EC ON EL ON PR ON");
    while (1)
        {
        i = read(ser_port3, buffer, 100);
        if (i > 0)
            {
            buffer[i] = '\0';
            printf("%s",buffer);
            }
        if (getbutton(CTRLKEY))
            {
            printf("Sending command: ");
            fgets(buffer, 190, stdin);
            send_command(ser_port3, buffer);
            }
        if (getbutton(BUT142))
            break;
        }
    printf("Using the force_sensor? (y/n) ");
    fgets(buffer, 4, stdin);
    if ((buffer[0] == 'y') || (buffer[0] == 'Y'))
        {
        force_switch = 1;
        send_command(ser_port3, "COM A EC OFF EL OFF PR OFF");
        for (i=1; i<=30000; i++)
```

```
            j++;
        i=ioctl(ser_port3, TCGETA, &port);
        port.c_cc[4] = 74;
        port.c_cc[5] = 1;
        i=ioctl(ser_port3, TCSETA, &port);
        ioctl(ser_port3, TCFLSH, 0);
        send_command(ser_port3, "DP S");
        }
    else
        force_switch = 0;
    for (i=1; i<=T_JOINT; i++)
        force_dtheta[i] = 0;
    return;
}
```

```
/*                    Source code file "goalp.c"                    */

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

#define TSTEPS 10000
#define TPATHS 40
#define MAT_DIM 10
#define T_JOINT 7

extern struct {int index; int sign; float obt_v;} obtctl;
extern struct {char st_con[2]; float goal_position[MAT_DIM];} g_state[TSTEPS];

/********** Linear path control between two trajectory points **********/
goal_lin(scheme_gain, dst)
float scheme_gain, *dst;
{
    extern float h_transf[5][5], linear_v;
    extern float theta_rd[MAT_DIM], path_distance;
    extern int steps;
    long mx, my;
    float distance, vels[8], dx[8];
    float temp1, temp2, temp3, tp1, origin[8];
    int i;

    difference(theta_rd, h_transf, dst, dx);
    get_position(origin, h_transf);
    distance=sqrt(dx[1]*dx[1]+dx[2]*dx[2]+dx[3]*dx[3]);
    if (distance <= linear_v)
        {
        goal_rpy(scheme_gain, dst);
        return(0);
        }
    tp1 = linear_v/distance;
    for (i=0; i<=6; i++)
        vels[i]=dx[i]*tp1;
    while (tp1<1)
        {
        steps++;
        origin[1]+=vels[1];
        origin[2]+=vels[2];
        origin[3]+=vels[3];
        temp1 = cos(origin[4]);
        temp2 = sin(origin[4]);
        origin[5] += vels[5]*temp1 - vels[4]*temp2;
        temp3 = (vels[4]*temp1 + vels[5]*temp2)/cos(origin[5]);
        origin[6] += temp3;
        origin[4] += (vels[6] + temp3*sin(origin[5]));
```

```
        temp1=sqrt(vels[1]*vels[1]+vels[2]*vels[2]+vels[3]*vels[3]);
        path_distance += temp1;
        k_inverse(1, scheme_gain, theta_rd, vels, origin);
        difference(theta_rd, h_transf, dst, dx);
        distance = sqrt(dx[1]*dx[1]+dx[2]*dx[2]+dx[3]*dx[3]);
        tp1 = linear_v/distance;
        for (i=1; i<=6; i++)
            vels[i]=dx[i]*tp1;
        if (getbutton(MOUSE2) || getbutton(MOUSE3))
            {
            mx=getvaluator(MOUSEX);
            my=getvaluator(MOUSEY);
            if ((1100<=mx) && (mx<=1170) && (my>=20) && (my<=70))
                return(1);
            }
        }
    return(0);
}

/************ Rol-Pitch-Yaw motion between two trajectory points **************/
goal_rpy(scheme_gain, dst)
float scheme_gain, *dst;
{
    extern float h_transf[5][5], joint_v;
    extern float theta_rd[MAT_DIM];
    extern float path_distance;
    extern int steps;
    long mx, my;
    float distance, vels[8], dx[8];
    float temp1, temp2, temp3, tp1, origin[8];
    int i;

    difference(theta_rd, h_transf, dst, dx);
    get_position(origin, h_transf);
    distance=sqrt(dx[4]*dx[4]+dx[5]*dx[5]+dx[6]*dx[6]);
    if (distance < joint_v)
        {
        origin[1]+=dx[1];
        origin[2]+=dx[2];
        origin[3]+=dx[3];
        temp1 = cos(origin[4]);
        temp2 = sin(origin[4]);
        origin[5] += dx[5]*temp1 - dx[4]*temp2;
        temp3 = (dx[4]*temp1 + dx[5]*temp2)/cos(origin[5]);
        origin[6] += temp3;
        origin[4] += (dx[6] + temp3*sin(origin[5]));
        path_distance += sqrt(dx[1]*dx[1] + dx[2]*dx[2] + dx[3]*dx[3]);
        k_inverse(1, scheme_gain, theta_rd, dx, origin);
        return(0);
        }
    tp1 = joint_v/distance;
```

```
    for (i=1; i<=6; i++)
        vels[i]=dx[i]*tp1;

    while (tp1 < 1)
        {
        steps++;
        origin[1]+=vels[1];
        origin[2]+=vels[2];
        origin[3]+=vels[3];
        temp1 = cos(origin[4]);
        temp2 = sin(origin[4]);
        origin[5] += vels[5]*temp1 - vels[4]*temp2;
        temp3 = (vels[4]*temp1 + vels[5]*temp2)/cos(origin[5]);
        origin[6] += temp3;
        origin[4] += (vels[6] + temp3*sin(origin[5]));
        path_distance += sqrt(vels[1]*vels[1] + vels[2]*vels[2] + vels[3]*vels[3]);
        k_inverse(1, scheme_gain, theta_rd, vels, origin);
        difference(theta_rd, h_transf, dst, dx);
        distance = sqrt(dx[4]*dx[4]+dx[5]*dx[5]+dx[6]*dx[6]);
        tp1 = joint_v/distance;
        for (i=1; i<=6; i++)
            vels[i] = dx[i]*tp1;
        if (getbutton(MOUSE2) || getbutton(MOUSE3))
            {
            mx=getvaluator(MOUSEX);
            my=getvaluator(MOUSEY);
            if ((1100<=mx) && (mx<=1170) && (my>=20) && (my<=70))
                return(1);
            }
        }
    return(0);
}

/*************** Joint motion between two trajectory points ******************/
goal_jnt(n, dst)
int n;
float *dst;
{
    extern float joint_v, theta_rd[MAT_DIM];
    extern int win_state, steps1;
    long mx, my;
    float tp1, temp1, temp2, dtheta[MAT_DIM];
    int i, maxv_joint;

    temp1=0;
    for (i=0; i<=n; i++)
        {
        temp2=fabs(dst[i]-theta_rd[i]);
        if (temp1<temp2)
            {
            maxv_joint = i;
```

```
            temp1=temp2;
            }
        }
    tp1 = joint_v/temp1;
    for (i=0; i<=n; i++)
        dtheta[i] = dst[i]-theta_rd[i];
    while (tp1 < 1)
        {
        for (i=0; i<=n; i++)
            theta_rd[i]+=dtheta[i]*tp1;
        draw_robot(theta_rd);
        dtheta[maxv_joint] = dst[maxv_joint] - theta_rd[maxv_joint];
        tp1 = joint_v/fabs(dtheta[maxv_joint]);
        if (win_state==1)
            {
            steps1=steps;
            draw_curv();
            }

        if (getbutton(MOUSE2) || getbutton(MOUSE3))
            {
            mx=getvaluator(MOUSEX);
            my=getvaluator(MOUSEY);
            if ((1100<=mx) && (mx<=1170) && (my>=20) && (my<=70))
            return(1);
            }
        }
    for (i=0; i<=n; i++)
        theta_rd[i]=dst[i];
    draw_robot(theta_rd);
    if (win_state==1)
        {
        steps1=steps;
        draw_curv();
        }
    return(0);
}

/************************* Open or close the gripper ***************************/
goal_gp(theta_rd, gp)
float *theta_rd, gp;
{
    extern float joint_v;
    long mx, my;
    float temp1, temp2;
    int i;

    temp1 = gp - theta_rd[0];
    temp2 = 20*joint_v;
    if (temp1 < 0)
        temp2 = -temp2;
```

```
    while(fabs(temp1) > fabs(temp2))
        {
        theta_rd[0] += temp2;
        temp1 = gp - theta_rd[0];
        draw_robot(theta_rd);
        if (getbutton(MOUSE2) || getbutton(MOUSE3))
            {
            mx=getvaluator(MOUSEX);
            my=getvaluator(MOUSEY);
            if ((1100<=mx) && (mx<=1170) && (my>=20) && (my<=70))
                return(1);
            }
        }
    theta_rd[0] = gp;
    draw_robot(theta_rd);
    return(0);
}

/***************** Orbit motion between two trajectory points ******************/
goal_obt(n, dst)
int n;
float *dst;
{
    extern struct obtctl;
    extern float theta_rd[MAT_DIM];
    extern int e_traj_count, win_state, steps1;
    long mx, my;
    float temp1, temp2, temp3, vels[8], position[8], dtheta[MAT_DIM];
    int i, j;

    temp1=0;
    for (i=1; i<=n; i++)
        {
        temp2=dst[i]-theta_rd[i];
        if (fabs(temp2)>fabs(temp1))
            {
            j=i;
            temp1=temp2;
            }
        }
    o_motion(theta_rd, dtheta);
    if (dtheta[j]*temp1<0)
        {
        obtctl.sign = -obtctl.sign;
        }
    get_position(position, h_transf);
    temp2=temp3=fabs(obtctl.obt_v);
    while (temp2<=temp3)
        {
        o_motion(theta_rd, dtheta);
        new_theta(theta_rd, dtheta);
```

```
        difference(theta_rd, h_transf, position, vels);
        least_norm(theta_rd, vels, dtheta);
        new_theta(theta_rd, dtheta);
        draw_robot(theta_rd);
        if (win_state==1)
            {
            steps1=steps;
            draw_curv();
            }
        if (getbutton(MOUSE2) || getbutton(MOUSE3))
            {
            mx=getvaluator(MOUSEX);
            my=getvaluator(MOUSEY);
            if ((1100<=mx) && (mx<=1170) && (my>=20) && (my<=70))
                return(1);
            }
        temp3=fabs(dst[j]-theta_rd[j]);
        }
    temp1=obtctl.obt_v;
    obtctl.obt_v=dst[j]-theta_rd[j];
    obtctl.index=j;
    o_motion(theta_rd, dtheta);
    new_theta(theta_rd, dtheta);
    obtctl.obt_v=temp1;
    difference(theta_rd, h_transf, position, vels);
    least_norm(theta_rd, vels, dtheta);
    new_theta(theta_rd, dtheta);
    draw_robot(theta_rd);
    if (win_state==1)
        {
        steps1=steps;
        draw_curv();
        }
    return(0);
}

/******************* Linear and circular paths calculation *********************/
linear_cont(scheme_gain, r, x2, x3, stop)
float scheme_gain, r, *x2, *x3;
int stop;
{
    extern float linear_v;
    float dx1, dx2, dy1, dy2, dz1, dz2, k1, k2, k3, r_new;
    float turn, v_theta_half, vect1, vect2, tp1, tp2;
    float a[8], b[8], c[4], center[4], norm_turn[4];

    r_new = r;
    dx1=x2[1]-h_transf[1][4];
    dy1=x2[2]-h_transf[2][4];
    dz1=x2[3]-h_transf[3][4];
    dx2=x3[1]-x2[1];
```

```
dy2=x3[2]-x2[2];
dz2=x3[3]-x2[3];
vect1=sqrt(dx1*dx1+dy1*dy1+dz1*dz1);
if (vect1 < linear_v)
    return(0);
vect2=sqrt(dx2*dx2+dy2*dy2+dz2*dz2);
tp1=dx1*dx2+dy1*dy2+dz1*dz2;
turn=acos(tp1/vect1/vect2);
v_theta_half=0.5*turn;
tp1=tan(v_theta_half);
k1 = vect1 - linear_v*2;
if (stop == 0)
    k2 = vect2/2 - linear_v*2;
else
    k2 = vect2 - linear_v*2;
if ((r/tp1 > k1) || (r/tp1 > k2))
    {
    if (k1 > k2)
        r_new = k2*tp1;
    else
        r_new = k1*tp1;
    if (r_new <= 0)
        {
        if (1 == goal_lin(scheme_gain, x2))
            return(1);
        else
            return(0);
        }
    }
k1=tp1*r_new/vect1;
k2=tp1*r_new/vect2;
a[0] = x2[0];
a[1]=x2[1]-k1*dx1;
a[2]=x2[2]-k1*dy1;
a[3]=x2[3]-k1*dz1;
b[1]=x2[1]+k2*dx2;
b[2]=x2[2]+k2*dy2;
b[3]=x2[3]+k2*dz2;
c[1]=0.5*(a[1]+b[1]);
c[2]=0.5*(a[2]+b[2]);
c[3]=0.5*(a[3]+b[3]);
a[4]=x2[4];
a[5]=x2[5];
a[6]=x2[6];
if (1==goal_lin(scheme_gain, a))
    return(1);
if ((linear_v/r_new)<turn)
    {
    k3=sin(v_theta_half)*sin(v_theta_half);
    center[1]=x2[1]+(c[1]-x2[1])/k3;
    center[2]=x2[2]+(c[2]-x2[2])/k3;
```

```
        center[3]=x2[3]+(c[3]-x2[3])/k3;
        norm_turn[1]=dy1*dz2-dz1*dy2;
        norm_turn[2]=dx2*dz1-dx1*dz2;
        norm_turn[3]=dx1*dy2-dx2*dy1;
        if (1==circle(scheme_gain, r_new, a, center, norm_turn, turn))
            return(1);
        }
    return(0);
}


/***************************** Circular path planning ************************/

circle(scheme_gain, r, coords, center, omega, turn)
float scheme_gain, r, *coords, *center, *omega, turn;
{
    extern float linear_v, theta_rd[MAT_DIM];
    extern int steps;
    long mx, my;
    float temp1, tp1, tp2, dx[8], x, y, x1, y1, angle, d_theta, origin[8];
    float vels[8], xx, yy, zz, xx1, yy1, zz1, norm_x[4], norm_y[4], norm_z[4];

    get_position(origin, h_transf);
    norm_x[1]=(coords[1]-center[1])/r;
    norm_x[2]=(coords[2]-center[2])/r;
    norm_x[3]=(coords[3]-center[3])/r;
    tp1=sqrt(omega[1]*omega[1]+omega[2]*omega[2]+omega[3]*omega[3]);
    norm_z[1]=omega[1]/tp1;
    norm_z[2]=omega[2]/tp1;
    norm_z[3]=omega[3]/tp1;
    norm_y[1]=norm_z[2]*norm_x[3]-norm_z[3]*norm_x[2];
    norm_y[2]=norm_z[3]*norm_x[1]-norm_z[1]*norm_x[3];
    norm_y[3]=norm_z[1]*norm_x[2]-norm_z[2]*norm_x[1];
    xx=coords[1]-h_transf[1][4];
    yy=coords[2]-h_transf[2][4];
    zz=coords[3]-h_transf[3][4];
    y=xx*xx+yy*yy+zz*zz;
    xx=sqrt(y+r*r);
    yy=(2*r*r+y-linear_v*linear_v)/(2*xx*r);
    zz=sqrt(y);
    angle=acos(yy)-atan2(zz, r);
    y=r*sin(angle);
    x=r*cos(angle);
    dx[1]=x-r;
    dx[2]=y+zz;
    xx=dx[1]*norm_x[1]+dx[2]*norm_y[1];
    yy=dx[1]*norm_x[2]+dx[2]*norm_y[2];
    zz=dx[1]*norm_x[3]+dx[2]*norm_y[3];
    dx[1]=xx;
    dx[2]=yy;
    dx[3]=zz;
```

```
dx[4]=dx[5]=dx[6]=0;
origin[1]+=xx;
origin[2]+=yy;
origin[3]+=zz;
steps++;
temp1=sqrt(xx*xx+yy*yy+zz*zz);
path_distance+=temp1;
k_inverse(1, scheme_gain, theta_rd, dx, origin);
d_theta=2*asin(linear_v/2/r);
tp1=sin(d_theta);
tp2=cos(d_theta);
xx=coords[1];
yy=coords[2];
zz=coords[3];

while (fabs(turn-angle)>=d_theta)
    {
    steps++;
    x1=x*tp2-y*tp1;
    y1=x*tp1+y*tp2;
    dx[1]=x1-x;
    dx[2]=y1-y;
    xx=dx[1]*norm_x[1]+dx[2]*norm_y[1];
    yy=dx[1]*norm_x[2]+dx[2]*norm_y[2];
    zz=dx[1]*norm_x[3]+dx[2]*norm_y[3];
    dx[1]=xx;
    dx[2]=yy;
    dx[3]=zz;
    dx[4]=dx[5]=dx[6]=0;
    origin[1]+=xx;
    origin[2]+=yy;
    origin[3]+=zz;
    dx[0]=1;
    temp1=sqrt(xx*xx+yy*yy+zz*zz);
    path_distance+=temp1;
    k_inverse(1, scheme_gain, theta_rd, dx, origin);
    if (dx[0]==0)
        {
        origin[1]-=xx;
        origin[2]-=yy;
        origin[3]-=zz;
        path_distance-=temp1;
        }
    else
        {
        x=x1;
        y=y1;
        angle+=d_theta;
        }
    if (getbutton(MOUSE2) || getbutton(MOUSE3))
        {
```

```
            mx=getvaluator(MOUSEX);
            my=getvaluator(MOUSEY);
            if ((1100<=mx) && (mx<=1170) && (my>=20) && (my<=70))
            return(1);
            }
        }
    d_theta=turn-angle;
    xx=r*(1-cos(d_theta));
    zz=sqrt(linear_v*linear_v-xx*xx)-r*sin(d_theta);
    xx=cos(turn);
    yy=sin(turn);
    x1=r*xx-zz*yy;
    y1=r*yy+zz*xx;
    dx[1]=x1-x;
    dx[2]=y1-y;
    xx=dx[1]*norm_x[1]+dx[2]*norm_y[1];
    yy=dx[1]*norm_x[2]+dx[2]*norm_y[2];
    zz=dx[1]*norm_x[3]+dx[2]*norm_y[3];
    dx[1]=xx;
    dx[2]=yy;
    dx[3]=zz;
    dx[4]=dx[5]=dx[6]=0;
    origin[1]+=xx;
    origin[2]+=yy;
    origin[3]+=zz;
    steps++;
    temp1=sqrt(xx*xx+yy*yy+zz*zz);
    path_distance+=temp1;
    k_inverse(1, scheme_gain, theta_rd, dx, origin);
    return(0);
}

/*************** Kinematics inverse with  local optimization ******************/

k_inverse(draw, scheme_gain, theta_rd, vels, origin)
int draw;
float *theta_rd, *vels, scheme_gain, *origin;
{
    extern struct obtctl;
    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;
    static float sigma6;
    int i, joint_v_lim;
    float dp1[MAT_DIM], dp2[MAT_DIM], dp3[MAT_DIM];
    float er[MAT_DIM], tp[MAT_DIM];
    float  norm2, norm3, norm_k, 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];
    float  vect11[4], vect12[4], vect21[4], vect22[4], link_p1[4], link_p2[4];
```

```
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];
least_norm(tp, vels, dp1);

new_theta(tp, dp1);
if (scheme_index == 0)                    /* No optimization        */
    {
    difference(tp, h_transf, origin, temp);
    least_norm(tp, temp, dp2);
    new_theta(tp, dp2);
    new_theta(dp1, dp2);
    for (i=1; i<=T_JOINT; i++)
        {
        theta_rd[i]+=dp1[i];
        control_path[steps][2]+=dp1[i]*dp1[i];
        }
    control_path[steps][2]=sqrt(control_path[steps][2])*1145.9156;
    total_cost += control_path[steps][2];
    if (draw == 1)
        draw_robot(theta_rd);
    if (win_state==1)
        {
        steps1=steps;
        draw_curv();
        }
    return;
    }
for (i=1; i<=T_JOINT; i++)
    tp[i]=theta_rd[i];
tp[0] = obtctl.obt_v;
i = obtctl.sign;
if (obtctl.obt_v>0)
    obtctl.obt_v = 0.05236;
else
    obtctl.obt_v = -0.05236;
get_position(position, h_transf);
o_motion(tp, dp2);
new_theta(tp, dp2);
difference(tp, h_transf, position, er);
least_norm(tp, er, dp3);
new_theta(tp, dp3);
```

```
new_theta(dp2, dp3);
obtctl.obt_v = tp[0];
obtctl.sign = i;
norm2 = 0;
for (i=1; i<=T_JOINT; i++)
    norm2 + =dp2[i]*dp2[i];

if (scheme_index == 1)        /* Minimize the norm of the least-norm */
    {
    least_norm(tp, vels, dp3);
    norm1 = norm3 = 0;
    for (i=1; i<=T_JOINT; i++)
        {
        norm1+=dp1[i]*dp1[i];
        norm3+=dp3[i]*dp3[i];
        }
    norm_k = (norm3*norm3-norm1*norm1)/norm2/(sqrt(norm3)+sqrt(norm1))
                /(norm3+norm1);
    norm1 = sqrt(norm1)*1145.9156;
    }
else if (scheme_index == 2)   /* Minimize the inverse of det(JJ^T)   */
    {
    jacobian(theta_rd,h_transf,jaco);
    cal_jjt(6, T_JOINT, jaco, jsqr);
    determ(6, jsqr);
    norm1=1/sqrt(jsqr[0][0]);
    jacobian(tp,h_transf,jaco);
    cal_jjt(6, T_JOINT, jaco, jsqr);
    determ(6, jsqr);
    norm3=1/sqrt(jsqr[0][0]);
    norm_k = (norm3*norm3-norm1*norm1)/norm2/(norm3+norm1);
    norm1 *= 1.5e6;
    }
else if (scheme_index == 3)   /* Minimize the inverse of min. SV (1) */
    {
    jacobian(theta_rd,h_transf,jaco);
    cal_jjt(6, T_JOINT, jaco, jsqr);
    svd(6, jsqr, eigen);
    eigen[0]=eigen[1];
    for (i=2; i<=6; i++)
        {
        if (eigen[0]>eigen[i])
            eigen[0]=eigen[i];
        }
    norm1=1/sqrt(eigen[0]);
    jacobian(tp,h_transf,jaco);
    cal_jjt(6, T_JOINT, jaco, jsqr);
    svd(6, jsqr, eigen);
    eigen[0]=eigen[1];
    for (i=2; i<=6; i++)
        {
```

```
        if (eigen[0]>eigen[i])
            eigen[0]=eigen[i];
        }
    norm3=1/sqrt(eigen[0]);
    norm_k = (norm3*norm3-norm1*norm1)/norm2/(norm3+norm1);
    norm1 *= 20;
    }
else if (scheme_index == 4)   /* Minimize the inverse of min. SV (2) */
    {
    jacobian(theta_rd,h_transf,jaco);
    cal_jjt(6, T_JOINT, jaco, jsqr);
    for (i=1; i<=6; i++)
        temp[i]=jsqr[i][i];
    x0=(-0.1);
    for (i=1; i<=6; i++)
        jsqr[i][i]=temp[i]-x0;
    determ(6, jsqr);
    y0=sqrt(jsqr[0][0]);
    if (steps==1)
        sigma6=0.1;
    x1=sigma6;
    for (i=1; i<=6; i++)
        jsqr[i][i]=temp[i]-x1;
    determ(6, jsqr);
    y1=jsqr[0][0];
    kk=(y1-y0)/(x1-x0);
    x=x1-y1/kk;
    while (fabs(y1/kk)>1e-6)
        {
        x0=x1;
        x1=x;
        y0=y1;
        for (i=1; i<=6; i++)
            jsqr[i][i]=temp[i]-x1;
        determ(6, jsqr);
        y1=jsqr[0][0];
        kk=(y1-y0)/(x1-x0);
        x=x1-y1/kk;
        }
    sigma6=x;
    norm1=1/sqrt(sigma6);
    jacobian(tp,h_transf,jaco);
    cal_jjt(6, T_JOINT, jaco, jsqr);
    for (i=1; i<=6; i++)
        temp[i]=jsqr[i][i];
    x0=(-0.1);
    for (i=1; i<=6; i++)
        jsqr[i][i]=temp[i]-x0;
    determ(6, jsqr);
    y0=sqrt(jsqr[0][0]);
    x1=sigma6;
```

```
        for (i=1; i<=6; i++)
            jsqr[i][i]=temp[i]-x1;
        determ(6, jsqr);
        y1=jsqr[0][0];
        kk=(y1-y0)/(x1-x0);
        x=x1-y1/kk;
        while (fabs(y1/kk)>1e-6)
            {
            x0=x1;
            x1=x;
            y0=y1;
            for (i=1; i<=6; i++)
                jsqr[i][i]=temp[i]-x1;
            determ(6, jsqr);
            y1=jsqr[0][0];
            kk=(y1-y0)/(x1-x0);
            x=x1-y1/kk;
            }
        norm3=1/sqrt(x);
        norm_k = (norm3*norm3-norm1*norm1)/norm2/(norm3+norm1);
        norm1 *= 20;
        }
    else if (scheme_index == 5)        /* Joint limit avoidance (1)    */
        {
        get_j_lim(&norm1, theta_rd);
        get_j_lim(&norm3, tp);
        norm_k = (norm3*norm3-norm1*norm1)/norm2/(norm3+norm1);
        }
    else if (scheme_index == 6)        /* Joint limit avoidance (2)    */
        {
        get_j_lim2(&norm1, theta_rd);
        get_j_lim2(&norm3, tp);
        norm_k = (norm3*norm3-norm1*norm1)/norm2/(norm3+norm1);
        }
    norm_k *= scheme_gain;

/* If give the homo. solution a limit */
    norm2 = sqrt(norm2);
    if (robot_index == 1)
        {
        if (norm_k>0.1/norm2)
            norm_k=0.1/norm2;
        else if (norm_k<-0.1/norm2)
            norm_k=(-0.1/norm2);
        }
    else if (robot_index == 2)
        {
        if (norm_k>0.2/norm2)
            norm_k=0.2/norm2;
        else if (norm_k<-0.2/norm2)
            norm_k=(-0.2/norm2);
```

```
        }
    for (i=1; i<=T_JOINT; i++)
        dp3[i] = dp1[i] - dp2[i]*norm_k;

    /* If using max_joint_v_limit method
        j_v_limit(dp1, dp2, norm_k, dp3); */

    new_theta(theta_rd, dp3);
    difference(theta_rd, h_transf, origin, er);
    least_norm(theta_rd, er, dp1);
    new_theta(theta_rd, dp1);
    new_theta(dp3, dp1);
    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;
        }
    control_path[steps][2] = norm1;
    if (draw == 1)
        draw_robot(theta_rd);
    if (win_state==1)
        {
        steps1=steps;
        draw_curv();
        }
    return;
}

/************************** 3-D map generator ****************************/

grad_data(ref_joint, self_inc, total_path, r, s_count, goal_steps)
int ref_joint, total_path, s_count, goal_steps;
float r, self_inc;
{
    extern float theta_rd[MAT_DIM], h_transf[5][5];
    extern float trajectory[TSTEPS][3];
    extern struct obtctl;
    extern struct g_state;
    extern float scheme_gain, path_distance;
    extern int total_steps;
    int i, j, k, l, stop;
    float init_theta[MAT_DIM], origin[8], joint1, temp1, temp2, temp3, tp[8], tpp[8];
    float dp[8], obv1, tp1, tp2, tp3, droll, dpitch, dyaw;

    for (i=0; i<=T_JOINT; i++)
        init_theta[i]=theta_rd[i];
    get_position(origin, h_transf);
    if (joint_v < fabs(self_inc))
```

```
    {
    if (obtctl.obt_v > 0)
        obtctl.obt_v = joint_v;
    else
        obtctl.obt_v = -joint_v;
    }
joint1=theta_rd[ref_joint]-self_inc;
for (i=1; i<=total_path; i++)
    {
    path_distance=0;
    e_traj_count=0;
    trajectory[0][0]=origin[1];
    trajectory[0][1]=origin[2];
    trajectory[0][2]=origin[3];
    joint1+=self_inc;
    for (j=0; j<=T_JOINT; j++)
        theta_rd[j]=init_theta[j];
    if (i>1)
        if (1 == self_motion(ref_joint, self_inc, joint1, origin, theta_rd))
            return(1);
    for (j=1; j<=T_JOINT; j++)
        init_theta[j]=theta_rd[j];
    k=s_count;
    steps=0;
    for (s_count; s_count<=goal_steps; s_count++)
        {
        if ((s_count==goal_steps) || (g_state[s_count].st_con[1]==0))
            {
            l=grad_lin(ref_joint, i, g_state[s_count].goal_position);
            }
        else
            {
            j=s_count+1;
            if ((g_state[j].st_con[1]==0) || (j==goal_steps)
                || (g_state[j].st_con[0]>=8) || (g_state[j].st_con[0] <= 2))
                stop = 1;
            else
                stop = 0;
            l=linear_cont_grad(ref_joint, i, r, g_state[s_count].goal_position,
                g_state[j].goal_position, stop);
            }
        if (l==1)
            return(1);
        }
    s_count=k;
    if (i==1)
        total_steps = steps - 1;
    }
for (i=1; i<=T_JOINT; i++)
    theta_rd[i]=init_theta[i];
return(0);
```

```
}

/********** Self motion function used by 3-D map generator ******************/
self_motion(ref_joint, self_inc, goal_rd, origin, theta_rd)
int ref_joint;
float self_inc, goal_rd, *origin, *theta_rd;
{
    extern float h_transf[5][5];
    extern struct obtctl;
    long mx, my;
    float tpp[MAT_DIM], dp[MAT_DIM], temp1=0, temp2, temp3, tp1, tp2, tp3;
    float vels[8], droll, dpitch, dyaw, temp_obt_v;
    int i;

    o_motion(theta_rd, tpp);
    if (tpp[ref_joint]*self_inc < 0)
        obtctl.sign = -obtctl.sign;

    if (fabs(obtctl.obt_v) >= fabs(self_inc))
        {
        if (obtctl.obt_v > 0)
            obtctl.obt_v = fabs(self_inc);
        else
            obtctl.obt_v = -fabs(self_inc);
        }
    temp_obt_v = obtctl.obt_v;
    temp1=theta_rd[ref_joint];
    while (fabs(goal_rd-temp1)>0.000017453293)
        {
        temp2=goal_rd-temp1;
        for (i=1; i<=7; i++)
            tpp[i]=theta_rd[i];
        o_motion(theta_rd, dp);
        new_theta(theta_rd, dp);
        difference(theta_rd, h_transf, origin, vels);
        least_norm(theta_rd, vels, dp);
        new_theta(theta_rd, dp);
        temp3=goal_rd-theta_rd[ref_joint];
        if ((temp3*temp2)<0)
            {
            obtctl.obt_v/=2.0;
            for (i=1; i<=7; i++)
                theta_rd[i]=tpp[i];
            }
        else
            temp1=theta_rd[ref_joint];
        if (getbutton(MOUSE2) || getbutton(MOUSE3))
            {
            mx=getvaluator(MOUSEX);
            my=getvaluator(MOUSEY);
            if ((1100<=mx) && (mx<=1170) && (my>=20) && (my<=70))
```

```
            return(1);
            }
        }
    obtctl.obt_v = temp_obt_v;
    draw_robot(theta_rd);
    return(0);
}

/******* Linear and circular path calculation used by 3-D map generator *******/
linear_cont_grad(ref_joint, path, r, x2, x3, stop)
float r, *x2, *x3;
int ref_joint, path, stop;
{
    extern float linear_v, linear_a;
    extern int steps;
    extern float h_transf[5][5];
    float dx1, dx2, dy1, dy2, dz1, dz2, k1, k2, k3, r_new;
    float turn, v_theta_half, vect1, vect2, tp1, tp2;
    float a[8], b[8], c[4], center[4], norm_turn[4];

    r_new = r;
    dx1=x2[1]-h_transf[1][4];
    dy1=x2[2]-h_transf[2][4];
    dz1=x2[3]-h_transf[3][4];
    if (sqrt(dx1*dx1+dy1*dy1+dz1*dz1)<linear_v*2)
        return(0);
    dx2=x3[1]-x2[1];
    dy2=x3[2]-x2[2];
    dz2=x3[3]-x2[3];
    vect1=sqrt(dx1*dx1+dy1*dy1+dz1*dz1);
    vect2=sqrt(dx2*dx2+dy2*dy2+dz2*dz2);
    tp1=dx1*dx2+dy1*dy2+dz1*dz2;
    turn=acos(tp1/vect1/vect2);
    v_theta_half=0.5*turn;
    tp1=tan(v_theta_half);
    if (stop == 0)
        tp2 = vect2/2;
    else
        tp2 = vect2;
    if ((r/tp1 > vect1) || (r/tp1 > tp2))
        {
        if (vect1 > tp2)
            r_new = (tp2 - linear_v)*tp1;
        else
            r_new = (vect1 - linear_v)*tp1;
        if (r_new <= 0)
            return(0);
        }
    k1=tp1*r_new/vect1;
    k2=tp1*r_new/vect2;
    a[1]=x2[1]-k1*dx1;
```

```
    a[2]=x2[2]-k1*dy1;
    a[3]=x2[3]-k1*dz1;
    b[1]=x2[1]+k2*dx2;
    b[2]=x2[2]+k2*dy2;
    b[3]=x2[3]+k2*dz2;
    c[1]=0.5*(a[1]+b[1]);
    c[2]=0.5*(a[2]+b[2]);
    c[3]=0.5*(a[3]+b[3]);
    rpy_angles(h_transf);
    a[4]=h_transf[0][1];
    a[5]=h_transf[0][2];
    a[6]=h_transf[0][3];
    if (1==grad_lin(ref_joint, path, a))
        return(1);
    if ((linear_v/r_new)<turn)
        {
        k3=sin(v_theta_half)*sin(v_theta_half);
        center[1]=x2[1]+(c[1]-x2[1])/k3;
        center[2]=x2[2]+(c[2]-x2[2])/k3;
        center[3]=x2[3]+(c[3]-x2[3])/k3;
        norm_turn[1]=dy1*dz2-dz1*dy2;
        norm_turn[2]=dx2*dz1-dx1*dz2;
        norm_turn[3]=dx1*dy2-dx2*dy1;
        if (1==grad_circle(ref_joint, path, r_new, a, center, norm_turn, turn))
            return(1);
        }
    return(0);
}

/************** Linear path control used by 3-D map generator ******************/

grad_lin(ref_joint, path, dst)
float *dst;
int ref_joint, path;
{
    extern float h_transf[5][5], linear_v, joint_v, linear_a;
    extern long mx, my;
    extern int steps, scheme_index;
    extern float norm_dat[TSTEPS][TPATHS][3], path_distance;
    float distance, diff[MAT_DIM], vels[MAT_DIM];
    float temp1, temp2, temp3, tp1, origin[MAT_DIM];
    int i;

    difference(theta_rd, h_transf, dst, diff);
    get_position(origin, h_transf);
    distance=sqrt(diff[1]*diff[1]+diff[2]*diff[2]+diff[3]*diff[3]);
    tp1=linear_v/distance;
    for (i=1; i<=6; i++)
        vels[i]=diff[i]*tp1;
    while (tp1<1)
        {
```

```
        steps++;
        origin[1] += vels[1];
        origin[2] += vels[2];
        origin[3] += vels[3];
        temp1 = cos(origin[4]);
        temp2 = sin(origin[4]);
        origin[5] += vels[5]*temp1 - vels[4]*temp2;
        temp3 = (vels[4]*temp1 + vels[5]*temp2)/cos(origin[5]);
        origin[6] += temp3;
        origin[4] += (vels[6] + temp3*sin(origin[5]));
        path_distance+=sqrt(vels[1]*vels[1]+vels[2]*vels[2]+vels[3]*vels[3]);
        get_grad_dat(ref_joint, path, steps, vels, origin);
        if (getbutton(MOUSE2) || getbutton(MOUSE3))
            {
            mx=getvaluator(MOUSEX);
            my=getvaluator(MOUSEY);
            if ((1100<=mx) && (mx<=1170) && (my>=20) && (my<=70))
            return(1);
            }
        diff[1] = dst[1]-h_transf[1][4];
        diff[2] = dst[2]-h_transf[2][4];
        diff[3] = dst[3]-h_transf[3][4];
        distance=sqrt(diff[1]*diff[1]+diff[2]*diff[2]+diff[3]*diff[3]);
        tp1 = linear_v/distance;
        for (i=1; i<=6; i++)
            vels[i]=diff[i]*tp1;
        }
    return(0);
}

/************* Circular path planning used by 3-D map generator ***************/
grad_circle(ref_joint, path, r, coords, center, omega, turn)
float r, *coords, *center, *omega, turn;
int ref_joint, path;
{
    extern float linear_v, linear_a, theta_rd[MAT_DIM];
    extern float scheme_gain, path_distance;
    extern float norm_dat[TSTEPS][TPATHS][3];
    extern int steps, scheme_index;
    float tp1, tp2, dx[8], x, y, x1, y1, angle, d_theta, origin[8];
    float xx, yy, zz, norm_x[4], norm_y[4], norm_z[4];
    int i, k;

    get_position(origin, h_transf);
    norm_x[1]=(coords[1]-center[1])/r;
    norm_x[2]=(coords[2]-center[2])/r;
    norm_x[3]=(coords[3]-center[3])/r;
    tp1=sqrt(omega[1]*omega[1]+omega[2]*omega[2]+omega[3]*omega[3]);
    norm_z[1]=omega[1]/tp1;
    norm_z[2]=omega[2]/tp1;
    norm_z[3]=omega[3]/tp1;
```

```
norm_y[1]=norm_z[2]*norm_x[3]-norm_z[3]*norm_x[2];
norm_y[2]=norm_z[3]*norm_x[1]-norm_z[1]*norm_x[3];
norm_y[3]=norm_z[1]*norm_x[2]-norm_z[2]*norm_x[1];
xx=coords[1]-h_transf[1][4];
yy=coords[2]-h_transf[2][4];
zz=coords[3]-h_transf[3][4];
y=xx*xx+yy*yy+zz*zz;
xx=sqrt(y+r*r);
yy=(2*r*r+y-linear_v*linear_v)/(2*xx*r);
zz=sqrt(y);
angle=acos(yy)-atan2(zz, r);
y=r*sin(angle);
x=r*cos(angle);
dx[1]=x-r;
dx[2]=y+zz;
xx=dx[1]*norm_x[1]+dx[2]*norm_y[1];
yy=dx[1]*norm_x[2]+dx[2]*norm_y[2];
zz=dx[1]*norm_x[3]+dx[2]*norm_y[3];
dx[1]=xx;
dx[2]=yy;
dx[3]=zz;
dx[4]=dx[5]=dx[6]=0;
origin[1]+=xx;
origin[2]+=yy;
origin[3]+=zz;
path_distance+=sqrt(xx*xx+yy*yy+zz*zz);
steps++;
get_grad_dat(ref_joint, path, steps, dx, origin);
d_theta=2*asin(linear_v/2/r);
tp1=sin(d_theta);
tp2=cos(d_theta);

while (fabs(turn-angle)>=d_theta)
    {
    steps++;
    x1=x*tp2-y*tp1;
    y1=x*tp1+y*tp2;
    dx[1]=x1-x;
    dx[2]=y1-y;
    x=x1;
    y=y1;
    xx=dx[1]*norm_x[1]+dx[2]*norm_y[1];
    yy=dx[1]*norm_x[2]+dx[2]*norm_y[2];
    zz=dx[1]*norm_x[3]+dx[2]*norm_y[3];
    dx[1]=xx;
    dx[2]=yy;
    dx[3]=zz;
    dx[4]=dx[5]=dx[6]=0;
    origin[1]+=xx;
    origin[2]+=yy;
    origin[3]+=zz;
```

