IDENTIFICATION OF PARAMETERS AND CONTROL OF ROBOTIC MANIPULATORS USING ARTIFICIAL NEURAL NETWORKS

RAJNISH AGGARWAL

IDENTIFICATION OF PARAMETERS AND CONTROL OF ROBOTIC MANIPULATORS USING ARTIFICIAL NEURAL NETWORKS

By

© Rajnish Aggarwal (B.E.)

A Thesis submitted to the School of Graduate Studies in partial fulfiiment of the requirements for the degree of Master of Engineering

Faculty of Engineering and Applied Science

Memorial University of Newfoundland

August 1995

St. John's Newfoundland Canada

ISBN 0-612-13878-X

ABSTRACT

In the present work the estimation of dynamic parameters and the trajectory control of a two link planar manipulator is carried out. The derivation of system equations involve the kinematic parameters such as joint positions, velocities and accelerations. For both the dynamic parameter estimation and the trajectory control an Artificial Neural Networks method called, Linear Programming (LP)-Neuro Method, is used. In this algorithm, the weights are obtained by a combination of linear programming having a sparse coefficient matrix and a single variable non-linear optimization routine.

The training set values required for parameter estimation are generated by the Iterative Newton-Euler Dynamics Algorithm. The Artificial Neural Network is trained to predict the dynamic parameters. The values of the forces and torques are recomputed based on the estimated dynamic parameters. This method is useful for the on line parameter estimation of the manipulators having odd mass distribution, which is the case in actual practice.

For the control problem non-linear optimization method is used to evaluate the gain parameters required for the manipulator to follow the desired trajectory. Then the Linear Programming (LP)-Neuro Method is used to obtain the weight matrix which relates the input (joint positions and velocities) and the output (gain parameters). This weight matrix, for each point along the trajectory, can be used on-line to evaluate the gain parameters thereby eliminating the time consuming

calculations at each point along the trajectory. Finally, the effect of the variation of the maximum tangential velocity and general control law are studied.

ACKNOWLEDGEMENTS

With sincere appreciation and gratitude I would like to thank my supervisor, Dr. Anand M. Sharan, for his invaluable direction, support, patient guidance and constant encouragement during the course of this work. Working with him was a valuable experience both personally and professionally.

I would also like to extend my thanks to Dr. John Malpas, Dean of School of Graduate Studies for supporting me financially through this period. I am also thankful to Dr. R. Seshadri, Dean of Engineering and Dr. J. J. Sharp, Associate Dean of Engineering for their cooperation and necessary help during this graduate program.

Finally, I would like to thank all my friends for their constant encouragement and moral support.

This work is dedicated to my parents.

TABLE OF CONTENTS

	PAGE
ABSTRACT	ii
ACKNOWLEDGEMENTS	iv
TABLE OF CONTENTS	vi
LIST OF FIGURES	ix
LIST OF TABLES	xiii
NOMENCLATURE	xv

CHAPTER 1

INTRODUCTION AND LITERATURE SURVEY

1.1	INTRODUCTION	1
1.2	LITERATURE SURVEY	4
	1.2.1 ARTIFICIAL NEURAL NETWORKS	4
	1.2.2 ESTIMATION OF PARAMETERS	5
	1.2.3 ARTIFICIAL NEURAL NETWORK CONTROL IN ROBOTICS	6
1.3	OBJECTIVES OF THE THESIS	8

CHAPTER 2

ESTIMATION OF DYNAMIC PARAMETERS

2.1	INTRODUCTION	10

2.2 ROBOT KINEMATICS 11

2.2.1 INVERSE KINEMATICS 11

2.3 ROBOT DYNAMICS 14

2.3.1 GENERATION OF INERTIA [I] - MATRIX 14

2.4 PARAMETER IDENTIFICATION 19

2.4.1 THE LINEAR PROGRAMMING - NEURO (LPN) METHOD 19

2.4.2 ARTIFICIAL NEURAL NETWORK (ANN) IN PARAMETER ESTIMATION 23

2.5 RESULTS AND DISCUSSIONS 26

2.6 CONCLUSIONS 29

CHAPTER 3

CONTROL OF ROBOTIC MANIPULATORS

3.1 INTRODUCTION 40

3.2 DYNAMIC EQUATIONS OF A PLANAR TWO LINK MANIPULATOR 41

3.3 TRAJECTORY CONTROL 43

3.3.1 INDEPENDENT PD CONTROL 43

3.3.2 EVALUATION OF GAIN PARAMETERS 44

3.3.3 HOOKE'S AND JEEVES NON-LINEAR OPTIMIZATION METHOD 46

3.3.4 NUMERICAL INTEGRATION TECHNIQUE 49

3.3.4.1 RUNGE-KUTTA METHOD FOR A SECOND ORDER DIFFERENTIAL EQUATION 49

3.3.4.2 RUNGE-KUTTA METHOD FOR THE CONTROL PROBLEM 52

3.4 TRAJECTORY GENERATION 54

3.5 OPTIMAL CONTROL METHOD (NON - LINEAR OPTIMIZATION) 62

3.6 ARTIFICIAL NEURAL NETWORKS IN TRAJECTORY CONTROL 64

3.7 THE EFFECT OF VARIATION OF MAXIMUM TANGENTIAL VELOCITY 77

3.8 THE GENERAL CONTROL LAW 87

3.9 CONCLUSIONS 92

CHAPTER 4

CONCLUSIONS AND RECOMMENDATIONS

4.1 DISCUSSION AND CONCLUSIONS 93

4.2 RECOMMENDATIONS FOR FUTURE WORK 95

REFERENCES 96

APPENDIX A	EXPRESSIONS FOR INERTIA TENSOR	101
APPENDIX B	FORMULA FOR ROTATION ABOUT THE PRINCIPAL AXIS BY θ	104
APPENDIX C	EXPRESSIONS FOR THE DERIVATION OF THE ITERATIVE NEWTON-EULER DYNAMIC ALGORITHM	105
APPENDIX D	PROGRAM LISTINGS	110

LIST OF FIGURES

NO	DESCRIPTION	PAGE
1.1	THE UNIMATION PUMA 500 IN OPERATION	2
1.2	THE PUMA-560 MANIPULATOR	3
2.1	PLANAR TWO-LINK MANIPULATOR	12
2.2	A LINK WITH REGULAR MASS DISTRIBUTION	16
2.3	PLANAR TWO-LINK MANIPULATOR WITH IRREGULAR MASS DISTRIBUTION	18
2.4	DIAGRAMMATIC REPRESENTATION OF THE NETWORK (LP-NEURO METHOD)	20
2.5	DESIRED TRAJECTORY	30
2.6	VARIATION OF TANGENTIAL VELOCITY ALONG THE TRAJECTORY	31
2.7	FLOW CHART : DYNAMIC PARAMETER ESTIMATION	32
2.8	FLOW CHART : COMPARISON OF FORCES AND TORQUES CORRESPONDING TO ESTIMATED PARAMETERS	33
2.9	ERROR VALUES IN F_x FOR LINKS 1 AND 2	34
2.10	ERROR VALUES IN F_y FOR LINKS 1 AND 2	35
2.11	ERROR VALUES IN τ_x FOR LINKS 1 AND 2	36
2.12	ERROR VALUES IN τ_y FOR LINKS 1 AND 2	37
2.13	ERROR VALUES IN τ_z FOR LINKS 1 AND 2	38

2.14	PERCENTAGE ERROR IN τ_2 FOR LINKS 1 AND 2	39
3.1	SPECIFICATIONS OF THE DESIRED AND ACTUAL TRAJECTORY	45
3.2	DESIRED TRAJECTORY	55
3.3	DESIRED TANGENTIAL VELOCITY PROFILE	58
3.4	DESIRED CARTESIAN TRAJECTORY ACCORDING TO TABLE 3.3	61
3.5	FLOW CHART : OPTIMAL CONTROL METHOD (NON-LINEAR OPTIMIZATION)	63
3.6	FLOW CHART : LINEAR PROGRAMMING (LP -NEURO) METHOD (ANN METHOD)	66
3.7	VARIATION OF θ_1 ALONG THE TRAJECTORY	68
3.8	VARIATION OF θ_2 ALONG THE TRAJECTORY	69
3.9	VARIATION OF $\dot{\theta}_1$ ALONG THE TRAJECTORY	70
3.10	COMPARISON OF K_{p1} VALUES OBTAINED USING OPTIMAL CONTROL AND ANN METHOD	72
3.11	COMPARISON OF K_{p2} VALUES OBTAINED USING OPTIMAL CONTROL AND ANN METHOD	73
3.12	COMPARISON OF K_{v1} VALUES OBTAINED USING OPTIMAL CONTROL AND ANN METHOD	74
3.13	COMPARISON OF K_{v2} VALUES OBTAINED USING OPTIMAL CONTROL AND ANN METHOD	75

3.14	TRAJECTORY CONTROL OBTAINED USING OPTIMAL CONTROL AND ANN METHOD	76
3.15	VELOCITY VARIATION ACCORDING TO TABLE 3.5	79
3.16	DESIRED TRAJECTORY AND THE TRAJECTORY OBTAINED USING ANN METHOD WHEN THE MAXIMUM VELOCITY IS 0.15 m/s	80
3.17	DESIRED TRAJECTORY AND THE TRAJECTORY OBTAINED USING ANN METHOD WHEN THE MAXIMUM VELOCITY IS 0.15 m/s (SMOOTH CORNER)	81
3.18	DESIRED TRAJECTORY AND THE TRAJECTORY OBTAINED USING ANN METHOD WHEN THE MAXIMUM VELOCITY IS 0.1 m/s	82
3.19	DESIRED TRAJECTORY AND THE TRAJECTORY OBTAINED USING ANN METHOD WHEN THE MAXIMUM VELOCITY IS 0.05 m/s	83
3.20	DESIRED TRAJECTORY AND THE TRAJECTORY OBTAINED USING ANN METHOD WHEN THE MAXIMUM VELOCITY IS 0.001 m/s	84
3.21	DESIRED TRAJECTORY AND THE TRAJECTORY OBTAINED USING ANN METHOD WHEN THE MAXIMUM VELOCITY IS 2 inch/min	

	(0.0008466 m/s)	85
3.22	DESIRED TRAJECTORY AND THE TRAJECTORY OBTAINED USING ANN METHOD WHEN THE MAXIMUM VELOCITY IS 2 inch/min (0.0008466 m/s : SMOOTH CORNER)	86
3.23	VARIATION OF THE NORMALIZED ERROR BETWEEN THE OBJECTIVE FUNCTIONS	91
A.1	RIGID BODY WITH AN ATTACHED FRAME	102
C.1	FREE BODY DIAGRAM OF A LINK	108

LIST OF TABLES

NO	DESCRIPTION	PAGE
2.1	THE ITERATIVE NEWTON-EULER DYNAMICS ALGORITHM	15
2.2	INPUT VECTOR, OUTPUT VECTOR AND WEIGHT MATRIX USED FOR TRAINING	25
2.3	VARIOUS PARAMETERS USED FOR A TWO-LINK MANIPULATOR	27
2.4	COMPARISON OF ESTIMATED DYNAMIC PARAMETERS	28
3.1	RUNGE-KUTTA METHOD IN VARIABLE FORM	51
3.2	VARIOUS PARAMETERS USED FOR A TWO-LINK MANIPULATOR	57
3.3	VARIATION OF JOINT DISPLACEMENT VALUES	60
3.4	COMPARISON OF GAIN VALUES OBTAINED USING OPTIMAL CONTROL (NON-LINEAR OPTIMIZATION) AND ANN METHOD	71
3.5	VARIOUS MAXIMUM TANGENTIAL VELOCITY VALUES	78
3.6	GAIN VALUES OBTAINED USING OPTIMAL CONTROL METHOD WITH 4 DESIGN VARIABLES	88
3.7	GAIN VALUES OBTAINED USING OPTIMAL CONTROL METHOD FOR THE GENERAL CONTROL LAW	89

3.8 COMPARISON OF OBJECTIVE FUNCTION VALUES OBTAINED USING 4 DESIGN VARIABLE AND GENERAL CONTROL LAW 90

NOMENCLATURE

{ }	vector or column matrix
[ ]	matrix
f(.)	activation function
[W], [V]	weight matrices
{I}, {D}, {O}	input, desired and output vectors respectively
l_1, l_2	link lengths
m_1, m_2	link masses
r	radius of the circular arc
a, b, c	length, height and width of link respectively
${}^{C}[I]$	inertia matrix in a frame located at the centre of mass of the body
${}^{A}[I]$	inertia matrix in an arbitrarily translated frame
${}^{B}[I]$	final inertia matrix involving mass moment of inertia and mass product of inertia terms
${}^{C}\{Pc\}$	position vector of the C.G. of the link in a frame located at the centre of mass of the body
${}^{A}\{Pc\}$	position vector of the C.G. of the link in an arbitrarily translated frame
${}^{B}\{Pc\}$	final position vector of the C.G. of the link
$[C_1]$	orthogonal transformation matrix, X-Y-Z fixed angle matrix
X, Y, $\dot{X}$, $\dot{Y}$, $\ddot{X}$, $\ddot{Y}$	cartesian displacements, velocities and accelerations

θ_i, $\dot{\theta}_i$, $\ddot{\theta}_i$	displacement, angular velocity and angular acceleration of the *i* th joint respectively
α, β, γ	rotation about Z, Y and X axes respectively
F, τ	force and torque
θ_d, $\dot{\theta}_d$	desired joint displacement and joint velocity
θ_a, $\dot{\theta}_a$	actual joint displacement and joint velocity
e	error in joint position
$\dot{e}$	error in joint velocity
k_{p1}, k_{p2}, k_{v1}, k_{v2}	gain parameters
$[K_p]$	proportional gain matrix
$[K_v]$	velocity gain matrix
U	objective function
V_r, V_t	radial and tangential velocity
A_r, A_t	radial and tangential acceleration
$[M(\theta)]$	n x n mass matrix of the manipulator
$\{V(\theta,\dot{\theta})\}$	n x 1 vector containing centrifugal and coriolis terms
$\{G(\theta)\}$	n x 1 vector of gravity terms
g	acceleration due to gravity
h, ΔT	time increment for Runge-Kutta method
N_1, N_2	exponents of the joint displacement error and the joint velocity error in the general control law

CHAPTER 1

INTRODUCTION AND LITERATURE SURVEY

1.1 INTRODUCTION

In the recent years the robots have found an ever increasing use in the field of industrial automation. Industrial robots are already assuming many hazardous, unpleasant or monotonous tasks, while simultaneously improving the productivity of factories in the industrialized world. Robots potentially find applications in the hazardous or inaccessible environments, such as, nuclear reactors, furnace operations, mines, deep sea and outer space. Fig. 1.1 shows the Unimation PUMA 500 (Programmable Universal Manipulator for Assembly) in operation and Fig. 1.2 shows the rigid body model of the 6 Degree-of-Freedom (DOF) PUMA-560 manipulator. The study of mechanics and control of manipulators is not a new science, but merely a collection of topics taken from *classical* fields. Control theory provides tools for designing and evaluating algorithms to realize desired motions or force application.

Although computers outperform both biological and artificial neural systems for tasks based on precise and fast arithmetic operations, artificial neural systems represent the promising new generation of information processing networks. Advances have been made in applying such systems for problems found

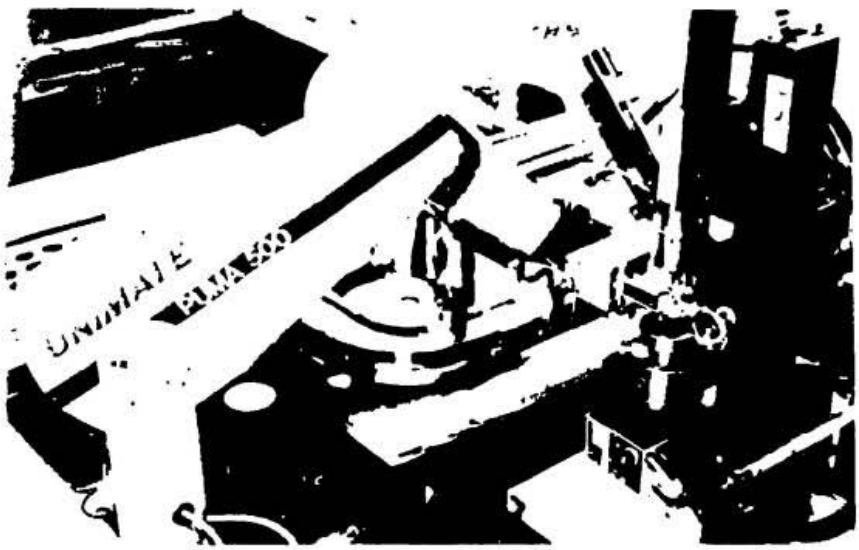

Fig. 1.1 THE UNIMATION PUMA 500 IN OPERATION

[Spong and Vidyasagar, 1989]

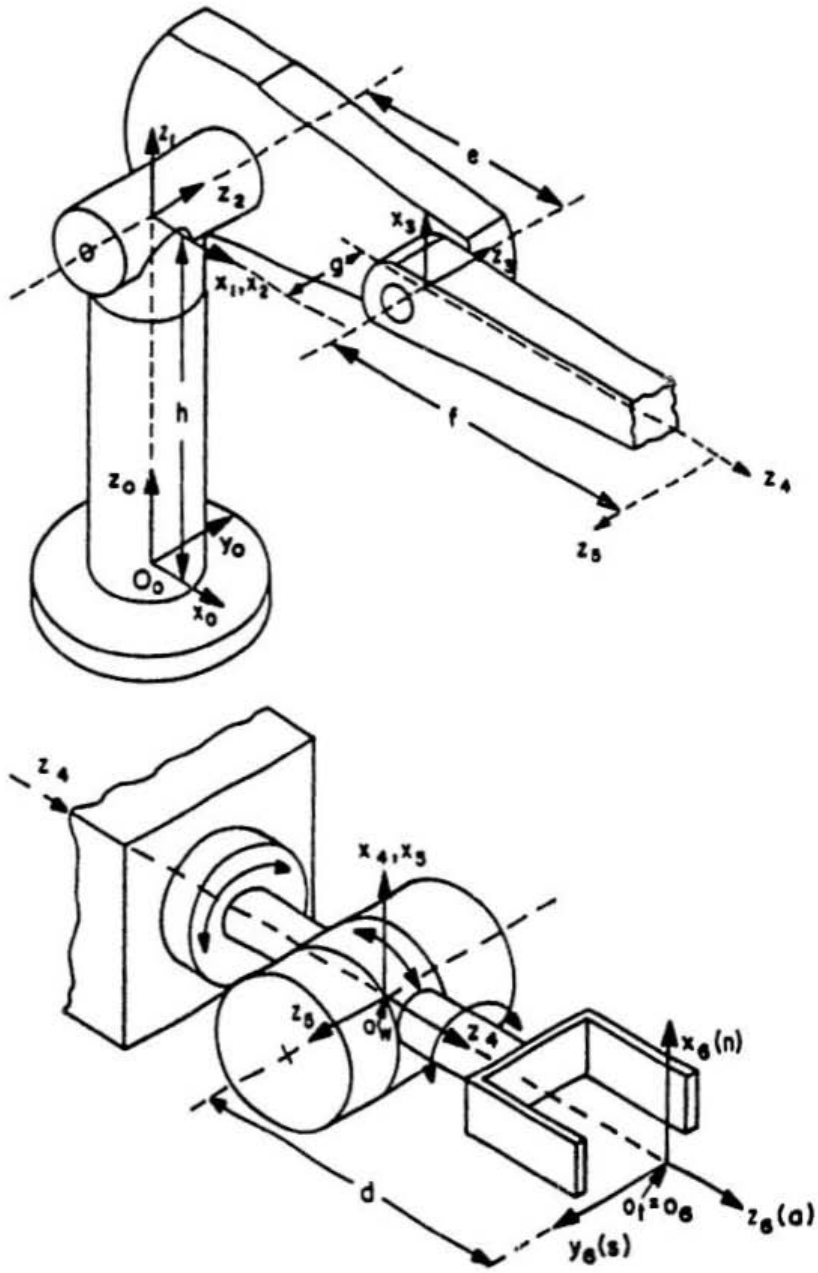

Fig. 1.2 THE PUMA-560 MANIPULATOR

intractable or difficult for traditional computation. Since, the area of robotics requires numerous and complicated on-line calculations so, Artificial Neural Networks (ANN) find an immediate application in this area.

1.2 LITERATURE SURVEY

1.2.1 ARTIFICIAL NEURAL NETWORKS

The use of Artificial Neural Networks has increased tremendously in recent years because of the availability of faster and parallel processors and the basic learning algorithms (Grossberg, 1982; Hopfield, 1982; Rumelhart and McClelland, 1986; Kohonen, 1988). Neural network can learn mapping between the input and output space and synthesize an associative memory that retrieves the appropriate output when presented with an input, and has the ability to generalize with new inputs. ANNs are being used to accomplish complex functions such as generalization, error correction, information reconstruction, pattern analysis and learning. Because of their massively parallel nature, neural networks can perform computations at very high speeds (Fukuda and Shibata, 1992).

Neural networks have also been used to successfully solve complex problems like the Travelling Salesman problem. It has been observed that neural networks have often been opportunistic, i.e. the network model is customized to serve the needs of the task at hand (Kulkarni, 1991). They represent a new approach that is robust and fault-tolerant.

Neural networks require basic algorithms for accomplishing the learning

task. Several algorithms are functional at present. One such algorithm which is widely used is backpropagation (BP) algorithm. In backpropagation algorithm, during the learning phase, the observed outputs are compared with the desired outputs, and the weights are optimized to minimize the error function. In competitive learning, the weights are updated with each new input (Rumelhart and McClelland, 1986). Barmann and Biegler-Kong (1992) discuss efficient learning algorithms for neural networks.

Neural networks can perform functional approximations that are beyond the scope of optimal linear techniques. Gulati et al., (1990) have introduced neural formalism to efficiently learn non-linear mapping using a mathematical construct called terminal attractors.

Neural networks have been found useful in the field of robotics in the recent times. Forward and inverse displacement analyses of robotic manipulators have been done by Nyugen et al. (1990) and Gulati et al., (1990). Neural networks seem to be a promising approach to solve non-linear control problems as well (Tabary and Salaun, 1992). Some other interesting applications in the control of robotic manipulators can be seen in Fukuda et al., 1991; and Akio et al., 1992.

1.2.2 ESTIMATION OF PARAMETERS

The kinematic parameter identification techniques were reviewed by Hollerbach (1989). Many researchers [Maveda et al. 1984; Khosla and Kandate 1985; Kawasaki and Nishimura 1986; An et. al. 1986; Atkenson et. al. 1986] have

worked to estimate the dynamic parameters. The method of calculation of the base inertial parameters of closed-loop robots through symbolic computations is given by Khalil and Bennis (1995). A method of estimating the mass properties of a manipulator by measuring the reaction moments at the base has been discussed by Dubowsky and Cheah (1989). These estimates are not accurate enough for dynamic control. A method of generating the variable forgetting factor for on-line parameter estimation, has been studied by Weiping Li and Slotine (1988). These estimates are applicable to linearly parameterized models only. One of the ways of ascertaining these (the dynamic parameters) would be by using Artificial Neural Networks (ANN). The application of Neural Networks for the identification and control of low order nonlinear dynamical systems is studied by Narendra and Parthasarathy (1990). In the present investigation, an ANN method is used to estimate the dynamic parameters and is discussed in detail in Chapter 2 of this thesis. This would be using a faster algorithm developed by Balasubramanian and Sharan (1993).

1.2.3 ARTIFICIAL NEURAL NETWORK CONTROL IN ROBOTICS

There has been recent trend within the robotics control literature to apply neural networks for the control of robotic systems. In many applications reported in the literature (Gu and Chan, 1989; Fukuda and Shibata, 1990; Helferty and Biswas, 1990; Jamshidi et al., 1990; Karakasoglu and Sundareshan, 1990; Yamamura et al., 1990) the process of neural network learning is conducted on-line (i.e. the dynamics of the neural network is embedded in the closed-loop with

the dynamics of the robotic system), yet there appears to be lack of studies focussing on the dynamic behavior of the neural network during learning and/or control when the neural network is used in such context.

Kawato (1990) used feedback error learning to compute the feedforward torques required for a manipulator to follow a path. The neural network implemented in this method uses the desired joint positions, velocities and accelerations as inputs and adjusts the network weights using the feedback torque as the error signal to a backpropagation parameter optimizing algorithm. Yuh (1992) also used a neural network for manipulator control. He used a "critic" equation, which is a function of the manipulator output error, to train the network to directly compute the manipulator input torques.

Asada (1990) used a multilayered feedforward network to learn a non-linear mapping for compliance control. From the measured forces and torques in an assembly task he used the network to compute the required velocities, which would allow the assembly task to be completed. Ozaki et. al. (1991) presents the method of trajectory control of robotic manipulators using a nonlinear compensator. The adaptive capability of neural network controller to compensate unstructured uncertainties is discussed. A method of using two neural networks for the control of a robotic arm is suggested by Shoham et. al. (1992). The neural network weights which are usually chosen arbitrarily have been determined in a systematic way based on a geometric interpretation of the neuron function. Balasubramanian (1993) used ANN over a limited range of trajectory control.

1.3 OBJECTIVES OF THE THESIS

We have seen in the last few sections that the neural networks are quite versatile tools to solve problems in a wide variety of areas. With this in mind, it was thought to apply this tool to solve problems in the areas of parameter estimation and control of robotic manipulators. Also, based on the review of the literature, the following are the objectives of this thesis:

1. The identification of the dynamic parameters of robotic manipulators.
2. The optimal control of the robotic manipulators using the Hookes and Jeeves Direct Search non-linear optimization method over a practical range of a trajectory.
3. Use of ANN method for the trajectory control over a practical range.
4. Study the effect of the variation of the maximum tangential velocity over a trajectory on the controllability (error) of a manipulator.
5. Study the general control law in the trajectory control.

In Chapter 2 the inverse kinematic relationship is established between the Cartesian and joint displacement, velocity and acceleration values on off-line basis which reduces on-line computation time. The ANN method, linear programming - neuro (LPN) method, is discussed in this chapter. This method is a non-linear method where most, but not all, of the computations are done using linear programming. This method is fast converging as compared to many other methods. In this chapter the generation of Inertia Matrix [I] for an irregular shaped

link is discussed. Finally, in this chapter the results of estimation of dynamic parameters are reported.

In Chapter 3 the various control issues for a two link planar manipulator are studied. Here the optimal control method (non-linear optimization) and linear programming - neuro (LPN) method is used in the on-line robotic control. The effect of variation of the maximum velocity over a trajectory and general control law are also included in this chapter.

Finally, conclusions and recommendations for future work are presented in Chapter 4.

CHAPTER 2

ESTIMATION OF DYNAMIC PARAMETERS

2.1 INTRODUCTION

A manipulator consisting of concatenated rigid links has highly non-linear dynamics. The joint torques are related to the joint angles, velocities and accelerations by a set of non-linear equations involving various constant parameters such as link lengths, masses, inertias and so on. These parameters are separated into two groups: the kinematic parameters, and the dynamic parameters. The kinematic parameters do not include any physical quantities such as mass or inertia, while the dynamic parameters include them. To control the robotic mechanisms it is required to know both groups of parameters.

In actual practice the mass distribution along any link is quite complex due to the various hydraulic drives or other accessories attached to these links. The theoretical values (dynamic parameters) are not accurately known as a result of this. Therefore, for precision control one has to incorporate these quantities in the model.

In this chapter, at first, the unknown dynamic parameters are identified using ANN method. The network is trained by considering the forces and torques acting on links as input parameters and position of the centre of gravity of the links

as well as the various moments of inertias as the output parameters. Then, the training the unknown output parameters of the links are obtained using the trained ANN.

2.2 ROBOT KINEMATICS

2.2.1 INVERSE KINEMATICS

Fig. 2.1 shows a planar two-link manipulator having revolute joints. In this figure, (X_o, Y_o) represents the global coordinate system and (X_i, Y_i) represent the local coordinate frame of link 'i' (i = 1,2). The joint variables are given by θ_i.

In this figure the position of the end effector is represented by the coordinates X and Y. The inverse solution at this point can be carried out by the use of following equations, in sequence:

Defining $c_1 = \cos\theta_1$; $c_{12} = \cos(\theta_1 + \theta_2)$; $s_1 = \sin\theta_1$; $s_{12} = \sin(\theta_1 + \theta_2)$; $c_2 = \cos\theta_2$; $s_2 = \sin\theta_2$; we start with

$$c_2 = \frac{X^2 + Y^2 - l_1^2 - l_2^2}{2 l_1 l_2}, \text{ and} \qquad (2.1)$$

$$s_2 = \pm\sqrt{1 - c_2^2} \quad . \qquad (2.2)$$

From this, we get θ_2 as

$$\theta_2 = \text{atan2}(s_2, c_2) \qquad (2.3)$$

If we have two intermediate variables k_1 and k_2 as

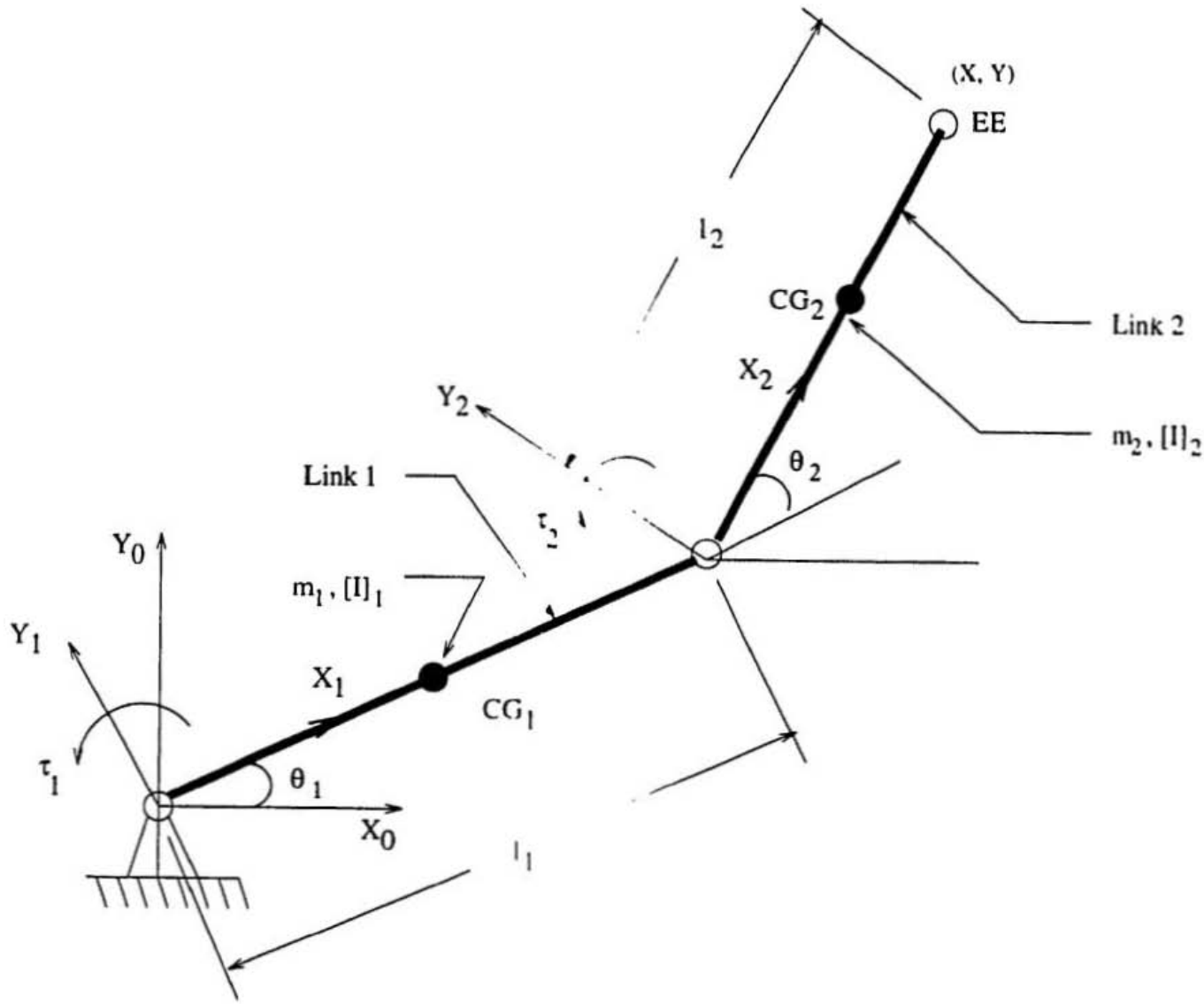

Fig. 2.1 PLANAR TWO-LINK MANIPULATOR

$$k_1 = l_1 + l_2 c_2 \,, \tag{2.4}$$

$$k_2 = l_2 s_2 \,, \tag{2.5}$$

then we can obtain

$$\theta_1 = \text{Atan2}(Y, X) - \text{Atan2}(k_2, k_1) \tag{2.6}$$

The above inverse kinematic equations can be written in vector form as

$$\begin{Bmatrix} X \\ Y \end{Bmatrix} = \begin{Bmatrix} l_1 c_1 + l_2 c_{12} \\ l_1 s_1 + l_2 s_{12} \end{Bmatrix} \tag{2.7}$$

where l_1, l_2 are the lengths of link 1 and 2 respectively.

Differentiating Eq. (2.7) with respect to time, we get

$$\begin{Bmatrix} \dot{X} \\ \dot{Y} \end{Bmatrix} = \begin{bmatrix} -l_1 s_1 - l_2 s_{12} & -l_2 s_{12} \\ l_1 c_1 + l_2 c_{12} & l_2 c_{12} \end{bmatrix} \begin{Bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{Bmatrix} \tag{2.8}$$

Finally, the equations for the acceleration of the end-effector can be expressed in terms of both the first and second derivatives of θ_1 and θ_2 as

$$\begin{aligned} \begin{Bmatrix} \ddot{X} \\ \ddot{Y} \end{Bmatrix} &= \begin{bmatrix} -l_1 s_1 - l_2 s_{12} & -l_2 s_{12} \\ l_1 c_1 + l_2 c_{12} & l_2 c_{12} \end{bmatrix} \begin{Bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{Bmatrix} \\ &+ \begin{bmatrix} -l_1 c_1 \dot{\theta}_1 - l_2 c_{12}(\dot{\theta}_1 + \dot{\theta}_2) & -l_2 c_{12}(\dot{\theta}_1 + \dot{\theta}_2) \\ -l_1 s_1 \dot{\theta}_1 - l_2 s_{12}(\dot{\theta}_1 + \dot{\theta}_2) & -l_2 s_{12}(\dot{\theta}_1 + \dot{\theta}_2) \end{bmatrix} \begin{Bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{Bmatrix} \end{aligned} \tag{2.9}$$

2.3 ROBOT DYNAMICS

2.3.1 GENERATION OF INERTIA [I] - MATRIX

As discussed in the introduction, the inertia matrices of the robotic manipulators are not known in most practical situations. In actual control of robotic manipulators one has to compute forces and torques which can be calculated using the Iterative Newton-Euler Dynamics Algorithm shown in Table 2.1. As one can see in this table, one needs to precisely know the numerical values of the matrices $[\,I\,]_i$ shown in Steps 6-8 in order to calculate the joint forces and torques.

Fig. 2.2 shows a link of regular geometry whose inertia matrices, in the frame ${}^{C}\{\ \}$ are given by [Rothbart, 1973]

$$ {}^{C}[I] = \begin{bmatrix} \frac{m}{12}(b^2+c^2) & 0 & 0 \\ 0 & \frac{m}{12}(c^2+a^2) & 0 \\ 0 & 0 & \frac{m}{12}(a^2+b^2) \end{bmatrix} \qquad (2.10) $$

This matrix in frame ${}^{A}\{\ \}$ will be given by [Craig, 1994]

$$ {}^{A}[I] = {}^{C}[I] + m\left[{}^{A}\{P_c\}^T\ {}^{A}\{P_c\}[I^*] - {}^{A}\{P_c\}\ {}^{A}\{P_c\}^T \right] \qquad (2.11) $$

where ${}^{A}\{Pc\} = \{X_c, Y_c, Z_c\}^T$ locates the centre of mass relative to ${}^{A}\{\ \}$ and $[I^*]$ is the 3 x 3 identity matrix.