```
        path_distance+=sqrt(xx*xx+yy*yy+zz*zz);
        get_grad_dat(ref_joint, path, steps, dx, origin);
        if (getbutton(MOUSE2) || getbutton(MOUSE3))
            {
            mx=getvaluator(MOUSEX);
            my=getvaluator(MOUSEY);
            if ((1100<=mx) && (mx<=1170) && (my>=20) && (my<=70))
            return(1);
            }
        angle+=d_theta;
        }
    d_theta=turn-angle;
    xx=r*(1-cos(d_theta));
    zz=sqrt(linear_v*linear_v-xx*xx)-r*sin(d_theta);
    xx=cos(turn);
    yy=sin(turn);
    x1=r*xx-zz*yy;
    y1=r*yy+zz*xx;
    dx[1]=x1-x;
    dx[2]=y1-y;
    xx=dx[1]*norm_x[1]+dx[2]*norm_y[1];
    yy=dx[1]*norm_x[2]+dx[2]*norm_y[2];
    zz=dx[1]*norm_x[3]+dx[2]*norm_y[3];
    dx[1]=xx;
    dx[2]=yy;
    dx[3]=zz;
    dx[4]=dx[5]=dx[6]=0;
    origin[1]+=xx;
    origin[2]+=yy;
    origin[3]+=zz;
    path_distance+=sqrt(xx*xx+yy*yy+zz*zz);
    steps++;
    get_grad_dat(ref_joint, path, steps, dx, origin);
    return(0);
}

/********* Kinematics inverse and criteria calculation for the 3-D map ********/
get_grad_dat(ref_joint, path, steps, vels, origin)
int ref_joint, path, steps;
float *vels, *origin;
{
    extern float norm_dat[TSTEPS][TPATHS][3];
    extern int scheme_index;
    int i;

    norm_dat[steps][path][1]=theta_rd[ref_joint]/0.017453293;
    norm_dat[steps][path][0]=path_distance*10;
    if (scheme_index == 1)
        gcost_data(ref_joint, path, steps, vels, origin);
    else if (scheme_index == 2)
        gdet_data(ref_joint, path, steps, vels, origin);
```

```
    else if (scheme_index == 3)
        geig_data(ref_joint, path, steps, vels, origin);
    else if (scheme_index == 4)
        gdet6_data(ref_joint, path, steps, vels, origin);
    else if (scheme_index == 5)
        get_jlim_data(ref_joint, path, steps, vels, origin);
    else if (scheme_index == 6)
        get_jlim_data2(ref_joint, path, steps, vels, origin);
    return;
}

/***************** Calculate the norm of the least-norm solution **************/
gcost_data(ref_joint, path, steps, vels, origin)
int ref_joint, path, steps;
float *vels, *origin;
{
    extern float norm_dat[TSTEPS][TPATHS][3];
    extern float path_distance, theta_rd[MAT_DIM];
    float tp1[MAT_DIM], tp2[MAT_DIM], er[8];
    int i;

    least_norm(theta_rd, vels, tp1);
    new_theta(theta_rd, tp1);
    difference(theta_rd, h_transf, origin, er);
    least_norm(theta_rd, er, tp2);
    new_theta(theta_rd, tp2);
    norm_dat[steps][path][2]=0;
    for (i=1; i<=T_JOINT; i++)
        {
        tp1[i]+=tp2[i];
        norm_dat[steps][path][2]+=tp1[i]*tp1[i];
        }
    norm_dat[steps][path][2]=sqrt(norm_dat[steps][path][2])*1145.9156;
    return;
}

/********************** Calculate the invers of det(JJ^T)  ********************/
gdet_data(ref_joint, path, steps, vels, origin)
int ref_joint, path, steps;
float *vels, *origin;
{
    extern float norm_dat[TSTEPS][TPATHS][3], theta_rd[MAT_DIM], h_transf[5][5];
    extern float path_distance;
    float tp1[MAT_DIM], er[8];
    float jaco[MAT_DIM][MAT_DIM], jsqr[MAT_DIM][MAT_DIM];
    int i;

    jacobian(theta_rd, h_transf, jaco);
    cal_jjt(6, T_JOINT, jaco, jsqr);
    determ(6, jsqr);
    norm_dat[steps][path][2]=1.5e6/sqrt(jsqr[0][0]);
```

```
    least_norm(theta_rd, vels, tp1);
    new_theta(theta_rd, tp1);
    difference(theta_rd, h_transf, origin, er);
    least_norm(theta_rd, er, tp1);
    new_theta(theta_rd, tp1);
    return;
}

/******************** Calculate the invers of min. SV (1) ********************/
geig_data(ref_joint, path, steps, vels, origin)
int ref_joint, path, steps;
float *vels, *origin;
{
    extern float norm_dat[TSTEPS][TPATHS][3], theta_rd[MAT_DIM], h_transf[5][5];
    extern float path_distance;
    float tp1[MAT_DIM], er[8];
    float jsqr[MAT_DIM][MAT_DIM], jaco[MAT_DIM][MAT_DIM];
    float eigen[MAT_DIM];
    int i, j;

    jacobian(theta_rd, h_transf, jaco);
    cal_jjt(6, T_JOINT, jaco, jsqr);
    svd(6, jsqr, eigen);
    eigen[0]=eigen[1];
    for (i=2; i<=6; i++)
        {
        if (eigen[0]>eigen[i])
            eigen[0]=eigen[i];
        }
    norm_dat[steps][path][2]=20/sqrt(eigen[0]);
    least_norm(theta_rd, vels, tp1);
    new_theta(theta_rd, tp1);
    difference(theta_rd, h_transf, origin, er);
    least_norm(theta_rd, er, tp1);
    new_theta(theta_rd, tp1);
    return;
}

/******************** Calculate the invers of min. SV (2) ********************/
gdet6_data(ref_joint, path, steps, vels, origin)
int ref_joint, path, steps;
float *vels, *origin;
{
    extern float norm_dat[TSTEPS][TPATHS][3], theta_rd[MAT_DIM];
    extern float path_distance;
    static float sigma6;
    float tp1[MAT_DIM], er[8], jsqr[MAT_DIM][MAT_DIM], h_transf[5][5];
    float jaco[MAT_DIM][MAT_DIM], x, x0, x1, y0, y1, kk;
    int i, j;

    jacobian(theta_rd, h_transf, jaco);
```

```
    cal_jjt(6, T_JOINT, jaco, jsqr);
    for (i=1; i<=6; i++)
        tp1[i]=jsqr[i][i];
    x0=(-0.1);
    for (i=1; i<=6; i++)
        jsqr[i][i]=tp1[i]-x0;
    determ(6, jsqr);
    y0=jsqr[0][0];
    if (steps==1)
        sigma6=0.1;
    x1=sigma6;
    for (i=1; i<=6; i++)
        jsqr[i][i]=tp1[i]-x1;
    determ(6, jsqr);
    y1=jsqr[0][0];
    kk=(y1-y0)/(x1-x0);
    x=x1-y1/kk;
    while (fabs(y1/kk)>1e-6)
        {
        x0=x1;
        x1=x;
        y0=y1;
        for (i=1; i<=6; i++)
            jsqr[i][i]=tp1[i]-x1;
        determ(6, jsqr);
        y1=jsqr[0][0];
        kk=(y1-y0)/(x1-x0);
        x=x1-y1/kk;
        }
    sigma6=x;
    norm_dat[steps][path][2]=20/sqrt(sigma6);
    least_norm(theta_rd, vels, tp1);
    new_theta(theta_rd, tp1);
    difference(theta_rd, h_transf, origin, er);
    least_norm(theta_rd, er, tp1);
    new_theta(theta_rd, tp1);
    return;
}

/*************** Calculate the criterion of joint limits (1) ****************/
get_jlim_data(ref_joint, path, steps, vels, origin)
int ref_joint, path, steps;
float *vels, *origin;
{
    extern float norm_dat[TSTEPS][TPATHS][3], theta_rd[MAT_DIM], h_transf[5][5];
    extern float path_distance;
    float tp1[MAT_DIM], er[8], j_lim_h;
    int i;

    get_j_lim(&j_lim_h, theta_rd);
    norm_dat[steps][path][2]= j_lim_h;
```

```
    least_norm(theta_rd, vels, tp1);
    new_theta(theta_rd, tp1);
    difference(theta_rd, h_transf, origin, er);
    least_norm(theta_rd, er, tp1);
    new_theta(theta_rd, tp1);
    return;
}

/*************** Calculate the criterion of joint limits (2) ****************/
get_jlim_data2(ref_joint, path, steps, vels, origin)
int ref_joint, path, steps;
float *vels, *origin;
{
    extern float norm_dat[TSTEPS][TPATHS][3], theta_rd[MAT_DIM], h_transf[5][5];
    extern float path_distance;
    float tp1[MAT_DIM], er[8], j_lim_h;
    int i;

    get_j_lim2(&j_lim_h, theta_rd);
    norm_dat[steps][path][2]= j_lim_h * 10;
    least_norm(theta_rd, vels, tp1);
    new_theta(theta_rd, tp1);
    difference(theta_rd, h_transf, origin, er);
    least_norm(theta_rd, er, tp1);
    new_theta(theta_rd, tp1);
    return;
}


/******************* Move the manipulator to its home position ***************/
home_robot()
{
    extern float theta_rd[MAT_DIM], trajectory[TSTEPS][3];
    extern float robot_home[MAT_DIM], force_dtheta[MAT_DIM];
    extern int robot_index, goal_steps, state, e_traj_count, control_robot;
    extern int  force_switch;
    extern struct  obtctl;
    int i, tp;
    float home[MAT_DIM];

    if (control_robot == 0)
        {
        for (i=0; i<=T_JOINT; i++)
            theta_rd[i] = robot_home[i];
        }
    else if ((control_robot == 1) || (robot_index == 1))
        {
        tp = force_switch;
        force_switch = 0;
        for (i=0; i<=T_JOINT; i++)
            home[i] = robot_home[i];
```

```
        i = goal_jnt(T_JOINT, home);
        force_switch = tp;
        }
    for (i=1; i<=T_JOINT; i++)
        force_dtheta[i] = 0;

    if (goal_steps==0)
        state=0;
    e_traj_count=0;
    init_obt_struct(robot_index);
    trajectory[0][0]=h_transf[1][4];
    trajectory[0][1]=h_transf[2][4];
    trajectory[0][2]=h_transf[3][4];
    fm_print();
    draw_robot(theta_rd);
}
```

```
/*                    Source code file "tools.c"                    */

#include <stdio.h>
#include <math.h>
#include <errno.h>
#include <fcntl.h>
#include <termio.h>
#include <gl/device.h>
#include <signal.h>
#include <poll.h>
#include <stropts.h>
#include <sys/time.h>

#define TSTEPS 10000          /* The max. steps of trajectory          */
#define MAT_DIM 10            /* The max. dimension of matrix          */
#define T_JOINT 7             /* The total joint of the manipulator    */
extern struct {int index; int sign; float obt_v;} obtctl;
extern struct gt {char st_con[2]; float goal_position[MAT_DIM];};
extern struct gt g_state[TSTEPS];

struct termio port;           /* The structure used by control commu.  */

/****************** Calculating the square root of the det(d) ******************/
determ(n,d)
float d[MAT_DIM][MAT_DIM];
int n;
{
    int i, j, k, l;
    float t[MAT_DIM], tp[MAT_DIM][MAT_DIM];
    double temp, det;

    for (i=1; i<=n; i++)
        for (j=1; j<=n; j++)
            tp[i][j]=d[i][j];
    det=1;
    for (i=1; i<n; i++)
        {
        l=i;
        temp=tp[i][i];
        for (j=i+1; j<=n; j++)
            {
            if (fabs(tp[j][i])>fabs(temp))
                {
                l=j;
                temp=tp[j][i];
                }
            continue;
            }
        if ((l==i) && (temp==0))
            {
```

```
            d[0][0]=0;
            return;
            }
        if (l!=i)
            {
            det=(-det);
            for (k=i; k<=n; k++)
                {
                t[0]=tp[i][k];
                tp[i][k]=tp[l][k];
                tp[l][k]=t[0];
                }
            }
        for (j=i+1; j<=n; j++)
            t[j]=tp[i][j]/temp;
        for (j=i+1; j<=n; j++)
            {
            for (k=i+1; k<=n; k++)
                tp[j][k]-=t[k]*tp[j][i];
            }
        det *= temp;
        }
    det *= tp[n][n];
    d[0][0]=det;
    return;
}

/************************** Solving n equitions ****************************/
equit(n,d,x)
float d[MAT_DIM][MAT_DIM], *x;
int n;
{
    int i, j, k, l;
    float temp, s, t[MAT_DIM], tp[MAT_DIM][MAT_DIM];

    for (i=1; i<=n; i++)
        for (j=0; j<=n; j++)
            tp[i][j]=d[i][j];

    for (i=1; i<n; i++)
        {
        l=i;
        temp=tp[i][i];
        for (j=i+1; j<=n; j++)
            {
            if (fabs(tp[j][i])>fabs(temp))
                {
                l=j;
                temp=tp[j][i];
                }
            continue;
```

```
            }
        if ((l==i) && (temp==0))
            {
            printf("Singularity!\n");
            return(-1);
            }
        if (l!=i)
            {
            for (k=i; k<=n; k++)
                {
                temp=tp[i][k];
                tp[i][k]=tp[l][k];
                tp[l][k]=temp;
                }
            temp=tp[i][0];
            tp[i][0]=tp[l][0];
            tp[l][0]=temp;
            }
        for (j=i+1; j<=n; j++)
            t[j]=tp[i][j]/tp[i][i];
        t[0]=tp[i][0]/tp[i][i];
        for (j=i+1; j<=n; j++)
            {
            for (k=i+1; k<=n; k++)
                tp[j][k]-=t[k]*tp[j][i];
            tp[j][0]-=t[0]*tp[j][i];
            }
        }
    if ((tp[n][n]==0) && (tp[n][0]!=0))
        {
        printf("\ntp(n,n)=%f \nNo solution for the equitions!\n",tp[n][n]);
        return(-1);
        }
    x[n]=tp[n][0]/tp[n][n];
    k=n-1;
    for (i=k; i>=1; i--)
        {
        temp=tp[i][0];
        for (j=n; j>=i+1; j--)
            temp-=tp[i][j]*x[j];
        x[i]=temp/tp[i][i];
        }
    return(0);
}

/************************** Calculating J*J^T ***************************/
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;
}

/************************ Least norm solution ******************************/

least_norm(theta_rd, dx, dtheta)
float *theta_rd, *dx, *dtheta;
{
    int i;
    double temp1, temp2;
    float tp, dtheta_h[MAT_DIM], dtheta_p[MAT_DIM], jaco[MAT_DIM][MAT_DIM];
    float h_transf[5][5];

    jacobian(theta_rd,h_transf,jaco);
    for (i=1; i<=6; i++)
        jaco[i][0] = dx[i];
    if (-1==equit(6, jaco, dtheta_p))
        {
        ringbell();
        return(-1);
        }
    dtheta_p[7]=0;
    tp=0.0017453293;
    for (i=1; i<=6; i++)
        jaco[i][0]=(-jaco[i][7])*tp;
    if (-1==equit(6,jaco, dtheta_h))
        {
        ringbell();
        return(-1);
        }
    dtheta_h[7]=tp;
    temp1=temp2=0;
    for (i=1; i<=T_JOINT; i++)
        {
        temp1+=(dtheta_p[i]*dtheta_h[i]);
        temp2+=(dtheta_h[i]*dtheta_h[i]);
        }
    temp1/=temp2;
    sighold(SIGPOLL);
    for (i=1; i<=T_JOINT; i++)
        {
        dtheta[i] = dtheta_p[i] - dtheta_h[i]*temp1;
        }
```

```
    sigrelse(SIGPOLL);
    return(0);
}


/***************************** Orbit motion ********************************/

o_motion(theta_rd, dtheta)
float *theta_rd, *dtheta;
{
    extern struct obtctl;
    int i, j, k, index1, index2;
    float temp1, temp2;
    float dtheta_h[MAT_DIM], jaco[MAT_DIM][MAT_DIM], h_transf[5][5];

    index1=obtctl.index;
    jacobian(theta_rd, h_transf, jaco);
    for (i=1; i<=6; i++)
        {
        jaco[i][0] = -jaco[i][index1]*obtctl.obt_v;
        jaco[i][index1] = jaco[i][7];
        }

    if(-1==equit(6, jaco, dtheta))
        return(-1);
    dtheta[7]=dtheta[index1];
    dtheta[index1]=obtctl.obt_v;
    temp1 = 0;
    for (i=1; i<=T_JOINT; i++)
        {
        temp2=fabs(dtheta[i]);
        if (temp2>temp1)
            {
            temp1=temp2;
            index2=i;
            }
        }
    obtctl.index=index2;
    temp2 = fabs(obtctl.obt_v);
    if (dtheta[index2]<0)
        obtctl.obt_v = -temp2;
    else
        obtctl.obt_v = temp2;
    temp1/=temp2;
    for (i=1; i<=T_JOINT; i++)
        {
        dtheta[i]/=temp1*obtctl.sign;
        }
    return(0);
}
```

```
/*********** This function define the rotation direction of the elbow *********/
elbow_direction(theta_rd)
float *theta_rd;
{
    extern struct obtctl;
    extern int robot_index;
    float elbow_dir1[4], elbow_pos1[4], elbow_dir2[4], elbow_pos2[4];
    float tp_theta[MAT_DIM], jaco[MAT_DIM][MAT_DIM], h_transf[5][5];
    int i, j;

    if (robot_index == 2)
        return;

    k2107_elbow_pos(theta_rd, elbow_dir1, elbow_pos1);
    obtctl.sign = 1;
    o_motion(theta_rd, tp_theta);
    new_theta(tp_theta, theta_rd);
    k2107_elbow_pos(tp_theta, elbow_dir2, elbow_pos2);
    elbow_pos2[1] -= elbow_pos1[1];
    elbow_pos2[2] -= elbow_pos1[2];
    elbow_pos2[3] -= elbow_pos1[3];
    elbow_pos2[0] = elbow_dir1[1]*elbow_pos2[1] + elbow_dir1[2]*elbow_pos2[2];
    elbow_pos2[0] += elbow_dir1[3]*elbow_pos2[3];
    if (elbow_pos2[0] < 0 )
        obtctl.obt_v = -obtctl.obt_v;
}

/************************ SVD (Using Given's rotation) ************************/
svd(n, a, eigenvalues)
int n;
float a[MAT_DIM][MAT_DIM], *eigenvalues;
{
    int i, j, k;
    float cs, sn, p, q, v, tp1, tp2, tp3;

    tp3=1;
    while (tp3>1e-6)
        {
        tp3=0;
        for (i=1; i<n; i++)
            {
            for (j=i+1; j<=n; j++)
                {
                p=tp1=tp2=0;
                for (k=1; k<=n; k++)
                    {
                    p+=a[k][i]*a[k][j];
                    tp1+=a[k][i]*a[k][i];
                    tp2+=a[k][j]*a[k][j];
                    }
                q=tp1-tp2;
```

```
                v=sqrt(4*p*p+q*q);
                if (q>=0)
                    {
                    cs=sqrt((v+q)/2/v);
                    sn=p/(v*cs);
                    }
                else
                    {
                    if (p>=0)
                        sn= sqrt((v-q)/2/v);
                    else
                        sn= -sqrt((v-q)/2/v);
                    cs=p/v/sn;
                    }
                v=p*p/(tp1*tp2);
                if (v > 1e-6)
                    for (k=1; k<=n; k++)
                        {
                        tp1=a[k][i]*cs+a[k][j]*sn;
                        a[k][j]= -a[k][i]*sn+a[k][j]*cs;
                        a[k][i]=tp1;
                        }
                if (tp3<v)
                    {
                    tp3=v;
                    }
                }
            }
        }
    for (i=1; i<=n; i++)
        {
        tp1=0;
        for (j=1; j<=n; j++)
            tp1+=a[j][i]*a[j][i];
        eigenvalues[i]=sqrt(tp1);
        for (j=1; j<=n; j++)
            a[j][i] = a[j][i]/eigenvalues[i];
        }
    return;
}

/***************************** Increasing theta ******************************/
new_theta(theta, dtheta)
float *theta, *dtheta;
{
    theta[1] += dtheta[1];
    theta[2] += dtheta[2];
    theta[3] += dtheta[3];
    theta[4] += dtheta[4];
    theta[5] += dtheta[5];
    theta[6] += dtheta[6];
```

```
    theta[7] += dtheta[7];
    return;
}

/************************ Roll Pitch Yaw Calculation ***************************/
rpy_angles(h_transf)
float h_transf[5][5];
{
    float tp;
    int i, j;

    if (fabs(h_transf[3][1]) <= 1)
        tp = h_transf[3][1];
    else if (h_transf[3][1] > 0)
        tp = 1;
    else
        tp = -1;
    h_transf[0][2] = asin(-tp);
    if ((1 - fabs(cos(h_transf[0][2]))) > 0 )
        {
        h_transf[0][1]=atan2(h_transf[2][1], h_transf[1][1]);
        h_transf[0][3]=atan2(h_transf[3][2], h_transf[3][3]);
        }
}

/************************ Recording the configuration ****************************/
rec_position(index, state, steps, theta_rd, h_transf)
int index, state, steps;
float *theta_rd, h_transf[5][5];
{
    extern struct g_state;

    g_state[steps].goal_position[0]=theta_rd[0];;
    if (state>=8)
        {
        g_state[steps].goal_position[1]=theta_rd[1];
        g_state[steps].goal_position[2]=theta_rd[2];
        g_state[steps].goal_position[3]=theta_rd[3];
        g_state[steps].goal_position[4]=theta_rd[4];
        g_state[steps].goal_position[5]=theta_rd[5];
        g_state[steps].goal_position[6]=theta_rd[6];
        g_state[steps].goal_position[7]=theta_rd[7];
        g_state[steps].st_con[1]=0;
        }
    else if (state<=7)
        {
        rpy_angles(h_transf);
        g_state[steps].goal_position[1]=h_transf[1][4];
        g_state[steps].goal_position[2]=h_transf[2][4];
        g_state[steps].goal_position[3]=h_transf[3][4];
        g_state[steps].goal_position[4]=h_transf[0][1];
```

```
        g_state[steps].goal_position[5]=h_transf[0][2];
        g_state[steps].goal_position[6]=h_transf[0][3];
        g_state[steps].st_con[1]=index;
        }
    g_state[steps].st_con[0]=state;
    state=0;
    return;
}
/************************* Get position of the end effector ******************/
get_position(origin, h_transf)
float *origin, h_transf[5][5];
{

    rpy_angles(h_transf);
    origin[1]=h_transf[1][4];
    origin[2]=h_transf[2][4];
    origin[3]=h_transf[3][4];
    origin[4]=h_transf[0][1];
    origin[5]=h_transf[0][2];
    origin[6]=h_transf[0][3];
    return;
}

/*****************************************************************************/
/*   Check the joint velosity limit and define the self_motion               */

j_v_limit(dtheta_m, dtheta_h, gain, dtheta)
float *dtheta_m, *dtheta_h, gain, *dtheta;
{
    extern float jv_lim[MAT_DIM];
    float temp1, temp2, th_h[MAT_DIM], k_max, k_min;
    int i, j, w;

    w = j = 0;
    k_max = 1e20;
    k_min = -1e20;
    for (i=1; i<=T_JOINT; i++)
        {
        th_h[i] = gain*dtheta_h[i];
        dtheta[i] = dtheta_m[i] - th_h[i];
        if (fabs(dtheta[i]) > jv_lim[i])
            j = -1;
        if (th_h[i] != 0)
            {
            temp1 = (jv_lim[i] - dtheta_m[i])/th_h[i];
            temp2 = (-jv_lim[i] - dtheta_m[i])/th_h[i];
            }
        if (temp1 > temp2)
            {
            if (temp1 < k_max)
                k_max = temp1;
```

```
            if (temp2 > k_min)
                k_min = temp2;
            }
        else
            {
            if (temp2 < k_max)
                k_max = temp2;
            if (temp1 > k_min)
                k_min = temp1;
            }
        }
    if (j == -1)
        {
        if (k_min <= 0)
            {
            for (i=1; i<=T_JOINT; i++)
                dtheta[i] = dtheta_m[i] + k_min*th_h[i];
            w = 0;
            }
        else
            {
            for (i=1; i<=T_JOINT; i++)
                dtheta[i] = dtheta_m[i];
            w = -1;
            }
        }
    return(w);
}

/************* Position and orientation differance of the end-effector *********/
difference(theta_rd, h_transf, goal, diff)
float *theta_rd, h_transf[5][5], *goal, *diff;
{
    float h[5][5];
    float cs1, sn1, cs2, sn2, sn3, cs3;


    transf(theta_rd, h_transf);
    diff[0] = goal[0] - theta_rd[0];
    diff[1]=goal[1]-h_transf[1][4];
    diff[2]=goal[2]-h_transf[2][4];
    diff[3]=goal[3]-h_transf[3][4];

    cs1 = cos(goal[4]);
    sn1 = sin(goal[4]);
    cs2 = cos(goal[5]);
    sn2 = sin(goal[5]);
    sn3 = sin(goal[6]);
    cs3 = cos(goal[6]);

            /****** If Z-Y-X rotation : **********/
```

```
    h[1][1] = cs1*cs2;
    h[2][1] = sn1*cs2;
    h[3][1] = -sn2;
    h[1][2] = cs1*sn2*sn3 - sn1*cs3;
    h[2][2] = sn1*sn2*sn3 + cs1*cs3;
    h[3][2] = cs2*sn3;
    h[1][3] = cs1*sn2*cs3 + sn1*sn3;
    h[2][3] = sn1*sn2*cs3 - cs1*sn3;
    h[3][3] = cs2*cs3;
    minimum_rot(h_transf, h, &diff[3]);
    return;
}
/*************************************************************************/
minimum_rot(h_transf, h, diff)
float h_transf[][5], h[][5], *diff;
{
    float nx, ny, nz, n_norm, phi;
    int i;

    nx = h_transf[1][3]*h[1][2] + h_transf[2][3]*h[2][2] + h_transf[3][3]*h[3][2];
    nx -= h_transf[1][2]*h[1][3] + h_transf[2][2]*h[2][3] + h_transf[3][2]*h[3][3];
    ny = h_transf[1][1]*h[1][3] + h_transf[2][1]*h[2][3] + h_transf[3][1]*h[3][3];
    ny -= h_transf[1][3]*h[1][1] + h_transf[2][3]*h[2][1] + h_transf[3][3]*h[3][1];
    nz = h_transf[1][2]*h[1][1] + h_transf[2][2]*h[2][1] + h_transf[3][2]*h[3][1];
    nz -= h_transf[1][1]*h[1][2] + h_transf[2][1]*h[2][2] + h_transf[3][1]*h[3][2];
    n_norm = sqrt(nx*nx + ny*ny + nz*nz);
    if (n_norm < 1e-20)
        {
        diff[1] = diff[2] = diff[3] = 0;
        return;
        }
    nx /= n_norm;
    ny /= n_norm;
    nz /= n_norm;
    phi = asin(n_norm/2);

    for (i=1; i<=3; i++)
        {
        diff[i] = (h_transf[i][1]*nx + h_transf[i][2]*ny + h_transf[i][3]*nz)*phi;
        }
}

/******************** Calculating the Jacobian matrix ************************/
jacobian(theta,h_transf,jaco)
float *theta, h_transf[5][5], jaco[MAT_DIM][MAT_DIM];
{
    extern int robot_index;

    if (robot_index == 1)
        k2107_jacobian(theta, h_transf, jaco);
    else if (robot_index == 2)
```

```
        LTM_jacobian(theta, h_transf, jaco);
}

/****************** Calculating the homogeneous transformation ******************/
transf(theta, h_transf)
float *theta, h_transf[5][5];
{
    extern int robot_index;

    if (robot_index == 1)
        k2107_transf(theta, h_transf);
    else if (robot_index == 2)
        LTM_transf(theta, h_transf);
}

/********************** Selecting the robor graphics model ********************/
select_robot(robot_index)
int robot_index;
{
    extern long color_back;

    if ( robot_index == 1 )    /* Robotics Research k2107 selected   */
        get_k2107_parameters();
    else if ( robot_index == 2 )   /* N.A.S.A. LTM selected    */
        get_LTM_parameters();
}

/********************** Save the present robot parameters      **********************/
save_parameters(robot_index)
int robot_index;
{

    if ( robot_index == 1 )    /* Robotics Research k2107 selected   */
        save_k2107_parameters();
    else if ( robot_index == 2 )   /* N.A.S.A. LTM selected    */
        save_LTM_parameters();
}

/******************** Initiation of the robots parameters **********************/
init_parameters()
{
    init_k2107_parameters();
    init_LTM_parameters();
}

/*************** Initiation of the robots orbit motion parameters *************/
init_obt_struct(robot_index)
int robot_index;
{

    if (robot_index == 1)
```

```
        init_k2107_obt_struct();
    else if (robot_index == 2)
        init_k2107_obt_struct();
}

/****************** Calculating the joint limit criterion (1) ******************/
get_j_lim(j_lim_h, theta_rd)
float *j_lim_h, *theta_rd;
{
    extern float dth_lim[MAT_DIM], th_lim1[MAT_DIM], th_lim2[MAT_DIM];
    float temp;
    int i, j;

    *j_lim_h = 0;
    for (i=1; i<=T_JOINT; i++)
        {
        temp = (th_lim2[i] - theta_rd[i])*(theta_rd[i] - th_lim1[i]);
        if (temp <= 0)
            {
            *j_lim_h = 10000;
            break;
            }
        else
            *j_lim_h += dth_lim[i]/temp;
        }
}

/****************** Calculating the joint limit criterion (2) ******************/
get_j_lim2(j_lim_h, theta_rd)
float *j_lim_h, *theta_rd;
{
    extern float dth_lim[MAT_DIM];
    float temp;
    int i, j;

    *j_lim_h = 0;
    for (i=1; i<=T_JOINT; i++)
        {
        temp = (th_lim1[i] + th_lim2[i])/2;
        temp = (theta_rd[i] - temp)/(temp - th_lim2[i]);
        *j_lim_h += temp*temp;
        }
}


/******************** Robot jogging communication function ********************/
robot_jog()
{
    extern float theta_rd[MAT_DIM];
    extern int ser_port2, control_robot, first_signal, first_comm_num;
    extern float jv_lim[MAT_DIM];
```

```
extern int joint_out[MAT_DIM];
extern char r_feed_back[200], r_command[200];
float joint_out_vel[MAT_DIM], temp;
int i, j, k, jv_limit_sign;
char buffer[200], prefix;
static int reading = 0;

if (ser_port2 < 0)
    return;
sigignore(SIGPOLL);
i = read(ser_port2, r_feed_back, 190);
if (i>=4)
    {
    jv_limit_sign = 0;
    for (i=1; i<=T_JOINT; i++)
        {
        temp = (float) joint_out[i];
        joint_out_vel[i] = (theta_rd[i] - temp/10000)*5;
        if (fabs(joint_out_vel[i])>jv_lim[i])
            jv_limit_sign = i;
        joint_out[i] = (float) (theta_rd[i]*10000);
        }
    if (jv_limit_sign == 0)
        {
        buffer[0] = 'n';
        buffer[1] = 'G';
        buffer[2] = 0x20;
        prefix = 'a';
        for (i=1; i<=T_JOINT; i++)
            {
            j = i*5 - 2;
            buffer[j] = prefix;
            memcpy(&buffer[(++j)], &joint_out[i], 4);
            prefix ++;
            }
        i = j + 4;
        buffer[i]=0x0d;
        i++;
        buffer[i]=0x0a;
        i++;
        buffer[i]= '\0';
        ioctl(ser_port2, TCFLSH, 0);
        write(ser_port2, buffer, i);
        signal(SIGPOLL, robot_jog);
        }
    else                        /* When joint vel. over the limit */
        {
        sigignore(SIGPOLL);
        tpon();
        pop_cursor();
        ringbell();
```

```
            printf("The joint %d velocity over the limit\n", jv_limit_sign);
            for (i=1; i<=T_JOINT; i++)
                printf("Joint %d velocity = %f deg/sec\n",
                        i,joint_out_vel[i]/0.017453293);
            printf("Press 'return' to continue\n");
            fgets(buffer, 10, stdin);
            control_robot = 0;
            tpoff();
            }
        }
    return;
}

/********************** Sending a command to the controller ********************/
send_command(ser_port, command)
int ser_port;
char *command;
{
    char buffer[200];
    int i;

    strcpy(buffer, command);
    i=strlen(buffer);
    i--;
    if (buffer[i]>'0')
        i++;
    buffer[i++]=0x0d;
    buffer[i++]=0x0a;
    buffer[i]='\0';
    write(ser_port, buffer, i);
}

/********************** Receive the feed back *********************************/
feed_back(time_delay, buffer)
long time_delay;
char *buffer;
{
    extern int ser_port2;
    int length;
    long i;

    length=0;
    i=time_delay;
    while(i>=0)
        {
        length=read(ser_port2, buffer, 190);
        if (length>0)
            {
            buffer[length]='\0';
            break;
            }
```

```
        i--;
        }
    return(length);
}

/******************** Initiate the communication port 2 ************************/
/*  The serial communication port 2 is used to connect the silican graphics   */
/* computer to the robot controller.                                          */

init_comm2()
{
    extern int ser_port2;
    int i, tp;

    ser_port2=open("/dev/ttyd2", (O_RDWR | O_NDELAY));
    if ( ser_port2 == -1 )
        {
        printf("\nOpen serial port 2 fai\nThe error is %d\n", errno);
        return;
        }
    tp=ioctl(ser_port2, TCGETA, &port);
    port.c_iflag = 0;
    port.c_oflag = 0;
    port.c_cflag = B9600 | CS8 | CREAD | PARENB | HUPCL | CLOCAL;
    port.c_lflag = ICANON;
    port.c_line = 0;
    for (i=0; i<=7; i++)
        port.c_cc[i] = 0;
    port.c_cc[4] = 0;
    port.c_cc[5] = 0;
    tp=ioctl(ser_port2, TCSETA, &port);
    if ( tp == -1 )
        {
        printf("Serial port 2: The TCSETA command errno is %d\n", errno);
        return;
        }
    ioctl(ser_port2, I_SETSIG, S_INPUT);
    sigignore(POLLIN);
}

/********************** Initiate the communication port 3 **********************/
/*  The serial communication port 3 is used for connecting the silican        */
/* graphics computer to the force senser.                                     */

init_comm3()
{
    extern int ser_port3;
    int i, tp;

    ser_port3=open("/dev/ttyd3", (O_RDWR | O_NDELAY));
    if ( ser_port3 == -1 )
```

```
        {
        printf("Open serial port 3 fail\nThe error is %d\n", errno);
        return;
        }
    tp=ioctl(ser_port3, TCGETA, &port);
    port.c_iflag = 0 ;
    port.c_oflag = 0;
    port.c_cflag = B9600 | CS8 | CREAD | HUPCL | CLOCAL;
    port.c_lflag = 0 ;
    port.c_line = 0;
    for (i=0; i<=7; i++)
        port.c_cc[i] = 0;
    port.c_cc[4] = 74;
    port.c_cc[5] = 1;
    tp=ioctl(ser_port3, TCSETA, &port);
    if ( tp == -1 )
        {
        printf("Serial port 3: The TCSETA command errno is %d\n", errno);
        return;
        }
}

/*************** Initiate the communication port 4 ***************************/
/*  The serial communication port 4 is used for connecting the silican       */
/* graphics computer to the ultrasonic senser.                                */

init_comm4()
{
    extern int ser_port4;
    int i, tp;

    ser_port4=open("/dev/ttyd4", (O_RDWR | O_NDELAY));
    if ( ser_port4 == -1 )
        {
        printf("Open serial port 4 fail\nThe error is %d\n", errno);
        return;
        }
    tp=ioctl(ser_port4, TCGETA, &port);
    port.c_iflag = 0 ;
    port.c_oflag = 0;
    port.c_cflag = B9600 | CS8 | CREAD | HUPCL | CLOCAL;
    port.c_lflag = ICANON ;
    port.c_line = 0;
    for (i=0; i<=7; i++)
        port.c_cc[i] = 0;
    tp=ioctl(ser_port4, TCSETA, &port);
    if ( tp == -1 )
        {
        printf("Serial port 4: The TCSETA command errno is %d\n", errno);
        return;
        }
```

```
}

/******************** Read ult_sensor and ask the new data **********************/
ult_sensor(buffer)
char *buffer;
{
    extern float ult_dtheta[MAT_DIM], theta_rd[MAT_DIM], obst_dis;
    long echo_time;

    echo_time = strtol (buffer, &buffer[4], 16);
    obst_dis = 0.038976378*((float) echo_time);
    if (obst_dis < 40)
        {
        elbow_direction(theta_rd);
        obtctl.sign = 1;
        o_motion(theta_rd, ult_dtheta);
        }
    else
        ult_dtheta[1] = ult_dtheta[2] = ult_dtheta[3] = ult_dtheta[4]
        = ult_dtheta[5] = ult_dtheta[6] = ult_dtheta[7] = 0;
}


/******************** Read force sensor and ask the new data ******************/
force_sensor(force_buffer)
unsigned char *force_buffer;
{
    extern int ser_port4, read_force, config_change, state, teach_orbit;
    extern float force_data[MAT_DIM], kf[8];
    extern float force_dtheta[MAT_DIM], theta_rd[MAT_DIM];
    int i, j;
    float h_transf[5][5];
    long delay;
    char tp[10];
    float temp1, temp2, temp3, dtheta[MAT_DIM], vels[MAT_DIM];


    transf(theta_rd, h_transf);
    for (i=1; i<=6; i++)
        {
        j = 2 + (i - 1)*11;
        strncpy(tp, &force_buffer[j], 7);
        tp[7] = 0;
        force_data[i] = atof(tp);
        }
    for (i=1; i<=3; i++)
        {
        j = i + 3;
        vels[i] = - kf[1]*force_data[1]*h_transf[i][1]
                  + kf[2]*force_data[2]*h_transf[i][2]
                  + kf[3]*force_data[3]*h_transf[i][3];
```

```
        vels[j] = - kf[4]*force_data[4]*h_transf[i][1]
                  + kf[5]*force_data[5]*h_transf[i][2]
                  + kf[6]*force_data[6]*h_transf[i][3];

        }
    least_norm(theta_rd, vels, force_dtheta);
    read_force = 0;
    teach_orbit = 0;
    return;
}

/************************ Port 3 & 4 manager function ***************************/

poll_manager(control, read_index)
int control, *read_index;
{
    extern int ult_switch, ser_port3, ser_port4;
    extern char force_buffer[200];
    unsigned char buffer[200];
    static struct pollfd poll_message[2];
    int i;


    if (control == 1)
        {
        poll_message[0].fd = ser_port3;
        poll_message[1].fd = ser_port4;
        poll_message[0].events = POLLIN;
        poll_message[1].events = POLLIN;
        }

    else if (control == 2)
        {
        if ((poll_message[0].fd <=0) || (poll_message[1].fd <=0))
            return;
        i = poll(poll_message, 2, 0);
        if (i < 0)
            perror("poll failed");
        else if (i>0)
            {
            read_index[0] = read_index[1] = 0;
            if (poll_message[0].revents == POLLIN)
                {
                sighold(SIGPOLL);
                i = read(poll_message[0].fd, force_buffer, 75);
                sigrelse(SIGPOLL);
                force_buffer[i] = 0;
                if (i>=74)
                    {
                    force_sensor(force_buffer);
                    read_index[0] = 1;
```

```
                    }
                else
                    {
                    read_index[0] = -1;
                    }
                sighold(SIGPOLL);
                ioctl(poll_message[0].fd, TCFLSH, 0);
                send_command(poll_message[0].fd, "DP S");
                sigrelse(SIGPOLL);
                }
            if (poll_message[1].revents == POLLIN)
                {
                sighold(SIGPOLL);
                i = read(poll_message[1].fd, buffer, 6);
                sigrelse(SIGPOLL);
                if (i >= 4)
                    {
                    ult_sensor(buffer);
                    read_index[1] = 1;
                    }
                else
                    {
                    read_index[1] = -1;
                    }
                buffer[0] = 0x41;
                sighold(SIGPOLL);
                write(poll_message[1].fd, buffer, 1);
                sigrelse(SIGPOLL);
                }
            }
        }
    return;
}

/***************** Place the cursor at the position x=800, y=250 **************/
pop_cursor()
{

    setvaluator(MOUSEX,800,0,1280);
    setvaluator(MOUSEY,250,0,1028);
}
```

```
/*                    Source code file "init_graph.c"                    */

#include <gl/gl.h>
#include <stdio.h>
#include <fmclient.h>
#include <gl/device.h>
#include <math.h>
#include <signal.h>
#include <poll.h>
#include <stropts.h>

#define TSTEPS 10000
#define TPATHS 40
#define MAT_DIM 10
#define T_JOINT 7

extern struct {int index; int sign; float obt_v;} obtctl;

/********************** The first window lightning data **********************/

float left_light[]=
    {LCOLOR, 1.0, 1.0, 1.0,
    POSITION, 0.433, 0.5, 0.75, 0.0,
    LMNULL};

float right_light[]=
    {LCOLOR, 1.0, 1.0, 1.0,
    POSITION, -0.415, 0.433, 0.80, 0.0,
    LMNULL};

float shiny_material[]=
    {SPECULAR, 0.9, 1.0, 1.0,
    DIFFUSE, 0.4, 0.43, 0.43,
/*  {SPECULAR, 0.5, 0.6, 0.6,
    DIFFUSE, 0.25, 0.30, 0.30, */
    SHININESS, 20.0,
    LMNULL};

float shiny_material2[]=
    {SPECULAR, 0.8, 0.8, 0.8,
    DIFFUSE, 0.4, 0.43, 0.43,
    SHININESS, 20.0,
    LMNULL};

float shiny_material3[]=
    {SPECULAR, 0.9, 0.8, 0.3,
    DIFFUSE, 0.3, 0.25, 0.1,
    SHININESS, 10.0,
    LMNULL};
```

```
float normal_material1[]=
    {SPECULAR, 0.2, 0.2, 0.2,
    DIFFUSE, 0.8, 0.8, 0.8,
    SHININESS, 10.0,
    LMNULL};

/********************** The second window lightning data **********************/

float view_point2[]= {0.0, 300.0, 0.0, -1500.0, -80.0, 0.0, 230.0};
float g_mat1[4][4], g_mat2[4][4];

float light1[]=
    {LCOLOR, 1.0, 1.0, 1.0,
    POSITION, 0.0, 0.0, -1.0, 0.0,
    LMNULL};

float light2[]=
    {LCOLOR, 1.0, 1.0, 1.0,
    POSITION, -1.0, 0.0, 0.0, 0.0,
    LMNULL};

float light3[]=
    {LCOLOR, 1.0, 1.0, 1.0,
    POSITION, 0.0, -1.0, 0.0, 0.0,
    LMNULL};

float light4[]=
    {LCOLOR, 1.0, 1.0, 1.0,
    POSITION, 0.0, 1.0, 0.0, 0.0,
    LMNULL};

float normal_material[]=
    {SPECULAR, 0.5, 0.5, 0.5,
    DIFFUSE, 0.6, 0.6, 0.6,
    SHININESS, 30.0,
    AMBIENT, 0.5, 0.5, 0.5,
    LMNULL};

/******************* The first window lightning define functions **************/
def_light()
{
    lmdef(DEFMATERIAL, 1, 14, shiny_material);
    lmdef(DEFMATERIAL, 2, 15, shiny_material2);
    lmdef(DEFMATERIAL, 3, 15, shiny_material3);
    lmdef(DEFMATERIAL, 4, 16, normal_material1);
    lmdef(DEFLIGHT, 1, 11, right_light);
    lmdef(DEFLIGHT, 2, 10, left_light);
    lmdef(DEFLMODEL, 1, 0, NULL);
}
use_light()
{
```

```
    lmbind(LIGHT0, 1);
    lmbind(LIGHT1, 2);
    lmbind(LMODEL, 1);
}

/****************** The second window lightning define functions ***************/
def_light2()
{
    lmdef(DEFMATERIAL, 5, 11, normal_material);
    lmdef(DEFLIGHT, 4, 6, light1);
    lmdef(DEFLIGHT, 5, 6, light2);
    lmdef(DEFLIGHT, 6, 6, light3);
    lmdef(DEFLIGHT, 7, 6, light4);
    lmdef(DEFLMODEL, 2, 0, NULL);
}
use_light2()
{
    lmbind(MATERIAL, 5);
    lmbind(LIGHT3, 4);
    lmbind(LIGHT4, 5);
    lmbind(LIGHT5, 6);
    lmbind(LIGHT6, 7);
    lmbind(LMODEL, 2);
}

/******************************** Identity matrix ******************************/
float id_mat[]=
    {1.0,0.0,0.0,0.0,
    0.0,1.0,0.0,0.0,
    0.0,0.0,1.0,0.0,
    0.0,0.0,0.0,1.0};

/********************* Initiate the main graphic window ************************/


init_rob_win()
{
    extern long rob_win, sub_win, *font_big, color_button, color_bt_char;
    extern long color_back, color_title, color_frame;
    extern int pic_delay;
    extern float view_point[MAT_DIM], theta_rd[MAT_DIM], view_mat[4][4];

    int i,j,k, btn1, btn2, btn3, btn4, btn5, btn6, btn7, btn8;
    char temp[20];

    btn1=1020;
    btn2=900;
    btn3=1070;
    btn4=950;
    btn5=1100;
    btn6=1170;
```

```
btn7=1200;
btn8=1270;
temp[0]=74;
prefposition(0,1280, 0,1024);
foreground();
rob_win=winopen("Robot");
winset(rob_win);
doublebuffer();
swapinterval(pic_delay);
linesmooth(TRUE);
RGBmode();
gconfig();
screenspace();
zbuffer(FALSE);
for (i=1; i<=2; i++)
    {
    czclear(color_frame, 0x7fffff);
    cpack(color_button);
    for (j=1; j<=11; j++)
        {
        rectfi(btn1, btn2, btn3, btn4);
        rectfi(btn5, btn2, btn6, btn4);
        rectfi(btn7, btn2, btn8, btn4);
        btn2 -= 80;
        btn4 -= 80;
        }
    btn2=900;
    btn4=950;
    cpack(color_bt_char);
    for (j=0; j<=6; j++)
        {
        k=923-j*80;
        cmov2i(1036, k);
        temp[1]=j+49;
        temp[2]=0;
        charstr(temp);
        }
    cmov2i(1036,363);
    charstr("Gp");
    cmov2i(1030,283);
    charstr("J-V");
    cmov2i(1023,203);
    charstr("U_SEN");
    cmov2i(1023,123);
    charstr("F_SEN");
    cmov2i(1134,923);
    charstr("X");
    cmov2i(1134,843);
    charstr("Y");
    cmov2i(1134,763);
    charstr("Z");
```

```
cmov2i(1120,683);
charstr("ROLL");
cmov2i(1115,603);
charstr("PITCH");
cmov2i(1125,523);
charstr("YAW");
cmov2i(1115,443);
charstr("ORBIT");
cmov2i(1116,363);
charstr("VIEW");
cmov2i(1115,283);
charstr("LIN-V");
cmov2i(1115,203);
charstr("LIN-A");
cmov2i(1116,123);
charstr("HOME");
cmov2i(1210,923);
charstr("REC-CT");
cmov2i(1210,843);
charstr("REC-ST");
cmov2i(1220,763);
charstr("GET");
cmov2i(1210,683);
charstr("STORE");
cmov2i(1210,603);
charstr("CLEAR");
cmov2i(1210,523);
charstr("BEGIN");
cmov2i(1216,443);
charstr("STEP");
cmov2i(1210,363);
charstr("DELETE");
cmov2i(1205,283);
charstr("OPTIONS");
cmov2i(1210,203);
charstr("3D-CAL");
cmov2i(1210,123);
charstr("CT_RBT");
cpack(0xaa88ee);
rectfi(1030, 980, 1165, 1000);
cmov2i(1038,985);
cpack(0x00ffff);
charstr("Coord_Switch");
cpack(0x8f0000);
rectfi(1020, 20, 1070, 70);
cmov2i(1030, 43);
cpack(0x00ffff);
charstr("RUN");
cpack(0x008fff);
rectfi(1100, 20, 1170, 70);
cmov2i(1115, 43);
```

```
        cpack(0x0);
        charstr("PAUSE");
        cpack(0x00ffff);
        rectfi(1200, 20, 1270, 70);
        cmov2i(1225, 43);
        cpack(0x0000ff);
        charstr("END");
        swapbuffers();
        }

    zbuffer(TRUE);
    viewport(10, 1000, 10, 1000);
    mmode(MVIEWING);
    loadmatrix(id_mat);
    perspective(450, 1, 0.01, 5000.0);
    lookat(0.0, 0.0,0.0, 0.0, 0.0, -150.0, 0.0);
    def_light();
    use_light();
    lsetdepth(0x7fffff, 0x0);
    zfunction(ZF_GEQUAL);
    czclear(color_back, 0x0);
    textport(480, 950, 100, 300);
    textcolor(0x88);
    pagecolor(0x33);
    fminit();
    font_big=fmscalefont(fmfindfont("Times-BoldItalic"),25);
    fm_print();
    pushmatrix();
    translate(view_point[1], view_point[2], view_point[3]);
    rot(view_point[4], 'x');
    rot(view_point[5], 'y');
    rot(view_point[6], 'z');
    getmatrix(view_mat);
    popmatrix();
    return;
}
/******************* Print the title to the graphic window *********************/
fm_print()
{
    extern int robot_index;
    extern long *font_big;

    fmsetfont(font_big);
    cpack(color_title);
    cmov(-0.0037, 0.0038, -0.01);
    if (robot_index == 1)
        fmprstr("Graphics Simulation of K-2107 Manipulator");
    else if (robot_index == 2)
        fmprstr("Graphics Simulation of LTM");
    cmov(-0.0037, 0.0035, -0.01);
    fmprstr("University of Tennessee, Knoxville");
```

```
}

/****************** View Jardgement of main graphic window ******************/

float box1[3]={0.0028, 0.0037, -0.01};
float box2[3]={0.0038, 0.0037, -0.01};
float box3[3]={0.0038, 0.0033, -0.01};
float box4[3]={0.0028, 0.0033, -0.01};
float box5[3]={0.0028, 0.0032, -0.01};
float box6[3]={0.0038, 0.0032, -0.01};
float box7[3]={0.0038, 0.0028, -0.01};
float box8[3]={0.0028, 0.0028, -0.01};
float box9[3]={0.0028, 0.0027, -0.01};
float box10[3]={0.0038, 0.0027, -0.01};
float box11[3]={0.0038, 0.0023, -0.01};
float box12[3]={0.0028, 0.0023, -0.01};
float box13[3]={0.0028, 0.0022, -0.01};
float box14[3]={0.0038, 0.0022, -0.01};
float box15[3]={0.0038, 0.0018, -0.01};
float box16[3]={0.0028, 0.0018, -0.01};
float box17[3]={0.0028, 0.0017, -0.01};
float box18[3]={0.0038, 0.0017, -0.01};
float box19[3]={0.0038, 0.0013, -0.01};
float box20[3]={0.0028, 0.0013, -0.01};
float box21[3]={0.0028, 0.0012, -0.01};
float box22[3]={0.0038, 0.0012, -0.01};
float box23[3]={0.0038, 0.0008, -0.01};
float box24[3]={0.0028, 0.0008, -0.01};
float box25[3]={0.0028, 0.0007, -0.01};
float box26[3]={0.0038, 0.0007, -0.01};
float box27[3]={0.0038, 0.0003, -0.01};
float box28[3]={0.0028, 0.0003, -0.01};
float box29[3]={0.0028, 0.0002, -0.01};
float box30[3]={0.0038, 0.0002, -0.01};
float box31[3]={0.0038, -0.0002, -0.01};
float box32[3]={0.0028, -0.0002, -0.01};
view_butn()
{
    cpack(0x8fa08f);
    bgnpolygon();
    v3f(box1); v3f(box2); v3f(box3); v3f(box4);
    endpolygon();
    bgnpolygon();
    v3f(box5); v3f(box6); v3f(box7); v3f(box8);
    endpolygon();
    bgnpolygon();
    v3f(box9); v3f(box10); v3f(box11); v3f(box12);
    endpolygon();
    bgnpolygon();
    v3f(box13); v3f(box14); v3f(box15); v3f(box16);
    endpolygon();
```

```
    bgnpolygon();
    v3f(box17); v3f(box18); v3f(box19); v3f(box20);
    endpolygon();
    bgnpolygon();
    v3f(box21); v3f(box22); v3f(box23); v3f(box24);
    endpolygon();
    bgnpolygon();
    v3f(box25); v3f(box26); v3f(box27); v3f(box28);
    endpolygon();
    bgnpolygon();
    v3f(box29); v3f(box30); v3f(box31); v3f(box32);
    endpolygon();
    cpack(0x0);
    cmov(0.0029,0.00345,-0.01);
    charstr("Left-Right");
    cmov(0.0029,0.00295,-0.01);
    charstr("Down-Up");
    cmov(0.0029,0.00245,-0.01);
    charstr("Near-Far");
    cmov(0.0029,0.00195,-0.01);
    charstr("Rotation 1");
    cmov(0.0029,0.00145,-0.01);
    charstr("Rotation 2");
    cmov(0.0029,0.00095,-0.01);
    charstr("Rotation 3");
    cmov(0.0029,0.00045,-0.01);
    charstr("Initiation");
    cmov(0.0029,-0.00005,-0.01);
    charstr("End View");
    return;
}

/****************** View Jardgement of main graphic window ********************/
change_view()
{
    extern long mx, my;
    extern float view_point[MAT_DIM], init_view_point[MAT_DIM], theta_rd[MAT_DIM];
    int i;

    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))
                    view_point[1]+=1;
                else
```

```
        view_point[1]-=1;
    c_view(view_point);
    }
else if ((841<=my) && (my<=889) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    if (getbutton(MOUSE2))
        view_point[2]+=1;
    else
        view_point[2]-=1;
    c_view(view_point);
    }
else if ((781<=my) && (my<=829) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    if (getbutton(MOUSE2))
        view_point[3]+=1;
    else
        view_point[3]-=1;
    c_view(view_point);
    }
else if ((721<=my) && (my<=769) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    if (getbutton(MOUSE2))
        view_point[4]+=2;
    else
        view_point[4]-=2;
    c_view(view_point);
    }
else if ((661<=my) && (my<=709) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    if (getbutton(MOUSE2))
        view_point[5]+=2;
    else
        view_point[5]-=2;
    c_view(view_point);
    }
else if ((601<=my) && (my<=649) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    if (getbutton(MOUSE2))
        view_point[6]+=2;
    else
        view_point[6]-=2;
    c_view(view_point);
    }
else if ((541<=my) && (my<=589) &&
    (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
```

```
                for (i=1; i<=6; i++)
                    view_point[i]=init_view_point[i];
                c_view(view_point);
                }
            else if ((481<=my) && (my<=529) &&
                (getbutton(MOUSE2) || getbutton(MOUSE3)))
                {
                draw_robot(theta_rd);
                return;
                }
            }
        final_check(theta_rd);
        }
}


c_view(view_point)
float *view_point;
{
    view_butn();
    c_view_cont(view_point);
}

c_view_cont(view_point)
float *view_point;
{
    pushmatrix();
    translate(view_point[1], view_point[2], view_point[3]);
    rot(view_point[4], 'x');
    rot(view_point[5], 'y');
    rot(view_point[6], 'z');
    getmatrix(view_mat);
    popmatrix();
    draw_robot(theta_rd);
}

/******************** Initiate the second graphic window **********************/
init_curv_win()
{
    extern long sub_win, sub_winx, sub_winy, color_back2;
    extern float view_point2[7], g_mat1[4][4], g_mat2[4][4];
    int i,j,k;
    char temp[20];

    prefposition(10,sub_winx, sub_winy,1000);
    foreground();
    noborder();
    sub_win=winopen();
    winset(sub_win);
    winpop();
    doublebuffer();
```

```
    RGBmode();
    gconfig();
    zbuffer(TRUE);
    czclear(color_back2, 0x7fffff);
    mmode(MVIEWING);
    loadmatrix(id_mat);
    perspective(450, 1, 0.01, 5000.0);
    lookat(0.0, 0.0,0.0, 0.0, 0.0, -150.0, 0.0);
    def_light2();
    use_light2();
    scale(0.1, 0.1, 0.1);
    getmatrix(g_mat1);
    translate(view_point2[1], view_point2[2], view_point2[3]);
    rot(view_point2[4], 'x');
    rot(view_point2[5], 'y');
    rot(view_point2[6], 'z');
    getmatrix(g_mat2);
    return;
}

int graph_state=0, step=0, path=0, path_ctl=0, prn_color=0;

/********** Main function of manipulating the second graphic window **********/
grad_plot(total_steps, total_paths)
int total_steps, total_paths;
{
    extern long mx, my, sub_win, sub_winx, sub_winy;
    extern float control_path[TSTEPS][3], theta_rd[MAT_DIM];
    extern int steps1, graph_state, step, path, prn_color, ellip;
    float jaco[MAT_DIM][MAT_DIM], jsqr[MAT_DIM][MAT_DIM], h_transf[5][5];
    float x_scaler, y_scaler, z_scaler;
    int i, j, t_path1, temp, plot_paths, plot_steps;
    static int part_plot;

    cpack(0x0);
    drawaxis();
aaa: mx=getvaluator(MOUSEX);
    my=getvaluator(MOUSEY);
    if (((10<=mx) && (mx<=sub_winx)) && ((sub_winy<=my) && (my<=1000)))
        {
        if (prn_color==0)
            {
            czclear(color_back2, 0x7fffff);
            cpack(0x0);
            }
        else
            {
            czclear(0xffffff, 0x7fffff);
            cpack(0x0);
            }
        linewidth(1);
```

```
drawaxis();
linewidth(3);
if (getbutton(F1KEY))
    steps1=0;
else if (getbutton(F2KEY))
    prn_color=1;
else if (getbutton(F3KEY))
    prn_color=0;
else if (getbutton(F4KEY))
    {
    total_steps=0;
    total_paths=0;
    }
else if (getbutton(F5KEY))
    {
    ellip=1;
    }
else if (getbutton(F6KEY))
    {
    ellip=0;
    }
else if (getbutton(AKEY))
    {
    part_plot = 0;
    }
else if (getbutton(PKEY))
    {
    part_plot = 1;
    }
else if (getbutton(SPACEKEY))
    {
    temp=0;
    while(temp==0)
        if (getbutton(MOUSE2) || getbutton(MOUSE3))
            temp=1;
    mx=getvaluator(MOUSEX);
    my=getvaluator(MOUSEY);
    if (mx-10>1000-my)
        {
        sub_winy=1010-mx;
        sub_winx=mx;
        winposition(10, mx, sub_winy, 1000);
        }
    else
        {
        sub_winx=1010-my;
        sub_winy=my;
        winposition(10, sub_winx, my, 1000);
        }
    ringbell();
    sleep(1);
```

```
    reshapeviewport();
    setvaluator(MOUSEX, sub_winx-1, 0, 1279);
    setvaluator(MOUSEY, sub_winy+1, 0, 1079);
    goto aaa;
    }

else if (getbutton(LKEY) ||    getbutton(RKEY))
    {
    if (getbutton(RKEY))
        view_point2[1]+=100;
    else
        view_point2[1]-=100;
    change_view2(view_point2);
    }
else if (getbutton(UKEY) || getbutton(DKEY))
    {
    if (getbutton(UKEY))
        view_point2[2]+=50;
    else
        view_point2[2]-=50;
    change_view2(view_point2);
    }
else if (getbutton(NKEY) || getbutton(FKEY))
    {
    if (getbutton(NKEY))
        view_point2[3]+=100;
    else
        view_point2[3]-=100;
    change_view2(view_point2);
    }
else if (getbutton(ONEKEY) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    if (getbutton(MOUSE2))
        view_point2[4]+=5;
    else
        view_point2[4]-=5;
    change_view2(view_point2);
    }
else if (getbutton(TWOKEY) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    if (getbutton(MOUSE2))
        view_point2[5]+=5;
    else
        view_point2[5]-=5;
    change_view2(view_point2);
    }
else if (getbutton(THREEKEY) && (getbutton(MOUSE2) || getbutton(MOUSE3)))
    {
    if (getbutton(MOUSE2))
        view_point2[6]+=5;
    else
```

```
        view_point2[6]-=5;
    change_view2(view_point2);
    }
else if (getbutton(IKEY))
    {
    view_point2[1]= 300.0;
    view_point2[2]= 0.0;
    view_point2[3]= -1500.0;
    view_point2[4]= -80.0;
    view_point2[5]= 0;
    view_point2[6]= 230.0;
    change_view2(view_point2);
    }
if (ellip==1)
    {
    jacobian(theta_rd,h_transf,jaco);
    cal_jjt(3, T_JOINT, jaco, jsqr);
    svd(3, jsqr, jsqr[0]);
    x_scaler=4.0*sqrt(jsqr[0][1]);
    y_scaler=4.0*sqrt(jsqr[0][2]);
    z_scaler=4.0*sqrt(jsqr[0][3]);
    for (i=1; i<=3; i++)
        for (j=1; j<=3; j++)
            h_transf[i][j]=jsqr[i][j];
    pushmatrix();
    rpy_angles(h_transf);
    rot( h_transf[0][1]/0.017453293, 'z');
    rot( h_transf[0][2]/0.017453293, 'y');
    rot( h_transf[0][3]/0.017453293, 'x');
    ellipse(x_scaler, y_scaler, z_scaler);
    popmatrix();
    }
 else
    {
    if (getbutton(LEFTARROWKEY))
        {
        path--;
        if (path<0)
            path=total_paths;
        }
    else if (getbutton(RIGHTARROWKEY))
        {
        path++;
        if (path>total_paths)
            path=0;
        }
    else if (getbutton(UPARROWKEY))
        {
        step++;
        if (step>total_steps)
                step=0;
```

```
        }
    else if (getbutton(DOWNARROWKEY))
        {
        step--;
        if (step<0)
            step=total_steps;
        }
    else if (getbutton(F12KEY))
        {
        n_cal(total_steps, total_paths, t_path1);
        ringbell();
        }
    else if (getbutton(MOUSE2) && getbutton(MOUSE3))
        {
        ringbell();
        if (graph_state==3)
            graph_state=0;
        else
            {
            graph_state++;
            }
        }
    }
if (ellip==0)
    {
    if (part_plot == 1)
        {
        plot_paths = path;
        plot_steps = step;
        t_path1 = path;
        }
    else
        {
        plot_paths = total_paths;
        plot_steps = total_steps;
        t_path1 = total_paths + 1;
        }
    if (graph_state==0)
        {
        if (prn_color==0)
            cpack(0x000f8f);
        else
            cpack(0x0);
        draw_grad(plot_steps, plot_paths);
        }
    else if (graph_state==1)
        grad_surf(part_plot, plot_steps, plot_paths, t_path1);
    else if (graph_state==2)
        {
        grad_surf(part_plot, plot_steps, plot_paths, t_path1);
        if (prn_color==0)
```

```
                cpack(0x000f8f);
            else
                cpack(0x0);
            least_curv(plot_steps, plot_paths);
            }
        else if (graph_state==3)
            {
            if (prn_color==0)
                cpack(0x000f8f);
            else
                cpack(0x0);
            least_curv(plot_steps, plot_paths);
            }
        if ((path>0) || (step>0))
            {
            if ((graph_state==1) || (graph_state==2))
                cpack(0x0);
            else
                cpack(0x80ff80);
            show_paths(plot_steps, path);
            show_steps(plot_paths, step);
            if ((path>0) && (step>0))
                show_data(path, step);
            }
        if (steps1>0)
            {
            if (prn_color==0)
                cpack(0xff0000);
            else
                cpack(0x0);
            c_path(steps1, control_path);
            }
        }
    swapbuffers();
    goto aaa;
    }
  winset(rob_win);
  linewidth(1);
  draw_robot(theta_rd);
  return;
}
/******************** Change View of second graphics window ******************/
change_view2(view_point2)
float *view_point2;
{

  loadmatrix(g_mat1);
  translate(view_point2[1], view_point2[2], view_point2[3]);
  rot(view_point2[4], 'x');
  rot(view_point2[5], 'y');
  rot(view_point2[6], 'z');
```

```
    return;
}

/*************** Draw axis in the second graphic window ***********************/
float norm_x1[]= {1.0, 0.0, 0.0, 0.0},
    norm_x2[]= {-1.0, 0.0, 0.0, 0.0};

float axis_x1[3]= {-200.0, 0.0, 0.0},
    axis_x2[3]= {1000.0, 0.0, 0.0},
    axis_x21[3]= {300.0, 0.0, 0.0},
    axis_y1[3]= {0.0, -200.0, 0.0},
    axis_y2[3]= {0.0, 400.0, 0.0},
    axis_y21[3]= {0.0, 300.0, 0.0},
    axis_z1[3]= {0.0, 0.0, -200.0},
    axis_z2[3]= {0.0, 0.0, 300.0},
    arrox1[3]= {980.0, -3.0, 0.0},
    arrox2[3]= {980.0, 3.0, 0.0},
    arroy1[3]= {-3.0, 380.0, 0.0},
    arroy2[3]= {3.0, 380.0, 0.0},
    arroz1[3]= {0.0, -3.0, 280.0},
    arroz2[3]= {0.0, 3.0, 280.0};
    arrox11[3]= {280.0, -3.0, 0.0},
    arrox21[3]= {280.0, 3.0, 0.0},
    arroy11[3]= {-3.0, 280.0, 0.0},
    arroy21[3]= {3.0, 280.0, 0.0},
    arroz11[3]= {0.0, -3.0, 280.0},
    arroz21[3]= {0.0, 3.0, 280.0};

drawaxis()
{
    extern int ellip;

    if (ellip==1)
        {
        cmov(axis_x21[0]+3,0.0, 0.0);
        charstr("X");
        cmov(0.0, axis_y21[1]+3, 0.0);
        charstr("Y");
        cmov(0.0, 0.0, axis_z2[2]+3);
        charstr("Z");
        bgnline();
        v3f(axis_x1); v3f(axis_x21);
        endline();
        bgnline();
        v3f(axis_y1); v3f(axis_y21);
        endline();
        bgnline();
        v3f(axis_z1); v3f(axis_z2);
        endline();
        }
    else
```

```
        {
        bgnline();
        v3f(axis_x1); v3f(axis_x2);
        endline();
        bgnline();
        v3f(arrox1); v3f(axis_x2); v3f(arrox2);
        endline();
        bgnline();
        v3f(axis_y1); v3f(axis_y2);
        endline();
        bgnline();
        v3f(arroy1); v3f(axis_y2); v3f(arroy2);
        endline();
        bgnline();
        v3f(axis_z1); v3f(axis_z2);
        endline();
        bgnline();
        v3f(arroz1); v3f(axis_z2); v3f(arroz2);
        endline();
        cmov(axis_x2[0]+30,0.0, -10.0);
        charstr("Trajectory");
        cmov(200.0, axis_y2[1], -10.0);
        charstr("Self motion");
        cmov(0.0, 0.0, axis_z2[2]+10);
        charstr("Cost");
        }
}
/********************* Draw the 3-D map by solid lines ************************/
draw_grad(total_steps, total_paths)
int total_steps, total_paths;
{
    extern float norm_dat[TSTEPS][TPATHS][3];
    int i, j, k, l;
    char buffer[10];
    float point1[3], point2[3];
    float tp1, tp2, pmin, pmax;

    for (i=1; i<=total_steps; i++)
        {
        for (j=1; j<total_paths; j++)
            {
            k=j+1;
            bgnline();
            v3f(norm_dat[i][j]); v3f(norm_dat[i][k]);
            endline();
            }
        }
    for (i=1; i<=total_paths; i++)
        for (j=1; j<total_steps; j++)
            {
            k=j+1;
```

```
            bgnline();
            v3f(norm_dat[j][i]);v3f(norm_dat[k][i]);
            endline();
            }
}

/******************** Draw the 3-D map by a solid surface **********************/
grad_surf(part_plot, total_steps, total_paths, t_path1)
int part_plot, total_steps, total_paths, t_path1;
{
    int i, i2, j, j2, k, l;
    char buffer[10];
    extern float norm_dat[TSTEPS][TPATHS][3], norm_light[TSTEPS][TPATHS][3];
    float point1[3], point2[3];
    float tp1, tp2, pmin, pmax;

    for (i=1; i<total_steps; i++)
        {
        i2=i+1;
        for (j=1; j<total_paths; j++)
            {
            j2=j+1;
            bgnpolygon();
            n3f(norm_light[i][j]); v3f(norm_dat[i][j]);
            n3f(norm_light[i][j2]); v3f(norm_dat[i][j2]);
            n3f(norm_light[i2][j2]); v3f(norm_dat[i2][j2]);
            n3f(norm_light[i2][j]); v3f(norm_dat[i2][j]);
            endpolygon();
            }
        if (part_plot == 0)
            {
            bgnpolygon();
            n3f(norm_light[i][0]); v3f(norm_dat[i][1]); v3f(norm_dat[i][0]);
            n3f(norm_light[i2][0]); v3f(norm_dat[i2][0]); v3f(norm_dat[i2][1]);
            endpolygon();
            bgnpolygon();
            n3f(norm_light[i][t_path1]); v3f(norm_dat[i][total_paths]);
            v3f(norm_dat[i][t_path1]); n3f(norm_light[i2][t_path1]);
            v3f(norm_dat[i2][t_path1]); v3f(norm_dat[i2][total_paths]);
            endpolygon();
            }
        }
    if (part_plot == 0)
        {
        bgnpolygon();
        n3f(norm_x2);
        for (i=0; i<=t_path1; i++)
            v3f(norm_dat[1][i]);
        endpolygon();
        bgnpolygon();
        n3f(norm_x1);
```

```
        for (i=0; i<=t_path1; i++)
            v3f(norm_dat[total_steps][i]);
        endpolygon();
        }
    return;
}

/****************** Calculate the norm of the 3-D map surface *****************/
n_cal(total_steps, total_paths, t_path1)
int total_steps, total_paths, t_path1;
{
    extern int graph_state;
    extern float norm_dat[TSTEPS][TPATHS][3], norm_light[TSTEPS][TPATHS][3];
    int i, i1, i2, j, j1, j2;
    float v, v1, v2, v3, vect1[4], vect2[4];

    for (i=1; i<=total_steps; i++)
        {
        i1=i-1;
        i2=i+1;
        for (j=1; j<=total_paths; j++)
            {
            j1=j-1;
            j2=j+1;
            if (i==1)
                {
                v1=norm_dat[i2][j][0]-norm_dat[i][j][0];
                v2=norm_dat[i2][j][1]-norm_dat[i][j][1];
                v3=norm_dat[i2][j][2]-norm_dat[i][j][2];
                }
            else if (i<total_steps)
                {
                v1=norm_dat[i2][j][0]-norm_dat[i1][j][0];
                v2=norm_dat[i2][j][1]-norm_dat[i1][j][1];
                v3=norm_dat[i2][j][2]-norm_dat[i1][j][2];
                }
            else
                {
                v1=norm_dat[i][j][0]-norm_dat[i1][j][0];
                v2=norm_dat[i][j][1]-norm_dat[i1][j][1];
                v3=norm_dat[i][j][2]-norm_dat[i1][j][2];
                }
            v=sqrt(v1*v1+v2*v2+v3*v3);
            vect1[1]=v1/v;
            vect1[2]=v2/v;
            vect1[3]=v3/v;
            if (j==1)
                {
                v1=norm_dat[i][j2][0]-norm_dat[i][j][0];
                v2=norm_dat[i][j2][1]-norm_dat[i][j][1];
                v3=norm_dat[i][j2][2]-norm_dat[i][j][2];
```

```
        }
    else if (j<total_paths)
        {
        v1=norm_dat[i][j2][0]-norm_dat[i][j1][0];
        v2=norm_dat[i][j2][1]-norm_dat[i][j1][1];
        v3=norm_dat[i][j2][2]-norm_dat[i][j1][2];
        }
    else
        {
        v1=norm_dat[i][j][0]-norm_dat[i][j1][0];
        v2=norm_dat[i][j][1]-norm_dat[i][j1][1];
        v3=norm_dat[i][j][2]-norm_dat[i][j1][2];
        }
    v=sqrt(v1*v1+v2*v2+v3*v3);
    vect2[1]=v1/v;
    vect2[2]=v2/v;
    vect2[3]=v3/v;
    norm_light[i][j][0]=vect2[2]*vect1[3]-vect2[3]*vect1[2];
    norm_light[i][j][1]=vect2[3]*vect1[1]-vect2[1]*vect1[3];
    norm_light[i][j][2]=vect2[1]*vect1[2]-vect2[2]*vect1[1];
    }
if (i==1)
    {
    v1=norm_dat[i2][1][0]-norm_dat[i][1][0];
    v2=norm_dat[i2][1][1]-norm_dat[i][1][1];
    v=sqrt(v1*v1+v2*v2);
    norm_light[i][0][0]=(-v1/v);
    norm_light[i][0][1]=(-v2/v);
    norm_light[i][0][2]=0;
    v1=norm_dat[i2][total_paths][0]-norm_dat[i][total_paths][0];
    v2=norm_dat[i2][total_paths][1]-norm_dat[i][total_paths][1];
    v=sqrt(v1*v1+v2*v2);
    norm_light[i][t_path1][0]=(-v1/v);
    norm_light[i][t_path1][1]=(-v2/v);
    norm_light[i][t_path1][2]=0;
    }
else if (i==total_steps)
    {
    v1=norm_dat[i][1][0]-norm_dat[i1][1][0];
    v2=norm_dat[i][1][1]-norm_dat[i1][1][1];
    v=sqrt(v1*v1+v2*v2);
    norm_light[i][0][0]=(-v1/v);
    norm_light[i][0][1]=(-v2/v);
    norm_light[i][0][2]=0;
    v1=norm_dat[i][total_paths][0]-norm_dat[i1][total_paths][0];
    v2=norm_dat[i][total_paths][1]-norm_dat[i1][total_paths][1];
    v=sqrt(v1*v1+v2*v2);
    norm_light[i][t_path1][0]=(-v1/v);
    norm_light[i][t_path1][1]=(-v2/v);
    norm_light[i][t_path1][2]=0;
    }
```

```
        else
            {
            v1=norm_dat[i2][1][0]-norm_dat[i1][1][0];
            v2=norm_dat[i2][1][1]-norm_dat[i1][1][1];
            v=sqrt(v1*v1+v2*v2);
            norm_light[i][0][0]=(-v1/v);
            norm_light[i][0][1]=(-v2/v);
            norm_light[i][0][2]=0;
            v1=norm_dat[i2][total_paths][0]-norm_dat[i][total_paths][0];
            v2=norm_dat[i2][total_paths][1]-norm_dat[i][total_paths][1];
            v=sqrt(v1*v1+v2*v2);
            norm_light[i][t_path1][0]=(-v1/v);
            norm_light[i][t_path1][1]=(-v2/v);
            norm_light[i][t_path1][2]=0;
            }
        norm_dat[i][0][0]=norm_dat[i][1][0];
        norm_dat[i][0][1]=norm_dat[i][1][1];
        norm_dat[i][0][2]=0;
        norm_dat[i][t_path1][0]=norm_dat[i][total_paths][0];
        norm_dat[i][t_path1][1]=norm_dat[i][total_paths][1];
        norm_dat[i][t_path1][2]=0;
        }
    graph_state = 1;
    return;
}

/************************** Plot least norm curves **************************/
least_curv(total_steps, total_paths)
int total_steps, total_paths;
{
    int i, j, k;
    extern float norm_dat[TSTEPS][TPATHS][3];

    for (i=1; i<=total_paths; i++)
        for (j=1; j<total_steps; j++)
            {
            k=j+1;
            bgnline();
            v3f(norm_dat[j][i]);v3f(norm_dat[k][i]);
            endline();
            }
    return;
}

/************************** Plot a single curve ********************************/
c_path(steps1, control_path)
int steps1;
float control_path[TSTEPS][3];
{
    int i, j;
```

```
    for (i=1; i<steps1; i++)
        {
        j=i+1;
        bgnline();
        v3f(control_path[i]);v3f(control_path[j]);
        endline();
        }
    return;
}

/********************** Show the path of a 3-D map ***************************/
show_paths(total_steps, path)
int total_steps, path;
{
    extern float norm_dat[TSTEPS][TPATHS][3];
    int i, j;

    for (i=1; i<total_steps; i++)
        {
        j=i+1;
        bgnline();
        v3f(norm_dat[i][path]);v3f(norm_dat[j][path]);
        endline();
        }
    return;
}

/************************ Show the step of a 3-D map ***************************/
show_steps(total_paths, step)
int total_paths, step;
{
    extern float norm_dat[TSTEPS][TPATHS][3];
    int i, j;

    for (i=1; i<total_paths; i++)
        {
        j=i+1;
        bgnline();
        v3f(norm_dat[step][i]);v3f(norm_dat[step][j]);
        endline();
        }
    return;
}

/*************** Print 3-D map data in the second graphic window **************/
show_data(path, step)
int path, step;
{
    extern float norm_dat[TSTEPS][TPATHS][3], g_mat1[4][4], g_mat2[4][4];
    char buffer[50];
```

```
    pushmatrix();
    loadmatrix(g_mat1);
    cpack(0x008fff);
    sprintf(buffer,"%d", step);
    cmov(-0.028,0.0375, -0.1);
    charstr(buffer);
    sprintf(buffer,"%7.3f", norm_dat[step][path][1]);
    cmov(-0.015,0.0375, -0.1);
    charstr(buffer);
    sprintf(buffer,"%7.3f", norm_dat[step][path][2]);
    cmov(0.005,0.0375, -0.1);
    charstr(buffer);
    popmatrix();
    return;
}

/********************** On line 3-D ploting function ************************/
draw_curv()
{
    extern float control_path[TSTEPS][3];
    extern int steps1, prn_color, ellip;
    extern sub_win;
    float jaco[MAT_DIM][MAT_DIM], jsqr[MAT_DIM][MAT_DIM], h_transf[5][5];
    float x_scaler, y_scaler, z_scaler;
    int i, j;

    linewidth(3);
    if (ellip==0)                /* Plot a 3-D curve   */
        {
        if (steps1 == 1)
            {
            control_path[0][0] = control_path[1][0];
            control_path[0][1] = control_path[1][1];
            control_path[0][2] = control_path[1][2];
            }
        i=steps1-1;
        if (i==0)
            return;
        winset(sub_win);
        frontbuffer(TRUE);
        if (prn_color==0)
            cpack(0xff0000);
        else
            cpack(0x0);
        bgnline();
        v3f(control_path[i]);v3f(control_path[steps1]);
        endline();
        frontbuffer(FALSE);
        }
    else                         /*   Draw the MVR ellipsoid  */
        {
```