In this figure we also have a frame ${}^{B}\{\ \}$ which is obtained by rotating the frame ${}^{A}\{\ \}$ about X_A by an angle γ, then about Y_A by an angle β and finally about Z_A by an angle α. Hence, the transformation matrix, $[C_1]$, relating the frames ${}^{A}\{\ \}$

TABLE 2.1 THE ITERATIVE NEWTON-EULER DYNAMICS ALGORITHM

[CRAIG, 1994]

FORWARD RECURSION

Step 1: $\{\omega\}_i = [R]_i^T \{\omega\}_{i-1} + \{z\}\,\dot{\Theta}_i$

Step 2: $\{\alpha\}_i = [R]_i^T \{\alpha\}_{i-1} + \{z\}\,\ddot{\Theta}_i + [R]_i^T \{\omega\}_i \times \{z\}\,\dot{\Theta}_i$

Step 3: $\{a\}_i = [R]_i^T \left(\{\alpha\}_{i-1} + \{\alpha\}_{i-1} \times \{p\}_{i-1} + \{\omega\}_{i-1} \times \{p\}_{i-1} \right)$
$+ \{z\}\,\ddot{\Theta}_i + 2 \times [R]_i^T \{\omega\}_{i_1} \times \{z\}\,\dot{\Theta}_i$

Step 4: $\{a\}_{ci} = \{a\}_i + \{\alpha\}_i \times \{s\}_i + \{\omega\}_i \times \{\omega\}_i \times \{s\}_i$

Step 5: $\{F\}_i = m_i \{a\}_{ci}$

Step 6: $\{N\}_i = [I]_i \{\alpha\}_i + \{\omega\}_i \times ([I]_i \{\omega\}_i)$

BACKWARD RECURSION

Step 7: $\{f\}_i = \{F\}_i + [R]_{i+1} \{f\}_{i+1}$

Step 8: $\{n\}_i = [R]_{i+1} \{n\}_{i+1} + \{N\}_i + \{s\}_i \times \{F\}_i + \{p\}_i \times ([R]_{i+1} \{f\}_{i+1})$

Step 9: $\tau_i = \{z\} \{n\}_i^T = n_{iz}$

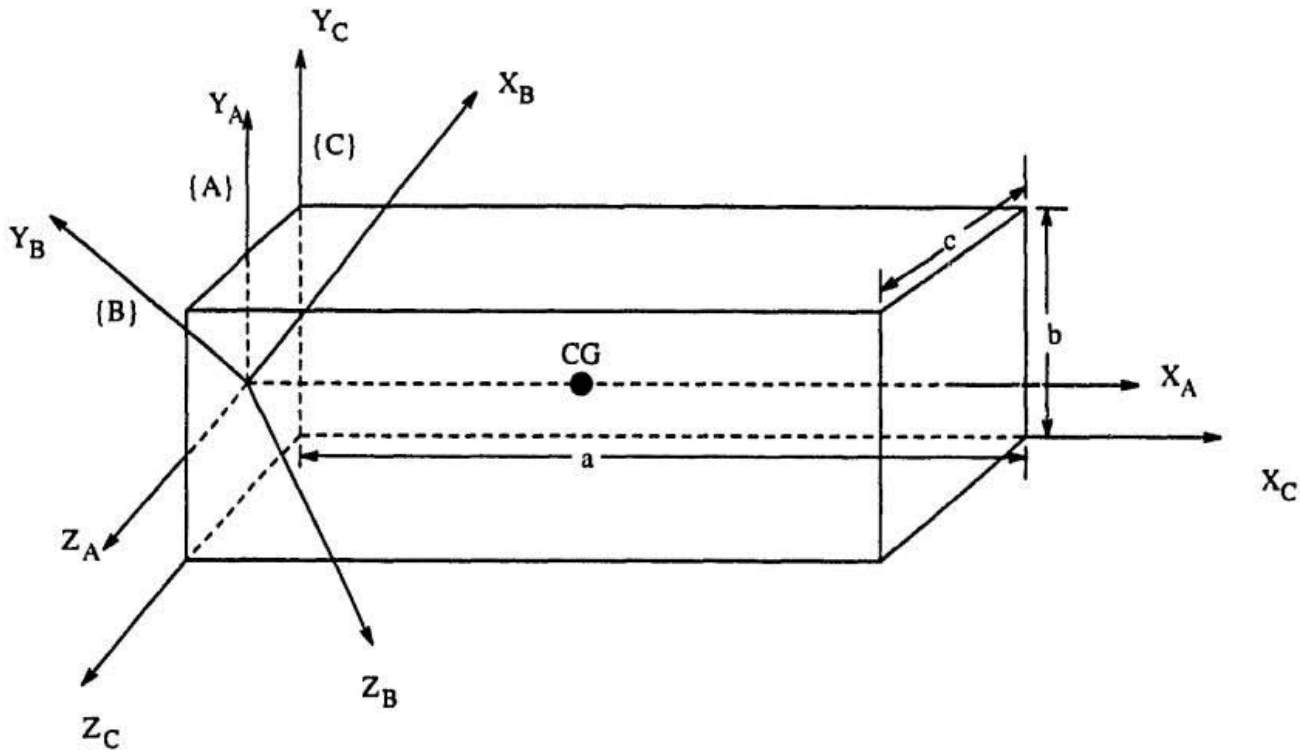

Fig. 2.2 A LINK WITH REGULAR MASS DISTRIBUTION

and ${}^{B}\{\ \}$ can be written as

$$
\begin{aligned}
{}^{A}_{B}[C_1] = {}^{A}_{B}[R_{XYZ}(\gamma,\beta,\alpha)] &= [R_Z(\alpha)]\,[R_Y(\beta)]\,[R_X(\gamma)] \\
&= \begin{bmatrix} c\alpha & -s\alpha & 0 \\ s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} c\beta & 0 & s\beta \\ 0 & 1 & 0 \\ -s\beta & 0 & c\beta \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & c\gamma & -s\gamma \\ 0 & s\gamma & c\gamma \end{bmatrix} \\
&= \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix}
\end{aligned}
\tag{2.12}
$$

which can be used to transform a vector in ${}^{B}\{\ \}$ space into ${}^{A}\{\ \}$ space.

To obtain the inertia matrix in frame ${}^{B}\{\ \}$ one has to use the equation [D'Souza and Garg, 1988; Greenwood, 1970]

$$
{}^{B}[I] = [C_1]^T\ {}^{A}[I]\ [C_1] \tag{2.13}
$$

Applying the same transformation matrix, $[C_1]$, to the position vector of the centroid of the link we get

$$
{}^{B}\{P_C\} = [C_1]^T\ {}^{A}\{P_C\} \tag{2.14}
$$

For this link the inertia matrix $[I]$ in both ${}^{C}\{\ \}$ and ${}^{A}\{\ \}$ frames will be a diagonal matrix. On the other hand, in the ${}^{B}\{\ \}$ frame it will be a symmetric but full matrix (all elements non-zero). A complex mass distribution of the robotic links would also have its inertia matrix, a full matrix as expressed in ${}^{B}[I]$. Fig. 2.3 shows the ${}^{B}\{\ \}$ frame and irregular mass distribution of a link. Thus, if the matrix ${}^{B}[I]$ is known then one can compute forces and torques from Table 2.1.

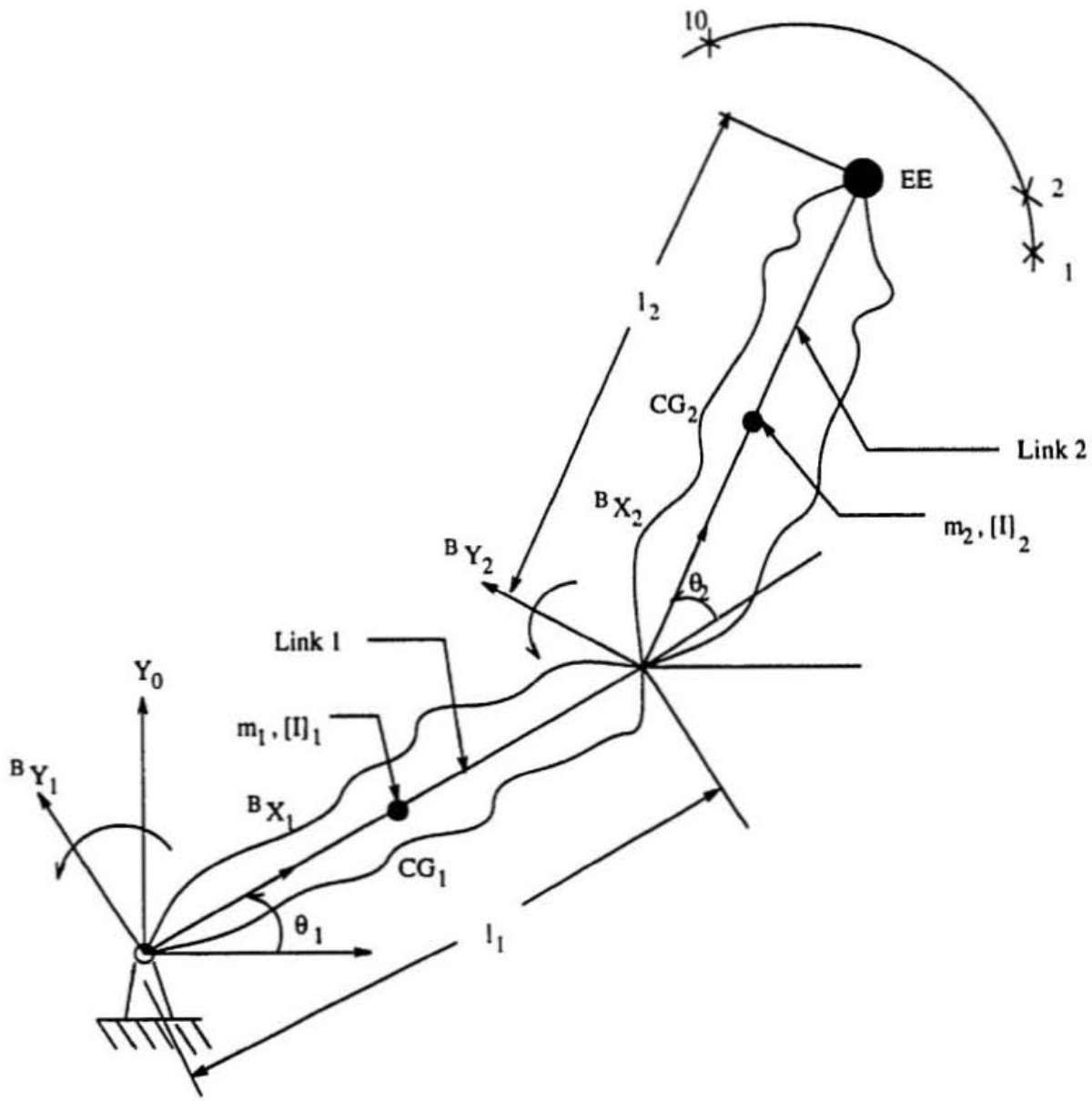

Fig. 2.3 PLANAR TWO-LINK MANIPULATOR WITH IRREGULAR MASS DISTRIBUTION

2.4 PARAMETER IDENTIFICATION

2.4.1 THE LINEAR PROGRAMMING - NEURO (LPN) METHOD

Before describing the details of the parameter identification by using ANN methods, let us discuss briefly one such method called the LP-Neuro method (LPN method). The basic details about the Artificial Neural Networks can be seen in any standard text such as [Hertz et al., 1991; Zurada, 1992]. This, the LPN method, is a non-linear method where most, but not all, of the computations are done using linear programming. This method utilizes the faster convergence property of linear programming which results in better error minimization.

The LPN method is diagrammatically explained in Fig. 2.4. Here the weight matrix [W] which relates {I} and {H} is given by [Balasubramanian and Sharan, 1993]

$$\{H\} = \begin{Bmatrix} h_1 \\ h_2 \\ \vdots \\ h_j \end{Bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1k} \\ w_{21} & w_{22} & \cdots & w_{2k} \\ . & \vdots & \vdots & . \\ w_{j1} & w_{j2} & \cdots & w_{jk} \end{bmatrix} \begin{Bmatrix} i_1 \\ i_2 \\ \vdots \\ i_k \end{Bmatrix} \qquad (2.15)$$

The activation function used in this case is

$$f(h_j) = m_j h_j + c_j \qquad (2.16)$$

In Eq. (2.16), each variable h_j has a corresponding scalar m_j and a constant c_j. The relationship between {O} (output) and {I} (input) is therefore given by

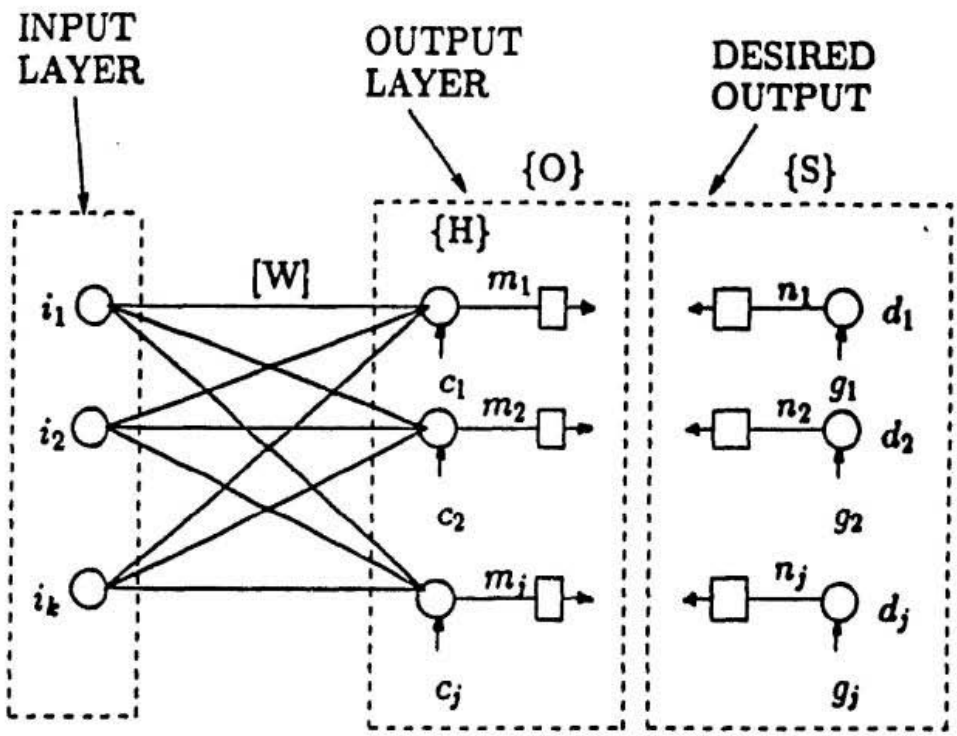

Fig. 2.4 DIAGRAMMATIC REPRESENTATION OF THE NETWORK

(LP-NEURO METHOD)

$$\{O\} = [M]\,[W]\,\{I\} - \{C\} \qquad (2.17)$$

where,

[M] is the diagonal slope matrix, which has scalar m_j as its diagonal elements, and {C} is the intercept vector.

A similar activation function for the desired output side (shown by the box on the extreme right in Fig. 2.4) can be written as

$$\{S\} = [N]\{D\} + \{G\} \qquad (2.18)$$

where [N] is a diagonal matrix. The matrices [N] and {G} are analogous to [M] and {C} in Eq. (2.17). One of the ways to obtain the set of weight matrices with minimum error would be by writing a cost function E_1 in the following form:

Minimize $E_1 = [N]\{D\} + \{G\} - [M][W]\{I\} - \{C\}$

subject to

$$[M][W]\{I\} + \{C\} = [N]\{D\} + \{G\}$$

or in the scalar form, it can be rewritten as

Minimize $E_1 = n_1d_1 + n_2d_2 + \ldots + n_jd_j - m_1\{w_{11}i_1 + w_{12}i_2 + \ldots + w_{1k}i_k\} - m_2\{w_{21}i_1 + w_{22}i_2 + \ldots + w_{2k}i_k\} - \ldots - m_j\{w_{j1}i_1 + w_{j2}i_2 + \ldots + w_{jk}i_k\} + g_1 + g_2 + \ldots + g_j - c_1 - c_2 - \ldots - c_j$

subject to

$$m_1\{w_{11}\,i_1 + w_{12}\,i_2 + \ldots + w_{1k}\,i_k\} + c_1 = n_1d_1 + g_1$$

$$\vdots \qquad \vdots \qquad \vdots$$

$$m_j\{w_{j1}\,i_1 + w_{j2}\,i_2 + \ldots + w_{jk}\,i_k\} + c_j = n_jd_j + g_j$$

where d_j are the elements of the desired output vector and w_{jk} are the weights. The weights w_{jk} can be expressed in a single dimensional array or a vector as

$$\{W\} = \begin{Bmatrix} w_{11} \\ w_{12} \\ \vdots \\ w_{k} \end{Bmatrix} = \begin{Bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{l*k} \end{Bmatrix} \qquad (2.19)$$

The weights w_{jk}, c_j and g_j when replaced by two positive numbers (in the linear programming, the variables are required to have non-negative values only) can be written as

$$w_i = v_i - v_{.1}$$

$$c_i = v_{ci} - v_{..1} \text{ and}$$

$$g_i = v_{gi} - v_{g} \qquad i = 1,2,...$$

$$l = 1,3,... \qquad (2.20)$$

After these substitutions, one arrives at

Minimize $E_1 = n_1d_1 + n_2d_2 + ... + nd - m_1(i_1v_1 - i_1v_2 + i_2v_3 - i_2v_4 + ...) - m_2(i_1v_{2k+1}$

$- i_1v_{2k+2} + ...) - m_3(i_1v_{4k+1} - i_1v_{4k+2} + ...) - m_j(...+ i_kv_{2jk-1} - i_kv_{2jk}) + v_{g1} -$

$v_{g2} + ... - v_{c1} + v_{c2} -$

subject to

$$[A]\{V\} = \{D\} \qquad (2.21)$$

where the coefficient matrix [A] is given by

$$\begin{bmatrix} m_1 i_1 & -m_1 j_1 & - & m_1 i_k & -m_1 i_n & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & m_2 j_1 & -m_2 j_1 & - & m_2 i_k & -m_1 i_n & 1 & -1 & -1 & 1 & - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & . & . & . & . & . & \vdots & . & . & . & . & \vdots & . & . & . & . & \vdots & . & . & . & \vdots & . & . & . & \vdots & . & \vdots & & \\ 0 & 0 & 0 & 0 & - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & - & 0 & 0 & 0 & 0 & - & m_k & -m_k & - & m_k & -m_k & 1 & -1 & -1 & 1 \end{bmatrix}$$

(2.21a)

Here, the coefficient matrix [A], is a sparse matrix. In the above equation, d_j are the desired values and i_k are the input values; v_i, v_{ci} and v_{gi} are the unknown variables and so are m_j and n_j. The problem shown in Eq. (2.21) is non-linear because of the occurrence of product terms such as $m_1 v_1$ ... etc. However, if this problem is combined with another multi-variable optimization problem containing all m_j only, then the problem involving the remaining variables can be solved by the linear programming method. Since, the number of variables far exceeds the number of constraints, it would be better to solve for m_j using non-linear optimization and the remaining variables which include weights, by linear method. This method clearly differs from others because, for the majority of the variables (other than m_j), the linear method yields faster convergence as compared to a totally non-linear method. The other details can be seen in [Balasubramanian, 1994].