```
        winset(sub_win);
        if (prn_color==0)
            {
            czclear(color_back2, 0x7fffff);
            cpack(0x0);
            }
        else
            {
            czclear(0xffffff, 0x7fffff);
            cpack(0x0);
            }
        linewidth(1);
        drawaxis();
        jacobian(theta_rd,h_transf,jaco);
        cal_jjt(3, T_JOINT, jaco, jsqr);
        svd(3, jsqr, jsqr[0]);
        x_scaler=4.0*sqrt(jsqr[0][1]);
        y_scaler=4.0*sqrt(jsqr[0][2]);
        z_scaler=4.0*sqrt(jsqr[0][3]);
        for (i=1; i<=3; i++)
            for (j=1; j<=3; j++)
                h_transf[i][j]=jsqr[i][j];
        pushmatrix();
        rpy_angles(h_transf);
        rot( h_transf[0][1]/0.017453293, 'z');
        rot( h_transf[0][2]/0.017453293, 'y');
        rot( h_transf[0][3]/0.017453293, 'x');
        ellipse(x_scaler, y_scaler, z_scaler);
        popmatrix();
        swapbuffers();
        }
    winset(rob_win);
    linewidth(1);
    return;
}
/********************** Data used by Drawing a ellipse ************************/
float ellipse0[]= {0.0, -1.0, 0.0},
    ellipse1[]= {0.2588, -0.9659, 0.0},
    ellipse2[]= {0.5, -0.866, 0.0},
    ellipse3[]= {0.7071, -0.7071, 0.0},
    ellipse4[]= {0.866, -0.5, 0.0},
    ellipse5[]= {0.9659, -0.2588, 0.0},
    ellipse6[]= {1.0, 0.0, 0.0},
    ellipse7[]= {0.9659, 0.2588, 0.0},
    ellipse8[]= {0.866, 0.5, 0.0},
    ellipse9[]= {0.7071, 0.7071, 0.0},
    ellipse10[]= {0.5, 0.866, 0.0},
    ellipse11[]= {0.2588, 0.9659, 0.0},
    ellipse12[]= {0.0, 1.0, 0.0},
    ellipse13[]= {0.0, -0.9659, 0.2588},
    ellipse14[]= {0.25, -0.933, 0.2588},
```

```
ellipse15[]= {0.483, -0.8365, 0.2588},
ellipse16[]= {0.683, -0.683, 0.2588},
ellipse17[]= {0.8365, -0.483, 0.2588},
ellipse18[]= {0.933, -0.25, 0.2588},
ellipse19[]= {0.9659, 0.0, 0.2588},
ellipse20[]= {0.933, 0.25, 0.2588},
ellipse21[]= {0.8365, 0.483, 0.2588},
ellipse22[]= {0.683, 0.683, 0.2588},
ellipse23[]= {0.483, 0.8365, 0.2588},
ellipse24[]= {0.25, 0.933, 0.2588},
ellipse25[]= {0.0, 0.9659, 0.2588},
ellipse26[]= {0.0, -0.866, 0.5},
ellipse27[]= {0.2241, -0.8365, 0.5},
ellipse28[]= {0.433, -0.75, 0.5},
ellipse29[]= {0.6124, -0.6124, 0.5},
ellipse30[]= {0.75, -0.433, 0.5},
ellipse31[]= {0.8365, -0.2241, 0.5},
ellipse32[]= {0.866, 0.0, 0.5},
ellipse33[]= {0.8365, 0.2241, 0.5},
ellipse34[]= {0.75, 0.433, 0.5},
ellipse35[]= {0.6124, 0.6124, 0.5},
ellipse36[]= {0.433, 0.75, 0.5},
ellipse37[]= {0.2241, 0.8365, 0.5},
ellipse38[]= {0.0, 0.866, 0.5},
ellipse39[]= {0.0, -0.7071, 0.7071},
ellipse40[]= {0.183, -0.683, 0.7071},
ellipse41[]= {0.3536, -0.6124, 0.7071},
ellipse42[]= {0.5, -0.5, 0.7071},
ellipse43[]= {0.6124, -0.3536, 0.7071},
ellipse44[]= {0.683, -0.183, 0.7071},
ellipse45[]= {0.7071, 0.0, 0.7071},
ellipse46[]= {0.683, 0.183, 0.7071},
ellipse47[]= {0.6124, 0.3536, 0.7071},
ellipse48[]= {0.5, 0.5, 0.7071},
ellipse49[]= {0.3536, 0.6124, 0.7071},
ellipse50[]= {0.183, 0.683, 0.7071},
ellipse51[]= {0.0, 0.7071, 0.7071},
ellipse52[]= {0.0, -0.5, 0.866},
ellipse53[]= {0.1294, -0.483, 0.866},
ellipse54[]= {0.25, -0.433, 0.866},
ellipse55[]= {0.3536, -0.3536, 0.866},
ellipse56[]= {0.433, -0.25, 0.866},
ellipse57[]= {0.483, -0.1294, 0.866},
ellipse58[]= {0.5, 0.0, 0.866},
ellipse59[]= {0.483, 0.1294, 0.866},
ellipse60[]= {0.433, 0.25, 0.866},
ellipse61[]= {0.3536, 0.3536, 0.866},
ellipse62[]= {0.25, 0.433, 0.866},
ellipse63[]= {0.1294, 0.483, 0.866},
ellipse64[]= {0.0, 0.5, 0.866},
ellipse65[]= {0.0, -0.2588, 0.9659},
```

```
    ellipse66[]= {0.067, -0.25, 0.9659},
    ellipse67[]= {0.1294, -0.2241, 0.9659},
    ellipse68[]= {0.183, -0.183, 0.9659},
    ellipse69[]= {0.2241, -0.1294, 0.9659},
    ellipse70[]= {0.25, -0.067, 0.9659},
    ellipse71[]= {0.2588, 0.0, 0.9659},
    ellipse72[]= {0.25, 0.067, 0.9659},
    ellipse73[]= {0.2241, 0.1294, 0.9659},
    ellipse74[]= {0.183, 0.183, 0.9659},
    ellipse75[]= {0.1294, 0.2241, 0.9659},
    ellipse76[]= {0.067, 0.25, 0.9659},
    ellipse77[]= {0.0, 0.2588, 0.9659},
    ellipse78[]= {0.0, 0.0, 1.0};

/************************** Draw part of an ellipsoid ***************************/
fourth_ellipse()
{
    bgnpolygon();
    n3f(ellipse0); v3f(ellipse0); n3f(ellipse1); v3f(ellipse1);
    n3f(ellipse14); v3f(ellipse14); n3f(ellipse13); v3f(ellipse13);
    endpolygon();
    bgnpolygon();
    n3f(ellipse1); v3f(ellipse1); n3f(ellipse2); v3f(ellipse2);
    n3f(ellipse15); v3f(ellipse15); n3f(ellipse14); v3f(ellipse14);
    endpolygon();
    bgnpolygon();
    n3f(ellipse2); v3f(ellipse2); n3f(ellipse3); v3f(ellipse3);
    n3f(ellipse16); v3f(ellipse16); n3f(ellipse15); v3f(ellipse15);
    endpolygon();
    bgnpolygon();
    n3f(ellipse3); v3f(ellipse3); n3f(ellipse4); v3f(ellipse4);
    n3f(ellipse17); v3f(ellipse17); n3f(ellipse16); v3f(ellipse16);
    endpolygon();
    bgnpolygon();
    n3f(ellipse4); v3f(ellipse4); n3f(ellipse5); v3f(ellipse5);
    n3f(ellipse18); v3f(ellipse18); n3f(ellipse17); v3f(ellipse17);
    endpolygon();
    bgnpolygon();
    n3f(ellipse5); v3f(ellipse5); n3f(ellipse6); v3f(ellipse6);
    n3f(ellipse19); v3f(ellipse19); n3f(ellipse18); v3f(ellipse18);
    endpolygon();
    bgnpolygon();
    n3f(ellipse6); v3f(ellipse6); n3f(ellipse7); v3f(ellipse7);
    n3f(ellipse20); v3f(ellipse20); n3f(ellipse19); v3f(ellipse19);
    endpolygon();
    bgnpolygon();
    n3f(ellipse7); v3f(ellipse7); n3f(ellipse8); v3f(ellipse8);
    n3f(ellipse21); v3f(ellipse21); n3f(ellipse20); v3f(ellipse20);
    endpolygon();
    bgnpolygon();
    n3f(ellipse8); v3f(ellipse8); n3f(ellipse9); v3f(ellipse9);
```

```
n3f(ellipse22); v3f(ellipse22); n3f(ellipse21); v3f(ellipse21);
endpolygon();
bgnpolygon();
n3f(ellipse9); v3f(ellipse9); n3f(ellipse10); v3f(ellipse10);
n3f(ellipse23); v3f(ellipse23); n3f(ellipse22); v3f(ellipse22);
endpolygon();
bgnpolygon();
n3f(ellipse10); v3f(ellipse10); n3f(ellipse11); v3f(ellipse11);
n3f(ellipse24); v3f(ellipse24); n3f(ellipse23); v3f(ellipse23);
endpolygon();
bgnpolygon();
n3f(ellipse11); v3f(ellipse11); n3f(ellipse12); v3f(ellipse12);
n3f(ellipse25); v3f(ellipse25); n3f(ellipse24); v3f(ellipse24);
endpolygon();
bgnpolygon();
n3f(ellipse13); v3f(ellipse13); n3f(ellipse14); v3f(ellipse14);
n3f(ellipse27); v3f(ellipse27); n3f(ellipse26); v3f(ellipse26);
endpolygon();
bgnpolygon();
n3f(ellipse14); v3f(ellipse14); n3f(ellipse15); v3f(ellipse15);
n3f(ellipse28); v3f(ellipse28); n3f(ellipse27); v3f(ellipse27);
endpolygon();
bgnpolygon();
n3f(ellipse15); v3f(ellipse15); n3f(ellipse16); v3f(ellipse16);
n3f(ellipse29); v3f(ellipse29); n3f(ellipse28); v3f(ellipse28);
endpolygon();
bgnpolygon();
n3f(ellipse16); v3f(ellipse16); n3f(ellipse17); v3f(ellipse17);
n3f(ellipse30); v3f(ellipse30); n3f(ellipse29); v3f(ellipse29);
endpolygon();
bgnpolygon();
n3f(ellipse17); v3f(ellipse17); n3f(ellipse18); v3f(ellipse18);
n3f(ellipse31); v3f(ellipse31); n3f(ellipse30); v3f(ellipse30);
endpolygon();
bgnpolygon();
n3f(ellipse18); v3f(ellipse18); n3f(ellipse19); v3f(ellipse19);
n3f(ellipse32); v3f(ellipse32); n3f(ellipse31); v3f(ellipse31);
endpolygon();
bgnpolygon();
n3f(ellipse19); v3f(ellipse19); n3f(ellipse20); v3f(ellipse20);
n3f(ellipse33); v3f(ellipse33); n3f(ellipse32); v3f(ellipse32);
endpolygon();
bgnpolygon();
n3f(ellipse20); v3f(ellipse20); n3f(ellipse21); v3f(ellipse21);
n3f(ellipse34); v3f(ellipse34); n3f(ellipse33); v3f(ellipse33);
endpolygon();
bgnpolygon();
n3f(ellipse21); v3f(ellipse21); n3f(ellipse22); v3f(ellipse22);
n3f(ellipse35); v3f(ellipse35); n3f(ellipse34); v3f(ellipse34);
endpolygon();
bgnpolygon();
```

```
n3f(ellipse22); v3f(ellipse22); n3f(ellipse23); v3f(ellipse23);
n3f(ellipse36); v3f(ellipse36); n3f(ellipse35); v3f(ellipse35);
endpolygon();
bgnpolygon();
n3f(ellipse23); v3f(ellipse23); n3f(ellipse24); v3f(ellipse24);
n3f(ellipse37); v3f(ellipse37); n3f(ellipse36); v3f(ellipse36);
endpolygon();
bgnpolygon();
n3f(ellipse24); v3f(ellipse24); n3f(ellipse25); v3f(ellipse25);
n3f(ellipse38); v3f(ellipse38); n3f(ellipse37); v3f(ellipse37);
endpolygon();
bgnpolygon();
n3f(ellipse26); v3f(ellipse26); n3f(ellipse27); v3f(ellipse27);
n3f(ellipse40); v3f(ellipse40); n3f(ellipse39); v3f(ellipse39);
endpolygon();
bgnpolygon();
n3f(ellipse27); v3f(ellipse27); n3f(ellipse28); v3f(ellipse28);
n3f(ellipse41); v3f(ellipse41); n3f(ellipse40); v3f(ellipse40);
endpolygon();
bgnpolygon();
n3f(ellipse28); v3f(ellipse28); n3f(ellipse29); v3f(ellipse29);
n3f(ellipse42); v3f(ellipse42); n3f(ellipse41); v3f(ellipse41);
endpolygon();
bgnpolygon();
n3f(ellipse29); v3f(ellipse29); n3f(ellipse30); v3f(ellipse30);
n3f(ellipse43); v3f(ellipse43); n3f(ellipse42); v3f(ellipse42);
endpolygon();
bgnpolygon();
n3f(ellipse30); v3f(ellipse30); n3f(ellipse31); v3f(ellipse31);
n3f(ellipse44); v3f(ellipse44); n3f(ellipse43); v3f(ellipse43);
endpolygon();
bgnpolygon();
n3f(ellipse31); v3f(ellipse31); n3f(ellipse32); v3f(ellipse32);
n3f(ellipse45); v3f(ellipse45); n3f(ellipse44); v3f(ellipse44);
endpolygon();
bgnpolygon();
n3f(ellipse32); v3f(ellipse32); n3f(ellipse33); v3f(ellipse33);
n3f(ellipse46); v3f(ellipse46); n3f(ellipse45); v3f(ellipse45);
endpolygon();
bgnpolygon();
n3f(ellipse33); v3f(ellipse33); n3f(ellipse34); v3f(ellipse34);
n3f(ellipse47); v3f(ellipse47); n3f(ellipse46); v3f(ellipse46);
endpolygon();
bgnpolygon();
n3f(ellipse34); v3f(ellipse34); n3f(ellipse35); v3f(ellipse35);
n3f(ellipse48); v3f(ellipse48); n3f(ellipse47); v3f(ellipse47);
endpolygon();
bgnpolygon();
n3f(ellipse35); v3f(ellipse35); n3f(ellipse36); v3f(ellipse36);
n3f(ellipse49); v3f(ellipse49); n3f(ellipse48); v3f(ellipse48);
endpolygon();
```

```
bgnpolygon();
n3f(ellipse36); v3f(ellipse36); n3f(ellipse37); v3f(ellipse37);
n3f(ellipse50); v3f(ellipse50); n3f(ellipse49); v3f(ellipse49);
endpolygon();
bgnpolygon();
n3f(ellipse37); v3f(ellipse37); n3f(ellipse38); v3f(ellipse38);
n3f(ellipse51); v3f(ellipse51); n3f(ellipse50); v3f(ellipse50);
endpolygon();
bgnpolygon();
n3f(ellipse39); v3f(ellipse39); n3f(ellipse40); v3f(ellipse40);
n3f(ellipse53); v3f(ellipse53); n3f(ellipse52); v3f(ellipse52);
endpolygon();
bgnpolygon();
n3f(ellipse40); v3f(ellipse40); n3f(ellipse41); v3f(ellipse41);
n3f(ellipse54); v3f(ellipse54); n3f(ellipse53); v3f(ellipse53);
endpolygon();
bgnpolygon();
n3f(ellipse41); v3f(ellipse41); n3f(ellipse42); v3f(ellipse42);
n3f(ellipse55); v3f(ellipse55); n3f(ellipse54); v3f(ellipse54);
endpolygon();
bgnpolygon();
n3f(ellipse42); v3f(ellipse42); n3f(ellipse43); v3f(ellipse43);
n3f(ellipse56); v3f(ellipse56); n3f(ellipse55); v3f(ellipse55);
endpolygon();
bgnpolygon();
n3f(ellipse43); v3f(ellipse43); n3f(ellipse44); v3f(ellipse44);
n3f(ellipse57); v3f(ellipse57); n3f(ellipse56); v3f(ellipse56);
endpolygon();
bgnpolygon();
n3f(ellipse44); v3f(ellipse44); n3f(ellipse45); v3f(ellipse45);
n3f(ellipse58); v3f(ellipse58); n3f(ellipse57); v3f(ellipse57);
endpolygon();
bgnpolygon();
n3f(ellipse45); v3f(ellipse45); n3f(ellipse46); v3f(ellipse46);
n3f(ellipse59); v3f(ellipse59); n3f(ellipse58); v3f(ellipse58);
endpolygon();
bgnpolygon();
n3f(ellipse46); v3f(ellipse46); n3f(ellipse47); v3f(ellipse47);
n3f(ellipse60); v3f(ellipse60); n3f(ellipse59); v3f(ellipse59);
endpolygon();
bgnpolygon();
n3f(ellipse47); v3f(ellipse47); n3f(ellipse48); v3f(ellipse48);
n3f(ellipse61); v3f(ellipse61); n3f(ellipse60); v3f(ellipse60);
endpolygon();
bgnpolygon();
n3f(ellipse48); v3f(ellipse48); n3f(ellipse49); v3f(ellipse49);
n3f(ellipse62); v3f(ellipse62); n3f(ellipse61); v3f(ellipse61);
endpolygon();
bgnpolygon();
n3f(ellipse49); v3f(ellipse49); n3f(ellipse50); v3f(ellipse50);
n3f(ellipse63); v3f(ellipse63); n3f(ellipse62); v3f(ellipse62);
```

```
endpolygon();
bgnpolygon();
n3f(ellipse50); v3f(ellipse50); n3f(ellipse51); v3f(ellipse51);
n3f(ellipse64); v3f(ellipse64); n3f(ellipse63); v3f(ellipse63);
endpolygon();
bgnpolygon();
n3f(ellipse52); v3f(ellipse52); n3f(ellipse53); v3f(ellipse53);
n3f(ellipse66); v3f(ellipse66); n3f(ellipse65); v3f(ellipse65);
endpolygon();
bgnpolygon();
n3f(ellipse53); v3f(ellipse53); n3f(ellipse54); v3f(ellipse54);
n3f(ellipse67); v3f(ellipse67); n3f(ellipse66); v3f(ellipse66);
endpolygon();
bgnpolygon();
n3f(ellipse54); v3f(ellipse54); n3f(ellipse55); v3f(ellipse55);
n3f(ellipse68); v3f(ellipse68); n3f(ellipse67); v3f(ellipse67);
endpolygon();
bgnpolygon();
n3f(ellipse55); v3f(ellipse55); n3f(ellipse56); v3f(ellipse56);
n3f(ellipse69); v3f(ellipse69); n3f(ellipse68); v3f(ellipse68);
endpolygon();
bgnpolygon();
n3f(ellipse56); v3f(ellipse56); n3f(ellipse57); v3f(ellipse57);
n3f(ellipse70); v3f(ellipse70); n3f(ellipse69); v3f(ellipse69);
endpolygon();
bgnpolygon();
n3f(ellipse57); v3f(ellipse57); n3f(ellipse58); v3f(ellipse58);
n3f(ellipse71); v3f(ellipse71); n3f(ellipse70); v3f(ellipse70);
endpolygon();
bgnpolygon();
n3f(ellipse58); v3f(ellipse58); n3f(ellipse59); v3f(ellipse59);
n3f(ellipse72); v3f(ellipse72); n3f(ellipse71); v3f(ellipse71);
endpolygon();
bgnpolygon();
n3f(ellipse59); v3f(ellipse59); n3f(ellipse60); v3f(ellipse60);
n3f(ellipse73); v3f(ellipse73); n3f(ellipse72); v3f(ellipse72);
endpolygon();
bgnpolygon();
n3f(ellipse60); v3f(ellipse60); n3f(ellipse61); v3f(ellipse61);
n3f(ellipse74); v3f(ellipse74); n3f(ellipse73); v3f(ellipse73);
endpolygon();
bgnpolygon();
n3f(ellipse61); v3f(ellipse61); n3f(ellipse62); v3f(ellipse62);
n3f(ellipse75); v3f(ellipse75); n3f(ellipse74); v3f(ellipse74);
endpolygon();
bgnpolygon();
n3f(ellipse62); v3f(ellipse62); n3f(ellipse63); v3f(ellipse63);
n3f(ellipse76); v3f(ellipse76); n3f(ellipse75); v3f(ellipse75);
endpolygon();
bgnpolygon();
n3f(ellipse63); v3f(ellipse63); n3f(ellipse64); v3f(ellipse64);
```

```
n3f(ellipse77); v3f(ellipse77); n3f(ellipse76); v3f(ellipse76);
endpolygon();
bgnpolygon();
n3f(ellipse65); v3f(ellipse65); n3f(ellipse66); v3f(ellipse66);
n3f(ellipse78); v3f(ellipse78);
endpolygon();
bgnpolygon();
n3f(ellipse66); v3f(ellipse66); n3f(ellipse67); v3f(ellipse67);
n3f(ellipse78); v3f(ellipse78);
endpolygon();
bgnpolygon();
n3f(ellipse67); v3f(ellipse67); n3f(ellipse68); v3f(ellipse68);
n3f(ellipse78); v3f(ellipse78);
endpolygon();
bgnpolygon();
n3f(ellipse68); v3f(ellipse68); n3f(ellipse69); v3f(ellipse69);
n3f(ellipse78); v3f(ellipse78);
endpolygon();
bgnpolygon();
n3f(ellipse69); v3f(ellipse69); n3f(ellipse70); v3f(ellipse70);
n3f(ellipse78); v3f(ellipse78);
endpolygon();
bgnpolygon();
n3f(ellipse70); v3f(ellipse70); n3f(ellipse71); v3f(ellipse71);
n3f(ellipse78); v3f(ellipse78);
endpolygon();
bgnpolygon();
n3f(ellipse71); v3f(ellipse71); n3f(ellipse72); v3f(ellipse72);
n3f(ellipse78); v3f(ellipse78);
endpolygon();
bgnpolygon();
n3f(ellipse72); v3f(ellipse72); n3f(ellipse73); v3f(ellipse73);
n3f(ellipse78); v3f(ellipse78);
endpolygon();
bgnpolygon();
n3f(ellipse73); v3f(ellipse73); n3f(ellipse74); v3f(ellipse74);
n3f(ellipse78); v3f(ellipse78);
endpolygon();
bgnpolygon();
n3f(ellipse74); v3f(ellipse74); n3f(ellipse75); v3f(ellipse75);
n3f(ellipse78); v3f(ellipse78);
endpolygon();
bgnpolygon();
n3f(ellipse75); v3f(ellipse75); n3f(ellipse76); v3f(ellipse76);
n3f(ellipse78); v3f(ellipse78);
endpolygon();
bgnpolygon();
n3f(ellipse76); v3f(ellipse76); n3f(ellipse77); v3f(ellipse77);
n3f(ellipse78); v3f(ellipse78);
endpolygon();
```

```
}

/***************************** Draw an ellipsoid *****************************/
ellipse(x_scaler, y_scaler, z_scaler)
float x_scaler, y_scaler, z_scaler;
{
    pushmatrix();
    scale(x_scaler, y_scaler, z_scaler);
    fourth_ellipse();
    rot(180, 'z');
    fourth_ellipse();
    rot(180, 'x');
    fourth_ellipse();
    rot(180, 'z');
    fourth_ellipse();
    popmatrix();
}

/***************************** Drawing the robot *****************************/
draw_robot(theta_rd)
float *theta_rd;
{
    extern float view_mat[4][4], h_transf[5][5], ult_dtheta[MAT_DIM];
    extern float trajectory[TSTEPS][3], eig_vect[4][4], force_dtheta[MAT_DIM];
    extern float th_lim1[MAT_DIM], th_lim2[MAT_DIM];
    extern float axis0[3], x_axis[3], y_axis[3], z_axis[3], obst_dis;
    extern int show_path, e_traj_count, ser_port1, control_robot, local_coord;
    extern long color_data, color_back;
    extern int path_plot_ctl, goal_steps, robot_index, force_switch;
    extern int control_robot, ult_switch;
    extern char force_buffer[200];

    float theta[8], tp1, tp2, roll, pitch, yaw;
    float tp_x, tp_y, tp_z;
    char buffer[100];
    long joint[8];
    static int error_data;
    int i, j, read_index[4];

    transf(theta_rd,h_transf);
    poll_manager(2, read_index);
    if ((read_index[0] < 0) || (read_index[1] < 0))
        error_data++;
    else
        error_data = 0;

    if (force_switch == 1)
        {
        cpack(color_data);
        cmov(-0.0036,-0.0034,-0.01);
        charstr(force_buffer);
```

```
    }
if ((read_index[0] != 0) && (error_data < 4))
    {
    new_theta(theta_rd, force_dtheta);
    }
if (read_index[1] != 0)
    {
    if ((ult_switch == 1) && (error_data < 4))
        {
        new_theta(theta_rd, ult_dtheta);
        }
    }

for (i=1; i<=7; i++)
    theta[i]=theta_rd[i]/0.017453293;
theta[0] = theta_rd[0];
rpy_angles(h_transf);
roll=h_transf[0][1]/0.017453293;
pitch=h_transf[0][2]/0.017453293;
yaw=h_transf[0][3]/0.017453293;
sprintf(buffer,
    "X=%8.3f   Y=%8.3f   Z=%8.3f   Roll=%8.3f   Pitch=%8.3f   Yaw=%8.3f",
    h_transf[1][4], h_transf[2][4], h_transf[3][4], roll, pitch, yaw);

cpack(color_data);
cmov(-0.0036,-0.0037,-0.01);
charstr(buffer);
cmov(-0.0036,-0.004,-0.01);
for (i=1; i<=7; i++)
    {
    sprintf(buffer,"J%d=%8.3f   ",i, theta[i]);
    if ((th_lim1[i] >= theta_rd[i]) || (th_lim2[i] <= theta_rd[i]))
        cpack(0x00ffff);
    else
        cpack(color_data);
    charstr(buffer);
    }
cpack(0xee00ee);
cmov(0.0028, 0.0038,-0.01);
if (local_coord == 0)
    charstr("Base coordinate");
else
    charstr("Hand coordinate");
pushmatrix();
loadmatrix(view_mat);
if (show_path==1)
    {
    cpack(0xff8888);
    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");
    e_traj_count++;
    trajectory[e_traj_count][0]=h_transf[1][4];
    trajectory[e_traj_count][1]=h_transf[2][4];
    trajectory[e_traj_count][2]=h_transf[3][4];
    for (i=0; i<e_traj_count; i++)
        {
        cpack(0xbfbf00);
        if(i==(i | 0x01))
            cpack(0xbf00bf);
        j=i+1;
        bgnline();
        v3f(trajectory[i]); v3f(trajectory[j]);
        endline();
        }
    }
if (robot_index == 1)
    {
    if ((ult_switch == 1) && (obst_dis <= 40))
        {
        pushmatrix();
        rot(theta[1], 'y');
        rot(theta[2], 'x');
        translate(0.0, 0.0, -5.625);
        rot(theta[3], 'y');
        translate(0.0, 37.985, -4.25);
        rot(theta[4], 'x');
        tp_x = -10.5 - obst_dis;
        translate(tp_x, 0, 0);
        tp_x = tp_y = tp_z = 3;
        ellipse(tp_x, tp_y, tp_z);
        popmatrix();
        }
    lmbind(MATERIAL, 1);
    draw_k2107_robot(theta);
    }
else if (robot_index == 2)
    {
    lmbind(MATERIAL, 2);
    draw_LTM_robot(theta);
```

```
        }
    popmatrix();
    swapbuffers();
    czclear(color_back, 0x0);
    return;
}


/***************** Final checking of changing configurations ******************/
final_check(theta_rd)
float *theta_rd;
{
    extern int ult_switch, force_switch, ult_sensor_fb, robot_index, control_robot;
    extern int drawing, changing_view, teach_orbit;
    extern float view_point[MAT_DIM], obst_dis;

    int i;
    char buffer[100];
    unsigned long echo_time;
    float tp_theta[MAT_DIM], jaco[MAT_DIM][MAT_DIM], h_transf[5][5];

    if (drawing == 0)
        {
        if ((force_switch == 1) || (ult_switch == 1) ||
            ((control_robot == 1) && (robot_index == 1)))

            {
            if (changing_view == 1)
                c_view(view_point);
            else
                draw_robot(theta_rd);
            }
        }
}
```

```
/*                    Source code file "draw_k2107.c"                    */

#include <gl/gl.h>
#include <math.h>
#include <stdio.h>
#define MAT_DIM 10

extern struct {int index; int sign; float obt_v;} obtctl;
extern float th_lim1[MAT_DIM], th_lim2[MAT_DIM], theta_rd[MAT_DIM], dth_lim[MAT_DIM];
extern float view_point[MAT_DIM], jv_lim[MAT_DIM];
extern float robot_home[MAT_DIM], init_view_point[MAT_DIM];
extern float joint_v;

/************************ Normal in X, Y and Z direction ************************/

float nx1[3]= {1.0, 0.0, 0.0},
    nx2[3]= {-1.0, 0.0, 0.0},
    ny1[3]= {0.0, 1.0, 0.0},
    ny2[3]= {0.0, -1.0, 0.0},
    nz1[3]= {0.0, 0.0, 1.0},
    nz2[3]= {0.0, 0.0, -1.0};

/***************************** Data for the base ********************************/

float v0[3]= {9.0, -8.1, -9.5},
    v1[3]= {8.0, -8.1, 0.0},
    v2[3]= {6.928, -8.1, 4.0},
    v3[3]= {4.0, -8.1, 6.928},
    v4[3]= {0.0, -8.1, 8.0},
    v5[3]= {-4.0, -8.1, 6.928},
    v6[3]= {-6.928, -8.1, 4.0},
    v7[3]= {-8.0, -8.1, 0.0},
    v8[3]= {-9.0, -8.1, -9.5},
    v9[3]= {9.0, -29.1, -9.5},
    v10[3]= {8.0, -29.1, 0.0},
    v11[3]= {6.928, -29.1, 4.0},
    v12[3]= {4.0, -29.1, 6.928},
    v13[3]= {0.0, -29.1, 8.0},
    v14[3]= {-4.0, -29.1, 6.928},
    v15[3]= {-6.928, -29.1, 4.0},
    v16[3]= {-8.0, -29.1, 0.0},
    v17[3]= {-9.0, -29.1, -9.5};

float n1[3]= {0.9945, 0.0, 0.1045},
    n2[3]= {0.866, 0.0, 0.5},
    n3[3]= {0.5, 0.0, 0.866},
    n5[3]= {-0.5, 0.0, 0.866},
    n6[3]= {-0.866, 0.0, 0.5},
    n7[3]= {-0.9945, 0.0, 0.1045};
```

```
/************************** Data for link 1 ********************************/

float av0[3]= {0.0, -8.0, -8.0},
    av1[3]= {-4.0, -8.0, -6.928},
    av2[3]= {-6.928, -8.0, -4.0},
    av3[3]= {-8.0, -8.0, 0.0},
    av4[3]= {-6.928, -8.0, 4.0},
    av5[3]= {-4.0, -8.0, 6.928},
    av6[3]= {0.0, -8.0, 8.0},
    av7[3]= {4.0, -8.0, 6.928},
    av8[3]= {6.928, -8.0, 4.0},
    av9[3]= {8.0, -8.0, 0.0},
    av10[3]= {6.928, -8.0, -4.0},
    av11[3]= {4.0, -8.0, -6.928},
    av12[3]= {0.0, -6.0, -8.0},
    av13[3]= {-4.0, -6.0, -6.928},
    av14[3]= {-6.928, -6.0, -4.0},
    av15[3]= {-8.0, -6.0, 0.0},
    av16[3]= {-6.928, -6.0, 4.0},
    av17[3]= {-4.0, -6.0, 6.928},
    av18[3]= {0.0, -6.0, 8.0},
    av19[3]= {4.0, -6.0, 6.928},
    av20[3]= {6.928, -6.0, 4.0},
    av21[3]= {8.0, -6.0, 0.0},
    av22[3]= {6.928, -6.0, -4.0},
    av23[3]= {4.0, -6.0, -6.928},
    av24[3]= {-6.0, -6.0, 4.0},
    av25[3]= {-6.0, 0.0, 4.0},
    av26[3]= {-6.0, 2.0, 3.464},
    av27[3]= {-6.0, 3.464, 2.0},
    av28[3]= {-6.0, 4.0, 0.0},
    av29[3]= {-6.0, 3.464, -2.0},
    av30[3]= {-6.0, 2.0, -3.464},
    av31[3]= {-6.0, 0.0, -4.0},
    av32[3]= {-6.0, -6.0, -4.0},
    av33[3]= {-4.0, -6.0, 4.0},
    av34[3]= {-4.0, 0.0, 4.0},
    av35[3]= {-4.0, 2.0, 3.464},
    av36[3]= {-4.0, 3.464, 2.0},
    av37[3]= {-4.0, 4.0, 0.0},
    av38[3]= {-4.0, 3.464, -2.0},
    av39[3]= {-4.0, 2.0, -3.464},
    av40[3]= {-4.0, 0.0, -4.0},
    av41[3]= {-4.0, -6.0, -4.0},
    av42[3]= {4.0, -6.0, 4.0},
    av43[3]= {4.0, 0.0, 4.0},
    av44[3]= {4.0, 2.0, 3.464},
    av45[3]= {4.0, 3.464, 2.0},
    av46[3]= {4.0, 4.0, 0.0},
    av47[3]= {4.0, 3.464, -2.0},
    av48[3]= {4.0, 2.0, -3.464},
```

```
    av49[3]= {4.0, 0.0, -4.0},
    av50[3]= {4.0, -6.0, -4.0},
    av51[3]= {6.0, -6.0, 4.0},
    av52[3]= {6.0, 0.0, 4.0},
    av53[3]= {6.0, 2.0, 3.464},
    av54[3]= {6.0, 3.464, 2.0},
    av55[3]= {6.0, 4.0, 0.0},
    av56[3]= {6.0, 3.464, -2.0},
    av57[3]= {6.0, 2.0, -3.464},
    av58[3]= {6.0, 0.0, -4.0},
    av59[3]= {6.0, -6.0, -4.0};


float an1[3]= {-0.5, 0.0, -0.866},
    an2[3]= {-0.866, 0.0, -0.5},
    an4[3]= {-0.866, 0.0, 0.5},
    an5[3]= {-0.5, 0.0, 0.866},
    an7[3]= {0.5, 0.0, 0.866},
    an8[3]= {0.866, 0.0, 0.5},
    an10[3]= {0.866, 0.0,-0.5},
    an11[3]= {0.5, 0.0, -0.866},
    an26[3]= {0.0, 0.5, 0.866},
    an27[3]= {0.0, 0.866, 0.5},
    an29[3]= {0.0, 0.866, -0.5},
    an30[3]= {0.0, 0.5, -0.866};

/************************** Data for link 2 ***********************************/

float bv0[3]= {-3.9, 3.75, -2.525},
    bv1[3]= {-3.9, 3.75, 0.0},
    bv2[3]= {-3.9, 3.248, 1.875},
    bv3[3]= {-3.9, 1.875, 3.248},
    bv4[3]= {-3.9, 0.0, 3.75},
    bv5[3]= {-3.9, -1.875, 3.248},
    bv6[3]= {-3.9, -3.248, 1.875},
    bv7[3]= {-3.9, -3.75, 0.0},
    bv8[3]= {-3.9, -3.248, -1.875},
    bv9[3]= {-3.9, 3.75, -8.725},
    bv10[3]= {3.9, 3.75, -2.525},
    bv11[3]= {3.9, 3.75, 0.0},
    bv12[3]= {3.9, 3.248, 1.875},
    bv13[3]= {3.9, 1.875, 3.248},
    bv14[3]= {3.9, 0.0, 3.75},
    bv15[3]= {3.9, -1.875, 3.248},
    bv16[3]= {3.9, -3.248, 1.875},
    bv17[3]= {3.9, -3.75, 0.0},
    bv18[3]= {3.9, -3.248, -1.875},
    bv19[3]= {3.9, 3.75, -8.725},
    bv20[3]= {0.0, 3.75, -10.625},
    bv21[3]= {2.5, 3.75, -9.955},
    bv22[3]= {4.33, 3.75, -8.125},
```

```
    bv23[3]= {5.0, 3.75, -5.625},
    bv24[3]= {4.33, 3.75, -3.125},
    bv25[3]= {2.5, 3.75, -1.295},
    bv26[3]= {0.0, 3.75, -0.625},
    bv27[3]= {-2.5, 3.75, -1.295},
    bv28[3]= {-4.33, 3.75, -3.125},
    bv29[3]= {-5.0, 3.75, -5.625},
    bv30[3]= {-4.33, 3.75, -8.125},
    bv31[3]= {-2.5, 3.75, -9.955},
    bv32[3]= {0.0, 7.2, -10.625},
    bv33[3]= {2.5, 7.2, -9.955},
    bv34[3]= {4.33, 7.2, -8.125},
    bv35[3]= {5.0, 7.2, -5.625},
    bv36[3]= {4.33, 7.2, -3.125},
    bv37[3]= {2.5, 7.2, -1.295},
    bv38[3]= {0.0, 7.2, -0.625},
    bv39[3]= {-2.5, 7.2, -1.295},
    bv40[3]= {-4.33, 7.2, -3.125},
    bv41[3]= {-5.0, 7.2, -5.625},
    bv42[3]= {-4.33, 7.2, -8.125},
    bv43[3]= {-2.5, 7.2, -9.955};


float bn2[3]= {0.0, 0.866,0.5},
    bn3[3]= {0.0, 0.5, 0.866},
    bn5[3]= {0.0, -0.5, 0.866},
    bn6[3]= {0.0, -0.866, 0.5},
    bn8[3]= {0.0, -0.866, -0.5},
    bn9[3]= {0.0, -0.7793, -0.6266},
    bn21[3]= {0.5, 0.0, -0.866},
    bn22[3]= {0.866, 0.0, -0.5},
    bn24[3]= {0.866, 0.0, 0.5},
    bn25[3]= {0.5, 0.0, 0.866},
    bn27[3]= {-0.5, 0.0, 0.866},
    bn28[3]= {-0.866, 0.0, 0.5},
    bn30[3]= {-0.866, 0.0, -0.5},
    bn31[3]= {-0.5, 0.0, -0.866};

/************************** Data for link 3 **********************************/

float cv0[3]= {0.0, 7.3, -5.0},
    cv1[3]= {2.5, 7.3, -4.33},
    cv2[3]= {4.33, 7.3, -2.5},
    cv3[3]= {5.0, 7.3, 0.0},
    cv4[3]= {4.33, 7.3, 2.5},
    cv5[3]= {2.5, 7.3, 4.33},
    cv6[3]= {0.0, 7.3, 5.0},
    cv7[3]= {-2.5, 7.3, 4.33},
    cv8[3]= {-4.33, 7.3, 2.5},
    cv9[3]= {-5.0, 7.3, 0.0},
    cv10[3]= {-4.33, 7.3, -2.5},
```

```
cv11[3]= {-2.5, 7.3, -4.33},
cv12[3]= {0.0, 34.235, -4.25},
cv13[3]= {2.125, 34.235, -3.681},
cv14[3]= {3.681, 34.235, -2.125},
cv15[3]= {4.25, 34.235, 0.0},
cv16[3]= {3.681, 34.235, 2.125},
cv17[3]= {2.125, 34.235, 3.681},
cv18[3]= {0.0, 34.235, 4.25},
cv19[3]= {-2.125, 34.235, 3.681},
cv20[3]= {-3.681, 34.235, 2.125},
cv21[3]= {-4.25, 34.235, 0.0},
cv22[3]= {-3.681, 34.235, -2.125},
cv23[3]= {-2.125, 34.235, -3.681},
cv24[3]= {-5.0, 34.235, -4.25},
cv25[3]= {-5.0, 34.737, -6.125},
cv26[3]= {-5.0, 36.11, -7.498},
cv27[3]= {-5.0, 37.985, -8.0},
cv28[3]= {-5.0, 39.86, -7.498},
cv29[3]= {-5.0, 41.232, -6.125},
cv30[3]= {-5.0, 41.735, -4.25},
cv31[3]= {-5.0, 41.232, -2.375},
cv32[3]= {-5.0, 39.86, -1.002},
cv33[3]= {-3.0, 34.235, 2.125},
cv34[3]= {-3.0, 34.235, -4.25},
cv35[3]= {-3.0, 34.737, -6.125},
cv36[3]= {-3.0, 36.11, -7.498},
cv37[3]= {-3.0, 37.985, -8.0},
cv38[3]= {-3.0, 39.86, -7.498},
cv39[3]= {-3.0, 41.232, -6.125},
cv40[3]= {-3.0, 41.735, -4.25},
cv41[3]= {-3.0, 41.232, -2.375},
cv42[3]= {-3.0, 39.86, -1.002},
cv43[3]= {3.0, 34.235, 2.125},
cv44[3]= {3.0, 34.235, -4.25},
cv45[3]= {3.0, 34.737, -6.125},
cv46[3]= {3.0, 36.11, -7.498},
cv47[3]= {3.0, 37.985, -8.0},
cv48[3]= {3.0, 39.86, -7.498},
cv49[3]= {3.0, 41.232, -6.125},
cv50[3]= {3.0, 41.735, -4.25},
cv51[3]= {3.0, 41.232, -2.375},
cv52[3]= {3.0, 39.86, -1.002},
cv53[3]= {5.0, 34.235, -4.25},
cv54[3]= {5.0, 34.737, -6.125},
cv55[3]= {5.0, 36.11, -7.498},
cv56[3]= {5.0, 37.985, -8.0},
cv57[3]= {5.0, 39.86, -7.498},
cv58[3]= {5.0, 41.232, -6.125},
cv59[3]= {5.0, 41.735, -4.25},
cv60[3]= {5.0, 41.232, -2.375},
cv61[3]= {5.0, 39.86, -1.002};
```

```
float cn0[3]= {0.0, 0.0329, -0.9994},
      cn1[3]= {0.4997, 0.0329, -0.8656},
      cn2[3]= {0.8656, 0.0329, -0.4997},
      cn3[3]= {0.9994, 0.0329, 0.0},
      cn4[3]= {0.8656, 0.0329, 0.4997},
      cn5[3]= {0.4997, 0.0329, 0.8656},
      cn6[3]= {0.0, 0.0329, 0.9994},
      cn7[3]= {-0.4997, 0.0329, 0.8636},
      cn8[3]= {-0.8656, 0.0329, 0.4997},
      cn9[3]= {-0.9994, 0.0329, 0.0},
      cn10[3]= {-0.8656, 0.0329, -0.4997},
      cn11[3]= {-0.4997, 0.0329, -0.8656},
      cn16[3]= {0.9898, -0.0506, 0.0876},
      cn20[3]= {-0.9898, -0.0506, 0.0876},
      cn25[3]= {0.0, -0.866, -0.5},
      cn26[3]= {0.0, -0.5, -0.866},
      cn28[3]= {0.0, 0.5, -0.866},
      cn29[3]= {0.0, 0.866, -0.5},
      cn31[3]= {0.0, 0.866, 0.5},
      cn32[3]= {0.0, 0.5, 0.866};

/************************** Data for link 4 **********************************/

float dv0[3]= {2.9, 3.5, 1.7},
      dv1[3]= {2.9, 3.5, 0.0},
      dv2[3]= {2.9, 3.031, -1.75},
      dv3[3]= {2.9, 1.75, -3.031},
      dv4[3]= {2.9, 0.0, -3.5},
      dv5[3]= {2.9, -1.875, -3.031},
      dv6[3]= {2.9, -3.248, -1.75},
      dv7[3]= {2.9, -3.5, 0.0},
      dv8[3]= {2.9, -3.031, 1.75},
      dv9[3]= {2.9, 3.5, 6.495},
      dv10[3]= {-2.9, 3.5, 1.7},
      dv11[3]= {-2.9, 3.5, 0.0},
      dv12[3]= {-2.9, 3.031, -1.75},
      dv13[3]= {-2.9, 1.75, -3.031},
      dv14[3]= {-2.9, 0.0, -3.5},
      dv15[3]= {-2.9, -1.875, -3.031},
      dv16[3]= {-2.9, -3.031, -1.75},
      dv17[3]= {-2.9, -3.5, 0.0},
      dv18[3]= {-2.9, -3.031, 1.75},
      dv19[3]= {-2.9, 3.5, 6.495},
      dv20[3]= {0.0, 3.5, 8.25},
      dv21[3]= {-2.0, 3.5, 7.714},
      dv22[3]= {-3.464, 3.5, 6.25},
      dv23[3]= {-4.0, 3.5, 4.25},
      dv24[3]= {-3.464, 3.5, 2.25},
      dv25[3]= {-2.0, 3.5, 0.786},
      dv26[3]= {0.0, 3.5, 0.25},
```

```
    dv27[3]= {2.0, 3.5, 0.786},
    dv28[3]= {3.464, 3.5, 2.25},
    dv29[3]= {4.0, 3.5, 4.25},
    dv30[3]= {3.464, 3.5, 6.25},
    dv31[3]= {2.0, 3.5, 7.714},
    dv32[3]= {0.0, 5.4, 8.25},
    dv33[3]= {-2.0, 5.4, 7.714},
    dv34[3]= {-3.464, 5.4, 6.25},
    dv35[3]= {-4.0, 5.4, 4.25},
    dv36[3]= {-3.464, 5.4, 2.25},
    dv37[3]= {-2.0, 5.4, 0.786},
    dv38[3]= {0.0, 5.4, 0.25},
    dv39[3]= {2.0, 5.4, 0.786},
    dv40[3]= {3.464, 5.4, 2.25},
    dv41[3]= {4.0, 5.4, 4.25},
    dv42[3]= {3.464, 5.4, 6.25},
    dv43[3]= {2.0, 5.4, 7.714};


float dn2[3]= {0.0, 0.866, -0.5},
    dn3[3]= {0.0, 0.5, -0.866},
    dn5[3]= {0.0, -0.5, -0.866},
    dn6[3]= {0.0, -0.866, -0.5},
    dn8[3]= {0.0, -0.866, 0.5},
    dn9[3]= {0.0, -0.7793, 0.6266},
    dn21[3]= {-0.5, 0.0, 0.866},
    dn22[3]= {-0.866, 0.0, 0.5},
    dn24[3]= {-0.866, 0.0, -0.5},
    dn25[3]= {-0.5, 0.0, -0.866},
    dn27[3]= {0.5, 0.0, -0.866},
    dn28[3]= {0.866, 0.0, -0.5},
    dn30[3]= {0.866, 0.0, 0.5},
    dn31[3]= {0.5, 0.0, 0.866};


/************************** Data for link 5 **********************************/

float ev0[3]= {0.0, 5.5, -4.0},
    ev1[3]= {2.0, 5.5, -3.464},
    ev2[3]= {3.464, 5.5, -2.0},
    ev3[3]= {4.0, 5.5, 0.0},
    ev4[3]= {3.464, 5.5, 2.0},
    ev5[3]= {2.0, 5.5, 3.464},
    ev6[3]= {0.0, 5.5, 4.0},
    ev7[3]= {-2.0, 5.5, 3.464},
    ev8[3]= {-3.464, 5.5, 2.0},
    ev9[3]= {-4.0, 5.5, 0.0},
    ev10[3]= {-3.464, 5.5, -2.0},
    ev11[3]= {-2.0, 5.5, -3.464},
    ev12[3]= {0.0, 36.246, -1.75},
    ev13[3]= {0.875, 36.246, -1.516},
```

ev14[3]= {1.516, 36.246, -0.875},
ev15[3]= {1.75, 36.246, 0.0},
ev16[3]= {1.516, 36.246, 0.875},
ev17[3]= {0.875, 36.246, 1.516},
ev18[3]= {0.0, 36.246, 1.75},
ev19[3]= {-0.875, 36.246, 1.516},
ev20[3]= {-1.516, 36.246, 0.875},
ev21[3]= {-1.75, 36.246, 0.0},
ev22[3]= {-1.516, 36.246, -0.875},
ev23[3]= {-0.875, 36.246, -1.516},
ev24[3]= {-2.5, 36.246, -1.937},
ev25[3]= {-2.5, 36.48, -2.812},
ev26[3]= {-2.5, 37.121, -3.453},
ev27[3]= {-2.5, 37.996, -3.687},
ev28[3]= {-2.5, 38.871, -3.453},
ev29[3]= {-2.5, 39.512, -2.812},
ev30[3]= {-2.5, 39.746, -1.937},
ev31[3]= {-2.5, 39.512, -1.062},
ev32[3]= {-2.5, 38.871, -0.421},
ev33[3]= {-1.4, 36.246, 0.871},
ev34[3]= {-1.4, 36.246, -1.937},
ev35[3]= {-1.4, 36.48, -2.812},
ev36[3]= {-1.4, 37.121, -3.453},
ev37[3]= {-1.4, 37.996, -3.687},
ev38[3]= {-1.4, 38.871, -3.453},
ev39[3]= {-1.4, 39.512, -2.812},
ev40[3]= {-1.4, 39.746, -1.937},
ev41[3]= {-1.4, 39.512, -1.062},
ev42[3]= {-1.4, 38.871, -0.421},
ev43[3]= {1.4, 36.246, 0.871},
ev44[3]= {1.4, 36.246, -1.937},
ev45[3]= {1.4, 36.48, -2.812},
ev46[3]= {1.4, 37.121, -3.453},
ev47[3]= {1.4, 37.996, -3.687},
ev48[3]= {1.4, 38.871, -3.453},
ev49[3]= {1.4, 39.512, -2.812},
ev50[3]= {1.4, 39.746, -1.937},
ev51[3]= {1.4, 39.512, -1.062},
ev52[3]= {1.4, 38.871, -0.421},
ev53[3]= {2.5, 36.246, -1.937},
ev54[3]= {2.5, 36.48, -2.812},
ev55[3]= {2.5, 37.121, -3.453},
ev56[3]= {2.5, 37.996, -3.687},
ev57[3]= {2.5, 38.871, -3.453},
ev58[3]= {2.5, 39.512, -2.812},
ev59[3]= {2.5, 39.746, -1.937},
ev60[3]= {2.5, 39.512, -1.062},
ev61[3]= {2.5, 38.871, -0.421};

float en0[3]= {0.0, 0.0748, -0.9972},

```
    en1[3]= {0.4986, 0.0748, -0.8636},
    en2[3]= {0.8636, 0.0748, -0.4986},
    en3[3]= {0.9972, 0.0748, 0.0},
    en4[3]= {0.8636, 0.0748, 0.4986},
    en5[3]= {0.4986, 0.0748, 0.8636},
    en6[3]= {0.0, 0.0748, 0.9972},
    en7[3]= {-0.4986, 0.0748, 0.8636},
    en8[3]= {-0.8636, 0.0748, 0.4986},
    en9[3]= {-0.9972, 0.0748, 0.0},
    en10[3]= {-0.8636, 0.0748, -0.4986},
    en11[3]= {-0.4986, 0.0748, -0.8636},
    en16[3]= {0.9898, -0.0506, 0.0876},
    en20[3]= {-0.9898, -0.0506, 0.0876},
    en25[3]= {0.0, -0.866, -0.5},
    en26[3]= {0.0, -0.5, -0.866},
    en28[3]= {0.0, 0.5, -0.866},
    en29[3]= {0.0, 0.866, -0.5},
    en31[3]= {0.0, 0.866, 0.5},
    en32[3]= {0.0, 0.5, 0.866};


/************************** Data for link 6 **********************************/

float fv0[3]= {-1.366, 1.75, 0.0},
    fv1[3]= {-1.366, 1.516, -0.875},
    fv2[3]= {-1.366, 0.875, -1.516},
    fv3[3]= {-1.366, 0.0, -1.75},
    fv4[3]= {-1.366, -0.875, -1.516},
    fv5[3]= {-1.366, -1.516, -0.875},
    fv6[3]= {-1.366, -1.75, 0.0},
    fv7[3]= {-1.366, -1.516, 0.875},
    fv8[3]= {-1.366, -0.875, 1.516},
    fv9[3]= {-1.366, 1.75, 3.031},
    fv10[3]= {1.366, 1.75, 0.0},
    fv11[3]= {1.366, 1.516, -0.875},
    fv12[3]= {1.366, 0.875, -1.516},
    fv13[3]= {1.366, 0.0, -1.75},
    fv14[3]= {1.366, -0.875, -1.516},
    fv15[3]= {1.366, -1.516, -0.875},
    fv16[3]= {1.366, -1.75, 0.0},
    fv17[3]= {1.366, -1.516, 0.875},
    fv18[3]= {1.366, -0.875, 1.516},
    fv19[3]= {1.366, 1.75, 3.031},
    fv20[3]= {0.0, 1.75, 3.687},
    fv21[3]= {0.875, 1.75, 3.453},
    fv22[3]= {1.516, 1.75, 2.812},
    fv23[3]= {1.75, 1.75, 1.937},
    fv24[3]= {1.516, 1.75, 1.062},
    fv25[3]= {0.875, 1.75, 0.421},
    fv26[3]= {0.0, 1.75, 0.187},
    fv27[3]= {-0.875, 1.75, 0.421},
```

```
    fv28[3]= {-1.516, 1.75, 1.062},
    fv29[3]= {-1.75, 1.75, 1.973},
    fv30[3]= {-1.516, 1.75, 2.812},
    fv31[3]= {-0.875, 1.75, 3.453},
    fv32[3]= {0.0, 4.78, 3.687},
    fv33[3]= {0.875, 4.78, 3.453},
    fv34[3]= {1.516, 4.78, 2.812},
    fv35[3]= {1.75, 4.78, 1.937},
    fv36[3]= {1.516, 4.78, 1.062},
    fv37[3]= {0.875, 4.78, 0.421},
    fv38[3]= {0.0, 4.78, 0.187},
    fv39[3]= {-0.875, 4.78, 0.421},
    fv40[3]= {-1.516, 4.78, 1.062},
    fv41[3]= {-1.75, 4.78, 1.973},
    fv42[3]= {-1.516, 4.78, 2.812},
    fv43[3]= {-0.875, 4.78, 3.453};

float fn1[3]= {0.0, 0.866, -0.5},
    fn2[3]= {0.0, 0.5, -0.866},
    fn4[3]= {0.0, -0.5, -0.866},
    fn5[3]= {0.0, -0.866, -0.5},
    fn7[3]= {0.0, -0.866, 0.5},
    fn8[3]= {0.0, -0.5, 0.866},
    fn21[3]= {0.5, 0.0, 0.866},
    fn22[3]= {0.866, 0.0, 0.5},
    fn24[3]= {0.866, 0.0, -0.5},
    fn25[3]= {0.5, 0.0, -0.866},
    fn27[3]= {-0.5, 0.0, -0.866},
    fn28[3]= {-0.866, 0.0, -0.5},
    fn30[3]= {-0.866, 0.0, 0.5},
    fn31[3]= {-0.5, 0.0, 0.866};


/************************ Data for the graper ******************************/

float gv0[3]= {0.0, -7.0, 1.5},
    gv1[3]= {0.75, -7.0, 1.299},
    gv2[3]= {1.299, -7.0, 0.75},
    gv3[3]= {1.5, -7.0, 0.0},
    gv4[3]= {1.299, -7.0, -0.75},
    gv5[3]= {0.75, -7.0, -1.299},
    gv6[3]= {0.0, -7.0, -1.5},
    gv7[3]= {-0.75, -7.0, -1.299},
    gv8[3]= {-1.299, -7.0, -0.75},
    gv9[3]= {-1.5, -7.0, 0.0},
    gv10[3]= {-1.299, -7.0, 0.75},
    gv11[3]= {-0.75, -7.0, 1.299},
    gv12[3]= {0.0, -3.0, 1.5},
    gv13[3]= {0.75, -3.0, 1.299},
    gv14[3]= {1.299, -3.0, 0.75},
    gv15[3]= {1.5, -3.0, 0.0},
```

```
    gv16[3]= {1.299, -3.0, -0.75},
    gv17[3]= {0.75, -3.0, -1.299},
    gv18[3]= {0.0, -3.0, -1.5},
    gv19[3]= {-0.75, -3.0, -1.299},
    gv20[3]= {-1.299, -3.0, -0.75},
    gv21[3]= {-1.5, -3.0, 0.0},
    gv22[3]= {-1.299, -3.0, 0.75},
    gv23[3]= {-0.75, -3.0, 1.299},
    gv24[3]= {-4.0, -3.0, 1.5},
    gv25[3]= {4.0, -3.0, 1.5},
    gv26[3]= {4.0, -3.0, -1.5},
    gv27[3]= {-4.0, -3.0, -1.5},
    gv28[3]= {-4.0, -2.0, 1.5},
    gv29[3]= {4.0, -2.0, 1.5},
    gv30[3]= {4.0, -2.0, 1.0},
    gv31[3]= {-4.0, -2.0, 1.0},
    gv32[3]= {-4.0, -2.5, 1.0},
    gv33[3]= {4.0, -2.5, 1.0},
    gv34[3]= {4.0, -2.5, -1.0},
    gv35[3]= {-4.0, -2.5, -1.0},
    gv36[3]= {-4.0, -2.0, -1.0},
    gv37[3]= {4.0, -2.0, -1.0},
    gv38[3]= {4.0, -2.0, -1.5},
    gv39[3]= {-4.0, -2.0, -1.5};

float gn1[3]= {0.5, 0.0, 0.866},
    gn2[3]= {0.866, 0.0, 0.5},
    gn4[3]= {0.866, 0.0, -0.5},
    gn5[3]= {0.5, 0.0, -0.866},
    gn7[3]= {-0.5, 0.0, -0.866},
    gn8[3]= {-0.866, 0.0, -0.5},
    gn10[3]= {-0.866, 0.0, 0.5},
    gn11[3]= {-0.5, 0.0, 0.866};


float ggv0[3]= {0.0, 0.0, 0.8},
    ggv1[3]= {2.0, 0.0, 0.8},
    ggv2[3]= {1.0, 4.5, 0.8},
    ggv3[3]= {0.0, 4.5, 0.8},
    ggv4[3]= {0.0, 0.0, -0.8},
    ggv5[3]= {2.0, 0.0, -0.8},
    ggv6[3]= {1.0, 4.5, -0.8},
    ggv7[3]= {0.0, 4.5, -0.8};

float ggn[3]= {0.9762, 0.2169, 0.0};


/****************************** Draw the base ***************************/

drawlink0()
{
```

```
    bgnpolygon();
    n3f(n1); v3f(v0); v3f(v9); v3f(v10); v3f(v1);
    endpolygon();
    bgnpolygon();
    v3f(v1); v3f(v10); n3f(n2); v3f(v11); v3f(v2);
    endpolygon();
    bgnpolygon();
    v3f(v2); v3f(v11); n3f(n3); v3f(v12); v3f(v3);
    endpolygon();
    bgnpolygon();
    v3f(v3); v3f(v12); n3f(nz1); v3f(v13); v3f(v4);
    endpolygon();
    bgnpolygon();
    v3f(v4); v3f(v13); n3f(n5); v3f(v14); v3f(v5);
    endpolygon();
    bgnpolygon();
    v3f(v5); v3f(v14); n3f(n6); v3f(v15); v3f(v6);
    endpolygon();
    bgnpolygon();
    v3f(v6); v3f(v15); n3f(n7); v3f(v16); v3f(v7);
    endpolygon();
    bgnpolygon();
    v3f(v7); v3f(v16); v3f(v17); v3f(v8);
    endpolygon();
    bgnpolygon();
    n3f(ny1); v3f(v0); v3f(v1); v3f(v2); v3f(v3);
    v3f(v4); v3f(v5); v3f(v6); v3f(v7); v3f(v8);
    endpolygon();
    bgnpolygon();
    n3f(ny2); v3f(v9); v3f(v10); v3f(v11); v3f(v12);
    v3f(v13); v3f(v14); v3f(v15); v3f(v16); v3f(v17);
    endpolygon();
    return;
}

/****************************** Draw link 1 ******************************/

drawlink1()
{
    bgnpolygon();
    n3f(nz2); v3f(av0); v3f(av12);n3f(an1);
    v3f(av13); v3f(av1);
    endpolygon();
    bgnpolygon();
    v3f(av1); v3f(av13); n3f(an2); v3f(av14); v3f(av2);
    endpolygon();
    bgnpolygon();
    v3f(av2); v3f(av14); n3f(nx2); v3f(av15); v3f(av3);
    endpolygon();
    bgnpolygon();
```

```
v3f(av3); v3f(av15); n3f(an4); v3f(av16);v3f(av4);
endpolygon();
bgnpolygon();
v3f(av4); v3f(av16); n3f(an5); v3f(av17); v3f(av5);
endpolygon();
bgnpolygon();
v3f(av5); v3f(av17); n3f(nz1);  v3f(av18); v3f(av6);
endpolygon();
bgnpolygon();
v3f(av6); v3f(av18); n3f(an7); v3f(av19); v3f(av7);
endpolygon();
bgnpolygon();
v3f(av7); v3f(av19); n3f(an8); v3f(av20); v3f(av8);
endpolygon();
bgnpolygon();
v3f(av8); v3f(av20); n3f(nx1); v3f(av21); v3f(av9);
endpolygon();
bgnpolygon();
v3f(av9); v3f(av21); n3f(an10); v3f(av22); v3f(av10);
endpolygon();
bgnpolygon();
v3f(av10); v3f(av22); n3f(an11); v3f(av23); v3f(av11);
endpolygon();
bgnpolygon();
v3f(av11); v3f(av23); n3f(nz2); v3f(av12); v3f(av0);
endpolygon();
bgnpolygon();
n3f(ny1); v3f(av12); v3f(av13); v3f(av14); v3f(av15);
v3f(av16); v3f(av17); v3f(av18); v3f(av19); v3f(av20);
v3f(av21); v3f(av22); v3f(av23); v3f(av23);
endpolygon();

bgnpolygon();
n3f(nx2); v3f(av24); v3f(av25); v3f(av26); v3f(av27); v3f(av28);
v3f(av29); v3f(av30); v3f(av31); v3f(av32);
endpolygon();
bgnpolygon();
n3f(nz1); v3f(av24); v3f(av25); v3f(av34); v3f(av33);
endpolygon();
bgnpolygon();
n3f(nz1); v3f(av25); v3f(av34);
n3f(an26); v3f(av35); v3f(av26);
endpolygon();
bgnpolygon();
v3f(av26); v3f(av35); n3f(an27);
v3f(av36); v3f(av27);
endpolygon();
bgnpolygon();
v3f(av27); v3f(av36); n3f(ny1); v3f(av37); v3f(av28);
endpolygon();
bgnpolygon();
```

```
    v3f(av28); v3f(av37); n3f(an29); v3f(av38); v3f(av29);
    endpolygon();
    bgnpolygon();
    v3f(av29); v3f(av38); n3f(an30); v3f(av39); v3f(av30);
    endpolygon();
    bgnpolygon();
    v3f(av30); v3f(av39); n3f(nz2); v3f(av40); v3f(av31);
    endpolygon();
    bgnpolygon();
    v3f(av31); v3f(av40); v3f(av41); v3f(av32);
    endpolygon();
    bgnpolygon();
    n3f(nx1); v3f(av33); v3f(av34); v3f(av35); v3f(av36);
    v3f(av37); v3f(av38); v3f(av39); v3f(av40); v3f(av41);
    endpolygon();
    bgnpolygon();
    n3f(nx2); v3f(av42); v3f(av43); v3f(av44); v3f(av45);
    v3f(av46); v3f(av47); v3f(av48); v3f(av49); v3f(av50);
    endpolygon();
    bgnpolygon();
    n3f(nz1); v3f(av42); v3f(av43); v3f(av52); v3f(av51);
    endpolygon();
    bgnpolygon();
    v3f(av43); v3f(av52); n3f(an26); v3f(av53); v3f(av44);
    endpolygon();
    bgnpolygon();
    v3f(av44); v3f(av53); n3f(an27); v3f(av54); v3f(av45);
    endpolygon();
    bgnpolygon();
    v3f(av45); v3f(av54); n3f(ny1); v3f(av55); v3f(av46);
    endpolygon();
    bgnpolygon();
    v3f(av46); v3f(av55); n3f(an29); v3f(av56); v3f(av47);
    endpolygon();
    bgnpolygon();
    v3f(av47); v3f(av56); n3f(an30); v3f(av57); v3f(av48);
    endpolygon();
    bgnpolygon();
    v3f(av48); v3f(av57); n3f(nz2); v3f(av58); v3f(av49);
    endpolygon();
    bgnpolygon();
    v3f(av49); v3f(av58); v3f(av59); v3f(av50);
    endpolygon();
    bgnpolygon();
    n3f(nx1); v3f(av51); v3f(av52); v3f(av53); v3f(av54);
    v3f(av55); v3f(av56); v3f(av57); v3f(av58); v3f(av59);
    endpolygon();
    return;
}
```

```
/****************************** Draw link 2 ********************************/

drawlink2()
{
    bgnpolygon();
    n3f(ny1); v3f(bv0); v3f(bv10);
    v3f(bv11); v3f(bv1);
    endpolygon();
    bgnpolygon();
    v3f(bv1); v3f(bv11); n3f(bn2); v3f(bv12); v3f(bv2);
    endpolygon();
    bgnpolygon();
    v3f(bv2); v3f(bv12);n3f(bn3); v3f(bv13); v3f(bv3);
    endpolygon();
    bgnpolygon();
    v3f(bv3); v3f(bv13); n3f(nz1); v3f(bv14); v3f(bv4);
    endpolygon();
    bgnpolygon();
    v3f(bv4); v3f(bv14); n3f(bn5); v3f(bv15); v3f(bv5);
    endpolygon();
    bgnpolygon();
    v3f(bv5); v3f(bv15); n3f(bn6);  v3f(bv16); v3f(bv6);
    endpolygon();
    bgnpolygon();
    v3f(bv6); v3f(bv16); n3f(ny2); v3f(bv17); v3f(bv7);
    endpolygon();
    bgnpolygon();
    v3f(bv7); v3f(bv17); n3f(bn8); v3f(bv18); v3f(bv8);
    endpolygon();
    bgnpolygon();
    n3f(bn9); v3f(bv8); v3f(bv18); v3f(bv19); v3f(bv9);
    endpolygon();

    bgnpolygon();
    n3f(nz2); v3f(bv20); v3f(bv32);
    n3f(bn21); v3f(bv33); v3f(bv21);
    endpolygon();
    bgnpolygon();
    v3f(bv21); v3f(bv33); n3f(bn22); v3f(bv34); v3f(bv22);
    endpolygon();
    bgnpolygon();
    v3f(bv22); v3f(bv34);n3f(nx1); v3f(bv35); v3f(bv23);
    endpolygon();
    bgnpolygon();
    v3f(bv23); v3f(bv35); n3f(bn24); v3f(bv36); v3f(bv24);
    endpolygon();
    bgnpolygon();
    v3f(bv24); v3f(bv36); n3f(bn25); v3f(bv37); v3f(bv25);
    endpolygon();
    bgnpolygon();
    v3f(bv25); v3f(bv37); n3f(nz1);  v3f(bv38); v3f(bv26);
```

```
    endpolygon();
    bgnpolygon();
    v3f(bv26); v3f(bv38); n3f(bn27); v3f(bv39); v3f(bv27);
    endpolygon();
    bgnpolygon();
    v3f(bv27); v3f(bv39); n3f(bn28); v3f(bv40); v3f(bv28);
    endpolygon();
    bgnpolygon();
    v3f(bv28); v3f(bv40); n3f(nx2); v3f(bv41); v3f(bv29);
    endpolygon();
    bgnpolygon();
    v3f(bv29); v3f(bv41); n3f(bn30); v3f(bv42); v3f(bv30);
    endpolygon();
    bgnpolygon();
    v3f(bv30); v3f(bv42); n3f(bn31); v3f(bv43); v3f(bv31);
    endpolygon();
    bgnpolygon();
    v3f(bv31); v3f(bv43); n3f(nz2); v3f(bv32); v3f(bv20);
    endpolygon();

    bgnpolygon();
    n3f(nx2); v3f(bv1); v3f(bv8); v3f(bv9);
    endpolygon();
    bgnpolygon();
    n3f(nx1); v3f(bv11); v3f(bv18); v3f(bv19);
    endpolygon();
    bgnpolygon();
    n3f(ny2); v3f(bv20); v3f(bv21); v3f(bv22);
    v3f(bv23); v3f(bv24); v3f(bv25);
    v3f(bv26); v3f(bv27); v3f(bv28);
    v3f(bv29); v3f(bv30); v3f(bv31);
    endpolygon();
    return;
}


/****************************** Draw link 3 *********************************/

drawlink30
{
    bgnpolygon();
    n3f(cn0); v3f(cv0); v3f(cv12);
    n3f(cn1); v3f(cv13); v3f(cv1);
    endpolygon();
    bgnpolygon();
    v3f(cv1); v3f(cv13); n3f(cn2); v3f(cv14); v3f(cv2);
    endpolygon();
    bgnpolygon();
    v3f(cv2); v3f(cv14); n3f(cn3); v3f(cv15); v3f(cv3);
    endpolygon();
    bgnpolygon();
```

```
v3f(cv3); v3f(cv15); n3f(cn4); v3f(cv16); v3f(cv4);
endpolygon();
bgnpolygon();
v3f(cv4); v3f(cv16); n3f(cn5); v3f(cv17); v3f(cv5);
endpolygon();
bgnpolygon();
v3f(cv5); v3f(cv17); n3f(cn6);  v3f(cv18); v3f(cv6);
endpolygon();
bgnpolygon();
v3f(cv6); v3f(cv18); n3f(cn7); v3f(cv19); v3f(cv7);
endpolygon();
bgnpolygon();
v3f(cv7); v3f(cv19); n3f(cn8); v3f(cv20); v3f(cv8);
endpolygon();
bgnpolygon();
v3f(cv8); v3f(cv20); n3f(cn9); v3f(cv21); v3f(cv9);
endpolygon();
bgnpolygon();
v3f(cv9); v3f(cv21); n3f(cn10); v3f(cv22); v3f(cv10);
endpolygon();
bgnpolygon();
v3f(cv10); v3f(cv22); n3f(cn11);  v3f(cv23); v3f(cv11);
endpolygon();
bgnpolygon();
v3f(cv11); v3f(cv23); n3f(cn0);  v3f(cv12); v3f(cv0);
endpolygon();
bgnpolygon();
n3f(ny1); v3f(cv12); v3f(cv13); v3f(cv14); v3f(cv15);
v3f(cv16); v3f(cv17); v3f(cv18); v3f(cv19); v3f(cv20);
v3f(cv21); v3f(cv22); v3f(cv23);
endpolygon();
bgnpolygon();
n3f(nx2); v3f(cv24); v3f(cv25); v3f(cv26); v3f(cv27);
v3f(cv28); v3f(cv29); v3f(cv30); v3f(cv31); v3f(cv32);
endpolygon();
bgnpolygon();
n3f(nx1); v3f(cv33); v3f(cv34); v3f(cv35); v3f(cv36); v3f(cv37);
v3f(cv38); v3f(cv39); v3f(cv40); v3f(cv41); v3f(cv42);
endpolygon();
bgnpolygon();
n3f(nx2); v3f(cv43); v3f(cv44); v3f(cv45); v3f(cv46); v3f(cv47);
v3f(cv48); v3f(cv49); v3f(cv50); v3f(cv51); v3f(cv52);
endpolygon();
bgnpolygon();
n3f(nx1); v3f(cv53); v3f(cv54); v3f(cv55); v3f(cv56);
v3f(cv57); v3f(cv58); v3f(cv59); v3f(cv60); v3f(cv61);
endpolygon();
bgnpolygon();
n3f(ny2); v3f(cv20); v3f(cv33); v3f(cv34); v3f(cv24);
endpolygon();
bgnpolygon();
```

```
v3f(cv24); v3f(cv34); n3f(cn25); v3f(cv35); v3f(cv25);
endpolygon();
bgnpolygon();
v3f(cv25); v3f(cv35); n3f(cn26); v3f(cv36); v3f(cv26);
endpolygon();
bgnpolygon();
v3f(cv26); v3f(cv36); n3f(nz2); v3f(cv37); v3f(cv27);
endpolygon();
bgnpolygon();
v3f(cv27); v3f(cv37); n3f(cn28); v3f(cv38); v3f(cv28);
endpolygon();
bgnpolygon();
v3f(cv28); v3f(cv38); n3f(cn29); v3f(cv39); v3f(cv29);
endpolygon();
bgnpolygon();
v3f(cv29); v3f(cv39); n3f(ny1); v3f(cv40); v3f(cv30);
endpolygon();
bgnpolygon();
v3f(cv30); v3f(cv40); n3f(cn31); v3f(cv41); v3f(cv31);
endpolygon();
bgnpolygon();
v3f(cv31); v3f(cv41); n3f(cn32); v3f(cv42); v3f(cv32);
endpolygon();
bgnpolygon();
v3f(cv32); v3f(cv42); v3f(cv33); v3f(cv20);
endpolygon();

bgnpolygon();
n3f(ny2); v3f(cv43); v3f(cv16); v3f(cv53); v3f(cv44);
endpolygon();
bgnpolygon();
v3f(cv44); v3f(cv53); n3f(cn25); v3f(cv54); v3f(cv45);
endpolygon();
bgnpolygon();
v3f(cv45); v3f(cv54); n3f(cn26); v3f(cv55); v3f(cv46);
endpolygon();
bgnpolygon();
v3f(cv46); v3f(cv55); n3f(nz2); v3f(cv56); v3f(cv47);
endpolygon();
bgnpolygon();
v3f(cv47); v3f(cv56); n3f(cn28); v3f(cv57); v3f(cv48);
endpolygon();
bgnpolygon();
v3f(cv48); v3f(cv57); n3f(cn29); v3f(cv58); v3f(cv49);
endpolygon();
bgnpolygon();
v3f(cv49); v3f(cv58); n3f(ny1); v3f(cv59); v3f(cv50);
endpolygon();
bgnpolygon();
v3f(cv50); v3f(cv59); n3f(cn31); v3f(cv60); v3f(cv51);
endpolygon();
```

```
    bgnpolygon();
    v3f(cv51); v3f(cv60); n3f(cn32); v3f(cv61); v3f(cv52);
    endpolygon();
    bgnpolygon();
    v3f(cv52); v3f(cv61); v3f(cv16); v3f(cv43);
    endpolygon();
    bgnpolygon();
    n3f(cn20); v3f(cv20); v3f(cv24); v3f(cv32);
    endpolygon();
    bgnpolygon();
    n3f(cn16); v3f(cv16); v3f(cv53); v3f(cv61);
    endpolygon();
    return;

}


/****************************** Draw link 4 *******************************/

drawlink4()
{
    bgnpolygon();
    n3f(ny1); v3f(dv0); v3f(dv10);
    v3f(dv11); v3f(dv1);
    endpolygon();
    bgnpolygon();
    v3f(dv1); v3f(dv11); n3f(dn2); v3f(dv12); v3f(dv2);
    endpolygon();
    bgnpolygon();
    v3f(dv2); v3f(dv12);n3f(dn3); v3f(dv13); v3f(dv3);
    endpolygon();
    bgnpolygon();
    v3f(dv3); v3f(dv13); n3f(nz2); v3f(dv14); v3f(dv4);
    endpolygon();
    bgnpolygon();
    v3f(dv4); v3f(dv14); n3f(dn5); v3f(dv15); v3f(dv5);
    endpolygon();
    bgnpolygon();
    v3f(dv5); v3f(dv15); n3f(dn6);  v3f(dv16); v3f(dv6);
    endpolygon();
    bgnpolygon();
    v3f(dv6); v3f(dv16); n3f(ny2); v3f(dv17); v3f(dv7);
    endpolygon();
    bgnpolygon();
    v3f(dv7); v3f(dv17); n3f(dn8); v3f(dv18); v3f(dv8);
    endpolygon();
    bgnpolygon();
    n3f(dn9); v3f(dv8); v3f(dv18); v3f(dv19); v3f(dv9);
    endpolygon();

    bgnpolygon();
```

```
n3f(nz1); v3f(dv20); v3f(dv32);
n3f(dn21); v3f(dv33); v3f(dv21);
endpolygon();
bgnpolygon();
v3f(dv21); v3f(dv33); n3f(dn22); v3f(dv34); v3f(dv22);
endpolygon();
bgnpolygon();
v3f(dv22); v3f(dv34); n3f(nx2); v3f(dv35); v3f(dv23);
endpolygon();
bgnpolygon();
v3f(dv23); v3f(dv35); n3f(dn24); v3f(dv36); v3f(dv24);
endpolygon();
bgnpolygon();
v3f(dv24); v3f(dv36); n3f(dn25); v3f(dv37); v3f(dv25);
endpolygon();
bgnpolygon();
v3f(dv25); v3f(dv37); n3f(nz2);  v3f(dv38); v3f(dv26);
endpolygon();
bgnpolygon();
v3f(dv26); v3f(dv38); n3f(dn27); v3f(dv39); v3f(dv27);
endpolygon();
bgnpolygon();
v3f(dv27); v3f(dv39); n3f(dn28); v3f(dv40); v3f(dv28);
endpolygon();
bgnpolygon();
v3f(dv28); v3f(dv40); n3f(nx1); v3f(dv41); v3f(dv29);
endpolygon();
bgnpolygon();
v3f(dv29); v3f(dv41); n3f(dn30); v3f(dv42); v3f(dv30);
endpolygon();
bgnpolygon();
v3f(dv30); v3f(dv42); n3f(dn31); v3f(dv43); v3f(dv31);
endpolygon();
bgnpolygon();
v3f(dv31); v3f(dv43); n3f(nz1); v3f(dv32); v3f(dv20);
endpolygon();

bgnpolygon();
n3f(nx1); v3f(dv1); v3f(dv8); v3f(dv9);
endpolygon();
bgnpolygon();
n3f(nx2); v3f(dv11); v3f(dv18); v3f(dv19);
endpolygon();
bgnpolygon();
n3f(ny2); v3f(dv20); v3f(dv21); v3f(dv22);
v3f(dv23); v3f(dv24); v3f(dv25);
v3f(dv26); v3f(dv27); v3f(dv28);
v3f(dv29); v3f(dv30); v3f(dv31);
endpolygon();
return;
```

```
}

/****************************** Draw link 5 ********************************/

drawlink5()
{
    bgnpolygon();
    n3f(en0); v3f(ev0); v3f(ev12);
    n3f(en1); v3f(ev13); v3f(ev1);
    endpolygon();
    bgnpolygon();
    v3f(ev1); v3f(ev13); n3f(en2); v3f(ev14); v3f(ev2);
    endpolygon();
    bgnpolygon();
    v3f(ev2); v3f(ev14); n3f(en3); v3f(ev15); v3f(ev3);
    endpolygon();
    bgnpolygon();
    v3f(ev3); v3f(ev15); n3f(en4); v3f(ev16); v3f(ev4);
    endpolygon();
    bgnpolygon();
    v3f(ev4); v3f(ev16); n3f(en5); v3f(ev17); v3f(ev5);
    endpolygon();
    bgnpolygon();
    v3f(ev5); v3f(ev17); n3f(en6);  v3f(ev18); v3f(ev6);
    endpolygon();
    bgnpolygon();
    v3f(ev6); v3f(ev18); n3f(en7); v3f(ev19); v3f(ev7);
    endpolygon();
    bgnpolygon();
    v3f(ev7); v3f(ev19);n3f(en8); v3f(ev20); v3f(ev8);
    endpolygon();
    bgnpolygon();
    v3f(ev8); v3f(ev20); n3f(en9); v3f(ev21); v3f(ev9);
    endpolygon();
    bgnpolygon();
    v3f(ev9); v3f(ev21); n3f(en10); v3f(ev22); v3f(ev10);
    endpolygon();
    bgnpolygon();
    v3f(ev10); v3f(ev22); n3f(en11);  v3f(ev23); v3f(ev11);
    endpolygon();
    bgnpolygon();
    v3f(ev11); v3f(ev23); n3f(en0);  v3f(ev12); v3f(ev0);
    endpolygon();

    bgnpolygon();
    n3f(ny1); v3f(ev12); v3f(ev13); v3f(ev14); v3f(ev15);
    v3f(ev16); v3f(ev17); v3f(ev18); v3f(ev19); v3f(ev20);
    v3f(ev21); v3f(ev22); v3f(ev23);
    endpolygon();
    bgnpolygon();
```