2.4.2 ARTIFICIAL NEURAL NETWORK (ANN) IN PARAMETER ESTIMATION

In the Artificial Neural Network method one obtains the weight matrix [W] mentioned in Eq. (2.15) by considering different set of vectors $\{H\}_i$ and $\{I\}_i$ and by carrying out various calculations as given by Eq. (2.21). One must remember that

the weights are expressed as the difference of two positive variables v_i and v_{i+1} in Eq. (2.20). The training of the neural network can be carried out by observing Table 2.2 where the parameters to be identified are shown as the output vector. To train the network, one can generate an irregular link inertia values from a regular link values using Eq. (2.13). It should be pointed out here that for different number of training sets, i, for example if i = 3, Eq. (2.21) can be expressed as [Sharan and Aggarwal, 1994]

$$\begin{matrix}\begin{Bmatrix}\{D\}_1 \\ \cdots\cdots \\ \{D\}_2 \\ \cdots\cdots \\ \{D\}_3\end{Bmatrix} \\ 3m \times 1\end{matrix} = \begin{matrix}\begin{bmatrix}[A]_1 \\ \cdots\cdots \\ [A]_2 \\ \cdots\cdots \\ [A]_3\end{bmatrix} \\ 3m \times n\end{matrix} \{V\}_{n \times 1} \qquad \textbf{(2.22)}$$

For example, here the dimension of the vector $\{D\}_1$ is $m \times 1$, the matrix $[A]_1$ is $m \times n$ and the vector $\{V\}$ is $n \times 1$. If we vary α, β, γ in certain sequential manner we would obtain different sets of inertia matrices in ${}^B\{\ \}$ space. For each of these sets, for a given set of displacement, velocity and acceleration values of the end effector, one can compute the forces and torques for that particular position of the end effector. It should be clarified here that these forces or torques are corresponding to a given point of the traversing end effector at a given position. Therefore, the forces or torques are the dynamic values at a given point on the

TABLE 2.2 INPUT VECTOR, OUTPUT VECTOR AND WEIGHT MATRIX USED FOR TRAINING

$$
\begin{Bmatrix}
I_{xx_1} \\ I_{yy_1} \\ I_{zz_1} \\ I_{xy_1} \\ I_{yz_1} \\ I_{zx_1} \\ Pc_{x_1} \\ Pc_{y_1} \\ Pc_{z_1} \\ I_{xx_2} \\ I_{yy_2} \\ I_{zz_2} \\ I_{xy_2} \\ I_{yz_2} \\ I_{zx_2} \\ Pc_{x_2} \\ Pc_{y_2} \\ Pc_{z_2}
\end{Bmatrix}
=
\begin{bmatrix}
W_{1,1} & W_{1,2} & W_{1,3} & \cdots & W_{1,12} \\
W_{2,1} & W_{2,2} & W_{2,3} & \cdots & W_{2,12} \\
W_{3,1} & W_{3,2} & W_{3,3} & \cdots & W_{3,12} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
W_{18,1} & W_{18,2} & W_{18,3} & \cdots & W_{18,12}
\end{bmatrix}
\begin{Bmatrix}
F_{x_1} \\ F_{y_1} \\ F_{z_1} \\ \tau_{x_1} \\ \tau_{y_1} \\ \tau_{z_1} \\ F_{x_2} \\ F_{y_2} \\ F_{z_2} \\ \tau_{x_2} \\ \tau_{y_2} \\ \tau_{z_2}
\end{Bmatrix}
$$

Here, the subscripts 1 and 2 refer to the link number

trajectory.

By varying α, β, γ one can generate the output vector shown in Table 2.2, whereas the input vector in this table can be calculated using Table 2.1 (for a given set of displacement, velocity and acceleration values). These vectors are then used to obtain the weights i.e. [W] matrix shown in Eq. (2.17). These weights can be used to identify the output vector of a manipulator whose dynamic quantities are unknown by measuring the forces and torques and using Table 2.2. It must be cautioned here that the kinematic parameters i.e. link lengths, velocity, acceleration, position, etc. must remain the same for both the known and unknown manipulators, otherwise the training has to be done differently.

2.5 RESULTS AND DISCUSSIONS

To illustrate the theory developed, a two-link manipulator was used and the various parameters for this manipulator are given in Table 2.3. At first, different values of the inertia matrices were generated by varying α, β and γ. Thereafter, the LPN method was used to identify the dynamic parameters. The comparison of the values of the dynamic parameters corresponding to one unknown set of values of α, β and γ and the LPN Method are depicted in Table 2.4. In this figure, the calculations using the known kinematic and dynamic parameters are done and it includes the determination of [W] matrix. After the training, the input vector mentioned in Table 2.2 for a manipulator whose [I] matrix was not known, was fed to the ANN. Then, the unknown (output) parameters are predicted by these

TABLE 2.3 VARIOUS PARAMETERS USED FOR A TWO-LINK MANIPULATOR

PROPERTY NAME	LINK 1	LINK 2
Link Lengths (m)	0.3	0.2
Mass (kg)	4.0	3.0
Height of Link ,b (m)	0.1	
Width of Link ,c (m)	0.1	
Radius of Circle (m)	0.2	
Maximum Tangential Velocity, V_t (m/s)	0.15	

TABLE 2.4 COMPARISON OF ESTIMATED DYNAMIC PARAMETERS

Kinematic Values : X = 0.08 m, Y = 0.08 m, $\dot{X}$ = 0.6 m/s, $\dot{Y}$ = 0.6 m/s, $\ddot{X}$ = 0.4 m^2/s, $\ddot{Y}$ = 0.6 m^2/s

S. No.	DYNAMIC PARAMETERS	UNKNOWN SET $\alpha, \beta, \gamma = 8°$		LPN METHOD VALUES	
		LINK 1	LINK 2	LINK 1	LINK 2
1	I_{xx} (kg-m^2)	0.011	0.00644	0.0097	-2.88
2	I_{yy} (kg-m^2)	0.121	0.0419	0.095	0.0431
3	I_{zz} (kg-m^2)	0.12	0.0416	0.122	0.042
4	I_{xy} (kg-m^2)	0.0135	0.00436	0.0156	0.002
5	I_{yz} (kg-m^2)	0.00215	0.000693	0.000957	0.000671
6	I_{zx} (kg-m^2)	-0.0178	-0.00573	-0.0177	-0.00572
7	Pc_x (m)	0.147	0.098	0.146	0.0977
8	Pc_y (m)	-0.0178	-0.0118	-0.0178	-0.0118
9	Pc_z (m)	0.0233	0.0156	0.0236	0.0157

weights, using Table 2.2. Its predicted inertia values were then used for the force and torque calculations along the trajectory shown in Figs. 2.5 and 2.6. Figs. 2.7 and 2.8 show the various programming details. The results of these calculations are shown in Figs. 2.9 to 2.14. These results clearly show the usefulness of the ANN method. It is quite obvious that as the end effector is made to travel along the trajectory, the errors in forces or torques are very small. The maximum error is less than 2% for all points having significant and comparable torque values. The only exception was the torque at point 3 for link 2 which was 0.015 N-m, and the error was 0.0041 N-m. In this case the torque itself was negligible.

2.6 CONCLUSIONS

In this chapter the problem of the identification of the dynamic parameters was carried out by using a link of regular geometry and associated inertia values. The inertia matrix of this link was then transformed to another set of axes to obtain a full matrix which a link having complicated mass distribution is normally expected to have. These matrices were then used to train an ANN. After the training, the ANN was used to predict the dynamic parameters of an unknown manipulator. The results obtained show that this technique of parameter estimation can be successfully used.

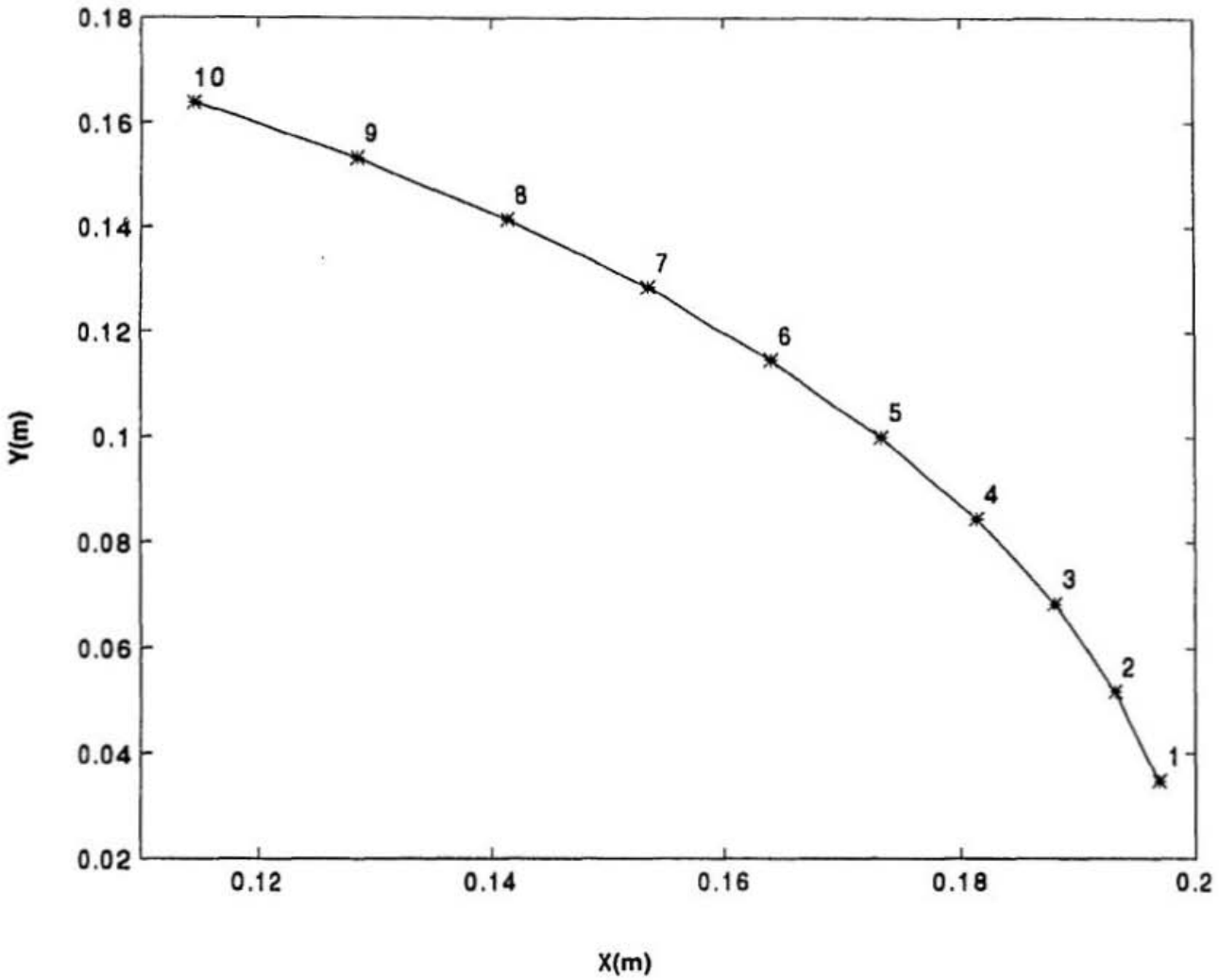

Fig. 2.5 DESIRED TRAJECTORY

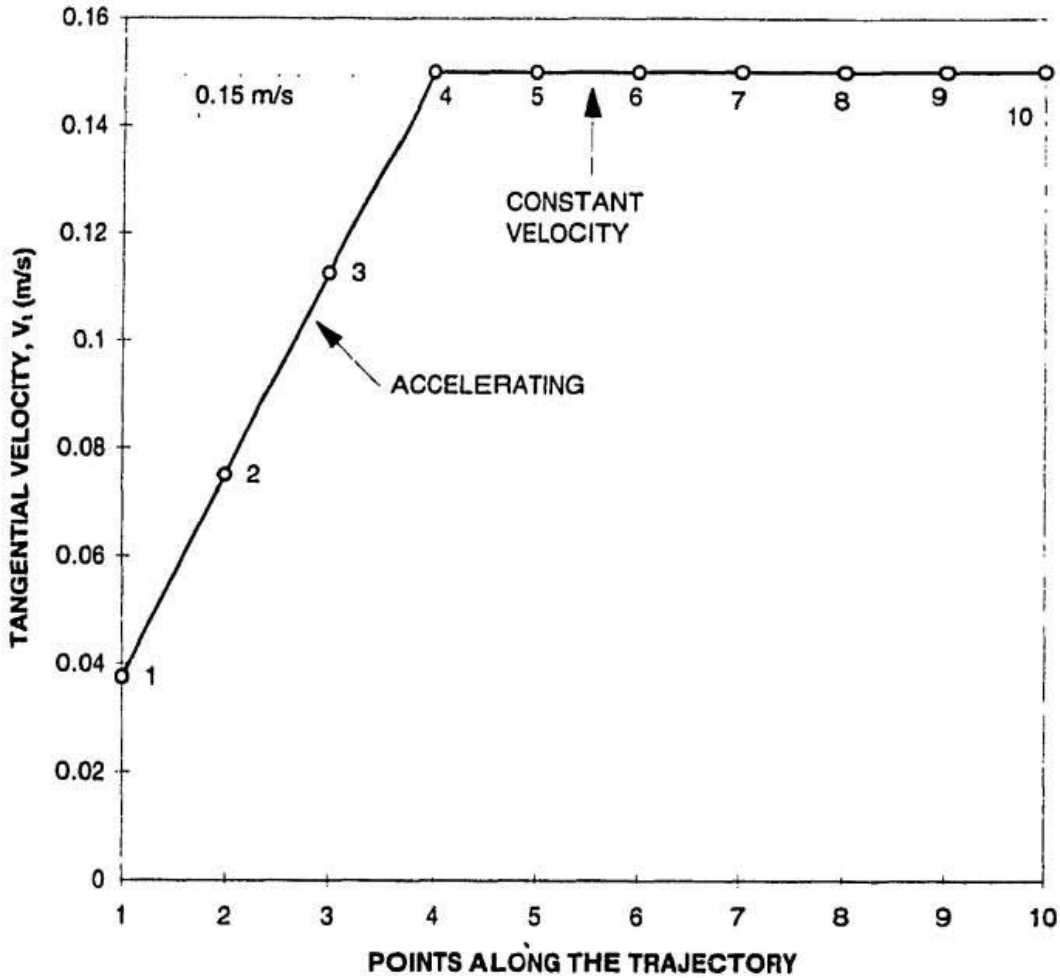

Fig. 2.6 VARIATION OF TANGENTIAL VELOCITY ALONG THE TRAJECTORY

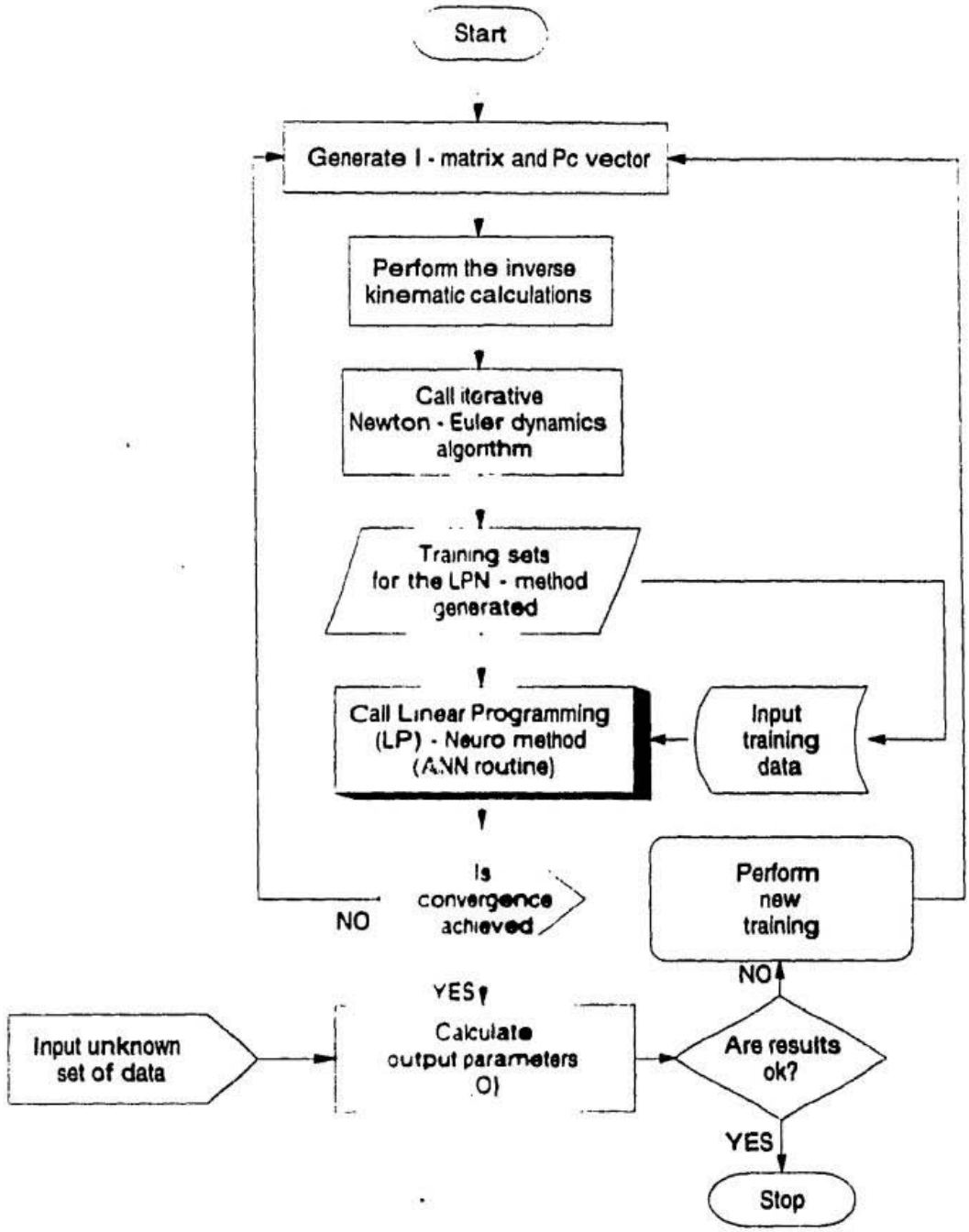

Fig. 2.7 FLOW CHART : DYNAMIC PARAMETER ESTIMATION

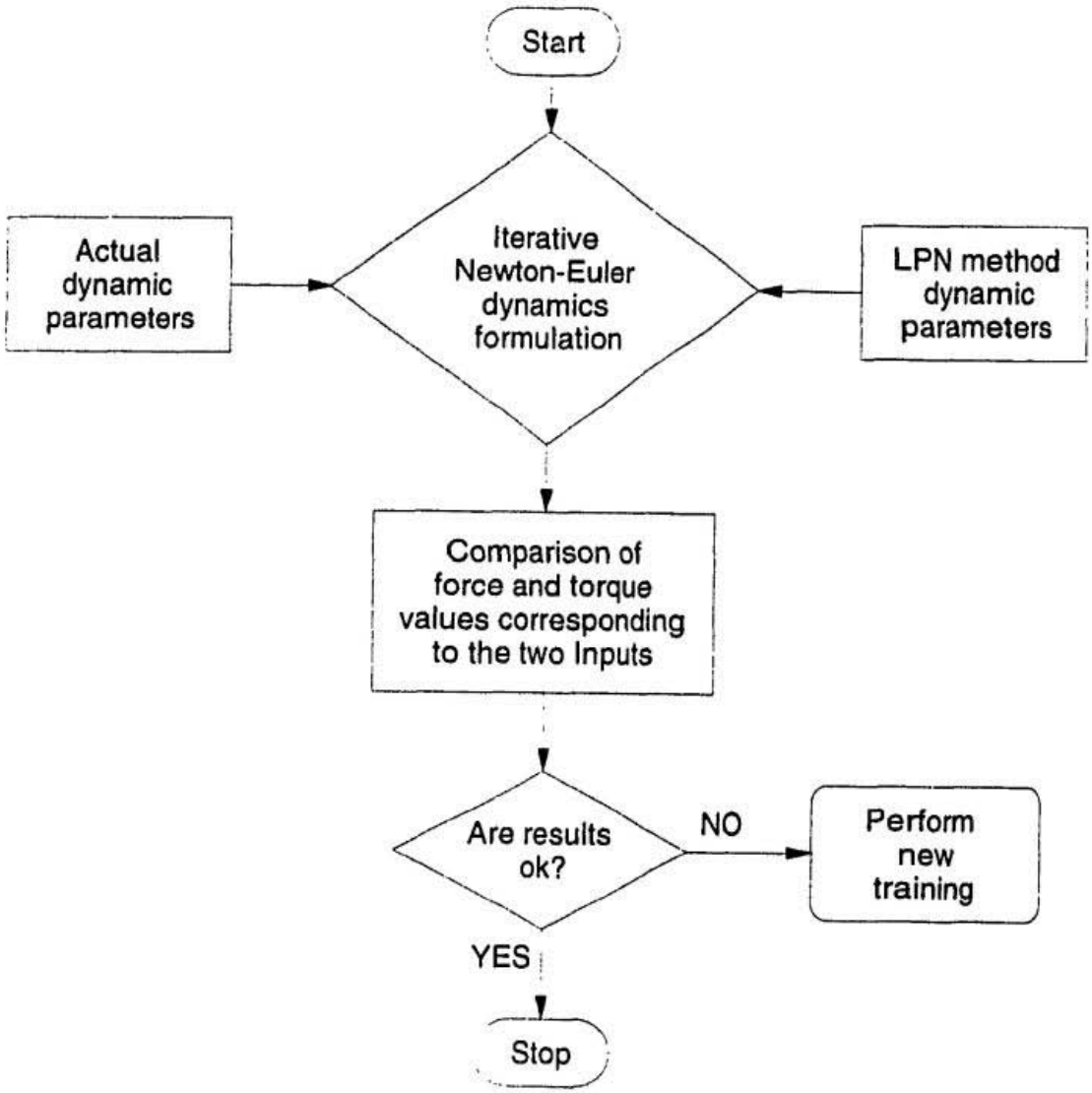

Fig. 2.8 FLOW CHART : COMPARISON OF FORCES AND TORQUES CORRESPONDING TO ESTIMATED PARAMETERS

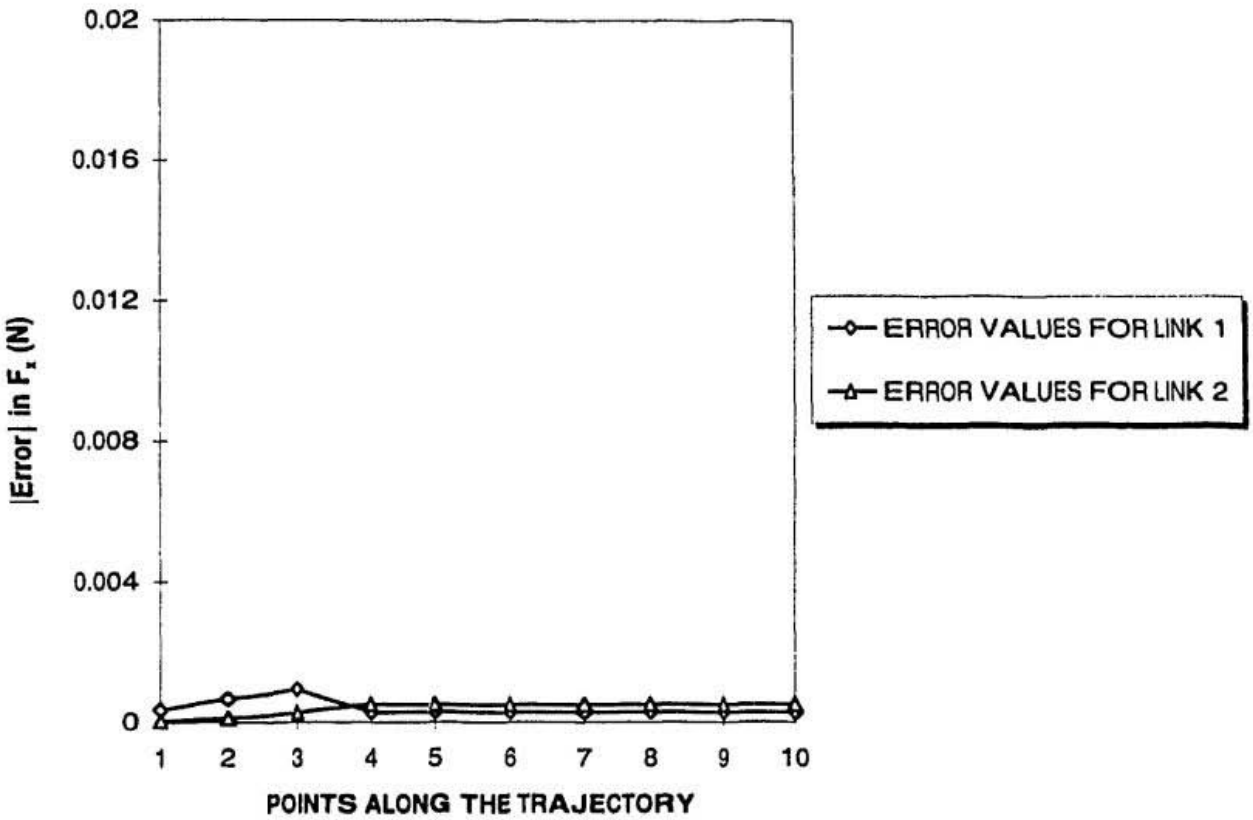

Fig. 2.9 ERROR VALUES IN F_x FOR LINKS 1 AND 2

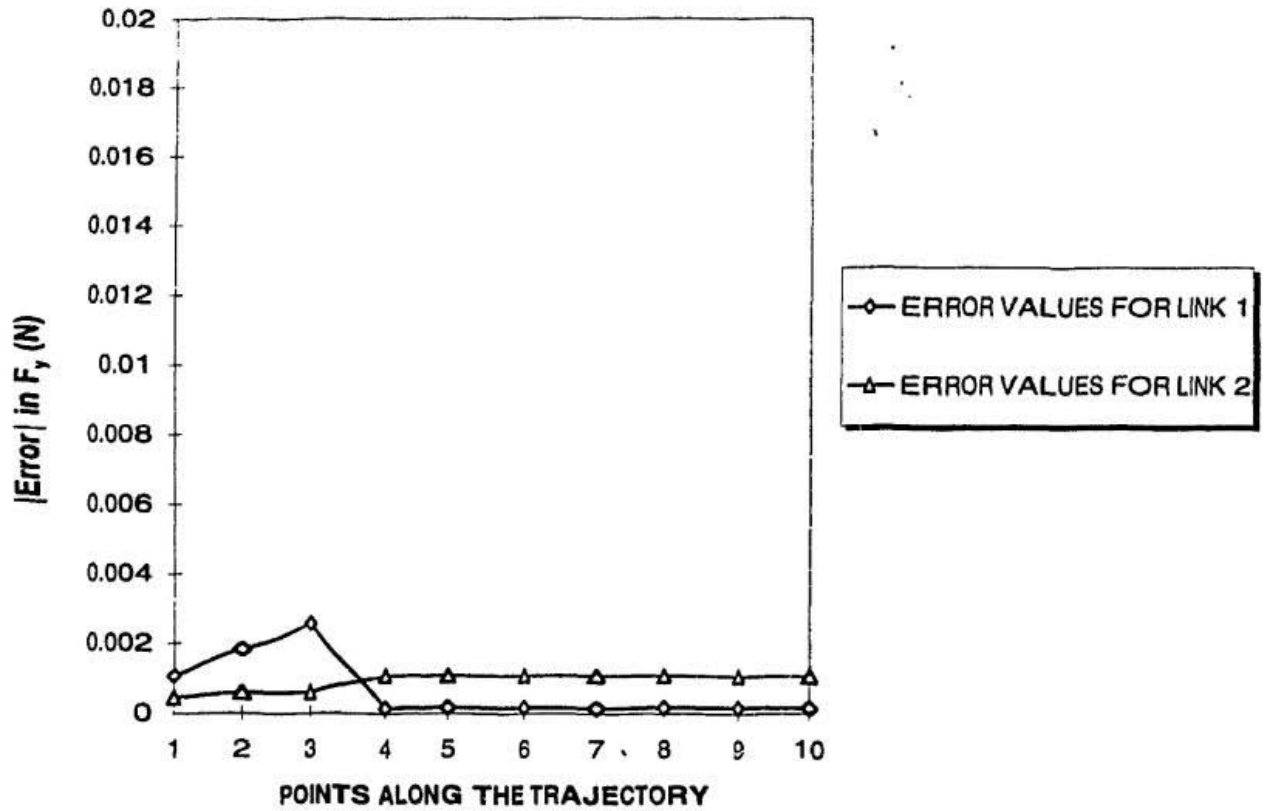

Fig. 2.10 ERROR VALUES IN F_y FOR LINKS 1 AND 2

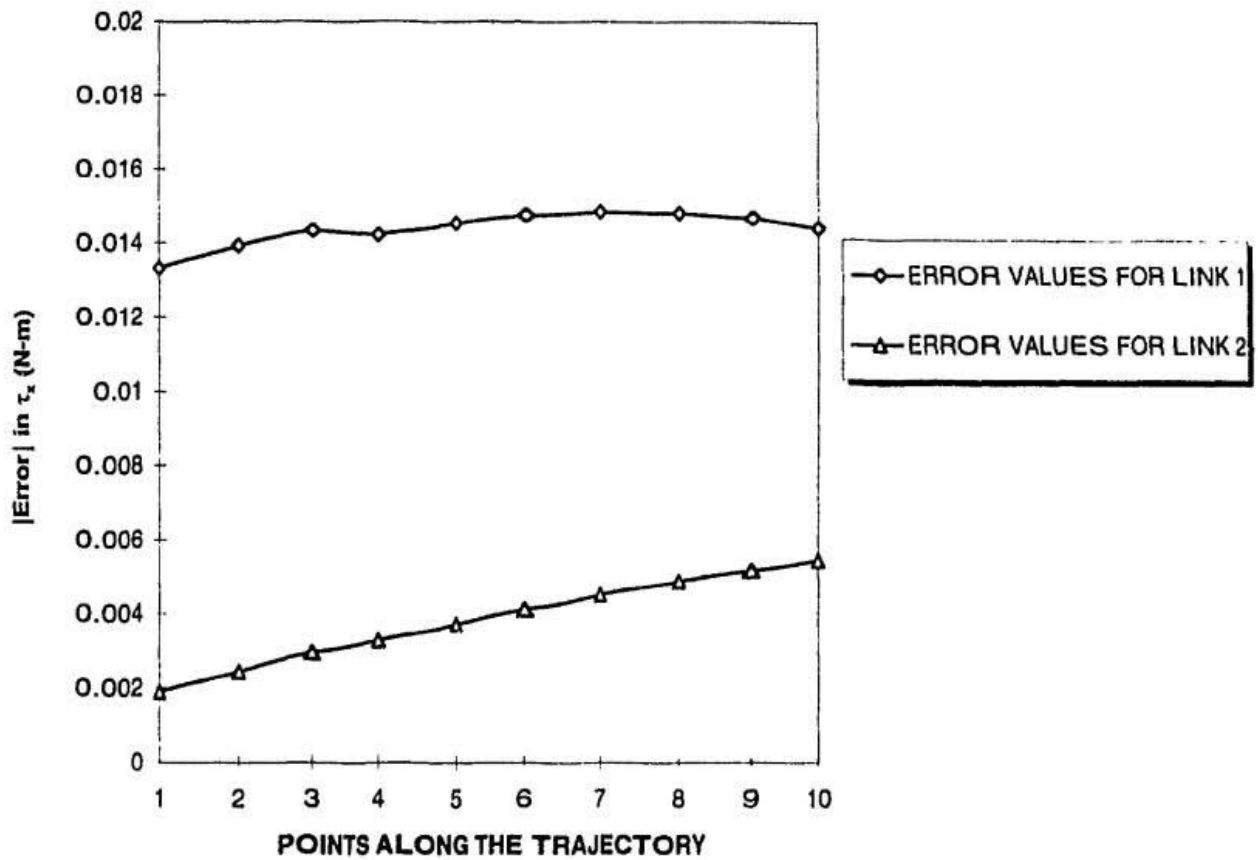

Fig. 2.11 ERROR VALUES IN τ_x FOR LINKS 1 AND 2

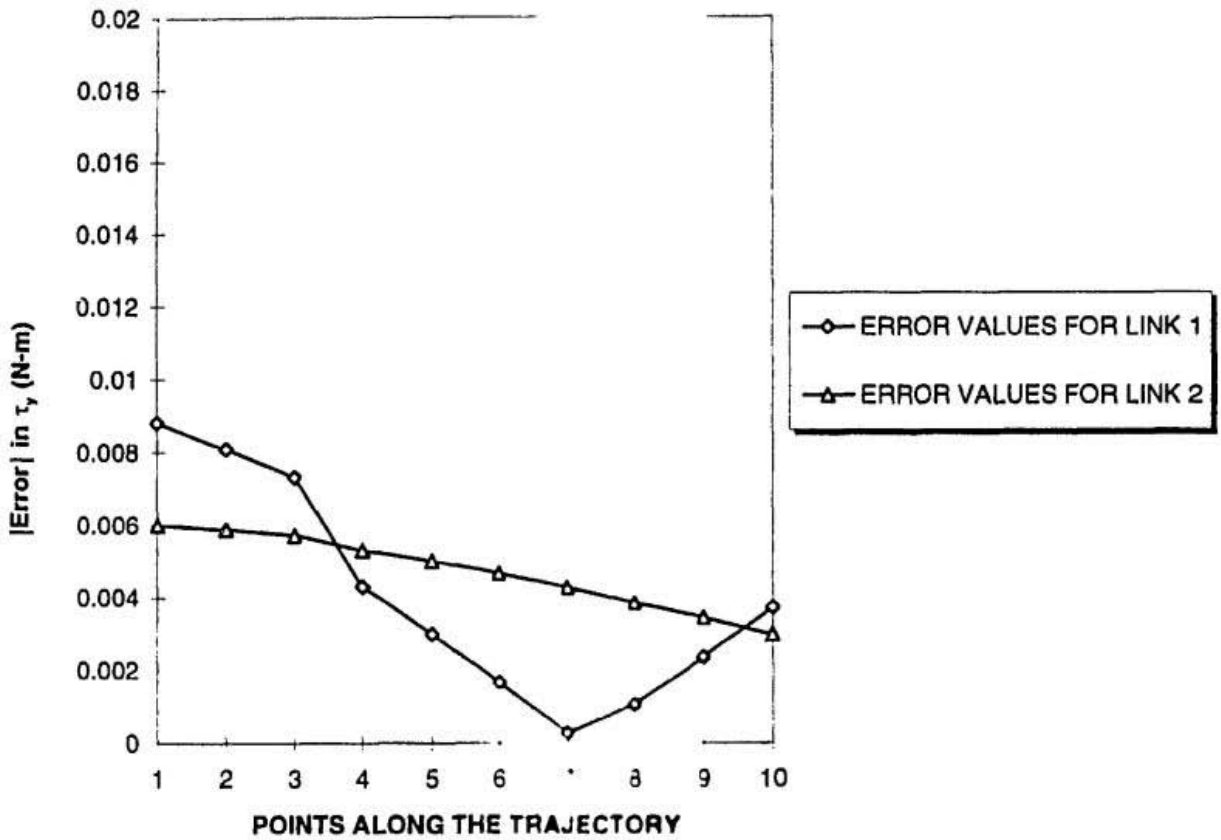

Fig. 2.12 ERROR VALUES IN τ_y FOR LINKS 1 AND 2

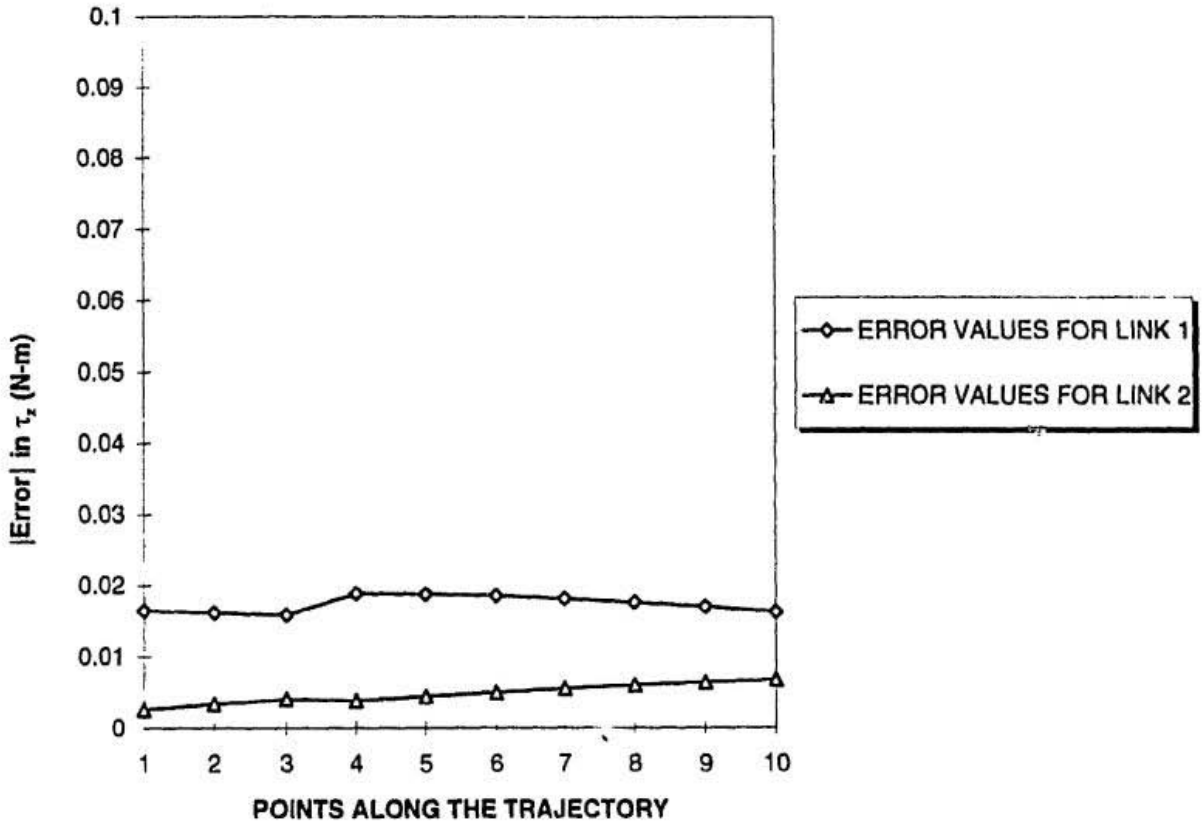

Fig. 2.13 ERROR VALUES IN τ_z FOR LINKS 1 AND 2

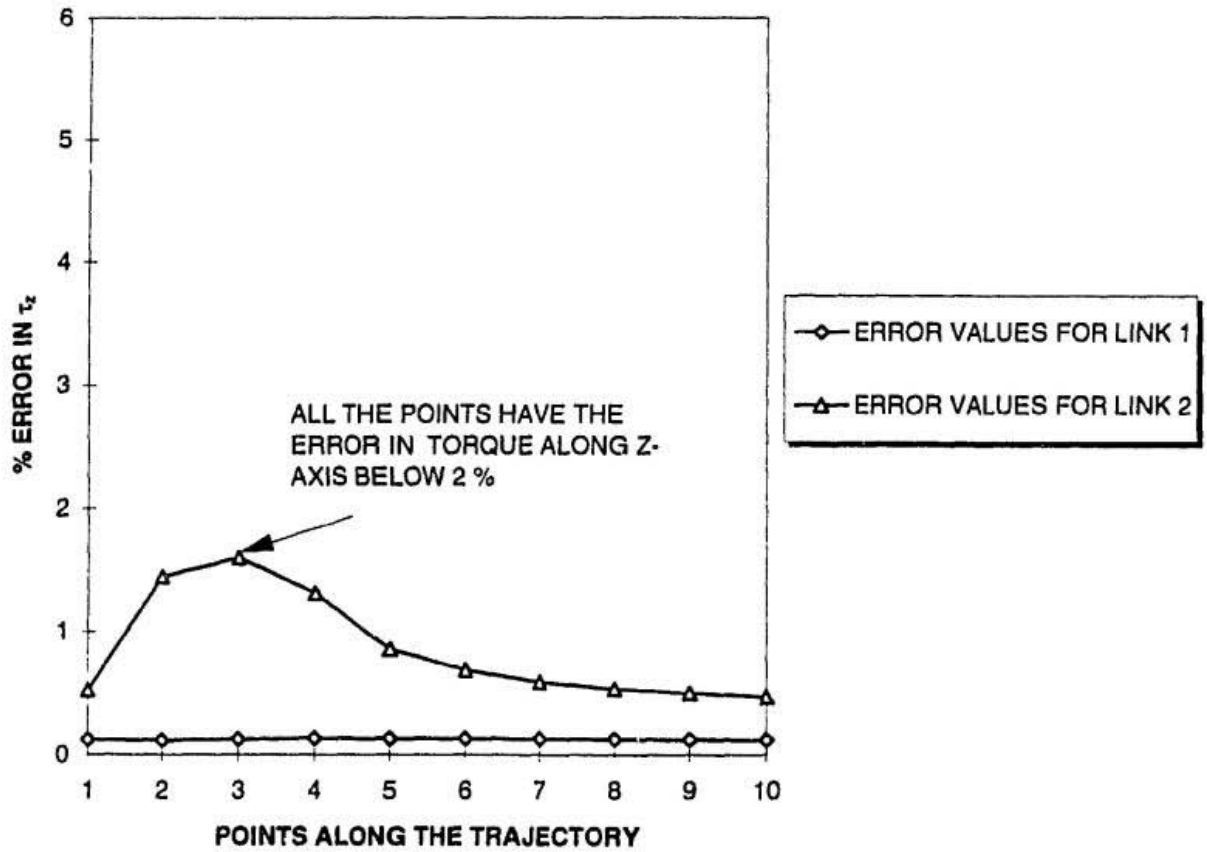

Fig. 2.14 PERCENTAGE ERROR IN τ_z FOR LINKS 1 AND 2

CHAPTER 3

CONTROL OF ROBOTIC MANIPULATORS

3.1 INTRODUCTION

Controlling a robot manipulator along a given trajectory can be divided into two subtasks. The first one is an inverse kinematic calculation which transforms the trajectory, usually specified in a Cartesian coordinate, into a sequence of required joint positions. The second one is the generation of the joint torques from required joint angles and their derivatives. For the intelligent robots the inverse kinematic calculations have to be done in real-time which is a time consuming computational procedure. Also, any attempt to upgrade the existing industrial robots controller imposes a much greater computational load on the control computer.

Recently, significant efforts were made in developing ANN controllers capable of controlling robotic manipulators. The ANN can be quite useful in this respect because the training of the network can be done off-line, in most of the situations where extensive computations can be done on a main-frame or fast computers. Once the training is complete then these networks can enable the manipulator to perform various tasks on the on-line basis.

In this chapter the ANN is successfully used by training the network by

obtaining the gain values, namely position gain values (k_p) and velocity gain values (k_v), corresponding to joint displacement (θ), joint velocity ($\dot{\theta}$), error in joint position values (e) and error in joint velocity values ($\dot{e}$) by using non-linear optimal control technique on off-line basis first. Then, the training of the network is performed using an Artificial Neural Network Method called the Linear Programming - Neuro (LPN) Method, which is faster and more accurate as compared to many other methods. The results obtained are tested on a two link planar manipulator whose end effector moves along a circular trajectory. The study includes the effect of the variation of the maximum velocity along the trajectory. The effects of the sharp changes in the velocity profile and the general control law are also included in this work.

3.2 DYNAMIC EQUATIONS OF A PLANAR TWO LINK MANIPULATOR

When the Newton-Euler equations are evaluated symbolically for any manipulator, they yield the dynamic equation which can be written in the form

$$\{\tau\} = [M(\theta)]\{\ddot{\theta}\} + \{V(\theta,\dot{\theta})\} + \{G(\theta)\} \qquad (3.1)$$

In other words, this is the dynamic equation written in matrix form for an n-link manipulator. Here, $[M(\theta)]$ is the n x n inertia matrix of the manipulator, $\{V(\theta,\dot{\theta})\}$ is an n x 1 vector containing centrifugal and coriolis terms and $\{G(\theta)\}$ is an n x 1 vector having gravity terms. The term $\{V(\theta,\dot{\theta})\}$, appearing in Eq. (3.1) has both the

position and velocity terms. In the case of a simple planar two-link manipulator shown in Fig. 2.1, the matrices involved are:

$$[M(\theta)] = \begin{bmatrix} l_2^2 m_2 + 2l_1 l_2 m_2 c_2 + l_1^2(m_1 + m_2) & l_2^2 m_2 + l_1 l_2 m_2 c_2 \\ l_2^2 m_2 + l_1 l_2 m_2 c_2 & l_2^2 m_2 \end{bmatrix}, \quad \textbf{(3.2)}$$

Any manipulator mass matrix is symmetric and positive definite, and is, therefore, always invertible [Craig, 1989]. Similarly, in the vector

$$\{V(\theta,\dot{\theta})\} = \begin{Bmatrix} -m_2 l_1 l_2 s_2 \dot{\theta}_2^2 - 2m_2 l_1 l_2 s_2 \dot{\theta}_1 \dot{\theta}_2 \\ m_2 l_1 l_2 s_2 \dot{\theta}_1^2 \end{Bmatrix}, \text{ and} \quad \textbf{(3.3)}$$

the term $-m_2 l_1 l_2 s_2 \dot{\theta}_2^2$ is caused by a *centrifugal force*, and is recognized as such because it depends on the square of the joint velocity. The next term, $-2m_2 l_1 l_2 s_2 \dot{\theta}_1 \dot{\theta}_2$ is due to the *Coriolis force* and always contains the product of two different joint velocities. Finally,

$$\{G(\theta)\} = \begin{Bmatrix} m_2 l_2 g c_{12} + (m_1 + m_2) l_1 g c_1 \\ m_2 l_2 g c_{12} \end{Bmatrix} \quad \textbf{(3.4)}$$

contains all those terms in which the gravitational constant, g, appears.