```
n3f(nx2); v3f(ev24); v3f(ev25); v3f(ev26); v3f(ev27);
v3f(ev28); v3f(ev29); v3f(ev30); v3f(ev31); v3f(ev32);
endpolygon();
bgnpolygon();
n3f(nx1); v3f(ev33); v3f(ev34); v3f(ev35); v3f(ev36); v3f(ev37);
v3f(ev38); v3f(ev39); v3f(ev40); v3f(ev41); v3f(ev42);
endpolygon();
bgnpolygon();
n3f(nx2); v3f(ev43); v3f(ev44); v3f(ev45); v3f(ev46); v3f(ev47);
v3f(ev48); v3f(ev49); v3f(ev50); v3f(ev51); v3f(ev52);
endpolygon();
bgnpolygon();
n3f(nx1); v3f(ev53); v3f(ev54); v3f(ev55); v3f(ev56);
v3f(ev57); v3f(ev58); v3f(ev59); v3f(ev60); v3f(ev61);
endpolygon();


bgnpolygon();
n3f(ny2); v3f(ev20); v3f(ev33); v3f(ev34); v3f(ev24);
endpolygon();
bgnpolygon();
v3f(ev24); v3f(ev34); n3f(en25); v3f(ev35); v3f(ev25);
endpolygon();
bgnpolygon();
v3f(ev25); v3f(ev35); n3f(en26); v3f(ev36); v3f(ev26);
endpolygon();
bgnpolygon();
v3f(ev26); v3f(ev36); n3f(nz2); v3f(ev37); v3f(ev27);
endpolygon();
bgnpolygon();
v3f(ev27); v3f(ev37); n3f(en28); v3f(ev38); v3f(ev28);
endpolygon();
bgnpolygon();
v3f(ev28); v3f(ev38); n3f(en29); v3f(ev39); v3f(ev29);
endpolygon();
bgnpolygon();
v3f(ev29); v3f(ev39); n3f(ny1); v3f(ev40); v3f(ev30);
endpolygon();
bgnpolygon();
v3f(ev30); v3f(ev40); n3f(en31); v3f(ev41); v3f(ev31);
endpolygon();
bgnpolygon();
v3f(ev31); v3f(ev41); n3f(en32); v3f(ev42); v3f(ev32);
endpolygon();
bgnpolygon();
v3f(ev32); v3f(ev42); v3f(ev33); v3f(ev20);
endpolygon();

bgnpolygon();
n3f(ny2); v3f(ev43); v3f(ev16); v3f(ev53); v3f(ev44);
endpolygon();
```

```
    bgnpolygon();
    v3f(ev44); v3f(ev53); n3f(en25); v3f(ev54); v3f(ev45);
    endpolygon();
    bgnpolygon();
    v3f(ev45); v3f(ev54); n3f(en26); v3f(ev55); v3f(ev46);
    endpolygon();
    bgnpolygon();
    v3f(ev46); v3f(ev55); n3f(nz2); v3f(ev56); v3f(ev47);
    endpolygon();
    bgnpolygon();
    v3f(ev47); v3f(ev56); n3f(en28); v3f(ev57); v3f(ev48);
    endpolygon();
    bgnpolygon();
    v3f(ev48); v3f(ev57); n3f(en29); v3f(ev58); v3f(ev49);
    endpolygon();
    bgnpolygon();
    v3f(ev49); v3f(ev58); n3f(ny1); v3f(ev59); v3f(ev50);
    endpolygon();
    bgnpolygon();
    v3f(ev50); v3f(ev59); n3f(en31); v3f(ev60); v3f(ev51);
    endpolygon();
    bgnpolygon();
    v3f(ev51); v3f(ev60); n3f(en32); v3f(ev61); v3f(ev52);
    endpolygon();
    bgnpolygon();
    v3f(ev52); v3f(ev61); v3f(ev16); v3f(ev43);
    endpolygon();
    bgnpolygon();
    n3f(en20); v3f(ev20); v3f(ev24); v3f(ev32);
    endpolygon();
    bgnpolygon();
    n3f(en16); v3f(ev16); v3f(ev53); v3f(ev61);
    endpolygon();
    return;
}


/****************************** Draw link 6 *********************************/

drawlink6()
{
    bgnpolygon();
    n3f(ny1); v3f(fv0); v3f(fv10); v3f(fv19); v3f(fv9);
    endpolygon();
    bgnpolygon();
    v3f(fv0); v3f(fv10); n3f(fn1); v3f(fv11); v3f(fv1);
    endpolygon();
    bgnpolygon();
    v3f(fv1); v3f(fv11); n3f(fn2); v3f(fv12); v3f(fv2);
    endpolygon();
    bgnpolygon();
```

```
v3f(fv2); v3f(fv12); n3f(nz2); v3f(fv13); v3f(fv3);
endpolygon();
bgnpolygon();
v3f(fv3); v3f(fv13); n3f(fn4); v3f(fv14); v3f(fv4);
endpolygon();
bgnpolygon();
v3f(fv4); v3f(fv14); n3f(fn5); v3f(fv15); v3f(fv5);
endpolygon();
bgnpolygon();
v3f(fv5); v3f(fv15); n3f(ny2); v3f(fv16); v3f(fv6);
endpolygon();
bgnpolygon();
v3f(fv6); v3f(fv16); n3f(fn7); v3f(fv17); v3f(fv7);
endpolygon();
bgnpolygon();
v3f(fv7); v3f(fv17); n3f(fn8); v3f(fv18); v3f(fv8);
endpolygon();
bgnpolygon();
v3f(fv8); v3f(fv18); v3f(fv19); v3f(fv9);
endpolygon();

bgnpolygon();
n3f(nz1); v3f(fv20); v3f(fv32);
n3f(fn21); v3f(fv33); v3f(fv21);
endpolygon();
bgnpolygon();
v3f(fv21); v3f(fv33); n3f(fn22); v3f(fv34); v3f(fv22);
endpolygon();
bgnpolygon();
v3f(fv22); v3f(fv34); n3f(nx1); v3f(fv35); v3f(fv23);
endpolygon();
bgnpolygon();
v3f(fv23); v3f(fv35); n3f(fn24); v3f(fv36); v3f(fv24);
endpolygon();
bgnpolygon();
v3f(fv24); v3f(fv36); n3f(fn25); v3f(fv37); v3f(fv25);
endpolygon();
bgnpolygon();
v3f(fv25); v3f(fv37); n3f(nz2); v3f(fv38); v3f(fv26);
endpolygon();
bgnpolygon();
v3f(fv26); v3f(fv38); n3f(fn27); v3f(fv39); v3f(fv27);
endpolygon();
bgnpolygon();
v3f(fv27); v3f(fv39); n3f(fn28); v3f(fv40); v3f(fv28);
endpolygon();
bgnpolygon();
v3f(fv28); v3f(fv40); n3f(nx2); v3f(fv41); v3f(fv29);
endpolygon();
bgnpolygon();
v3f(fv29); v3f(fv41); n3f(fn30); v3f(fv42); v3f(fv30);
```

```
    endpolygon();
    bgnpolygon();
    v3f(fv30); v3f(fv42); n3f(fn31); v3f(fv43); v3f(fv31);
    endpolygon();
    bgnpolygon();
    v3f(fv31); v3f(fv43); n3f(nz1); v3f(fv32); v3f(fv20);
    endpolygon();

    bgnpolygon();
    n3f(ny2); v3f(fv20); v3f(fv21); v3f(fv22); v3f(fv23);
    v3f(fv24); v3f(fv25); v3f(fv26); v3f(fv27); v3f(fv28);
    v3f(fv29); v3f(fv30); v3f(fv31);
    endpolygon();
    bgnpolygon();
    n3f(nx2); v3f(fv0); v3f(fv8); v3f(fv9);
    endpolygon();
    bgnpolygon();
    n3f(nx1); v3f(fv10); v3f(fv18); v3f(fv19);
    endpolygon();
    return;
}

/****************************** Draw gripper ********************************/

drawgripper()
{
    bgnpolygon();
    n3f(nz1); v3f(gv0); v3f(gv12); n3f(gn1); v3f(gv13); v3f(gv1);
    endpolygon();
    bgnpolygon();
    v3f(gv1); v3f(gv13); n3f(gn2); v3f(gv14); v3f(gv2);
    endpolygon();
    bgnpolygon();
    v3f(gv2); v3f(gv14); n3f(nx1); v3f(gv15); v3f(gv3);
    endpolygon();
    bgnpolygon();
    v3f(gv3); v3f(gv15); n3f(gn4); v3f(gv16); v3f(gv4);
    endpolygon();
    bgnpolygon();
    v3f(gv4); v3f(gv16); n3f(gn5); v3f(gv17); v3f(gv5);
    endpolygon();
    bgnpolygon();
    v3f(gv5); v3f(gv17); n3f(nz2); v3f(gv18); v3f(gv6);
    endpolygon();
    bgnpolygon();
    v3f(gv6); v3f(gv18); n3f(gn7); v3f(gv19); v3f(gv7);
    endpolygon();
    bgnpolygon();
    v3f(gv7); v3f(gv19); n3f(gn8); v3f(gv20); v3f(gv8);
    endpolygon();
    bgnpolygon();
```

```
v3f(gv8); v3f(gv20); n3f(nx2); v3f(gv21); v3f(gv9);
endpolygon();
bgnpolygon();
v3f(gv9); v3f(gv21); n3f(gn10); v3f(gv22); v3f(gv10);
endpolygon();
bgnpolygon();
v3f(gv10); v3f(gv22); n3f(gn11); v3f(gv23); v3f(gv11);
endpolygon();
bgnpolygon();
v3f(gv11); v3f(gv23); n3f(nz1); v3f(gv12); v3f(gv0);
endpolygon();


bgnpolygon();
n3f(ny2);  v3f(gv24); v3f(gv25); v3f(gv26); v3f(gv27);
endpolygon();
bgnpolygon();
n3f(nx2); v3f(gv24); v3f(gv27); v3f(gv35); v3f(gv32);
endpolygon();
bgnpolygon();
v3f(gv27); v3f(gv39); v3f(gv36); v3f(gv35);
endpolygon();
bgnpolygon();
v3f(gv32); v3f(gv31); v3f(gv28); v3f(gv24);
endpolygon();
bgnpolygon();
n3f(nx1); v3f(gv25); v3f(gv26); v3f(gv34); v3f(gv33);
endpolygon();
bgnpolygon();
v3f(gv26); v3f(gv38); v3f(gv37); v3f(gv34);
endpolygon();
bgnpolygon();
v3f(gv25); v3f(gv33); v3f(gv30); v3f(gv29);
endpolygon();
bgnpolygon();
n3f(nz1);  v3f(gv24); v3f(gv28); v3f(gv29); v3f(gv25);
endpolygon();
bgnpolygon();
n3f(nz2);  v3f(gv27); v3f(gv39); v3f(gv38); v3f(gv26);
endpolygon();
bgnpolygon();
n3f(ny1);  v3f(gv28); v3f(gv29); v3f(gv30); v3f(gv31);
endpolygon();
bgnpolygon();
v3f(gv32); v3f(gv33); v3f(gv34); v3f(gv35);
endpolygon();
bgnpolygon();
v3f(gv36); v3f(gv37); v3f(gv38); v3f(gv39);
endpolygon();
bgnpolygon();
n3f(nz2);  v3f(gv30); v3f(gv31); v3f(gv32); v3f(gv33);
```

```
    endpolygon();
    bgnpolygon();
    n3f(nz1);  v3f(gv34); v3f(gv35); v3f(gv36); v3f(gv37);
    endpolygon();
}


/***************** Draw the moving part of the gripper ************************/
drawfig()
{
    bgnpolygon();
    n3f(nz1); v3f(ggv0); v3f(ggv1); v3f(ggv2); v3f(ggv3);
    endpolygon();
    bgnpolygon();
    n3f(nz2); v3f(ggv4); v3f(ggv5); v3f(ggv6); v3f(ggv7);
    endpolygon();
    bgnpolygon();
    n3f(nx2); v3f(ggv0); v3f(ggv3); v3f(ggv7); v3f(ggv4);
    endpolygon();
    bgnpolygon();
    n3f(ggn); v3f(ggv1); v3f(ggv5); v3f(ggv6); v3f(ggv2);
    endpolygon();
    bgnpolygon();
    n3f(ny1); v3f(ggv2); v3f(ggv3); v3f(ggv7); v3f(ggv6);
    endpolygon();
}

/************************** Draw K-2107 manipulator ************************/

draw_k2107_robot(theta)
float *theta;
{

    drawlink0();
    rot(theta[1], 'y');
    drawlink1();
    rot(theta[2], 'x');
    drawlink2();
    translate(0.0, 0.0, -5.625);
    rot(theta[3], 'y');
    drawlink3();
    translate(0.0, 37.985, -4.25);
    rot(theta[4], 'x');
    drawlink4();
    translate(0.0, 0.0, 4.25);
    rot(theta[5], 'y');
    drawlink5();
    translate(0.0, 37.996, -1.937);
    rot(theta[6], 'x');
    drawlink6();
    translate(0.0, 10.619, 1.937);
```

```
    rot(theta[7], 'y');
    drawgripper();
    translate(theta[0], -2.5, 0.0);
    drawfig();
    rot(180.0, 'y');
    translate(2*theta[0], 0.0, 0.0);
    drawfig();
}

/********************* Parameters of K-2107 manipulator ***********************/

#define A2 (-5.625)
#define A3 (-4.25)
#define D3 37.985
#define A4 4.25
#define A5 (-1.937)
#define D5 37.996
#define A6 1.937
#define D7 10.619

float k2107_low_lim[]= {0.0, -3.1415926, -0.78539816, -3.1415926, -3.1415926,
                        -6.2831853, -3.1415926, -12.566371};
float k2107_up_lim[]= {2.0, 3.1415926,  2.3561945, 3.1415926, 0.0, 6.2831853,
                        0.0, 12.566371};

/*  4.0          75       75       120          120          135          135          135  */
float k2107_jv_lim[]={2.0, 1.308997, 1.308997, 2.0943951, 2.0943951, 2.3561945,
                        2.3561945, 2.3561945};
float k2107_home[]={2.0, 0.0, 1.5707963, 0.0, -1.5707963, 0.0, -1.5707963, 0.0};
float k2107_init_vpoint[] = {0.0, 10.0, -15.0, -150.0, -90.0, 0.0, 150.0};
int k2107_init_obt_index = 1;
int k2107_init_obt_sign = 1;
float k2107_theta[MAT_DIM], k2107_view_point[MAT_DIM];
float k2107_obt_vel;
int k2107_obt_sign, k2107_obt_index;

/************** Load the parameters of RRC K-2107 manioulator *****************/
init_k2107_parameters()
{
    int i;

    for (i=0; i<=7; i++)
        {
        k2107_theta[i] = k2107_home[i];
        k2107_view_point[i] = k2107_init_vpoint[i];
        }
    k2107_obt_sign = k2107_init_obt_sign;
    k2107_obt_index = k2107_init_obt_index;
    k2107_obt_vel = joint_v;
}
```

```
/**************** Transfer parameters to the main program ********************/
get_k2107_parameters()
{
    int i;

    for (i=0; i<=7; i++)
        {
        th_lim1[i] = k2107_low_lim[i];
        th_lim2[i] = k2107_up_lim[i];
        dth_lim[i] = k2107_up_lim[i] - k2107_low_lim[i];
        dth_lim[i] *= dth_lim[i];
        jv_lim[i] = k2107_jv_lim[i];
        theta_rd[i] = k2107_theta[i];
        robot_home[i] = k2107_home[i];
        view_point[i] = k2107_view_point[i];
        init_view_point[i] = k2107_init_vpoint[i];
        }
    obtctl.index = k2107_obt_index;
    obtctl.sign = k2107_obt_sign;
    obtctl.obt_v = k2107_obt_vel;
}

/******************* get parameters from the main program **********************/
save_k2107_parameters()
{
    int i;

    for (i=0; i<=7; i++)
        {
        k2107_theta[i] = theta_rd[i];
        k2107_view_point[i] = view_point[i];
        }
    k2107_obt_index = obtctl.index;
    k2107_obt_sign = obtctl.sign;
    k2107_obt_vel = obtctl.obt_v;
}

/******************* Initiate the orbit control parameters *******************/
init_k2107_obt_struct()
{

    obtctl.sign = k2107_init_obt_sign;
    obtctl.index = k2107_init_obt_index;
    obtctl.obt_v = joint_v;
}

/*************** Calculate the Jacobian of RRC K-2107 manipulator *************/
k2107_jacobian(theta,t,jc)
float *theta, t[5][5], jc[MAT_DIM][MAT_DIM];
{
    unsigned int i;
```

```
float th, r1, r2, r3;
float s1, s2, s3, s4, s5, s6, s7;
float c1, c2, c3, c4, c5, c6, c7;
float u1, u2, u3, v3;



th=theta[1];
s1=sin(th);
c1=cos(th);
th=theta[2];
s2=sin(th);
c2=cos(th);
th=theta[3];
s3=sin(th);
c3=cos(th);
th=theta[4];
s4=sin(th);
c4=cos(th);
th=theta[5];
s5=sin(th);
c5=cos(th);
th=theta[6];
s6=sin(th);
c6=cos(th);
th=theta[7];
s7=sin(th);
c7=cos(th);

jc[4][1]=jc[6][1]=0;
jc[5][1]=1;
jc[4][2]=c1;
jc[5][2]=0;
jc[6][2]= -s1;
jc[1][3]=s1*c2*A2;
jc[2][3]= -s2*A2;
jc[3][3]=c1*c2*A2;
jc[4][3]=s1*s2;
jc[5][3]=c2;
jc[6][3]=c1*s2;
jc[1][4]=s1*(c2*(A3*c3+A2)+s2*D3)+c1*A3*s3;
jc[2][4]=c2*D3-s2*(c3*A3+A2);
jc[3][4]=c1*(c2*(A3*c3+A2)+s2*D3)-s1*A3*s3;
jc[4][4]=c1*c3-s1*c2*s3;
jc[5][4]=s2*s3;
jc[6][4]= -c1*c2*s3-s1*c3;
u1=s1*c2*c3+c1*s3;
u2= -s2*c3;
u3=c1*c2*c3-s1*s3;
v3= -c1*s2;
r1=u1*c4-jc[4][3]*s4;
```

```
r2=u2*c4-c2*s4;
r3=u3*c4+v3*s4;
jc[4][5]=u1*s4+jc[4][3]*c4;
jc[5][5]=u2*s4+c2*c4;
jc[6][5]=u3*s4-v3*c4;
jc[1][5]=r1*A4+jc[1][4];
jc[2][5]=r2*A4+jc[2][4];
jc[3][5]=r3*A4+jc[3][4];
u1=r1*c5+jc[4][4]*s5;
u2=r2*c5+jc[5][4]*s5;
u3=r3*c5+jc[6][4]*s5;
jc[4][6]=jc[4][4]*c5-r1*s5;
jc[5][6]=jc[5][4]*c5-r2*s5;
jc[6][6]=jc[6][4]*c5-r3*s5;
jc[1][6]=u1*A5+jc[4][5]*D5+r1*A4+jc[1][4];
jc[2][6]=u2*A5+jc[5][5]*D5+r2*A4+jc[2][4];
jc[3][6]=u3*A5+jc[6][5]*D5+r3*A4+jc[3][4];
jc[4][7]=u1*s6+jc[4][5]*c6;
jc[5][7]=u2*s6+jc[5][5]*c6;
jc[6][7]=u3*s6+jc[6][5]*c6;
r1=u1*c6-jc[4][5]*s6;
t[1][1]=r1*c7+jc[4][6]*s7;
t[1][2]=jc[4][6]*c7-r1*s7;
t[1][3]=jc[4][7];
t[1][4]=jc[4][7]*D7+A6*(u1*c6-jc[4][5]*s6)+jc[1][6];
r2=u2*c6-jc[5][5]*s6;
t[2][1]=r2*c7+jc[5][6]*s7;
t[2][2]=jc[5][6]*c7-r2*s7;
t[2][3]=jc[5][7];
t[2][4]=jc[5][7]*D7+A6*(u2*c6-jc[5][5]*s6)+jc[2][6];
r3=u3*c6-jc[6][5]*s6;
t[3][1]=r3*c7+jc[6][6]*s7;
t[3][2]=jc[6][6]*c7-r3*s7;
t[3][3]=jc[6][7];
t[3][4]=jc[6][7]*D7+A6*(u3*c6-jc[6][5]*s6)+jc[3][6];
t[4][1]=t[4][2]=t[4][3]=0;
t[4][4]=1;
jc[1][1]=t[3][4];
jc[2][1]=0;
jc[3][1]= -t[1][4];
jc[1][2]= -t[2][4]*jc[6][2];
jc[2][2]=t[1][4]*jc[6][2]-t[3][4]*jc[4][2];
jc[3][2]=t[2][4]*jc[4][2];
jc[1][7]=jc[2][7]=jc[3][7]=0;
for (i=3; i<=6; i++)
    {
    r1=t[1][4]-jc[1][i];
    r2=t[2][4]-jc[2][i];
    r3=t[3][4]-jc[3][i];
    jc[1][i]=jc[5][i]*r3-jc[6][i]*r2;
    jc[2][i]=jc[6][i]*r1-jc[4][i]*r3;
```

```
        jc[3][i]=jc[4][i]*r2-jc[5][i]*r1;
        }
    return;
}

/***** Calculate the homogeneous transformation of RRC K-2107 manipulator *****/

k2107_transf(theta,t)
float *theta, t[5][5];
{
    unsigned int i;
    float th, r1, r2, r3, t1, t2, t3, w1, w2, w3, x1, x2, x3;
    float s1, s2, s3, s4, s5, s6, s7;
    float c1, c2, c3, c4, c5, c6, c7;
    float u1, u2, u3, v3, y1, y2, y3;



    th=theta[1];
    s1=sin(th);
    c1=cos(th);
    th=theta[2];
    s2=sin(th);
    c2=cos(th);
    th=theta[3];
    s3=sin(th);
    c3=cos(th);
    th=theta[4];
    s4=sin(th);
    c4=cos(th);
    th=theta[5];
    s5=sin(th);
    c5=cos(th);
    th=theta[6];
    s6=sin(th);
    c6=cos(th);
    th=theta[7];
    s7=sin(th);
    c7=cos(th);

    th=s1*s2;
    x1=s1*(c2*(A3*c3+A2)+s2*D3)+c1*A3*s3;
    x2=c2*D3-s2*(c3*A3+A2);
    x3=c1*(c2*(A3*c3+A2)+s2*D3)-s1*A3*s3;
    t1=c1*c3-s1*c2*s3;
    t2=s2*s3;
    t3= -c1*c2*s3-s1*c3;
    u1=s1*c2*c3+c1*s3;
    u2= -s2*c3;
    u3=c1*c2*c3-s1*s3;
    v3= -c1*s2;
```

```
    r1=u1*c4-th*s4;
    r2=u2*c4-c2*s4;
    r3=u3*c4+v3*s4;
    w1=u1*s4+th*c4;
    w2=u2*s4+c2*c4;
    w3=u3*s4-v3*c4;
    u1=r1*c5+t1*s5;
    u2=r2*c5+t2*s5;
    u3=r3*c5+t3*s5;
    y1=t1*c5-r1*s5;
    y2=t2*c5-r2*s5;
    y3=t3*c5-r3*s5;
    x1=u1*A5+w1*D5+r1*A4+x1;
    x2=u2*A5+w2*D5+r2*A4+x2;
    x3=u3*A5+w3*D5+r3*A4+x3;
    r1=u1*c6-w1*s6;
    t[1][1]=r1*c7+y1*s7;
    t[1][2]=y1*c7-r1*s7;
    t[1][3]=u1*s6+w1*c6;
    t[1][4]=t[1][3]*D7+A6*(u1*c6-w1*s6)+x1;
    r2=u2*c6-w2*s6;
    t[2][1]=r2*c7+y2*s7;
    t[2][2]=y2*c7-r2*s7;
    t[2][3]=u2*s6+w2*c6;
    t[2][4]=t[2][3]*D7+A6*(u2*c6-w2*s6)+x2;
    r3=u3*c6-w3*s6;
    t[3][1]=r3*c7+y3*s7;
    t[3][2]=y3*c7-r3*s7;
    t[3][3]=u3*s6+w3*c6;
    t[3][4]=t[3][3]*D7+A6*(u3*c6-w3*s6)+x3;
    t[4][1]=t[4][2]=t[4][3]=0;
    t[4][4]=1;
    return;
}


/********** Calculate the elbow position of RRC K-2107 manipulator ************/

k2107_elbow_pos(theta, elbow_dir, elbow_pos)
float *theta, *elbow_dir, *elbow_pos;
{
    float th;
    float s1,s2,s3;
    float c1,c2,c3;

    th=theta[1];
    s1=sin(th);
    c1=cos(th);
    th=theta[2];
    s2=sin(th);
    c2=cos(th);
```

```
    th=theta[3];
    s3=sin(th);
    c3=cos(th);

    elbow_dir[1] = s2*s3;
    elbow_dir[2] = s1*c2*s3-c1*c3;
    elbow_dir[3] = -c1*c2*s3-s1*c3;

    elbow_pos[1] = -s2*A3*c3+c2*D3-A2*s2;
    elbow_pos[2] = -s1*c2*A3*c3-c1*A3*s3-s1*s2*D3-s1*A2*c2;
    elbow_pos[3] = c1*c2*A3*c3-s1*A3*s3+c1*s2*D3+c1*A2*c2;
}
```

```
/*                    Source code file "draw_LTM.c"                    */

#include <gl/gl.h>
#include <math.h>
#include <stdio.h>
#define MAT_DIM 10
#define A2 2.30000e+01
#define A4 2.00000e+01
#define D7 1.20000e+01
extern struct {int index; int sign; float obt_v;} obtctl;
extern float th_lim1[MAT_DIM], th_lim2[MAT_DIM], theta_rd[MAT_DIM];
extern float view_point[MAT_DIM], jv_lim[MAT_DIM];
extern float robot_home[MAT_DIM], init_view_point[MAT_DIM];
extern float joint_v;
extern float nx1[3], nx2[3], ny1[3], ny2[3], nz1[3], nz2[3];


/***************************** Cylinder data ********************************/
float cyl1[]= {0.0, 1.0, 0.0},
     cyl2[]= {0.5, 0.866, 0.0},
     cyl3[]= {0.866, 0.5, 0.0},
     cyl4[]= {1.0, 0.0, 0.0},
     cyl5[]= {0.866, -0.5, 0.0},
     cyl6[]= {0.5, -0.866, 0.0},
     cyl7[]= {0.0, -1.0, 0.0},
     cyl8[]= {-0.5, -0.866, 0.0},
     cyl9[]= {-0.866, -0.5, 0.0},
     cyl10[]= {-1.0, 0.0, 0.0},
     cyl11[]= {-0.866, 0.5, 0.0},
     cyl12[]= {-0.5, 0.866, 0.0},
     cyl13[]= {0.0, 1.0, 0.5},
     cyl14[]= {0.5, 0.866, 0.5},
     cyl15[]= {0.866, 0.5, 0.5},
     cyl16[]= {1.0, 0.0, 0.5},
     cyl17[]= {0.866, -0.5, 0.5},
     cyl18[]= {0.5, -0.866, 0.5},
     cyl19[]= {0.0, -1.0, 0.5},
     cyl20[]= {-0.5, -0.866, 0.5},
     cyl21[]= {-0.866, -0.5, 0.5},
     cyl22[]= {-1.0, 0.0, 0.5},
     cyl23[]= {-0.866, 0.5, 0.5},
     cyl24[]= {-0.5, 0.866, 0.5},
     cylct1[] = {0.0, 0.0, 0.0},
     cylct2[] = {0.0, 0.0, 1.00};

/* Surface Normals for the Cylinders */
float cyn1[]= {0.0, 1.0, 0.0},
     cyn2[]= {0.5, 0.866, 0.0},
     cyn3[]= {0.866, 0.5, 0.0},
     cyn4[]= {1.0, 0.0, 0.0},
```

```
    cyn5[]= {0.866, -0.5, 0.0},
    cyn6[]= {0.5, -0.866, 0.0},
    cyn7[]= {0.0, -1.0, 0.0},
    cyn8[]= {-0.5, -0.866, 0.0},
    cyn9[]= {-0.866, -0.5, 0.0},
    cyn10[]= {-1.0, 0.0, 0.0},
    cyn11[]= {-0.866, 0.5, 0.0},
    cyn12[]= {-0.5, 0.866, 0.0},
    cyz1[]= {0.0, 0.0, 1.0},
    cyz2[]= {0.0, 0.0, -1.0};

float cy_fs_n1[]= {0.0, 0.1736, 0.9848},
    cy_fs_n2[]= {0.08682, 0.1504, 0.9848},
    cy_fs_n3[]= {0.1504, 0.08682, 0.9848},
    cy_fs_n4[]= {0.1736, 0.0, 0.9848},
    cy_fs_n5[]= {0.1504, -0.08682, 0.9848},
    cy_fs_n6[]= {0.08682, -0.1504, 0.9848},
    cy_fs_n7[]= {0.0, -0.1736, 0.9848},
    cy_fs_n8[]= {-0.08682, -0.1504, 0.9848},
    cy_fs_n9[]= {-0.1504, -0.08682, 0.9848},
    cy_fs_n10[]= {-0.1736, 0.0, 0.9848},
    cy_fs_n11[]= {-0.1504, 0.08682, 0.9848},
    cy_fs_n12[]= {-0.08682, 0.1504, 0.9848};


/****************** Positions and the sizses of the cylinders ******************/
float cyl_array[][7]= {2.0, 0.0, 0.0, 0.0, 90.0, 3.0, 1.5,
                       -2.0, 0.0, 0.0, 0.0, -90.0, 3.0, 1.5,
                       1.625, 20.0, 0.0, 0.0, 90.0, 2.25, 1.125,
                       -1.625, 20.0, 0.0, 0.0, -90.0, 2.25, 1.125,
                       -6.0, 0.0, 0.0, 0.0, -90.0, 1.0, 1.76};

/****************** The normal vector used by the graphics model ***************/

float nxy1[] = {0.7071, 0.7071, 0.0},
    nxy2[] = {0.7071, -0.7071, 0.0},
    nxy3[] = {-0.7071, -0.7071, 0.0},
    nxy4[] = {-0.7071, 0.7071, 0.0},
    nyz1[] = {0.0, 0.7071, 0.7071},
    nyz2[] = {0.0, 0.7071, -0.7071},
    nyz3[] = {0.0, -0.7071, -0.7071},
    nyz4[] = {0.0, -0.7071, 0.7071};

float nxz1[] = {0.7071, 0.0, 0.7071},
    nxz2[] = {0.7071, 0.0, -0.7071},
    nxz3[] = {-0.7071, 0.0, -0.7071},
    nxz4[] = {-0.7071, 0.0, 0.7071};

float nxzp1[] = {0.9962, 0.0, 0.0872},
    nxzp2[] = {0.9962, 0.0, -0.0872},
    nxzn1[] = {-0.9962, 0.0, 0.0872},
```

```
    nxzn2[] = {-0.9962, 0.0, -0.0872};

float nzxp1[] = {0.0872, 0.0, 0.9962},
    nzxp2[] = {-0.0872, 0.0, 0.9962},
    nzxn1[] = {0.0872, 0.0, -0.9962},
    nzxn2[] = {-0.0872, 0.0,-0.9962};

float cxy_n1[] = {0.0000, 1.0000, 0.0000},
    cxy_n2[] = {0.5000, 0.8660, 0.0000},
    cxy_n3[] = {0.8660, 0.5000, 0.0000},
    cxy_n4[] = {1.0000, 0.0000, 0.0000},
    cxy_n5[] = {0.8660, -0.5000, 0.0000},
    cxy_n6[] = {0.5000, -0.8660, 0.0000},
    cxy_n7[] = {0.0000, -1.0000, 0.0000},
    cxy_n8[] = {-0.5000, -0.8660, 0.0000},
    cxy_n9[] = {-0.8660, -0.5000, 0.0000},
    cxy_n10[] = {-1.0000, 0.0000, 0.0000},
    cxy_n11[] = {-0.8660, 0.5000, 0.0000},
    cxy_n12[] = {-0.5000, 0.8660, 0.0000};

float cxz_n1[] = {0.0000, 0.0000, 1.0000},
    cxz_n2[] = {0.5000, 0.0000, 0.8660},
    cxz_n3[] = {0.8660, 0.0000, 0.5000},
    cxz_n4[] = {1.0000, 0.0000, 0.0000},
    cxz_n5[] = {0.8660, 0.0000, -0.5000},
    cxz_n6[] = {0.5000, 0.0000, -0.8660},
    cxz_n7[] = {0.0000, 0.0000, -1.0000},
    cxz_n8[] = {-0.5000, 0.0000, -0.8660},
    cxz_n9[] = {-0.8660, 0.0000, -0.5000},
    cxz_n10[] = {-1.0000, 0.0000, 0.0000},
    cxz_n11[] = {-0.8660, 0.0000, 0.5000},
    cxz_n12[] = {-0.5000, 0.0000, 0.8660};

float cyz_n1[] = {0.0000, 0.0000, 1.0000},
    cyz_n2[] = {0.0000, 0.5000, 0.8660},
    cyz_n3[] = {0.0000, 0.8660, 0.5000},
    cyz_n4[] = {0.0000, 1.0000, 0.0000},
    cyz_n5[] = {0.0000, 0.8660, -0.5000},
    cyz_n6[] = {0.0000, 0.5000, -0.8660},
    cyz_n7[] = {0.0000, 0.0000, -1.0000},
    cyz_n8[] = {0.0000, -0.5000, -0.8660},
    cyz_n9[] = {0.0000, -0.8660, -0.5000},
    cyz_n10[] = {0.0000, -1.0000, 0.0000},
    cyz_n11[] = {0.0000, -0.8660, 0.5000},
    cyz_n12[] = {0.0000, -0.5000, 0.8660};

/****************** Data Used by the LTM Graphics model ******************/

/****************************** Stand Data ********************************/

float LTM_stand1[] = {2.5, -15.531,  3.0},
```

```
    LTM_stand2[] = {  3.0, -15.531, 2.5},
    LTM_stand3[] = {  3.0, -15.531, -2.5},
    LTM_stand4[] = { 2.5, -15.531, -3.0},
    LTM_stand5[] = { -2.5, -15.531, -3.0},
    LTM_stand6[] = { -3.0, -15.531, -2.5},
    LTM_stand7[] = { -3.0, -15.531, 2.5},
    LTM_stand8[] = { -2.5, -15.531,  3.0},
    LTM_stand9[] = { 4.0, -18.0,  3.0},
    LTM_stand10[] = { 5.0, -18.0, 2.0},
    LTM_stand11[] = { 5.0, -18.0, -6.0},
    LTM_stand12[] = { 4.0, -18.0, -7.0},
    LTM_stand13[] = { -4.0, -18.0, -7.0},
    LTM_stand14[] = { -5.0, -18.0, -6.0},
    LTM_stand15[] = { -5.0, -18.0, 2.0},
    LTM_stand16[] = { -4.0, -18.0,  3.0},
    LTM_stand17[] = { 4.0, -43.0,  3.0},
    LTM_stand18[] = { 5.0, -43.0, 2.0},
    LTM_stand19[] = { 5.0, -43.0, -6.0},
    LTM_stand20[] = { 4.0, -43.0, -7.0},
    LTM_stand21[] = { -4.0, -43.0, -7.0},
    LTM_stand22[] = { -5.0, -43.0, -6.0},
    LTM_stand23[] = { -5.0, -43.0, 2.0},
    LTM_stand24[] = { -4.0, -43.0,  3.0};

/************** Data used by drawing axes of the LTM manipulator ***************/

float LTM_axis1[] = {2.0, 1.8478, 0.7654},
    LTM_axis2[] = {2.0, 0.7654, 1.8478},
    LTM_axis3[] = {2.0, -0.7654, 1.8478},
    LTM_axis4[] = {2.0, -1.8478, 0.7654},
    LTM_axis5[] = {2.0, -1.8478, -0.7654},
    LTM_axis6[] = {2.0, -0.7654, -1.8478},
    LTM_axis7[] = {2.0, 0.7654, -1.8478},
    LTM_axis8[] = {2.0, 1.8478, -0.7654},
    LTM_axis9[] = {1.0, 0.9239, 0.3827},
    LTM_axis10[] = {1.0, 0.3827, 0.9239},
    LTM_axis11[] = {1.0, -0.3827, 0.9239},
    LTM_axis12[] = {1.0, -0.9239, 0.3827},
    LTM_axis13[] = {1.0, -0.9239, -0.3827},
    LTM_axis14[] = {1.0, -0.3827, -0.9239},
    LTM_axis15[] = {1.0, 0.3827, -0.9239},
    LTM_axis16[] = {1.0, 0.9239, -0.3827},
    LTM_axis17[] = {-2.0, 1.8478, 0.7654},
    LTM_axis18[] = {-2.0, 0.7654, 1.8478},
    LTM_axis19[] = {-2.0, -0.7654, 1.8478},
    LTM_axis20[] = {-2.0, -1.8478, 0.7654},
    LTM_axis21[] = {-2.0, -1.8478, -0.7654},
    LTM_axis22[] = {-2.0, -0.7654, -1.8478},
    LTM_axis23[] = {-2.0, 0.7654, -1.8478},
    LTM_axis24[] = {-2.0, 1.8478, -0.7654},
    LTM_axis25[] = {-1.0, 0.9239, 0.3827},
```

```
    LTM_axis26[] = {-1.0, 0.3827, 0.9239},
    LTM_axis27[] = {-1.0, -0.3827, 0.9239},
    LTM_axis28[] = {-1.0, -0.9239, 0.3827},
    LTM_axis29[] = {-1.0, -0.9239, -0.3827},
    LTM_axis30[] = {-1.0, -0.3827, -0.9239},
    LTM_axis31[] = {-1.0, 0.3827, -0.9239},
    LTM_axis32[] = {-1.0, 0.9239, -0.3827},
    LTM_axis33[] = {0.7654, 1.8478, 2.0},
    LTM_axis34[] = {1.8478, 0.7654, 2.0},
    LTM_axis35[] = {1.8478, -0.7654, 2.0},
    LTM_axis36[] = {0.7654, -1.8478, 2.0},
    LTM_axis37[] = {-0.7654, -1.8478, 2.0},
    LTM_axis38[] = {-1.8478, -0.7654, 2.0},
    LTM_axis39[] = {-1.8478, 0.7654, 2.0},
    LTM_axis40[] = {-0.7654, 1.8478, 2.0},
    LTM_axis41[] = {0.3827, 0.9239, 1.0},
    LTM_axis42[] = {0.9239, 0.3827, 1.0},
    LTM_axis43[] = {0.9239, -0.3827, 1.0},
    LTM_axis44[] = {0.3827, -0.9239, 1.0},
    LTM_axis45[] = {-0.3827, -0.9239, 1.0},
    LTM_axis46[] = {-0.9239, -0.3827, 1.0},
    LTM_axis47[] = {-0.9239, 0.3827, 1.0},
    LTM_axis48[] = {-0.3827, 0.9239, 1.0},
    LTM_axis49[] = {0.7654, 1.8478, 3.5},
    LTM_axis50[] = {1.8478, 0.7654, 3.5},
    LTM_axis51[] = {1.8478, -0.7654, 3.5},
    LTM_axis52[] = {0.7654, -1.8478, 3.5},
    LTM_axis53[] = {-0.7654, -1.8478, 3.5},
    LTM_axis54[] = {-1.8478, -0.7654, 3.5},
    LTM_axis55[] = {-1.8478, 0.7654, 3.5},
    LTM_axis56[] = {-0.7654, 1.8478, 3.5},