In all of these equations, Eqs. (3.2), (3.3) and (3.4)

$c_1 = \cos\theta_1$; $c_{12} = \cos(\theta_1 + \theta_2)$; $s_1 = \sin\theta_1$, $s_{12} = \sin(\theta_1 + \theta_2)$

These equations are derived using the Newton-Euler algorithm (shown in Table 2.1) based on the following assumptions:

1. All mass exists as a point mass at the distal end of the link.

2. Inertial tensor written at the center of mass for each link is the null matrix.

3. There are no forces acting on the end-effector.

The idea of inverse dynamics is to seek a non-linear feedback control law

$$\tau = f(\theta,\dot{\theta}), \tag{3.5}$$

which when substituted in Eq. (3.1), results in a linear closed loop system. It is quite difficult, if not impossible to find control parameters for general non-linear systems. Since $[M(\theta)]$ is invertible, we may solve for joint acceleration $\{\ddot{\theta}\}$ of the manipulator as

$$\{\ddot{\theta}\} = [M(\theta)]^{-1} \{\{\tau\} - \{V(\theta,\dot{\theta})\} - \{G(\theta)\}\} \tag{3.6}$$

We then apply any of the several known *numerical integration techniques*, where $\{\dot{\theta}\}$ and $\{\theta\}$ are expressed in terms of $\{\ddot{\theta}\}$ mentioned above, to integrate the acceleration to compute the future positions and velocities.

3.3 TRAJECTORY CONTROL

3.3.1 INDEPENDENT PD CONTROL

An independent Proportional-Plus-Derivative (PD) Control scheme (classical control scheme) is used to control the movement of the manipulator. While PD schemes are adequate in most control applications, there is overshooting i.e. the end-effector could go beyond the specified position before actually settling down. Overshooting is quite undesirable, because in order to eliminate overshooting, an

integrator is used which introduces damping and causes the end-effector to move slowly through a number of intermediate set points, thus considerably delaying the completion of the task, and the quality of the displacement etc. The controller design can thus become more sophisticated on account of the involvement of non-linear system dynamics. This type of control algorithm is one class of computed-torque-like controllers.

3.3.2 EVALUATION OF GAIN PARAMETERS

In a PD control scheme, the torque equation is given by

$$\{\tau\} = [K_p]\{e\} + [K_v]\{\dot{e}\} \qquad (3.7)$$

where $\{e\} = \{\theta_d(t+1)\} - \{\theta_a(t)\}$ and

$$\{\dot{e}\} = \{\dot{\theta}_d(t+1)\} - \{\dot{\theta}_a(t)\}$$

From Fig. 3.1, the following observations can be drawn:

1. The kinematic parameters with the subscript *d* represent the desired values on the trajectory. These values are computed based on Table 2.1. The subscript *a* refers to the actual values of the kinematic parameters obtained by solving the control equations which involve the finite difference scheme discussed in Section 3.3.4.

2. It should be noted that the desired values obtained using Table 2.1 can be computed in an off-line based trajectory planning. On the other hand, the actual values and the errors etc., have to be computed on-line. One should try to minimize the on-line computations to increase the speed with which the task is

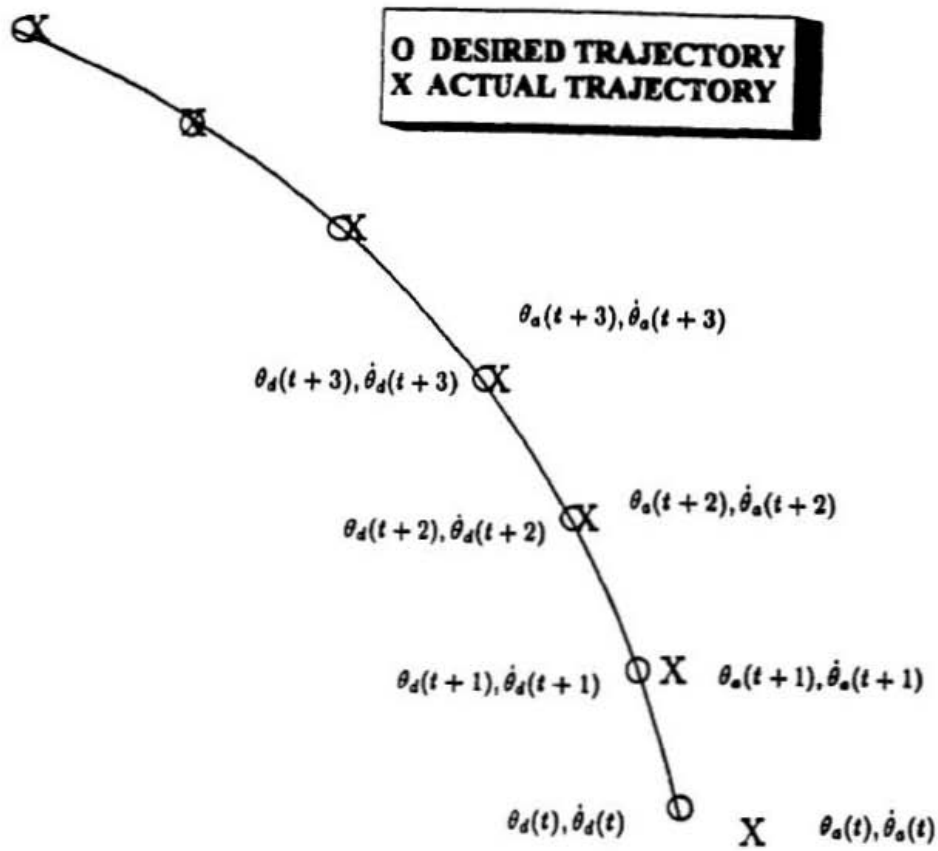

Fig. 3.1 SPECIFICATIONS OF THE DESIRED AND ACTUAL TRAJECTORY

performed.

3. $[K_p]$ and $[K_v]$ are diagonal matrices with diagonal elements consisting of position gain k_p and velocity gain k_v values respectively. Mathematically, they are written as

$$[K_p] = \begin{bmatrix} k_{p1} & 0 \\ 0 & k_{p2} \end{bmatrix} \text{ and} \tag{3.8}$$

$$[K_v] = \begin{bmatrix} k_{v1} & 0 \\ 0 & k_{v2} \end{bmatrix} \tag{3.9}$$

The use of a single value respectively for the entire trajectory for k_p and k_v may not be able to produce torques to follow the desired trajectory. The trajectory control can be achieved by evaluating the set of gain parameters for the entire trajectory using non-linear optimization method (the optimal control method) as described in the Section 3.5 on a point by point basis. This requires the gain values to be different for each point along the trajectory of the manipulator. The objective of the optimal control is to minimize the errors in joint positions and joint velocities between the actual values and the desired values, based on the gain variables $[K_p]$ and $[K_v]$.

3.3.3 HOOKES AND JEEVES NON-LINEAR OPTIMIZATION METHOD

The gain values are evaluated using non-linear optimization routine that would minimize $\|\theta_a(t+1) - \theta_d(t+1)\|$ and $\|\dot{\theta}_a(t+1) - \dot{\theta}_d(t+1)\|$. $[K_p]$ and $[K_v]$ are obtained using the Hookes and Jeeves method (Rao, 1978). The pattern search

method of Hooke and Jeeves is a sequential technique each step of which consists of two kinds of moves, one called the exploratory move and the other called the pattern move. The first kind of move is included to explore the local behaviour of the objective function and the second kind of move is included to take advantage of the pattern direction.

Let us define

$$\{X\} = \begin{Bmatrix} k_{p1} \\ k_{p2} \\ k_{,1} \\ k_{,2} \end{Bmatrix}, \tag{3.10}$$

and

$$F(\{X\}) = \{\theta_a(t+1) - \theta_s(t+1)\}^2 + \{\dot{\theta}_a(t+1) - \dot{\theta}_d(t+1)\}^2 \tag{3.11}$$

The objective function can be mathematically written as

$$U(\{X\}) = F(\{X\}) + \sum_{k=1}^{m} P_k g_k^2 H(g_k) \tag{3.12}$$

where {X} is the design vector, k constraints are represented as g_k and $H(g_k)$ is the Heavyside unit step function defined so that

$$H(g_k) = 1 \quad \text{for } g_k \geq 0 \quad \text{or,}$$

$$H(g_k) = 0 \quad \text{for } g_k < 0 \tag{3.13}$$

In Eq. (3.12), P_k are large penalty constants which are positive because the present problem is a minimization problem. Next, one needs to solve for the

minimum of U({X}) using the Hookes and Jeeves method and the step-by-step procedure for a design vector {X} having n components is mentioned below:

(i) Start with an initial estimate of the design vector

$$\{X\} = \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{Bmatrix} \qquad \textbf{(3.14)}$$

and choose Δx_i, i = 1,2,...,n as step lengths in each of the coordinate directions u_i, i = 1,2,...,n.

(ii) Set temporary base point $\{Y_{k,0}\} = \{X_k\}$

(iii) Start the exploratory move by perturbing one design variable at a time in order to find the improved value of the objective function. Set:

$$\{Y_{k,i}\} = \begin{cases} \{Y_{k,i-1}\} + \Delta x_i\{u_i\} & \text{if } U^+ = U(\{Y_{k,i-1}\} + \Delta x_i\{u_i\}) \\ & \quad < U = U(\{Y_{k,i-1}\}) \\ \{Y_{k,i-1}\} - \Delta x_i\{u_i\} & \text{if } U^- = U(\{Y_{k,i-1}\} - \Delta x_i\{u_i\}) \\ & \quad < U = U(\{Y_{k,i-1}\}) \\ & \quad < U^+ = U(\{Y_{k,i-1}\} + \Delta x_i\{u_i\}) \\ \{Y_{k,i-1}\} & \text{if } U = U(\{Y_{k,i-1}\}) < \min(U^+, U^-) \end{cases} \qquad \textbf{(3.15)}$$

In this way, all the design variables x_n are perturbed and the improved position $\{Y_{k,n}\}$ found.

(iv) If the point $\{Y_{k,n}\}$ is not different from $\{X_k\}$, reduce the step lengths Δx_i; set i = 1 and go to step (iii). If $\{Y_{k,n}\}$ is different from $\{X_k\}$ obtain the new base point as

$$\{X_{k+1}\} = \{Y_{k,n}\} \tag{3.16}$$

(v) Find the pattern direction {S} using

$$\{S\} = \{X_{k+1}\} - \{X_k\} \tag{3.17}$$

Find the point $\{Y_{k+1,0}\}$ as

$$\{Y_{k+1,0}\} = \{X_{k+1}\} + \lambda \{S\} \tag{3.18}$$

Find λ^*, the optimum step length in the direction {S} and use λ^* in Eq.(3.18).

(vi) Set k = k+1, $U_k = U(\{Y_{k0}\})$ and i = 1; repeat step (iii). If at the end of step (iii), $U(\{Y_{k,n}\}) < U(X_k)$ use the new base point as $\{X_{k+1}\} = \{Y_{k,n}\}$ and go to step (v). If $U(\{Y_{k,n}\}) >= U(\{X_k^-\})$, set $\{X_{k+1}\} = \{X_k\}$ and reduce step lengths; set k = k+1 and go to step (ii).

(vii) The process is terminated if the step lengths become less than ε, a very small quantity.

3.3.4 NUMERICAL INTEGRATION TECHNIQUE

3.3.4.1 RUNGE-KUTTA METHOD FOR A SECOND ORDER DIFFERENTIAL EQUATION

The Runge-Kutta method is self-starting and gives quite accurate results. If we have a single second order differential equation as

$$\ddot{x} = \frac{1}{m}\left[F(t) - c\dot{x} - kx\right] = f(x, \dot{x}, t) \tag{3.19}$$

By defining $\dot{x} = y$. This can be written as two first order equations:

$$\dot{x} = y \tag{3.20}$$

$$\dot{y} = f(x, y, t) \tag{3.21}$$

By defining

$$\{X(t)\} = \begin{Bmatrix} x(t) \\ y(t) \end{Bmatrix} \tag{3.22}$$

$$\{F(t)\} = \begin{Bmatrix} y \\ f(x,y,t) \end{Bmatrix} \tag{3.23}$$

the following recurrence formula is used to find the values of X(t) at different grid points t_i according to the fourth order Runge-Kutta method.

$$\{X_{i+1}\} = \{X_i\} + \frac{1}{6}\left[\{K_1\} + 2\{K_2\} + 2\{K_3\} + \{K_4\}\right] \tag{3.24}$$

where

$$\{K_1\} = h\ F\left(\{X_i\}, t_i\right) \tag{3.25}$$

$$\{K_2\} = h\ F\left(\{X_i\} + \frac{1}{2}\{K_1\}, t_i + \frac{1}{2}h\right) \tag{3.26}$$

$$\{K_3\} = h\ F\left(\{X_i\} + \frac{1}{2}\{K_2\}, t_i + \frac{1}{2}h\right) \tag{3.27}$$

$$\{K_4\} = h\ F\left(\{X_i\} + \{K_3\}, t_{i+1}\right) \tag{3.28}$$

In the variable form the Runge-Kutta Method is shown in Table 3.1.

These quantities evaluated in the table are then used in the following recurrence

TABLE 3.1 RUNGE-KUTTA METHOD IN VARIABLE FORM

t	x	$y = \dot{x}$	$f = \dot{y} = \ddot{x}$
$T_1 = t_i$	$X_1 = x_i$	$Y_1 = y_i$	$F_1 = f(T_1, X_1, Y_1)$
$T_2 = t_i + h/2$	$X_2 = x_i + Y_1.(h/2)$	$Y_2 = y_i + F_1.(h/2)$	$F_2 = f(T_2, X_2, Y_2)$
$T_3 = t_i + h/2$	$X_3 = x_i + Y_2.(h/2)$	$Y_3 = y_i + F_2.(h/2)$	$F_3 = f(T_3, X_3, Y_3)$
$T_4 = t_i + h$	$X_4 = x_i + Y_3.(h/2)$	$Y_4 = y_i + F_3.(h/2)$	$F_4 = f(T_4, X_4, Y_4)$

formula to calculate the values of x and $\dot{x}$ at the next time step as:

$$x_{i+1} = x_i + \frac{h}{6}[Y_1 + 2Y_2 + 2Y_3 + Y_4] \quad \textbf{(3.29)}$$

$$y_{i+1} = y_i + \frac{h}{6}[F_1 + 2F_2 + 2F_3 + F_4] \quad \textbf{(3.30)}$$

3.3.4.2 RUNGE-KUTTA METHOD FOR THE CONTROL PROBLEM

Rewriting Eq. (3.6), the differential equation for the present problem is given by

$$\{\ddot{\theta}\} = [M(\theta)]^{-1}\{\{\tau\} - \{V(\theta,\dot{\theta})\} - \{G(\theta)\}\} \quad \textbf{(3.6)}$$

here the quantities $[M(\theta)]$, $\{V(\theta,\dot{\theta})\}$ and $\{G(\theta)\}$ are given by Eqs. (3.2), (3.3) and (3.4) respectively. After substituting Eq. (3.7) into Eq. (3.6) one can write Eq. (3.6) for the planar two link manipulator as [Aggarwal et. al., 1995]

$$\begin{Bmatrix}\ddot{\theta}_1\\ \ddot{\theta}_2\end{Bmatrix} = \begin{bmatrix}a(\theta_2) & b(\theta_2)\\ c(\theta_2) & d(\theta_2)\end{bmatrix} \times \left\{\left([K_p]\begin{Bmatrix}\theta_{1_r} - \theta_1\\ \theta_{2_r} - \theta_2\end{Bmatrix} - [K_v]\begin{Bmatrix}\dot{\theta}_{1_r} - \dot{\theta}_1\\ \dot{\theta}_{2_r} - \dot{\theta}_2\end{Bmatrix}\right) - \begin{Bmatrix}V_1(\dot{\theta}_1, \dot{\theta}_2)\\ V_2(\dot{\theta}_1)\end{Bmatrix} - \begin{Bmatrix}G_1(\theta_1,\theta_2)\\ G_2(\theta_1,\theta_2)\end{Bmatrix}\right\} \quad \textbf{(3.31)}$$

Where $\begin{bmatrix}a(\theta_2) & b(\theta_2)\\ c(\theta_2) & d(\theta_2)\end{bmatrix}$ is $[M(\theta)]^{-1}$ **(3.32)**

and $[M(\theta)]^{-1}$ can also be written as

$$[M(\theta)]^{-1} = \frac{1}{\det[M(\theta)]} \begin{bmatrix} I_2^2 m_2 & -\left(I_2^2 m_2 + I_1 I_2 m_2 c_2\right) \\ -\left(I_2^2 m_2 + I_1 I_2 m_2 c_2\right) & I_2^2 m_2 + 2 I_1 I_2 m_2 c_2 + I_1^2 (m_1 + m_2) \end{bmatrix} \quad \textbf{(3.33)}$$

Eq. (3.31) is a combination of two second order differential equations so, we will use the fourth Order Runge-Kutta Method discussed in Section 3.3.4.1. Defining:

$$\dot{\theta}_1 = \Psi_1 \quad \text{and} \quad \dot{\theta}_2 = \Psi_2 \quad \textbf{(3.34)}$$

the two second order differential equations in Eq. (3.31) can now be written as four first order differential equations as

$$\left.\begin{aligned} \dot{\theta}_1 &= \Psi_1 \\ \dot{\Psi}_1 = \ddot{\theta}_1 &= F(\theta_1, \theta_2, \Psi_1, \Psi_2) \\ \dot{\theta}_2 &= \Psi_2 \\ \dot{\Psi}_2 = \ddot{\theta}_2 &= G(\theta_1, \theta_2, \Psi_1, \Psi_2) \end{aligned}\right\} \quad \textbf{(3.35)}$$

Therefore, one can define

$$\{X(t)\} = \begin{Bmatrix} \theta_1(t) \\ \Psi_1(t) \\ \theta_2(t) \\ \Psi_2(t) \end{Bmatrix}, \quad \text{and} \tag{3.36}$$

$$\{F(t)\} = \begin{Bmatrix} \Psi_1 \\ \dot{\Psi}_1 \\ \Psi_2 \\ \dot{\Psi}_2 \end{Bmatrix} \tag{3.37}$$

along the similar lines as in Eqs. (3.22) and (3.23). Finally, using Eq. (3.24) with Eqs. (3.25), (3.26), (3.27) and (3.28) we numerically integrate to calculate {X(t)} for the next time step.

3.4 TRAJECTORY GENERATION

In robotic applications, a desired task is usually specified in the work space or Cartesian space, as this is where the motion of the manipulator is easily described in relation to the external environment and workpiece. However, trajectory-following control is easily performed in the joint space as this is where the arm dynamics is more easily formulated. Therefore, it is important to be able to find the desired joint space trajectory given the Cartesian trajectory. This is accomplished using the inverse kinematics.

Fig. 2.1 shows the planar two link manipulator. For this investigation we have chosen a circular arc as the desired trajectory (Fig. 3.2). In Fig. 3.2 the

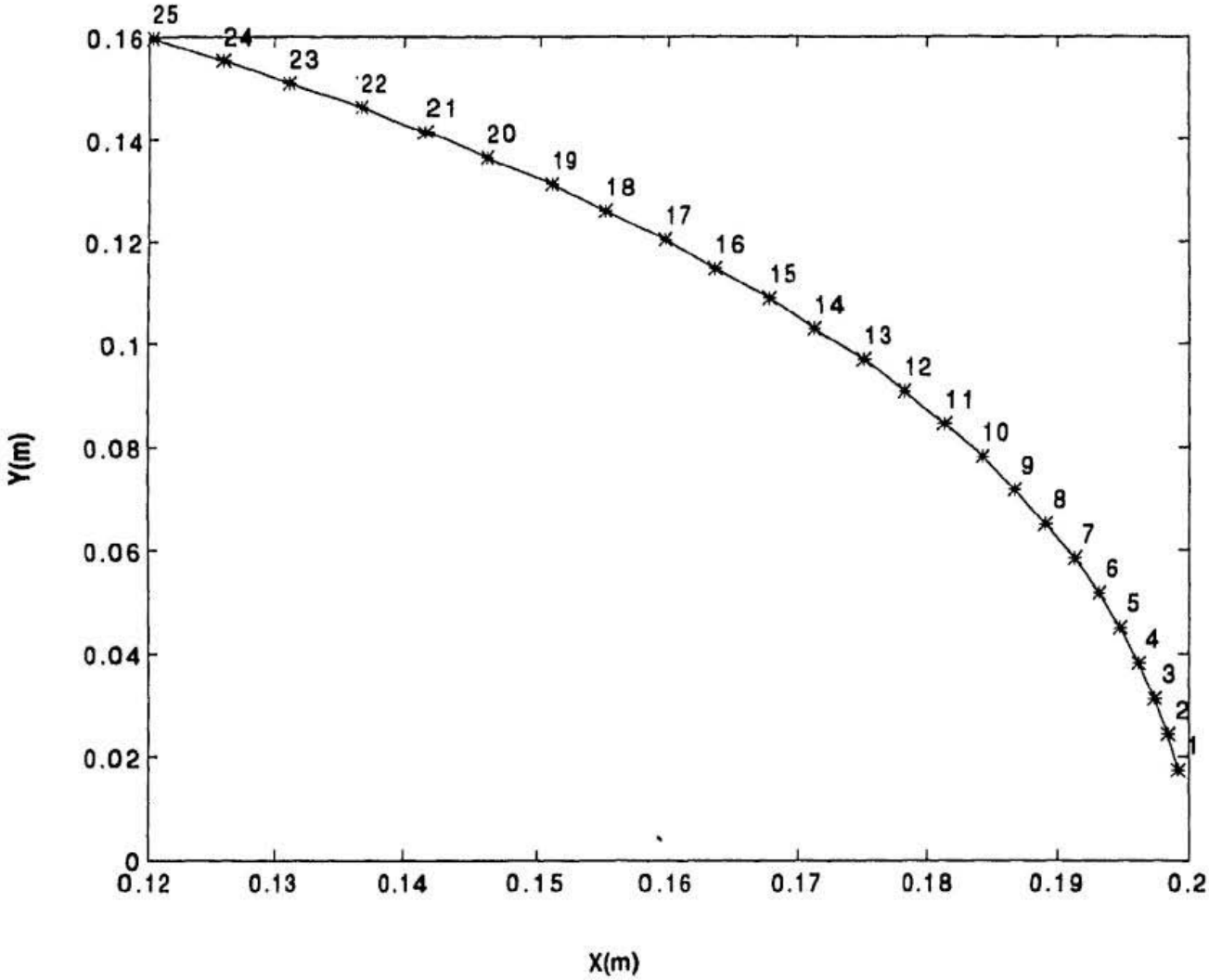

Fig. 3.2 DESIRED TRAJECTORY

manipulator end effector starts at point 1 and goes to point 25 on the trajectory with the variation of angle from 5° to 53° with increments of 2°. The radius of the circular arc is 0.2 m. Table 3.2 shows the various link parameters used. Fig. 3.3 shows the variation of tangential velocity along the trajectory.

If θ (5° to 53°) is the angle with which the end effector moves along the trajectory then the Cartesian coordinates of the end effector of the two link manipulator is given by

$$\begin{aligned} x &= r\cos(\theta) \\ y &= r\sin(\theta) \end{aligned} \tag{3.38}$$

where r is the radius of the circular arc.