    LTM_axis57[] = {1.0, -1.0, 1.0},
    LTM_axis58[] = {1.0, -1.0, -1.0},
    LTM_axis59[] = {-1.0, -1.0, -1.0},
    LTM_axis60[] = {-1.0, -1.0, 1.0},
    LTM_axis61[] = {1.0, 1.0, 1.0},
    LTM_axis62[] = {1.0, 1.0, -1.0},
    LTM_axis63[] = {-1.0, 1.0, -1.0},
    LTM_axis64[] = {-1.0, 1.0, 1.0};

float axis_n1[] = {-0.7071, 0.6533, 0.2706},
    axis_n2[] = {-0.7071, 0.2706, 0.6533},
    axis_n3[] = {-0.7071, -0.2706, 0.6533},
    axis_n4[] = {-0.7071, -0.6533, 0.2706},
    axis_n5[] = {-0.7071, -0.6533, -0.2706},
    axis_n6[] = {-0.7071, -0.2706, -0.6533},
    axis_n7[] = {-0.7071, 0.2706, -0.6533},
    axis_n8[] = {-0.7071, 0.6533, -0.2706},
    axis_n17[] = {0.7071, 0.6533, 0.2706},
    axis_n18[] = {0.7071, 0.2706, 0.6533},
```

```
    axis_n19[] = {0.7071, -0.2706, 0.6533},
    axis_n20[] = {0.7071, -0.6533, 0.2706},
    axis_n21[] = {0.7071, -0.6533, -0.2706},
    axis_n22[] = {0.7071, -0.2706, -0.6533},
    axis_n23[] = {0.7071, 0.2706, -0.6533},
    axis_n24[] = {0.7071, 0.6533, -0.2706},
    axis_n33[] = {0.2706, 0.6533, -0.7071},
    axis_n34[] = {0.6533, 0.2706, -0.7071},
    axis_n35[] = {0.6533, -0.2706, -0.7071},
    axis_n36[] = {0.2706, -0.6533, -0.7071},
    axis_n37[] = {-0.2706, -0.6533, -0.7071},
    axis_n38[] = {-0.6533, -0.2706, -0.7071},
    axis_n39[] = {-0.6533, 0.2706, -0.7071},
    axis_n40[] = {-0.2706, 0.6533, -0.7071},
    axis_n49[] = { 0.3827, 0.9239, 0.0},
    axis_n50[] = { 0.9239, 0.3827, 0.0},
    axis_n51[] = { 0.9239, -0.3827, 0.0},
    axis_n52[] = { 0.3827, -0.9239, 0.0},
    axis_n53[] = { -0.3827, -0.9239, 0.0},
    axis_n54[] = { -0.9239, -0.3827, 0.0},
    axis_n55[] = { -0.9239, 0.3827, 0.0},
    axis_n56[] = { -0.3827, 0.9239, 0.0};


/****************************** Base Data *********************************/

float LTM_bas1[] = {  2.5, -15.531,  3.0},
    LTM_bas2[] = {  3.0, -15.531, 2.5},
    LTM_bas3[] = {  3.0, -15.531, -2.5},
    LTM_bas4[] = { 2.5, -15.531, -3.0},
    LTM_bas5[] = { -2.5, -15.531, -3.0},
    LTM_bas6[] = { -3.0, -15.531, -2.5},
    LTM_bas7[] = { -3.0, -15.531, 2.5},
    LTM_bas8[] = { -2.5, -15.531,  3.0},
    LTM_bas9[] = {  2.5, 0.0,  3.0},
    LTM_bas10[] = {  3.0, 0.0, 2.5},
    LTM_bas11[] = {  3.0, 0.0, -2.5},
    LTM_bas12[] = { 2.5, 0.0, -3.0},
    LTM_bas13[] = { -2.5, 0.0, -3.0},
    LTM_bas14[] = { -3.0, 0.0, -2.5},
    LTM_bas15[] = { -3.0, 0.0, 2.5},
    LTM_bas16[] = { -2.5, 0.0,  3.0},
    LTM_bas17[] = { -2.1,  0.0,  3.0},
    LTM_bas18[] = { -2.1,  -3.0,  3.0},
    LTM_bas19[] = { -1.1,  -4.0,  3.0},
    LTM_bas20[] = { 1.1,  -4.0,  3.0},
    LTM_bas21[] = { 2.1,  -3.0,  3.0},
    LTM_bas22[] = { 2.1,  0.0,  3.0},
    LTM_bas23[] = { -2.1,  0.0, -3.0},
    LTM_bas24[] = { -2.1,  -3.0, -3.0},
    LTM_bas25[] = { -1.1,  -4.0, -3.0},
```

```
    LTM_bas26[] = { 1.1,  -4.0, -3.0},
    LTM_bas27[] = { 2.1,  -3.0, -3.0},
    LTM_bas28[] = { 2.1,  0.0, -3.0};


/****************************** LTM link2 data ******************************/

float LTM_uparm1[] = {  2.5, 7.469, 3.0},
    LTM_uparm2[] = { 3.0, 7.469, 2.5},
    LTM_uparm3[] = { 3.0, 7.469, -2.5},
    LTM_uparm4[] = { 2.5, 7.469, -3.0},
    LTM_uparm5[] = { -2.5, 7.469, -3.0},
    LTM_uparm6[] = { -3.0, 7.469, -2.5},
    LTM_uparm7[] = { -3.0, 7.469, 2.5},
    LTM_uparm8[] = { -2.5, 7.469, 3.0},
    LTM_uparm9[] = { 2.0, 5.0, 6.5},
    LTM_uparm10[] = { 2.5, 5.0, 6.0},
    LTM_uparm11[] = { 2.5, 5.0, 4.0},
    LTM_uparm12[] = { 2.0, 5.0, 3.5},
    LTM_uparm13[] = { -2.0, 5.0, 3.5},
    LTM_uparm14[] = { -2.5, 5.0, 4.0},
    LTM_uparm15[] = { -2.5, 5.0, 6.0},
    LTM_uparm16[] = { -2.0, 5.0, 6.5},
    LTM_uparm17[] = { 2.0, 0.0, 6.5},
    LTM_uparm18[] = { 2.5, 0.0, 6.0},
    LTM_uparm19[] = { 2.5, 0.0, 4.0},
    LTM_uparm20[] = { 2.0, 0.0, 3.5},
    LTM_uparm21[] = { -2.0, 0.0, 3.5},
    LTM_uparm22[] = { -2.5, 0.0, 4.0},
    LTM_uparm23[] = { -2.5, 0.0, 6.0},
    LTM_uparm24[] = { -2.0, 0.0, 6.5},
    LTM_uparm25[] = { -1.5, -2.0, 6.5},
    LTM_uparm26[] = { 1.5, -2.0, 6.5},
    LTM_uparm27[] = { -1.5, -2.5, 6.0},
    LTM_uparm28[] = { 1.5, -2.5, 6.0},
    LTM_uparm29[] = { -1.5, -2.5, 4.0},
    LTM_uparm30[] = { 1.5, -2.5, 4.0},
    LTM_uparm31[] = { -1.5, -2.0, 3.5},
    LTM_uparm32[] = { 1.5, -2.0, 3.5};

float LTM_uparm_n1[] = { 0.5749,  0.5822, 0.5749},
    LTM_uparm_n2[] = { 0.9801,  -0.1985, 0.0},
    LTM_uparm_n3[] = { 0.3557,  -0.8643, -0.3557},
    LTM_uparm_n4[] = { 0.0,  -0.9431, -0.3326},
    LTM_uparm_n5[] = { -0.3557,  -0.8643, -0.3557},
    LTM_uparm_n6[] = { -0.9801,  -0.1985, 0.0},
    LTM_uparm_n7[] = { -0.5749,  0.5822, 0.5749},
    LTM_uparm_n8[] = { 0.0,  0.8509, 0.5252},
    LTM_uparm_n17[] = { 0.4444,  -0.7777, 0.4444},
    LTM_uparm_n19[] = { 0.3234,  0.8893, -0.3234},
```

```
    LTM_uparm_n20[] = { 0.0,  0.9558, -0.2941},
    LTM_uparm_n21[] = { -0.3234,  0.8893, -0.3234},
    LTM_uparm_n23[] = { -0.4444,  -0.7777, 0.4444},
    LTM_uparm_n24[] = { 0.0,  -0.9138, 0.4061},
    LTM_uparm_n27[] = {-0.3827, -0.9239, 0.0},
    LTM_uparm_n28[] = { 0.3827,-0.9239, 0.0},
    LTM_uparm_n29[] = {-0.3827, -0.9239, 0.0},
    LTM_uparm_n30[] = { 0.3827,-0.9239, 0.0};


/************************** LTM lower arm data ****************************/

float LTM_low_arm1[] = {2.25, 7.0, 1.625},
    LTM_low_arm2[] = {2.25, 3.5, -3.35},
    LTM_low_arm3[] = {2.25, -1.5, -3.35},
    LTM_low_arm4[] = {1.25, -2.5, -3.35},
    LTM_low_arm5[] = {-1.25, -2.5, -3.35},
    LTM_low_arm6[] = {-2.25, -1.5, -3.35},
    LTM_low_arm7[] = {-2.25, 3.5, -3.35},
    LTM_low_arm8[] = {-2.25, 7.0, 1.625},
    LTM_low_arm9[] = {2.0, 7.0, 2.125},
    LTM_low_arm10[] = {2.0, 3.5, -3.1},
    LTM_low_arm11[] = {2.0, -1.5, -3.1},
    LTM_low_arm12[] = {1.0, -2.25, -3.1},
    LTM_low_arm13[] = {-1.0, -2.25, -3.1},
    LTM_low_arm14[] = {-2.0, -1.5, -3.1},
    LTM_low_arm15[] = {-2.0, 3.5, -3.1},
    LTM_low_arm16[] = {-2.0, 7.0, 2.125},
    LTM_low_arm17[] = {2.25, 7.0, -1.625},
    LTM_low_arm18[] = {2.25, 3.5, -4.75},
    LTM_low_arm19[] = {2.25, -1.5, -4.75},
    LTM_low_arm20[] = {1.25, -2.5, -4.75},
    LTM_low_arm21[] = {-1.25, -2.5, -4.75},
    LTM_low_arm22[] = {-2.25, -1.5, -4.75},
    LTM_low_arm23[] = {-2.25, 3.5, -4.75},
    LTM_low_arm24[] = {-2.25, 7.0, -1.625},
    LTM_low_arm25[] = {2.0, 7.0, -2.125},
    LTM_low_arm26[] = {2.0, 3.5, -5.0},
    LTM_low_arm27[] = {2.0, -1.5, -5.0},
    LTM_low_arm28[] = {1.0, -2.25, -5.0},
    LTM_low_arm29[] = {-1.0, -2.25, -5.0},
    LTM_low_arm30[] = {-2.0, -1.5, -5.0},
    LTM_low_arm31[] = {-2.0, 3.5, -5.0},
    LTM_low_arm32[] = {-2.0, 7.0, -2.125},
    LTM_low_arm33[] = {-2.25, 20.0, 1.625},
    LTM_low_arm34[] = {-2.0, 20.0, 2.125},
    LTM_low_arm35[] = {-1.625, 20.0, 2.125},
    LTM_low_arm36[] = {-1.625, 18.5, 2.125},
    LTM_low_arm37[] = {-0.5, 17.5, 2.125},
    LTM_low_arm38[] = { 0.5, 17.5, 2.125},
    LTM_low_arm39[] = { 1.625, 18.5, 2.125},
```

```
    LTM_low_arm40[] = { 1.625, 20.0, 2.125},
    LTM_low_arm41[] = { 2.0, 20.0, 2.125},
    LTM_low_arm42[] = {2.25, 20.0, 1.625},
    LTM_low_arm43[] = {-2.25, 20.0, -1.625},
    LTM_low_arm44[] = {-2.0, 20.0, -2.125},
    LTM_low_arm45[] = {-1.625, 20.0, -2.125},
    LTM_low_arm46[] = {-1.625, 18.5, -2.125},
    LTM_low_arm47[] = {-0.5, 17.5, -2.125},
    LTM_low_arm48[] = { 0.5, 17.5, -2.125},
    LTM_low_arm49[] = { 1.625, 18.5, -2.125},
    LTM_low_arm50[] = { 1.625, 20.0, -2.125},
    LTM_low_arm51[] = { 2.0, 20.0, -2.125},
    LTM_low_arm52[] = {2.25, 20.0, -1.625};


float LTM_lown1[] = {0.2773, -0.9199, 0.2773},

    LTM_lown4[] = {0.3827, -0.9239, 0.0},
    LTM_lown5[] = {-0.3827,-0.9239, 0.0},
    LTM_lown6[] = {-0.9239, -0.3827, 0.0},
    LTM_lown9[] = {0.0, -0.2887, 0.9574},
    LTM_lown19[] = {0.9239, -0.3827, 0.0},
    LTM_lown20[] = {0.3827,-0.9239, 0.0},
    LTM_lown21[] = {-0.3827,-0.9239, 0.0},
    LTM_lown22[] = {-0.9239, -0.3827,0.0},
    LTM_lown23[] = {-0.4198, 0.8047, -0.4198},
    LTM_lown25[] = {0.0, 0.8866, -0.4626};

/****************************** LTM wrist data ******************************/

float LTM_rst1[] = {-3.5, 2.0, -1.5},
    LTM_rst2[] = {-2.5, 2.0, -2.5},
    LTM_rst3[] = {-0.0, 2.0, -2.5},
    LTM_rst4[] = {1.0, 1.732, -2.5},
    LTM_rst5[] = {1.732, 1.0, -2.5},
    LTM_rst6[] = {2.0, 0.0, -2.5},
    LTM_rst7[] = {1.732, -1.0, -2.5},
    LTM_rst8[] = {1.0, -1.732, -2.5},
    LTM_rst9[] = {0.0, -2.0, -2.5},
    LTM_rst10[] = {-2.5, -2.0, -2.5},
    LTM_rst11[] = {-3.5, -2.0, -1.5},
    LTM_rst12[] = {-3.5, 2.0, -3.5},
    LTM_rst13[] = {0.0, 2.0, -3.5},
    LTM_rst14[] = {1.0, 1.732, -3.5},
    LTM_rst15[] = {1.732, 1.0, -3.5},
    LTM_rst16[] = {2.0, 0.0, -3.5},
    LTM_rst17[] = {1.732, -1.0, -3.5},
    LTM_rst18[] = {1.0, -1.732, -3.5},
    LTM_rst19[] = {0.0, -2.0, -3.5},
    LTM_rst20[] = {-3.5, -2.0, -3.5},
    LTM_rst21[] = {-3.5, -2.5, -3.5},
```

```
    LTM_rst22[] = {-3.5, -2.5, 0.0},
    LTM_rst23[] = {-3.5, -2.1651, 1.25},
    LTM_rst24[] = {-4.4151, -1.25, 2.1651},
    LTM_rst25[] = {-4.75, 0.0, 2.5},
    LTM_rst26[] = {-4.4151, 1.25, 2.1651},
    LTM_rst27[] = {-3.5, 2.1651, 1.25},
    LTM_rst28[] = {-3.5, 2.5, 0.0},
    LTM_rst29[] = {-3.5, 2.5, -3.5},
    LTM_rst30[] = {-6.0, -2.5, -3.5},
    LTM_rst31[] = {-6.0, -2.5, 0.0},
    LTM_rst32[] = {-6.0, -2.1651, 1.25},
    LTM_rst33[] = {-6.0, -1.25, 2.1651},
    LTM_rst34[] = {-6.0, 0.0, 2.5},
    LTM_rst35[] = {-6.0, 1.25, 2.1651},
    LTM_rst36[] = {-6.0, 2.1651, 1.25},
    LTM_rst37[] = {-6.0, 2.5, 0.0},
    LTM_rst38[] = {-6.0, 2.5, -3.5};


/************************** LTM gripper Data ***************************/


float LTM_gp1[] = {-7.0, 1.5, 2.0},
    LTM_gp2[] = {-7.0, 2.5, 1.0},
    LTM_gp3[] = {-7.0, 2.5, -1.0},
    LTM_gp4[] = {-7.0, 1.5, -2.0},
    LTM_gp5[] = {-7.0, -1.5, -2.0},
    LTM_gp6[] = {-7.0, -2.5, -1.0},
    LTM_gp7[] = {-7.0, -2.5, 1.0},
    LTM_gp8[] = {-7.0, -1.5, 2.0},
    LTM_gp9[] = {-9.0, 1.5, 2.0},
    LTM_gp10[] = {-9.0, 2.5, 1.0},
    LTM_gp11[] = {-9.0, 2.5, -1.0},
    LTM_gp12[] = {-9.0, 1.5, -2.0},
    LTM_gp13[] = {-9.0, -1.5, -2.0},
    LTM_gp14[] = {-9.0, -2.5, -1.0},
    LTM_gp15[] = {-9.0, -2.5, 1.0},
    LTM_gp16[] = {-9.0, -1.5, 2.0},
    LTM_gp17[] = {-10.0, 1.5, 1.0},
    LTM_gp18[] = {-10.0, 1.5, -1.0},
    LTM_gp19[] = {-10.0, -1.5, -1.0},
    LTM_gp20[] = {-10.0, -1.5, 1.0};


float LTM_fg1[] = {-10.0, 0.0, 0.5},
    LTM_fg2[] = {-12.5, 0.0, 0.5},
    LTM_fg3[] = {-12.5, 0.5, 0.5},
    LTM_fg4[] = {-9.0, 1.4, 0.5},
    LTM_fg5[] = {-9.0, -1.0, 0.5},
    LTM_fg6[] = {-10.0, -1.0, 0.5},
```

```
    LTM_fg7[] = {-10.0, 0.0, -0.5},
    LTM_fg8[] = {-12.5, 0.0, -0.5},
    LTM_fg9[] = {-12.5, 0.5, -0.5},
    LTM_fg10[] = {-9.0, 1.5, -0.5},
    LTM_fg11[] = {-9.0, -1.0, -0.5},
    LTM_fg12[] = {-10.0, -1.0, -0.5};

float LTM_fg_n[] = { -0.3162, 0.9487, 0.0000};

/****************************** Draw a cylinder ******************************/
cylinder(array)
float *array;
{
    pushmatrix();
    translate(array[0], array[1], array[2]);
    rot(array[3], 'x');
    rot(array[4], 'y');
    scale(array[5], array[5], array[5]);

    bgnpolygon();
    n3f(cyn1); v3f(cyl1); v3f(cyl13); n3f(cyn2); v3f(cyl14); v3f(cyl2);
    endpolygon();
    bgnpolygon();
    v3f(cyl2); v3f(cyl14); n3f(cyn3); v3f(cyl15); v3f(cyl3);
    endpolygon();
    bgnpolygon();
    v3f(cyl3); v3f(cyl15); n3f(cyn4); v3f(cyl16); v3f(cyl4);
    endpolygon();
    bgnpolygon();
    v3f(cyl4); v3f(cyl16); n3f(cyn5); v3f(cyl17); v3f(cyl5);
    endpolygon();
    bgnpolygon();
    v3f(cyl5); v3f(cyl17); n3f(cyn6); v3f(cyl18); v3f(cyl6);
    endpolygon();
    bgnpolygon();
    v3f(cyl6); v3f(cyl18); n3f(cyn7); v3f(cyl19); v3f(cyl7);
    endpolygon();
    bgnpolygon();
    v3f(cyl7); v3f(cyl19); n3f(cyn8); v3f(cyl20); v3f(cyl8);
    endpolygon();
    bgnpolygon();
    v3f(cyl8); v3f(cyl20); n3f(cyn9); v3f(cyl21); v3f(cyl9);
    endpolygon();
    bgnpolygon();
    v3f(cyl9); v3f(cyl21); n3f(cyn10); v3f(cyl22); v3f(cyl10);
    endpolygon();
    bgnpolygon();
    v3f(cyl10); v3f(cyl22); n3f(cyn11); v3f(cyl23); v3f(cyl11);
    endpolygon();
    bgnpolygon();
    v3f(cyl11); v3f(cyl23); n3f(cyn12); v3f(cyl24); v3f(cyl12);
```

```
    endpolygon();
    bgnpolygon();
    v3f(cyl12); v3f(cyl24); n3f(cyn1); v3f(cyl13); v3f(cyl1);
    endpolygon();
    bgnpolygon();
    n3f(cyz2); v3f(cyl1); v3f(cyl2); v3f(cyl3); v3f(cyl4);
    v3f(cyl5); v3f(cyl6); v3f(cyl7); v3f(cyl8); v3f(cyl9);
    v3f(cyl10); v3f(cyl11); v3f(cyl12);
    endpolygon();

    bgnpolygon();
    n3f(cy_fs_n1); v3f(cyl13); n3f(cy_fs_n2); v3f(cyl14);
    n3f(cy_fs_n3); v3f(cyl15); n3f(cy_fs_n4); v3f(cyl16);
    n3f(cy_fs_n5); v3f(cyl17); n3f(cy_fs_n6); v3f(cyl18);
    n3f(cy_fs_n7); v3f(cyl19); n3f(cy_fs_n8); v3f(cyl20);
    n3f(cy_fs_n9); v3f(cyl21); n3f(cy_fs_n10); v3f(cyl22);
    n3f(cy_fs_n11); v3f(cyl23); n3f(cy_fs_n12); v3f(cyl24);
    endpolygon();
    popmatrix();
}

cylinder1(array)
float *array;
{
    pushmatrix();
    translate(array[0], array[1], array[2]);
    rot(array[3], 'x');
    rot(array[4], 'y');
    scale(array[5], array[5], array[6]);

    bgnpolygon();
    n3f(cyn1); v3f(cyl1); v3f(cyl13); n3f(cyn2); v3f(cyl14); v3f(cyl2);
    endpolygon();
    bgnpolygon();
    v3f(cyl2); v3f(cyl14); n3f(cyn3); v3f(cyl15); v3f(cyl3);
    endpolygon();
    bgnpolygon();
    v3f(cyl3); v3f(cyl15); n3f(cyn4); v3f(cyl16); v3f(cyl4);
    endpolygon();
    bgnpolygon();
    v3f(cyl4); v3f(cyl16); n3f(cyn5); v3f(cyl17); v3f(cyl5);
    endpolygon();
    bgnpolygon();
    v3f(cyl5); v3f(cyl17); n3f(cyn6); v3f(cyl18); v3f(cyl6);
    endpolygon();
    bgnpolygon();
    v3f(cyl6); v3f(cyl18); n3f(cyn7); v3f(cyl19); v3f(cyl7);
    endpolygon();
    bgnpolygon();
    v3f(cyl7); v3f(cyl19); n3f(cyn8); v3f(cyl20); v3f(cyl8);
    endpolygon();
```

```
    bgnpolygon();
    v3f(cyl8); v3f(cyl20); n3f(cyn9); v3f(cyl21); v3f(cyl9);
    endpolygon();
    bgnpolygon();
    v3f(cyl9); v3f(cyl21); n3f(cyn10); v3f(cyl22); v3f(cyl10);
    endpolygon();
    bgnpolygon();
    v3f(cyl10); v3f(cyl22); n3f(cyn11); v3f(cyl23); v3f(cyl11);
    endpolygon();
    bgnpolygon();
    v3f(cyl11); v3f(cyl23); n3f(cyn12); v3f(cyl24); v3f(cyl12);
    endpolygon();
    bgnpolygon();
    v3f(cyl12); v3f(cyl24); n3f(cyn1); v3f(cyl13); v3f(cyl1);
    endpolygon();
    bgnpolygon();
    n3f(cyz2); v3f(cyl1); v3f(cyl2); v3f(cyl3); v3f(cyl4);
    v3f(cyl5); v3f(cyl6); v3f(cyl7); v3f(cyl8); v3f(cyl9);
    v3f(cyl10); v3f(cyl11); v3f(cyl12);
    endpolygon();

    bgnpolygon();
    n3f(cy_fs_n1); v3f(cyl13); n3f(cy_fs_n2); v3f(cyl14);
    n3f(cy_fs_n3); v3f(cyl15); n3f(cy_fs_n4); v3f(cyl16);
    n3f(cy_fs_n5); v3f(cyl17); n3f(cy_fs_n6); v3f(cyl18);
    n3f(cy_fs_n7); v3f(cyl19); n3f(cy_fs_n8); v3f(cyl20);
    n3f(cy_fs_n9); v3f(cyl21); n3f(cy_fs_n10); v3f(cyl22);
    n3f(cy_fs_n11); v3f(cyl23); n3f(cy_fs_n12); v3f(cyl24);
    endpolygon();
    popmatrix();
}

/****************************** Draw Axis *******************************/

draw_LTM_axis()
{
    lmbind(MATERIAL, 3);
    bgnpolygon();
    n3f(axis_n1); v3f(LTM_axis1); v3f(LTM_axis9);
    n3f(axis_n2); v3f(LTM_axis10); v3f(LTM_axis2);
    endpolygon();
    bgnpolygon();
    v3f(LTM_axis2); v3f(LTM_axis10); n3f(axis_n3);
    v3f(LTM_axis11); v3f(LTM_axis3);
    endpolygon();
    bgnpolygon();
    v3f(LTM_axis3); v3f(LTM_axis11); n3f(axis_n4);
    v3f(LTM_axis12); v3f(LTM_axis4);
    endpolygon();
    bgnpolygon();
```

```
v3f(LTM_axis4); v3f(LTM_axis12); n3f(axis_n5);
v3f(LTM_axis13); v3f(LTM_axis5);
endpolygon();
bgnpolygon();
v3f(LTM_axis5); v3f(LTM_axis13); n3f(axis_n6);
v3f(LTM_axis14); v3f(LTM_axis6);
endpolygon();
bgnpolygon();
v3f(LTM_axis6); v3f(LTM_axis14); n3f(axis_n7);
v3f(LTM_axis15); v3f(LTM_axis7);
endpolygon();
bgnpolygon();
v3f(LTM_axis7); v3f(LTM_axis15); n3f(axis_n8);
v3f(LTM_axis16); v3f(LTM_axis8);
endpolygon();
bgnpolygon();
v3f(LTM_axis8); v3f(LTM_axis16); n3f(axis_n1);
v3f(LTM_axis9); v3f(LTM_axis1);
endpolygon();

bgnpolygon();
n3f(axis_n17); v3f(LTM_axis17); v3f(LTM_axis25);
n3f(axis_n18); v3f(LTM_axis26); v3f(LTM_axis18);
endpolygon();
bgnpolygon();
v3f(LTM_axis18); v3f(LTM_axis26); n3f(axis_n19);
v3f(LTM_axis27); v3f(LTM_axis19);
endpolygon();
bgnpolygon();
v3f(LTM_axis19); v3f(LTM_axis27); n3f(axis_n20);
v3f(LTM_axis28); v3f(LTM_axis20);
endpolygon();
bgnpolygon();
v3f(LTM_axis20); v3f(LTM_axis28); n3f(axis_n21);
v3f(LTM_axis29); v3f(LTM_axis21);
endpolygon();
bgnpolygon();
v3f(LTM_axis21); v3f(LTM_axis29); n3f(axis_n22);
v3f(LTM_axis30); v3f(LTM_axis22);
endpolygon();
bgnpolygon();
v3f(LTM_axis22); v3f(LTM_axis30); n3f(axis_n23);
v3f(LTM_axis31); v3f(LTM_axis23);
endpolygon();
bgnpolygon();
v3f(LTM_axis23); v3f(LTM_axis31); n3f(axis_n24);
v3f(LTM_axis32); v3f(LTM_axis24);
endpolygon();
bgnpolygon();
v3f(LTM_axis24); v3f(LTM_axis32); n3f(axis_n17);
v3f(LTM_axis25); v3f(LTM_axis17);
```

```
endpolygon();

bgnpolygon();
n3f(axis_n33); v3f(LTM_axis33); v3f(LTM_axis41);
n3f(axis_n34); v3f(LTM_axis42); v3f(LTM_axis34);
endpolygon();
bgnpolygon();
v3f(LTM_axis34); v3f(LTM_axis42); n3f(axis_n35);
v3f(LTM_axis43); v3f(LTM_axis35);
endpolygon();
bgnpolygon();
v3f(LTM_axis35); v3f(LTM_axis43); n3f(axis_n36);
v3f(LTM_axis44); v3f(LTM_axis36);
endpolygon();
bgnpolygon();
v3f(LTM_axis36); v3f(LTM_axis44); n3f(axis_n37);
v3f(LTM_axis45); v3f(LTM_axis37);
endpolygon();
bgnpolygon();
v3f(LTM_axis37); v3f(LTM_axis45); n3f(axis_n38);
v3f(LTM_axis46); v3f(LTM_axis38);
endpolygon();
bgnpolygon();
v3f(LTM_axis38); v3f(LTM_axis46); n3f(axis_n39);
v3f(LTM_axis47); v3f(LTM_axis39);
endpolygon();
bgnpolygon();
v3f(LTM_axis39); v3f(LTM_axis47); n3f(axis_n40);
v3f(LTM_axis48); v3f(LTM_axis40);
endpolygon();
bgnpolygon();
v3f(LTM_axis40); v3f(LTM_axis48); n3f(axis_n33);
v3f(LTM_axis41); v3f(LTM_axis33);
endpolygon();

bgnpolygon();
n3f(axis_n49); v3f(LTM_axis33); v3f(LTM_axis49);
n3f(axis_n50); v3f(LTM_axis50); v3f(LTM_axis34);
endpolygon();
bgnpolygon();
v3f(LTM_axis34); v3f(LTM_axis50); n3f(axis_n51);
v3f(LTM_axis51); v3f(LTM_axis35);
endpolygon();
bgnpolygon();
v3f(LTM_axis35); v3f(LTM_axis51); n3f(axis_n52);
v3f(LTM_axis52); v3f(LTM_axis36);
endpolygon();
bgnpolygon();
v3f(LTM_axis36); v3f(LTM_axis52); n3f(axis_n53);
v3f(LTM_axis53); v3f(LTM_axis37);
endpolygon();
```

```
    bgnpolygon();
    v3f(LTM_axis37); v3f(LTM_axis53); n3f(axis_n54);
    v3f(LTM_axis54); v3f(LTM_axis38);
    endpolygon();
    bgnpolygon();
    v3f(LTM_axis38); v3f(LTM_axis54); n3f(axis_n55);
    v3f(LTM_axis55); v3f(LTM_axis39);
    endpolygon();
    bgnpolygon();
    v3f(LTM_axis39); v3f(LTM_axis55); n3f(axis_n56);
    v3f(LTM_axis56); v3f(LTM_axis40);
    endpolygon();
    bgnpolygon();
    v3f(LTM_axis40); v3f(LTM_axis56); n3f(axis_n49);
    v3f(LTM_axis49); v3f(LTM_axis33);
    endpolygon();



    lmbind(MATERIAL, 2);
    bgnpolygon();
    n3f(ny2); v3f(LTM_axis57); v3f(LTM_axis58); v3f(LTM_axis59); v3f(LTM_axis60);
    endpolygon();
    bgnpolygon();
    n3f(ny1); v3f(LTM_axis61); v3f(LTM_axis62); v3f(LTM_axis63); v3f(LTM_axis64);
    endpolygon();
    bgnpolygon();
    n3f(nz2); v3f(LTM_axis58); v3f(LTM_axis62); v3f(LTM_axis63); v3f(LTM_axis59);
    endpolygon();
}


/********************** Draw the base connection of LTM **********************/
draw_LTM_stand()
{
    bgnpolygon();
    n3f(nz1); v3f(LTM_stand1); v3f(LTM_stand9);
    n3f(nxy1); v3f(LTM_stand10); v3f(LTM_stand2);
    endpolygon();
    bgnpolygon();
    v3f(LTM_stand2); v3f(LTM_stand10); v3f(LTM_stand11); v3f(LTM_stand3);
    endpolygon();
    bgnpolygon();
    v3f(LTM_stand3); v3f(LTM_stand11);
    n3f(nyz2); v3f(LTM_stand12); v3f(LTM_stand4);
    endpolygon();
    bgnpolygon();
    v3f(LTM_stand4); v3f(LTM_stand12); v3f(LTM_stand13); v3f(LTM_stand5);
    endpolygon();
    bgnpolygon();
    v3f(LTM_stand5); v3f(LTM_stand13);
```

```
    n3f(nxy4); v3f(LTM_stand14); v3f(LTM_stand6);
    endpolygon();
    bgnpolygon();
    v3f(LTM_stand6); v3f(LTM_stand14); v3f(LTM_stand15); v3f(LTM_stand7);
    endpolygon();
    bgnpolygon();
    v3f(LTM_stand7); v3f(LTM_stand15);
    n3f(nz1); v3f(LTM_stand16); v3f(LTM_stand8);
    endpolygon();
    bgnpolygon();
    v3f(LTM_stand8); v3f(LTM_stand16); v3f(LTM_stand9); v3f(LTM_stand1);
    endpolygon();

    lmbind(MATERIAL, 4);

    bgnpolygon();
    n3f(nz1); v3f(LTM_stand9); v3f(LTM_stand17);
    n3f(nx1); v3f(LTM_stand18); v3f(LTM_stand10);
    endpolygon();
    bgnpolygon();
    v3f(LTM_stand10); v3f(LTM_stand18); v3f(LTM_stand19); v3f(LTM_stand11);
    endpolygon();
    bgnpolygon();
    v3f(LTM_stand11); v3f(LTM_stand19);
    n3f(nz2); v3f(LTM_stand20); v3f(LTM_stand12);
    endpolygon();
    bgnpolygon();
    v3f(LTM_stand12); v3f(LTM_stand20); v3f(LTM_stand21); v3f(LTM_stand13);
    endpolygon();
    bgnpolygon();
    v3f(LTM_stand13); v3f(LTM_stand21);
    n3f(nx2); v3f(LTM_stand22); v3f(LTM_stand14);
    endpolygon();
    bgnpolygon();
    v3f(LTM_stand14); v3f(LTM_stand22); v3f(LTM_stand23); v3f(LTM_stand15);
    endpolygon();
    bgnpolygon();
    v3f(LTM_stand15); v3f(LTM_stand23);
    n3f(nz1); v3f(LTM_stand24); v3f(LTM_stand16);
    endpolygon();
    bgnpolygon();
    v3f(LTM_stand16); v3f(LTM_stand24); v3f(LTM_stand17); v3f(LTM_stand9);
    endpolygon();
    bgnpolygon();
    n3f(ny2); v3f(LTM_stand17); v3f(LTM_stand18);
    v3f(LTM_stand19); v3f(LTM_stand20); v3f(LTM_stand21);
    v3f(LTM_stand22); v3f(LTM_stand23); v3f(LTM_stand24);
    endpolygon();
    lmbind(MATERIAL, 2);

}
```

```
/****************************** Draw the Base ******************************/

draw_LTM_part()
{
    bgnpolygon();
    n3f(nzxp1); v3f(LTM_bas1); v3f(LTM_bas9);
    n3f(nxzp1); v3f(LTM_bas10); v3f(LTM_bas2);
    endpolygon();
    bgnpolygon();
    v3f(LTM_bas2); v3f(LTM_bas10); n3f(nxzp2);
    v3f(LTM_bas11); v3f(LTM_bas3);
    endpolygon();
    bgnpolygon();
    v3f(LTM_bas3); v3f(LTM_bas11); n3f(nzxn1);
    v3f(LTM_bas12); v3f(LTM_bas4);
    endpolygon();
    bgnpolygon();
    v3f(LTM_bas4); v3f(LTM_bas12); v3f(LTM_bas28); v3f(LTM_bas27);
    endpolygon();
    bgnpolygon();
    v3f(LTM_bas4); v3f(LTM_bas27); v3f(LTM_bas26);
    endpolygon();
    bgnpolygon();
    v3f(LTM_bas4); v3f(LTM_bas26); n3f(nzxn2);
    v3f(LTM_bas25); v3f(LTM_bas5);
    endpolygon();
    bgnpolygon();
    v3f(LTM_bas5); v3f(LTM_bas25); v3f(LTM_bas24);
    endpolygon();
    bgnpolygon();
    v3f(LTM_bas5); v3f(LTM_bas24); v3f(LTM_bas23); v3f(LTM_bas13);
    endpolygon();
    bgnpolygon();
    n3f(nzxn2);
    v3f(LTM_bas5); v3f(LTM_bas13);  n3f(nxzn2);
    v3f(LTM_bas14); v3f(LTM_bas6);
    endpolygon();
    bgnpolygon();
    v3f(LTM_bas6); v3f(LTM_bas14); n3f(nxzn1);
    v3f(LTM_bas15); v3f(LTM_bas7);
    endpolygon();
    bgnpolygon();
    v3f(LTM_bas7); v3f(LTM_bas15); n3f(nzxp2);
    v3f(LTM_bas16); v3f(LTM_bas8);
    endpolygon();
    bgnpolygon();
    v3f(LTM_bas8); v3f(LTM_bas16); v3f(LTM_bas17); v3f(LTM_bas18);
    endpolygon();
    bgnpolygon();
    v3f(LTM_bas8); v3f(LTM_bas18); v3f(LTM_bas19);
    endpolygon();
```

```
    bgnpolygon();
    v3f(LTM_bas8); v3f(LTM_bas19); n3f(nzxp1);
    v3f(LTM_bas20); v3f(LTM_bas1);
    endpolygon();
    bgnpolygon();
    v3f(LTM_bas1); v3f(LTM_bas20); v3f(LTM_bas21);
    endpolygon();
    bgnpolygon();
    v3f(LTM_bas1); v3f(LTM_bas21); v3f(LTM_bas22); v3f(LTM_bas9);
    endpolygon();
    bgnpolygon();
    n3f(nx1);
    v3f(LTM_bas17); v3f(LTM_bas18); v3f(LTM_bas24); v3f(LTM_bas23);
    endpolygon();
    bgnpolygon();
    n3f(nx2);
    v3f(LTM_bas21); v3f(LTM_bas22); v3f(LTM_bas28); v3f(LTM_bas27);
    endpolygon();
    bgnpolygon();
    n3f(ny1);
    v3f(LTM_bas19); v3f(LTM_bas20); v3f(LTM_bas26); v3f(LTM_bas25);
    endpolygon();
    bgnpolygon();
    n3f(nx2); v3f(LTM_bas21); v3f(LTM_bas27);
    n3f(ny1);  v3f(LTM_bas26); v3f(LTM_bas20);
    endpolygon();
    bgnpolygon();
    v3f(LTM_bas19); v3f(LTM_bas25);
    n3f(nx1);  v3f(LTM_bas24); v3f(LTM_bas18);
    endpolygon();
    cylinder(cyl_array[0]);
    cylinder(cyl_array[1]);
}

/*************************** Draw Upper Arm ****************************/

draw_LTM_uparm()
{
    bgnpolygon();
    n3f(LTM_uparm_n8); v3f(LTM_uparm1); v3f(LTM_uparm9);
    n3f(LTM_uparm_n2); v3f(LTM_uparm10); v3f(LTM_uparm2);
    endpolygon();
    bgnpolygon();
    v3f(LTM_uparm2); v3f(LTM_uparm10); v3f(LTM_uparm11); v3f(LTM_uparm3);
    endpolygon();
    bgnpolygon();
    v3f(LTM_uparm3); v3f(LTM_uparm11); n3f(LTM_uparm_n4);
    v3f(LTM_uparm12); v3f(LTM_uparm4);
    endpolygon();
    bgnpolygon();
    v3f(LTM_uparm4); v3f(LTM_uparm12); v3f(LTM_uparm13); v3f(LTM_uparm5);
```

```
endpolygon();
bgnpolygon();
v3f(LTM_uparm5); v3f(LTM_uparm13); n3f(LTM_uparm_n6);
v3f(LTM_uparm14); v3f(LTM_uparm6);
endpolygon();
bgnpolygon();
v3f(LTM_uparm6); v3f(LTM_uparm14); v3f(LTM_uparm15); v3f(LTM_uparm7);
endpolygon();
bgnpolygon();
v3f(LTM_uparm7); v3f(LTM_uparm15); n3f(LTM_uparm_n8);
v3f(LTM_uparm16); v3f(LTM_uparm8);
endpolygon();
bgnpolygon();
v3f(LTM_uparm8); v3f(LTM_uparm1);  v3f(LTM_uparm9); v3f(LTM_uparm16);
endpolygon();
bgnpolygon();
n3f(nz1); v3f(LTM_uparm9); v3f(LTM_uparm17);
n3f(nx1); v3f(LTM_uparm18); v3f(LTM_uparm10);
endpolygon();
bgnpolygon();
v3f(LTM_uparm10); v3f(LTM_uparm18); v3f(LTM_uparm19); v3f(LTM_uparm11);
endpolygon();
bgnpolygon();
v3f(LTM_uparm11); v3f(LTM_uparm19); n3f(nz2);
v3f(LTM_uparm20); v3f(LTM_uparm12);
endpolygon();
bgnpolygon();
v3f(LTM_uparm12); v3f(LTM_uparm20); v3f(LTM_uparm21); v3f(LTM_uparm13);
endpolygon();
bgnpolygon();
v3f(LTM_uparm13); v3f(LTM_uparm21); n3f(nx2);
v3f(LTM_uparm22); v3f(LTM_uparm14);
endpolygon();
bgnpolygon();
v3f(LTM_uparm14); v3f(LTM_uparm22); v3f(LTM_uparm23); v3f(LTM_uparm15);
endpolygon();
bgnpolygon();
v3f(LTM_uparm15); v3f(LTM_uparm23); n3f(nz1);
v3f(LTM_uparm24); v3f(LTM_uparm16);
endpolygon();
bgnpolygon();
v3f(LTM_uparm16); v3f(LTM_uparm24); v3f(LTM_uparm17); v3f(LTM_uparm9);
endpolygon();
bgnpolygon();
n3f(nz1); v3f(LTM_uparm24); v3f(LTM_uparm25);
n3f(LTM_uparm_n27); v3f(LTM_uparm27);
n3f(nx2); v3f(LTM_uparm23);
endpolygon();
bgnpolygon();
n3f(nz1); v3f(LTM_uparm25); v3f(LTM_uparm26);
n3f(LTM_uparm_n28); v3f(LTM_uparm28);
```

```
    n3f(LTM_uparm_n27); v3f(LTM_uparm27);
    endpolygon();
    bgnpolygon();
    n3f(nz1); v3f(LTM_uparm26); v3f(LTM_uparm17);
    n3f(nx1); v3f(LTM_uparm18);
    n3f(LTM_uparm_n28); v3f(LTM_uparm28);
    endpolygon();
    bgnpolygon();
    n3f(nz2); v3f(LTM_uparm21); v3f(LTM_uparm31);
    n3f(LTM_uparm_n29); v3f(LTM_uparm29);
    n3f(nx2); v3f(LTM_uparm22);
    endpolygon();
    bgnpolygon();
    n3f(nz2); v3f(LTM_uparm31); v3f(LTM_uparm32);
    n3f(LTM_uparm_n30); v3f(LTM_uparm30);
    n3f(LTM_uparm_n29); v3f(LTM_uparm29);
    endpolygon();
    bgnpolygon();
    n3f(nz2); v3f(LTM_uparm32); v3f(LTM_uparm20);
    n3f(nx1); v3f(LTM_uparm19);
    n3f(LTM_uparm_n30); v3f(LTM_uparm30);
    endpolygon();
    bgnpolygon();
    n3f(nx2); v3f(LTM_uparm22); v3f(LTM_uparm23);
    n3f(LTM_uparm_n27); v3f(LTM_uparm27);
    n3f(LTM_uparm_n29); v3f(LTM_uparm29);
    endpolygon();
    bgnpolygon();
    n3f(LTM_uparm_n27); v3f(LTM_uparm27);
    n3f(LTM_uparm_n28); v3f(LTM_uparm28);
    n3f(LTM_uparm_n30); v3f(LTM_uparm30);
    n3f(LTM_uparm_n29); v3f(LTM_uparm29);
    endpolygon();
    bgnpolygon();
    n3f(LTM_uparm_n28); v3f(LTM_uparm28);
    n3f(nx1); v3f(LTM_uparm18); v3f(LTM_uparm19);
    n3f(LTM_uparm_n30); v3f(LTM_uparm30);
    endpolygon();
    bgnpolygon();
    n3f(nz1);
    v3f(LTM_uparm24); v3f(LTM_uparm25); v3f(LTM_uparm26); v3f(LTM_uparm17);
    endpolygon();
    bgnpolygon();
    n3f(nz2);
    v3f(LTM_uparm21); v3f(LTM_uparm31); v3f(LTM_uparm32); v3f(LTM_uparm20);
    endpolygon();
    translate(0.0, 23.0, 0.0);
    draw_LTM_part();

}
```

```
/*************************** Draw Lower Arm ****************************/

draw_LTM_lowarm()
{
    bgnpolygon();
    n3f(nx1); v3f(LTM_low_arm1); v3f(LTM_low_arm2);
    n3f(LTM_lown9); v3f(LTM_low_arm10); v3f(LTM_low_arm9);
    endpolygon();
    bgnpolygon();
    n3f(nx1); v3f(LTM_low_arm2); v3f(LTM_low_arm3);
    n3f(nz1); v3f(LTM_low_arm11); v3f(LTM_low_arm10);
    endpolygon();
    bgnpolygon();
    n3f(nx1); v3f(LTM_low_arm3); n3f(LTM_lown4); v3f(LTM_low_arm4);
    n3f(nz1); v3f(LTM_low_arm12); v3f(LTM_low_arm11);
    endpolygon();
    bgnpolygon();
    n3f(LTM_lown4); v3f(LTM_low_arm4); n3f(LTM_lown5); v3f(LTM_low_arm5);
    n3f(nz1);  v3f(LTM_low_arm13); v3f(LTM_low_arm12);
    endpolygon();
    bgnpolygon();
    n3f(LTM_lown5); v3f(LTM_low_arm5); n3f(nx2); v3f(LTM_low_arm6);
    n3f(nz1); v3f(LTM_low_arm14); v3f(LTM_low_arm13);
    endpolygon();
    bgnpolygon();
    n3f(nx2); v3f(LTM_low_arm6); v3f(LTM_low_arm7);
    n3f(nz1); v3f(LTM_low_arm15); v3f(LTM_low_arm14);
    endpolygon();
    bgnpolygon();
    n3f(nx2); v3f(LTM_low_arm7); v3f(LTM_low_arm8);
    n3f(LTM_lown9); v3f(LTM_low_arm16); v3f(LTM_low_arm15);
    endpolygon();
    bgnpolygon();
    n3f(nx1); v3f(LTM_low_arm17); v3f(LTM_low_arm18);
    n3f(LTM_lown25); v3f(LTM_low_arm26); v3f(LTM_low_arm25);
    endpolygon();
    bgnpolygon();
    n3f(nx1); v3f(LTM_low_arm18); v3f(LTM_low_arm19);
    n3f(nz2); v3f(LTM_low_arm27); v3f(LTM_low_arm26);
    endpolygon();
    bgnpolygon();
    n3f(nx1); v3f(LTM_low_arm19); n3f(LTM_lown20); v3f(LTM_low_arm20);
    n3f(nz2); v3f(LTM_low_arm28); v3f(LTM_low_arm27);
    endpolygon();
    bgnpolygon();
    n3f(LTM_lown20); v3f(LTM_low_arm20); n3f(LTM_lown21); v3f(LTM_low_arm21);
    n3f(nz2); v3f(LTM_low_arm29); v3f(LTM_low_arm28);
    endpolygon();
    bgnpolygon();
    n3f(LTM_lown21); v3f(LTM_low_arm21); n3f(nx2); v3f(LTM_low_arm22);
    n3f(nz2);  v3f(LTM_low_arm30); v3f(LTM_low_arm29);
```

```
endpolygon();
bgnpolygon();
n3f(nx2); v3f(LTM_low_arm22); v3f(LTM_low_arm23);
n3f(nz2); v3f(LTM_low_arm31); v3f(LTM_low_arm30);
endpolygon();
bgnpolygon();
n3f(nx2); v3f(LTM_low_arm23); v3f(LTM_low_arm24);
n3f(LTM_lown25); v3f(LTM_low_arm32); v3f(LTM_low_arm31);
endpolygon();
bgnpolygon();
n3f(nx1);
v3f(LTM_low_arm1); v3f(LTM_low_arm2);
v3f(LTM_low_arm18); v3f(LTM_low_arm17);
endpolygon();
bgnpolygon();
v3f(LTM_low_arm2); v3f(LTM_low_arm3);
v3f(LTM_low_arm19); v3f(LTM_low_arm18);
endpolygon();
bgnpolygon();
n3f(nx2);
v3f(LTM_low_arm6); v3f(LTM_low_arm7);
v3f(LTM_low_arm23); v3f(LTM_low_arm22);
endpolygon();
bgnpolygon();
v3f(LTM_low_arm7); v3f(LTM_low_arm8);
v3f(LTM_low_arm24); v3f(LTM_low_arm23);
endpolygon();
bgnpolygon();
n3f(nx1);
v3f(LTM_low_arm3); n3f(LTM_lown4); v3f(LTM_low_arm4);
n3f(LTM_lown20); v3f(LTM_low_arm20); n3f(nx1); v3f(LTM_low_arm19);
endpolygon();
bgnpolygon();
n3f(LTM_lown4); v3f(LTM_low_arm4); n3f(LTM_lown5); v3f(LTM_low_arm5);
n3f(LTM_lown21); v3f(LTM_low_arm21); n3f(LTM_lown20); v3f(LTM_low_arm20);
endpolygon();
bgnpolygon();
n3f(LTM_lown5); v3f(LTM_low_arm5);
n3f(nx2); v3f(LTM_low_arm6); v3f(LTM_low_arm22);
n3f(LTM_lown21); v3f(LTM_low_arm21);
endpolygon();
bgnpolygon();
n3f(LTM_lown9);
v3f(LTM_low_arm9); v3f(LTM_low_arm10);
v3f(LTM_low_arm15); v3f(LTM_low_arm16);
endpolygon();
bgnpolygon();
n3f(LTM_lown25);
v3f(LTM_low_arm25); v3f(LTM_low_arm26);
v3f(LTM_low_arm31); v3f(LTM_low_arm32);
endpolygon();
```

```
bgnpolygon();
n3f(nz1);
v3f(LTM_low_arm10); v3f(LTM_low_arm11); v3f(LTM_low_arm12);
v3f(LTM_low_arm13); v3f(LTM_low_arm14); v3f(LTM_low_arm15);
endpolygon();
bgnpolygon();
n3f(nz2);
v3f(LTM_low_arm26); v3f(LTM_low_arm27); v3f(LTM_low_arm28);
v3f(LTM_low_arm29); v3f(LTM_low_arm30); v3f(LTM_low_arm31);
endpolygon();

bgnpolygon();
n3f(nx2); v3f(LTM_low_arm8); v3f(LTM_low_arm33);
n3f(nz1); v3f(LTM_low_arm34); v3f(LTM_low_arm16);
endpolygon();
bgnpolygon();
v3f(LTM_low_arm9); v3f(LTM_low_arm41);
n3f(nx1); v3f(LTM_low_arm42); v3f(LTM_low_arm1);
endpolygon();
bgnpolygon();
v3f(LTM_low_arm1); v3f(LTM_low_arm42);
v3f(LTM_low_arm52); v3f(LTM_low_arm17);
endpolygon();
bgnpolygon();
v3f(LTM_low_arm17); v3f(LTM_low_arm52);
n3f(nz2); v3f(LTM_low_arm51); v3f(LTM_low_arm25);
endpolygon();
bgnpolygon();
v3f(LTM_low_arm32); v3f(LTM_low_arm44);
n3f(nx2); v3f(LTM_low_arm43); v3f(LTM_low_arm24);
endpolygon();
bgnpolygon();
v3f(LTM_low_arm24); v3f(LTM_low_arm43);
v3f(LTM_low_arm33); v3f(LTM_low_arm8);
endpolygon();
bgnpolygon();
n3f(nz1);
v3f(LTM_low_arm16); v3f(LTM_low_arm34);
v3f(LTM_low_arm35); v3f(LTM_low_arm36);
endpolygon();
bgnpolygon();
v3f(LTM_low_arm16); v3f(LTM_low_arm36); v3f(LTM_low_arm37);
endpolygon();
bgnpolygon();
v3f(LTM_low_arm16); v3f(LTM_low_arm37);
v3f(LTM_low_arm38); v3f(LTM_low_arm9);
endpolygon();
bgnpolygon();
v3f(LTM_low_arm9); v3f(LTM_low_arm38); v3f(LTM_low_arm39);
endpolygon();
bgnpolygon();
```

```
    v3f(LTM_low_arm9); v3f(LTM_low_arm39);
    v3f(LTM_low_arm40); v3f(LTM_low_arm41);
    endpolygon();
    bgnpolygon();
    n3f(nz2);
    v3f(LTM_low_arm32); v3f(LTM_low_arm44);
    v3f(LTM_low_arm45); v3f(LTM_low_arm46);
    endpolygon();
    bgnpolygon();
    v3f(LTM_low_arm32); v3f(LTM_low_arm46); v3f(LTM_low_arm47);
    endpolygon();
    bgnpolygon();
    v3f(LTM_low_arm32); v3f(LTM_low_arm47);
    v3f(LTM_low_arm48); v3f(LTM_low_arm25);
    endpolygon();
    bgnpolygon();
    v3f(LTM_low_arm25); v3f(LTM_low_arm48); v3f(LTM_low_arm49);
    endpolygon();
    bgnpolygon();
    v3f(LTM_low_arm25); v3f(LTM_low_arm49);
    v3f(LTM_low_arm50); v3f(LTM_low_arm51);
    endpolygon();
    bgnpolygon();
    n3f(nx2);
    v3f(LTM_low_arm39); v3f(LTM_low_arm40);
    v3f(LTM_low_arm50); v3f(LTM_low_arm49);
    endpolygon();
    bgnpolygon();
    n3f(nx1);
    v3f(LTM_low_arm35); v3f(LTM_low_arm36);
    v3f(LTM_low_arm46); v3f(LTM_low_arm45);
    endpolygon();
    bgnpolygon();
    n3f(ny1);
    v3f(LTM_low_arm37); v3f(LTM_low_arm38);
    v3f(LTM_low_arm48); v3f(LTM_low_arm47);
    endpolygon();
    bgnpolygon();
    n3f(nx2); v3f(LTM_low_arm39); v3f(LTM_low_arm49);
    n3f(ny1); v3f(LTM_low_arm48); v3f(LTM_low_arm38);
    endpolygon();
    bgnpolygon();
    n3f(nx1); v3f(LTM_low_arm36); v3f(LTM_low_arm46);
    n3f(ny1); v3f(LTM_low_arm47); v3f(LTM_low_arm37);
    endpolygon();
    cylinder(cyl_array[2]);
    cylinder(cyl_array[3]);

}
```

```
/****************************** Draw Wrist *********************************/

draw_LTM_rst()
{
    bgnpolygon();
    n3f(nx1); v3f(LTM_rst1);  v3f(LTM_rst11);
    n3f(nz1); v3f(LTM_rst10);  v3f(LTM_rst2);
    endpolygon();
    bgnpolygon();
    v3f(LTM_rst2); v3f(LTM_rst3); v3f(LTM_rst4); v3f(LTM_rst5);
    v3f(LTM_rst6); v3f(LTM_rst7); v3f(LTM_rst8); v3f(LTM_rst9);
    v3f(LTM_rst10);
    endpolygon();
    bgnpolygon();
    n3f(nz2); v3f(LTM_rst12); v3f(LTM_rst13); v3f(LTM_rst14); v3f(LTM_rst15);
    v3f(LTM_rst16); v3f(LTM_rst17); v3f(LTM_rst18); v3f(LTM_rst19);
    v3f(LTM_rst20);
    endpolygon();
    bgnpolygon();
    n3f(ny1); v3f(LTM_rst1);  v3f(LTM_rst2); v3f(LTM_rst12);
    endpolygon();
    bgnpolygon();
    v3f(LTM_rst2);  v3f(LTM_rst12); v3f(LTM_rst13); v3f(LTM_rst3);
    endpolygon();
    bgnpolygon();
    v3f(LTM_rst3);  v3f(LTM_rst13); n3f(cxy_n2);
    v3f(LTM_rst14); v3f(LTM_rst4);
    endpolygon();
    bgnpolygon();
    v3f(LTM_rst4);  v3f(LTM_rst14); n3f(cxy_n3);
    v3f(LTM_rst15); v3f(LTM_rst5);
    endpolygon();
    bgnpolygon();
    v3f(LTM_rst5);  v3f(LTM_rst15); n3f(cxy_n4);
    v3f(LTM_rst16); v3f(LTM_rst6);
    endpolygon();
    bgnpolygon();
    v3f(LTM_rst6);  v3f(LTM_rst16); n3f(cxy_n5);
    v3f(LTM_rst17); v3f(LTM_rst7);
    endpolygon();
    bgnpolygon();
    v3f(LTM_rst7);  v3f(LTM_rst17); n3f(cxy_n6);
    v3f(LTM_rst18); v3f(LTM_rst8);
    endpolygon();
    bgnpolygon();
    v3f(LTM_rst8);  v3f(LTM_rst18); n3f(ny2);
    v3f(LTM_rst19); v3f(LTM_rst9);
    endpolygon();
    bgnpolygon();
    v3f(LTM_rst9);  v3f(LTM_rst19); v3f(LTM_rst20); v3f(LTM_rst10);
    endpolygon();
```

```
    bgnpolygon();
    v3f(LTM_rst10);  v3f(LTM_rst20); v3f(LTM_rst11);
    endpolygon();
    bgnpolygon();
    n3f(nx1); v3f(LTM_rst21);  v3f(LTM_rst22); v3f(LTM_rst23);
    v3f(LTM_rst27); v3f(LTM_rst28); v3f(LTM_rst29);
    endpolygon();
    bgnpolygon();
    n3f(nxz1);  v3f(LTM_rst23); v3f(LTM_rst24);
    v3f(LTM_rst25); v3f(LTM_rst26);  v3f(LTM_rst27);
    endpolygon();
    bgnpolygon();
    n3f(nx2); v3f(LTM_rst30);  v3f(LTM_rst31); v3f(LTM_rst32);
    v3f(LTM_rst33); v3f(LTM_rst34); v3f(LTM_rst35); v3f(LTM_rst36);
    v3f(LTM_rst37); v3f(LTM_rst38);
    endpolygon();
    bgnpolygon();
    n3f(ny2); v3f(LTM_rst21);  v3f(LTM_rst22); v3f(LTM_rst31); v3f(LTM_rst30);
    endpolygon();
    bgnpolygon();
    v3f(LTM_rst22);  v3f(LTM_rst31); n3f(cyz_n11);
    v3f(LTM_rst32); v3f(LTM_rst23);
    endpolygon();
    bgnpolygon();
    v3f(LTM_rst23);  v3f(LTM_rst32); n3f(cyz_n12);
    v3f(LTM_rst33); v3f(LTM_rst24);
    endpolygon();
    bgnpolygon();
    v3f(LTM_rst24);  v3f(LTM_rst33); n3f(cyz_n1);
    v3f(LTM_rst34); v3f(LTM_rst25);
    endpolygon();
    bgnpolygon();
    v3f(LTM_rst25);  v3f(LTM_rst34); n3f(cyz_n2);
    v3f(LTM_rst35); v3f(LTM_rst26);
    endpolygon();
    bgnpolygon();
    v3f(LTM_rst26);  v3f(LTM_rst35); n3f(cyz_n3);
    v3f(LTM_rst36); v3f(LTM_rst27);
    endpolygon();
    bgnpolygon();
    v3f(LTM_rst27);  v3f(LTM_rst36); n3f(ny1);
    v3f(LTM_rst37); v3f(LTM_rst28);
    endpolygon();
    bgnpolygon();
    v3f(LTM_rst28);  v3f(LTM_rst37); v3f(LTM_rst38); v3f(LTM_rst29);
    endpolygon();
    bgnpolygon();
    n3f(nz2);
    v3f(LTM_rst21);  v3f(LTM_rst30); v3f(LTM_rst38); v3f(LTM_rst29);
    endpolygon();
}
```

```
/****************************** Draw gripper ******************************/

draw_LTM_grip()
{
    bgnpolygon();
    n3f(nx1); v3f(LTM_gp1); v3f(LTM_gp2); v3f(LTM_gp3); v3f(LTM_gp4);
    v3f(LTM_gp5); v3f(LTM_gp6); v3f(LTM_gp7); v3f(LTM_gp8);
    endpolygon();
    bgnpolygon();
    n3f(nz1); v3f(LTM_gp1); v3f(LTM_gp9);
    n3f(ny1); v3f(LTM_gp10); v3f(LTM_gp2);
    endpolygon();
    bgnpolygon();
    v3f(LTM_gp2); v3f(LTM_gp10); v3f(LTM_gp11); v3f(LTM_gp3);
    endpolygon();
    bgnpolygon();
    v3f(LTM_gp3); v3f(LTM_gp11);
    n3f(nz2); v3f(LTM_gp12); v3f(LTM_gp4);
    endpolygon();
    bgnpolygon();
    v3f(LTM_gp4); v3f(LTM_gp12); v3f(LTM_gp13); v3f(LTM_gp5);
    endpolygon();
    bgnpolygon();
    v3f(LTM_gp5); v3f(LTM_gp13);
    n3f(ny2); v3f(LTM_gp14); v3f(LTM_gp6);
    endpolygon();
    bgnpolygon();
    v3f(LTM_gp6); v3f(LTM_gp14); v3f(LTM_gp15); v3f(LTM_gp7);
    endpolygon();
    bgnpolygon();
    v3f(LTM_gp7); v3f(LTM_gp15);
    n3f(nz1); v3f(LTM_gp16); v3f(LTM_gp8);
    endpolygon();
    bgnpolygon();
    v3f(LTM_gp8); v3f(LTM_gp16); v3f(LTM_gp9); v3f(LTM_gp1);
    endpolygon();
    bgnpolygon();
    n3f(nxz4); v3f(LTM_gp9); v3f(LTM_gp17);
    n3f(nxy4); v3f(LTM_gp10);
    endpolygon();
    bgnpolygon();
    v3f(LTM_gp10); v3f(LTM_gp17); v3f(LTM_gp18); v3f(LTM_gp11);
    endpolygon();
    bgnpolygon();
    v3f(LTM_gp11); n3f(nxz3); v3f(LTM_gp18); v3f(LTM_gp12);
    endpolygon();
    bgnpolygon();
    v3f(LTM_gp12); v3f(LTM_gp18); v3f(LTM_gp19); v3f(LTM_gp13);
    endpolygon();
```

```
    bgnpolygon();
    v3f(LTM_gp13); v3f(LTM_gp19); n3f(nxy3); v3f(LTM_gp14);
    endpolygon();
    bgnpolygon();
    v3f(LTM_gp14); v3f(LTM_gp19); v3f(LTM_gp20); v3f(LTM_gp15);
    endpolygon();
    bgnpolygon();
    v3f(LTM_gp15); n3f(nxz4); v3f(LTM_gp20); v3f(LTM_gp16);
    endpolygon();
    bgnpolygon();
    v3f(LTM_gp16); v3f(LTM_gp20); v3f(LTM_gp17); v3f(LTM_gp9);
    endpolygon();
    bgnpolygon();
    n3f(nx2);
    v3f(LTM_gp17); v3f(LTM_gp18); v3f(LTM_gp19); v3f(LTM_gp20);
    endpolygon();
    cylinder1(cyl_array[4]);
}


/******************* Draw the moving part of the gripper **********************/
draw_LTM_fg()
{
    bgnpolygon();
    n3f(nz1); v3f(LTM_fg1); v3f(LTM_fg2); v3f(LTM_fg3); v3f(LTM_fg4);
    endpolygon();
    bgnpolygon();
    v3f(LTM_fg1); v3f(LTM_fg4); v3f(LTM_fg5); v3f(LTM_fg6);
    endpolygon();
    bgnpolygon();
    n3f(nz2); v3f(LTM_fg7); v3f(LTM_fg8); v3f(LTM_fg9); v3f(LTM_fg10);
    endpolygon();
    bgnpolygon();
    v3f(LTM_fg7); v3f(LTM_fg10); v3f(LTM_fg11); v3f(LTM_fg12);
    endpolygon();
    bgnpolygon();
    n3f(nz1); v3f(LTM_fg1); v3f(LTM_fg2); v3f(LTM_fg3); v3f(LTM_fg4);
    endpolygon();
    bgnpolygon();
    n3f(nx2);
    v3f(LTM_fg1); v3f(LTM_fg6); v3f(LTM_fg12); v3f(LTM_fg7);
    endpolygon();
    bgnpolygon();
    v3f(LTM_fg2); v3f(LTM_fg3); v3f(LTM_fg9); v3f(LTM_fg8);
    endpolygon();
    bgnpolygon();
    n3f(ny2);
    v3f(LTM_fg1); v3f(LTM_fg2); v3f(LTM_fg8); v3f(LTM_fg7);
    endpolygon();
    bgnpolygon();
    n3f(nx1);
```

```
    v3f(LTM_fg4); v3f(LTM_fg5); v3f(LTM_fg11); v3f(LTM_fg10);
    endpolygon();
    bgnpolygon();
    n3f(LTM_fg_n);
    v3f(LTM_fg3); v3f(LTM_fg4); v3f(LTM_fg10); v3f(LTM_fg9);
    endpolygon();
}


/* ***** The function for actually drawing LTM. Calls all sub. functions ******/
/*                                                                            */

draw_LTM_robot(theta)
float *theta;

{

    draw_LTM_stand();
    draw_LTM_part();
    rot(theta[1], 'x');
    draw_LTM_axis();
    rot(theta[2], 'z');
    draw_LTM_uparm();
    rot(theta[3], 'x');
    pushmatrix();
    rot(180, 'y');
    draw_LTM_axis();
    popmatrix();
    rot(-theta[4], 'z');
    draw_LTM_lowarm();
    translate(0.0, 20.0, 0.0);
    rot(theta[5], 'x');
    pushmatrix();
    rot(180, 'y');
    scale(0.8, 0.8, 0.8);
    draw_LTM_axis();
    popmatrix();
    rot(-theta[6], 'z');
    draw_LTM_rst();
    rot(-theta[7], 'x');
    draw_LTM_grip();
    translate(0.0, theta[0], 0.0);
    draw_LTM_fg();
    rot(180.0, 'x');
    translate(0.0, 2*theta[0], 0.0);
    draw_LTM_fg();
    return;
}

/******************** Parameters of the LTM manipulator **********************/
```

```
float LTM_low_lim[]= {0.0, -2.3561945, -3.1415926, -0.523599, -3.1415926,
                -0.523599, 0.0, -3.1415926};
float LTM_up_lim[]= {1.5, 0.523599,  3.1415926, 2.3561945, 3.1415926,
                2.3561945, 3.1415926, 3.1415926};

float LTM_jv_lim[]= {1.0, 0.06545, 0.06545, 0.10908, 0.10908,
                              0.12654, 0.12654, 0.12654};
float LTM_home[] = {2.0, -1.5707963, 0.0, 1.5707963, 0.0, 0.0, 1.5707963, 0.0};
float LTM_init_vpoint[] = {0.0, 0.0, 13.0, -120.0, -60.0, 0.0, 150.0};
int LTM_init_obt_index = 2;
int LTM_init_obt_sign = 1;
float LTM_theta[MAT_DIM], LTM_view_point[MAT_DIM];
float LTM_obt_vel;
int LTM_obt_sign, LTM_obt_index;

/******************** Load the parameters of LTM manioulator ******************/
init_LTM_parameters()
{
    int i;

    for (i=0; i<=7; i++)
        {
        LTM_theta[i] = LTM_home[i];
        LTM_view_point[i] = LTM_init_vpoint[i];
        }
    LTM_obt_sign = LTM_init_obt_sign;
    LTM_obt_index = LTM_init_obt_index;
    LTM_obt_vel = joint_v;
}

/***************** Transfer  LTM parameters to the main program ****************/
get_LTM_parameters()
{
    int i;

    for (i=0; i<=7; i++)
        {
        th_lim1[i] = LTM_low_lim[i];
        th_lim2[i] = LTM_up_lim[i];
        jv_lim[i] = LTM_jv_lim[i];
        theta_rd[i] = LTM_theta[i];
        robot_home[i] = LTM_home[i];
        view_point[i] = LTM_view_point[i];
        init_view_point[i] = LTM_init_vpoint[i];
        }
    obtctl.index = LTM_obt_index;
    obtctl.sign = LTM_obt_sign;
    obtctl.obt_v = LTM_obt_vel;
}

/****************** get LTM parameters from the main program *****************/
```

```
save_LTM_parameters()
{
    int i;

    for (i=0; i<=7; i++)
        {
        LTM_theta[i] = theta_rd[i];
        LTM_view_point[i] = view_point[i];
        }
    LTM_obt_index = obtctl.index;
    LTM_obt_sign = obtctl.sign;
    LTM_obt_vel = obtctl.obt_v;
}

/******************** Initiate the orbit control parameters ********************/
init_LTM_obt_struct()
{

    obtctl.sign = LTM_init_obt_sign;
    obtctl.index = LTM_init_obt_index;
    obtctl.obt_v = joint_v;
}


/******************** Calculate the Jacobian of LTM manipulator ******************/

LTM_jacobian(theta, t, jc)
float *theta, t[5][5], jc[MAT_DIM][MAT_DIM];
{
    int i, j;
    float th, r1, r2, r3;
    float s1,s2,s3,s4,s5,s6,s7;
    float c1,c2,c3,c4,c5,c6,c7;
    float u1,u2,u3,v1,v2,v3,w1,w2,w3,t1,t2,t3;

    th=theta[1];
    s1=sin(th);
    c1=cos(th);
    th=theta[2];
    s2=sin(th);
    c2=cos(th);
    th=theta[3];
    s3=sin(th);
    c3=cos(th);
    th=theta[4];
    s4=sin(th);
    c4=cos(th);
    th=theta[5];
    s5=sin(th);
    c5=cos(th);
    th=theta[6];
```

```
s6=sin(th);
c6=cos(th);
th=theta[7];
s7=sin(th);
c7=cos(th);
jc[4][1]=1;
jc[1][1]=jc[2][1]=jc[3][1]=0;
jc[5][1]=jc[6][1]=0;
jc[1][2]=jc[2][2]=jc[3][2]=0;
jc[4][2]=0;
jc[5][2]= -s1;
jc[6][2]=c1;
jc[4][3]=c2;
jc[1][3]= -A2*s2;
jc[5][3]=c1*s2;
jc[2][3]=c1*A2*c2;
jc[6][3]=s1*s2;
jc[3][3]=s1*A2*c2;
jc[4][4]= -s2*s3;
jc[1][4]= -A2*s2;
jc[5][4]=c1*c2*s3+s1*c3;
jc[2][4]=c1*A2*c2;
jc[6][4]=s1*c2*s3-c1*c3;
jc[3][4]=s1*A2*c2;
jc[4][5]=s2*c3*s4+c2*c4;
jc[1][5]=c2*A4*s4-s2*c3*A4*c4-A2*s2;
u2 = c1*c2*c3-s1*s3;
jc[5][5] = c1*s2*c4 - u2*s4;
jc[2][5]= u2*A4*c4+c1*s2*A4*s4+c1*A2*c2;
u3 = s1*c2*c3+c1*s3;
jc[6][5] = s1*s2*c4 - u3*s4;
jc[3][5] = u3*A4*c4 + s1*s2*A4*s4 + s1*A2*c2;
u1=c2*s4-s2*c3*c4;
v1=s2*s3;
w1=jc[4][5];
t1=jc[1][5];
u2= u2*c4+c1*s2*s4;
v2= -(c1*c2*s3+s1*c3);
w2=jc[5][5];
t2=jc[2][5];
u3= u3*c4+s1*s2*s4;
v3=c1*c3-s1*c2*s3;
w3=jc[6][5];
t3=jc[3][5];
jc[4][6]=u1*s5-v1*c5;
jc[1][6]=t1;
jc[5][6]=u2*s5-v2*c5;
jc[2][6]=t2;
jc[6][6]=u3*s5-v3*c5;
jc[3][6]=t3;
r1 = u1*c5+v1*s5;
```

```
    jc[4][7]= r1*s6-w1*c6;
    jc[1][7]=t1;
    r2 = u2*c5+v2*s5;
    jc[5][7]= r2*s6-w2*c6;
    jc[2][7]=t2;
    r3 = u3*c5+v3*s5;
    jc[6][7]= r3*s6-w3*c6;
    jc[3][7]=t3;
    t[1][1]=(r1*c6+w1*s6)*c7+(u1*s5-v1*c5)*s7;
    t[1][2]=(u1*s5-v1*c5)*c7-(r1*c6+w1*s6)*s7;
    t[1][3]=jc[4][7];
    t[1][4]=(r1*s6-w1*c6)*D7+t1;
    t[2][1]=(r2*c6+w2*s6)*c7+(u2*s5-v2*c5)*s7;
    t[2][2]=(u2*s5-v2*c5)*c7-(r2*c6+w2*s6)*s7;
    t[2][3]=jc[5][7];
    t[2][4]=(r2*s6-w2*c6)*D7+t2;
    t[3][1]=(r3*c6+w3*s6)*c7+(u3*s5-v3*c5)*s7;
    t[3][2]=(u3*s5-v3*c5)*c7-(r3*c6+w3*s6)*s7;
    t[3][3]=jc[6][7];
    t[3][4]=(r3*s6-w3*c6)*D7+t3;
    t[4][1]=0;
    t[4][2]=0;
    t[4][3]=0;
    t[4][4]=1;
    for (i=1; i<=7; i++)
        {
        r1=t[1][4]-jc[1][i];
        r2=t[2][4]-jc[2][i];
        r3=t[3][4]-jc[3][i];
        jc[1][i]=jc[5][i]*r3-jc[6][i]*r2;
        jc[2][i]=jc[6][i]*r1-jc[4][i]*r3;
        jc[3][i]=jc[4][i]*r2-jc[5][i]*r1;
        }
    return;
}

/******** Calculate the homogeneous transformation of LTM manipulator *********/

LTM_transf(theta, t)
float *theta, t[5][5];
{
    float th, r1, r2, r3;
    float s1,s2,s3,s4,s5,s6,s7;
    float c1,c2,c3,c4,c5,c6,c7;
    float u1,u2,u3,v1,v2,v3,w1,w2,w3,t1,t2,t3;

    th=theta[1];
    s1=sin(th);
    c1=cos(th);
    th=theta[2];
    s2=sin(th);
```

```
    c2=cos(th);
    th=theta[3];
    s3=sin(th);
    c3=cos(th);
    th=theta[4];
    s4=sin(th);
    c4=cos(th);
    th=theta[5];
    s5=sin(th);
    c5=cos(th);
    th=theta[6];
    s6=sin(th);
    c6=cos(th);
    th=theta[7];
    s7=sin(th);
    c7=cos(th);
    u1=c2*s4-s2*c3*c4;
    v1=s2*s3;
    w1=s2*c3*s4+c2*c4;
    t1= c2*A4*s4-s2*c3*A4*c4-A2*s2;
    r2 = c1*c2*c3-s1*s3;
    u2= r2*c4+c1*s2*s4;
    v2= -(c1*c2*s3+s1*c3);
    w2=c1*s2*c4-r2*s4;
    t2= r2*A4*c4+c1*s2*A4*s4+c1*A2*c2;
    r3= s1*c2*c3+c1*s3;
    u3= r3*c4+s1*s2*s4;
    v3=c1*c3-s1*c2*s3;
    w3=s1*s2*c4-r3*s4;
    t3= r3*A4*c4+s1*s2*A4*s4+s1*A2*c2;
    r1 = u1*c5+v1*s5;
    r2 = u2*c5+v2*s5;
    r3 = u3*c5+v3*s5;
    t[1][1]=(r1*c6+w1*s6)*c7+(u1*s5-v1*c5)*s7;
    t[1][2]=(u1*s5-v1*c5)*c7-(r1*c6+w1*s6)*s7;
    t[1][3]= r1*s6-w1*c6;
    t[1][4]=(r1*s6-w1*c6)*D7+t1;
    t[2][1]=(r2*c6+w2*s6)*c7+(u2*s5-v2*c5)*s7;
    t[2][2]=(u2*s5-v2*c5)*c7-(r2*c6+w2*s6)*s7;
    t[2][3]= r2*s6-w2*c6;
    t[2][4]=(r2*s6-w2*c6)*D7+t2;
    t[3][1]=(r3*c6+w3*s6)*c7+(u3*s5-v3*c5)*s7;
    t[3][2]=(u3*s5-v3*c5)*c7-(r3*c6+w3*s6)*s7;
    t[3][3]= r3*s6-w3*c6;
    t[3][4]=(r3*s6-w3*c6)*D7+t3;
    t[4][1]=0;
    t[4][2]=0;
    t[4][3]=0;
    t[4][4]=1;
    return;
}
```

MAKEFILE FOR THE PROGRAM

```
libs =-lfm_s -lgl_s -lm -lc_s

rbt1 = k2107

rbt2 = LTM

mk_robot: makefile                      \
                draw_$(rbt1).o          \
                draw_$(rbt2).o          \
                init_graph.o            \
                tools.o                 \
                goalp.o                 \
                main_pro.o
        cc -O1 -o robot                 \
                draw_$(rbt1).o          \
                draw_$(rbt2).o          \
                init_graph.o            \
                tools.o                 \
                goalp.o                 \
                main_pro.o $(libs)

draw_$(rbt1).o: draw_$(rbt1).c
        cc -O1 -c draw_$(rbt1).c -o draw_$(rbt1).o

draw_$(rbt2).o: draw_$(rbt2).c
        cc -O1 -c draw_$(rbt2).c -o draw_$(rbt2).o

tools.o: tools.c
        cc -O1 -c tools.c -o tools.o

goalp.o: goalp.c
        cc -O1 -c goalp.c -o goalp.o

init_graph.o: init_graph.c
        cc -O1 -c init_graph.c -o init_graph.o

main_pro.o: main_pro.c
        cc -O1 -c main_pro.c -o main_pro.o
```

VITA

T. F. Chan was born in China 1957. After graduating from high school in Beijing China in 1975, he worked as a farmer in the suburb of Beijing for four years. He entered Beijing Polytechnic University in 1979, and received his undergraduate degree in Engineering Science in 1984. In the same year he joined the "Ming Tai Hong" company in Hong Kong, and worked as a marketing engineer and computer programmer. T. F. Chan entered the graduate school in Mechanical Engineering at the University of Tennessee in August 1989. He will continue his research work for Ph.D. of Mechanical & Aerospace Engineering at the University of Tennessee.