Knowing the tangential velocity V_t the Cartesian velocities of the end effector are given by

$$\begin{Bmatrix} \dot{x} \\ \dot{y} \end{Bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{Bmatrix} 0 \\ V_t \end{Bmatrix} \tag{3.39}$$

The radial and tangential accelerations are given by

$$\begin{aligned} A_r &= \frac{V_t^2}{r} \\ A_t &= \frac{V_{t_{i+1}}^2 - V_{t_i}^2}{2r\,\Delta\left(\theta_{t_{i+1}} - \theta_{t_i}\right)} \end{aligned} \tag{3.40}$$

where V_{ti+1} is the tangential velocity at the next time step and V_{ti} is the tangential velocity at the previous value of time. Therefore, the Cartesian acceleration values

Table 3.2 VARIOUS PARAMETERS USED FOR A TWO-LINK MANIPULATOR

PROPERTY NAME	LINK 1	LINK 2
Link Lengths (m)	0.3	0.2
Mass (kg)	4.0	3.0
Height of Link ,b (m)	0.1	
Width of Link ,c (m)	0.1	
Radius of Circle (m)	0.2	
Maximum Tangential Velocity, V_t (m/s)	0.15	
Initial Position Gain Values	K_{p1} = 150.0; K_{p2} = 150.0	
Initial Velocity Gain Values	K_{v1} = 150.0; K_{v2} = 150.0	
Step Size in Time, ΔT (s) for RK Method	0.001	

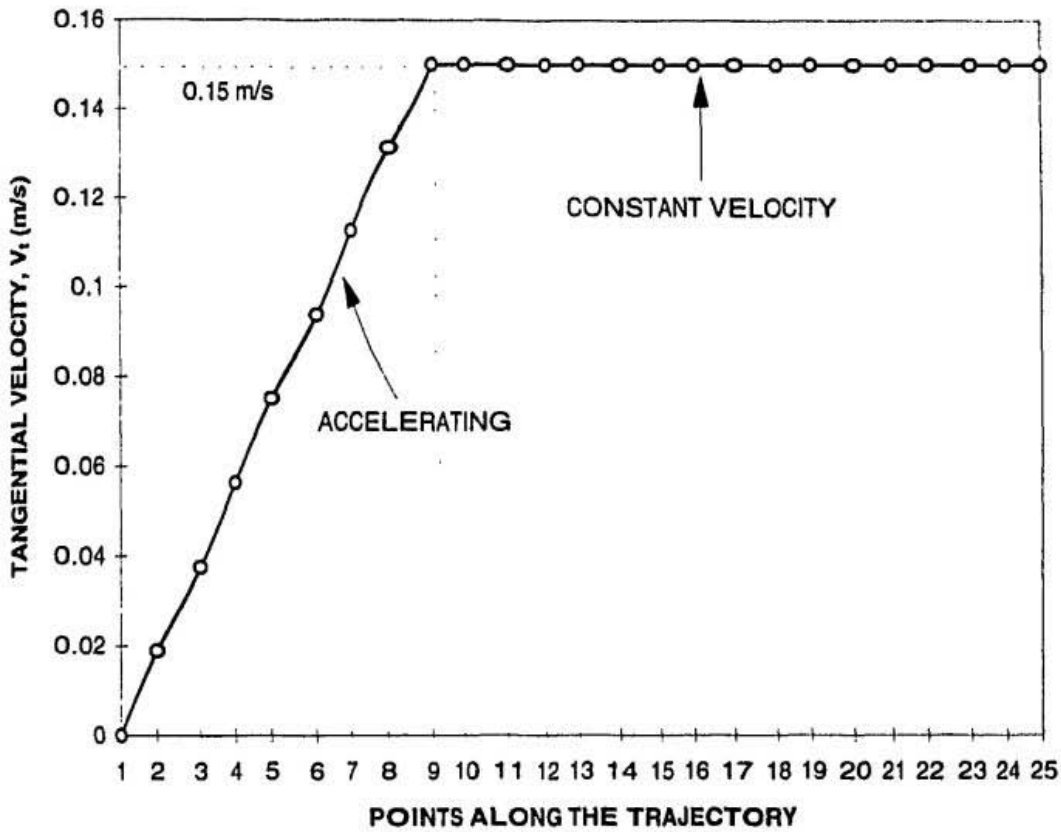

Fig. 3.3 DESIRED TANGENTIAL VELOCITY PROFILE

are given by

$$\begin{Bmatrix} \ddot{x} \\ \ddot{y} \end{Bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{Bmatrix} A_r \\ A_t \end{Bmatrix} \qquad (3.41)$$

Finally, knowing the Cartesian displacement, velocity and acceleration values, we use the inverse kinematic calculations as described below to calculate the joint displacement, velocity and acceleration values which are to be used in Eq. (3.31). The variation of joint displacement values for links 1 and 2 is shown in Table 3.3. Fig. 3.4 shows the orientation of the end effector of the planar two link manipulator following the desired trajectory in accordance with the joint displacement values given in Table 3.3.

A brief review of the computation procedure which for the sake of clarity can be stated in the step-by-step forms as:

STEP 1 : Compute the left hand side terms in Eqs. (3.38) to (3.41) in a sequential manner.

STEP 2 : Compute θ_1 and θ_2 using Eqs. (2.1) to (2.6).

STEP 3 : Use $\dot{X}$ and $\dot{Y}$ from Eq. (3.39) and compute $\dot{\theta}_1$ and $\dot{\theta}_2$ from Eq. (2.8) which is a set of two simultaneous linear algebraic equations.

STEP 4 : Similarly use $\ddot{X}$ and $\ddot{Y}$ from Eq. (3.41) and compute $\ddot{\theta}_1$ and $\ddot{\theta}_2$ using Eq. (2.9). One has to remember that $\dot{\theta}_1$ and $\dot{\theta}_2$ have already been calculated in step 3 above.

Table 3.3 VARIATION OF JOINT DISPLACEMENT VALUES

POINTS	θ_1 (rad)	θ_1 (deg)	θ_2 (rad)	θ_2 (deg)
1	-0.64	-36.41	2.42	138.59
2	-0.60	-34.41	2.42	138.59
3	-0.57	-32.41	2.42	138.59
4	-0.53	-30.41	2.42	138.59
5	-0.50	-28.41	2.42	138.59
6	-0.46	-26.41	2.42	138.59
7	-0.43	-24.41	2.42	138.59
8	-0.39	-22.41	2.42	138.59
9	-0.36	-20.41	2.42	138.59
10	-0.32	-18.41	2.42	138.59
11	-0.29	-16.41	2.42	138.59
12	-0.25	-14.41	2.42	138.59
13	-0.22	-12.41	2.42	138.59
14	-0.18	-10.41	2.42	138.59
15	-0.15	-8.41	2.42	138.59
16	-0.11	-6.41	2.42	138.59
17	-0.08	-4.41	2.42	138.59
18	-0.04	-2.41	2.42	138.59
19	-0.01	-0.41	2.42	138.59
20	0.03	1.59	2.42	138.59
21	0.06	3.59	2.42	138.59
22	0.10	5.59	2.42	138.59
23	0.13	7.59	2.42	138.59
24	0.17	9.59	2.42	138.59
25	0.20	11.59	2.42	138.59

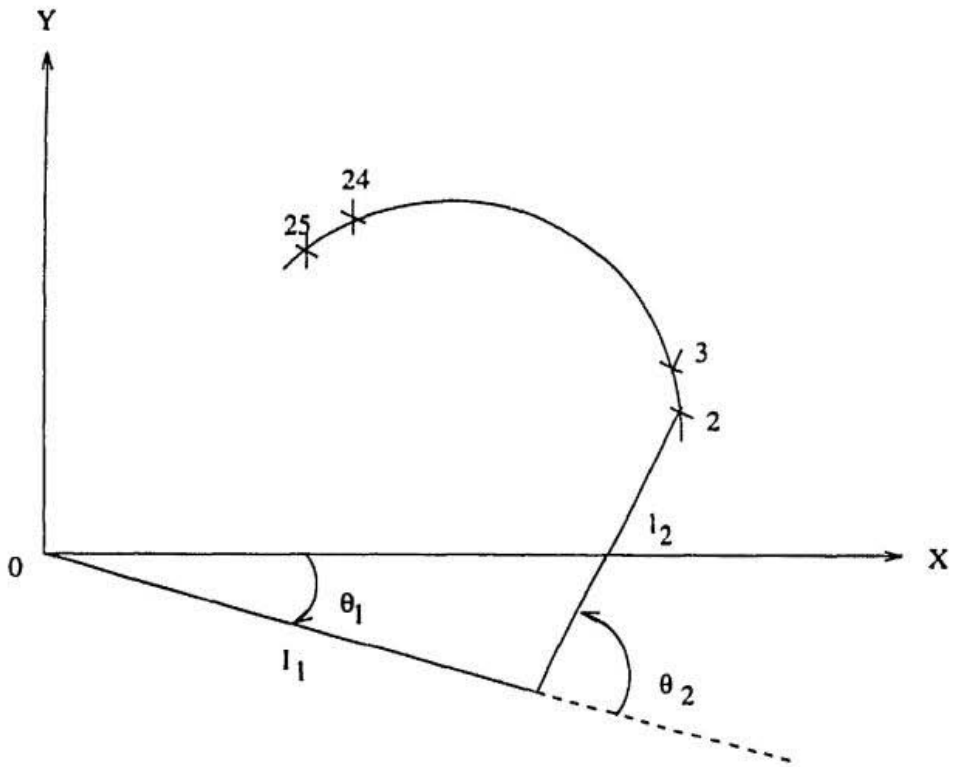

Fig. 3.4 DESIRED CARTESIAN TRAJECTORY ACCORDING TO TABLE 3.3

3.5 OPTIMAL CONTROL METHOD (NON - LINEAR OPTIMIZATION)

Optimal Control (non-linear optimization) method is a method used to achieve the trajectory control and obtain the position and velocity gain values on the off-line basis using the Hookes and Jeeves direct search non-linear optimization method. This method minimizes the error in the joint displacement and joint velocity values in accordance with Eq. (3.7). It makes use of the fourth order Runge-Kutta method (section 3.3.4) to calculate the actual joint displacement and velocity values at the next time step. The calculated gain values are then used for training the ANN as discussed in section 3.6. The various steps involved in the optimal control method are shown in Fig. 3.5. The various steps involved in this method are as follows:

STEP 1 : Read the link parameters (Table 3.2) and let the end-effector be at some initial position having coordinates (x,y).

STEP 2 : Use the inverse kinematic equations (Section 2.2.1) to calculate the actual joint position and joint velocity values.

STEP 3 : Calculate the joint torque using Eq. (3.7) based on the error in the desired and actual joint displacement and joint velocity values and thereafter compute the joint acceleration from Eq (3.6) It should be noted here that we use the actual values of θ, $\dot{\theta}$, etc. in Eq. (3.6)

STEP 4 : Numerically integrate forward in steps of ΔT and calculate the joint position and joint velocity values at the next time step.

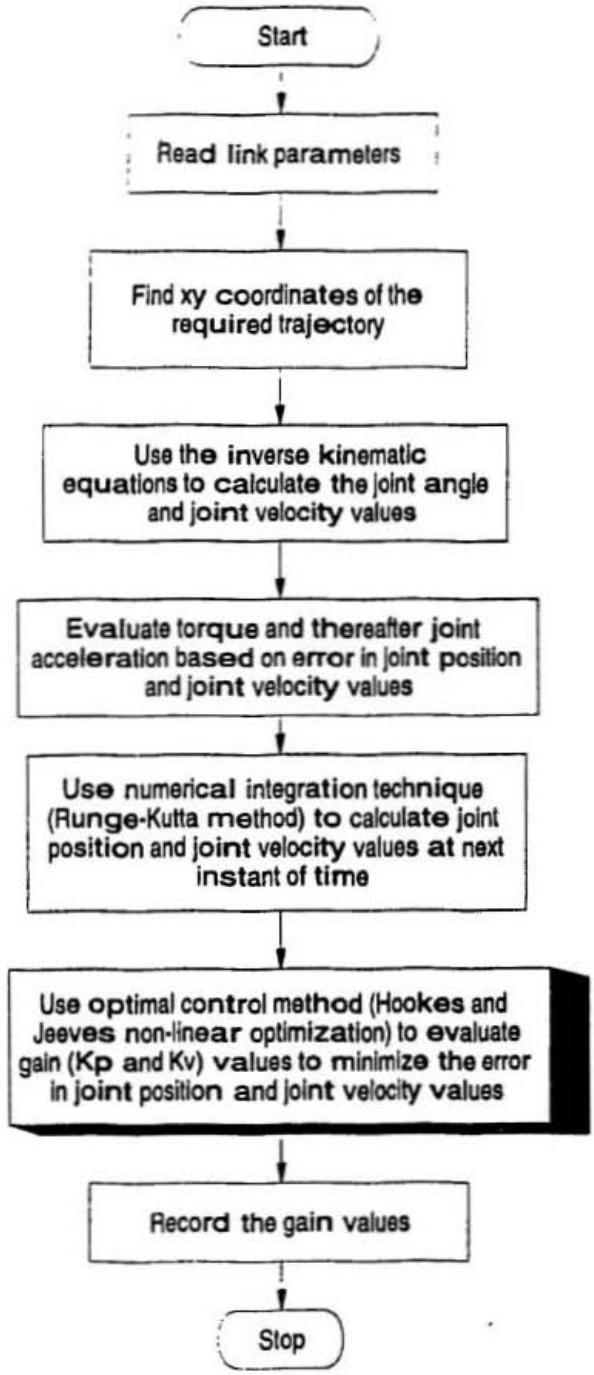

Fig. 3.5 FLOW CHART : OPTIMAL CONTROL METHOD (NON-LINEAR OPTIMIZATION)

STEP 5 : Finally use the Hookes and Jeeves direct search (non-linear optimization) method to minimize the error in the actual (obtained in Step 4) and the desired joint position and joint velocity values. Record the gain values. Here, we use $[K_p]$ and $[K_v]$ as design variables as shown in Eq. (3.10), and the objective function for the present case is given by Eqs. (3.11) and (3.12).

These gain values when substituted in Eq. (3.7) would result in the desired torque. The Steps 2 to 5 were repeated for the entire trajectory and a set of gain values was obtained. The objective here was to have the position control primarily, which was successfully attained. The gain values for a trajectory are shown in Figs. 3.10 to 3.13 (Section 3.6). Here, the gain values change quite significantly along the trajectory. This is because the matrices $[M(\Theta)]$, $\{V(\Theta,\dot{\Theta})\}$ and $\{G(\theta)\}$ undergo continuous change from position to position which also includes the changes in the velocity vector.

3.6 ARTIFICIAL NEURAL NETWORKS IN TRAJECTORY CONTROL

Neural network method has been widely used in many control applications. In the present case we are using the LP-neuro method which was found to be very effective in the parameter estimation problem discussed in Chapter 2. It would be quite beneficial to have a weight matrix which relates the input vector

$$\{I\} = \{\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2, e_1, e_2, \dot{e}_1, \dot{e}_2\}^T \qquad (3.42)$$

and the output vector

$$\{O\} = \{k_{p1}, k_{p2}, k_{v1}, k_{v2}\}^T \quad \textbf{(3.43)}$$

Several sets of {I} and {O} can be computed by starting with different initial conditions of θ_1, θ_2, $\dot{\theta}_1$ and $\dot{\theta}_2$ but the same trajectory as shown in Figs. 3.2 and 3.3. The training was done in such a way that different weight matrices were generated for all the 25 points along the trajectory. These weight matrices [W] were obtained from the 9 sets of {I} and {O} using the Optimal Control Method. The problem of trajectory control computationally becomes a lot simpler with the known weights [W]. The steps involved in obtaining the weight matrix as shown in Fig. 3.6 are as follows:

STEP 1 : Let the end-effector be at some initial position having coordinates (x,y).

STEP 2 : Use non-linear optimization method to evaluate gain values off-line for different trajectories.

STEP 3 : Note the input and output parameters for various trajectories.

STEP 4 : Compute [W] separately for each point along the trajectory using the LP-neuro method

STEP 5 : Use [W] on-line to evaluate the output parameters (gain values k_{p1}, k_{p2}, k_{v1}, k_{v2}, etc.).

Here, the weight matrix relates the input and output vectors as

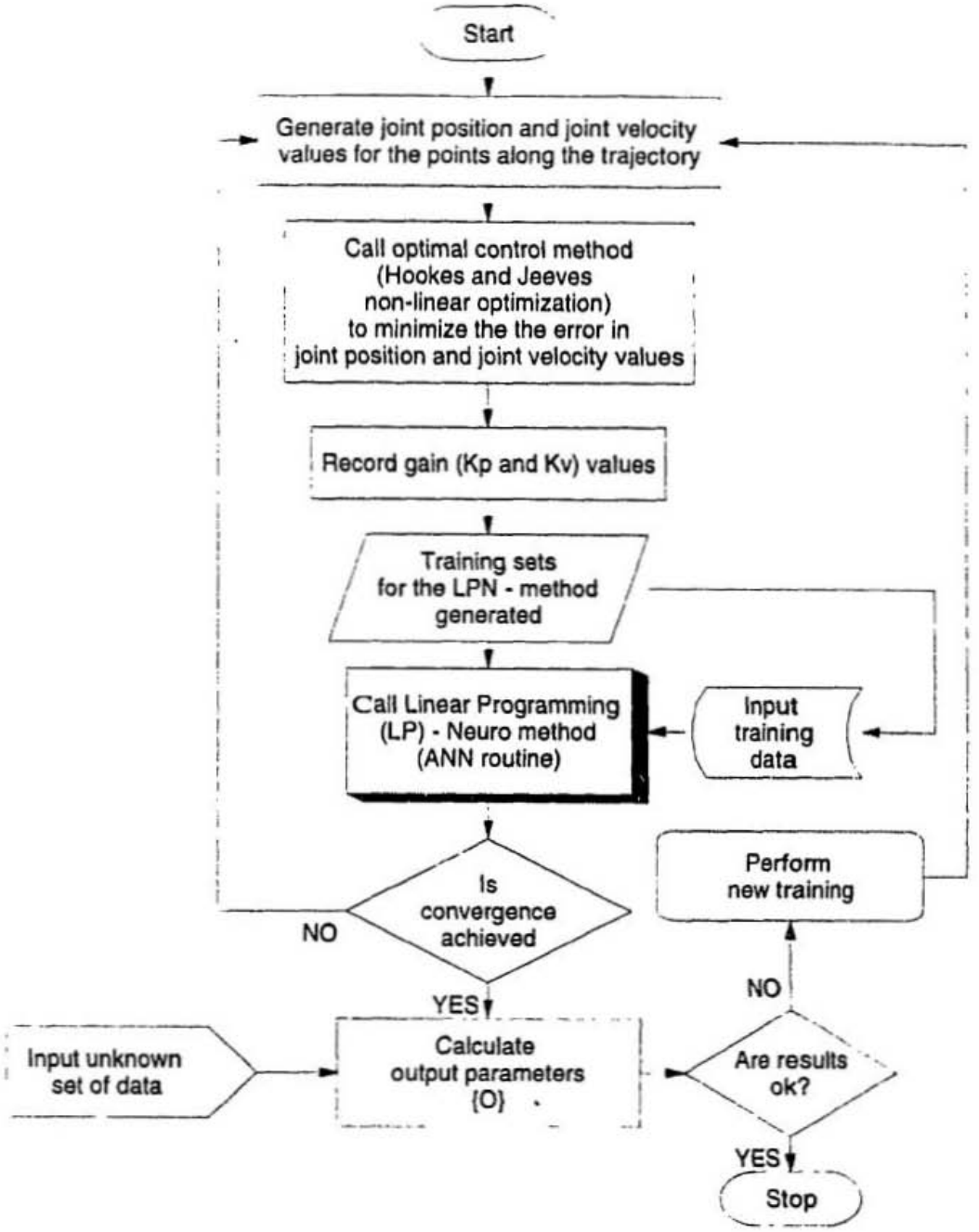

Fig. 3.6 FLOW CHART : LINEAR PROGRAMMING (LP -NEURO) METHOD (ANN METHOD)

$$\begin{Bmatrix} k_{p1} \\ k_{p2} \\ k_{v1} \\ k_{v2} \end{Bmatrix} = [W] \begin{Bmatrix} \theta_1 \\ \theta_2 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \\ e_1 \\ e_2 \\ \dot{e}_1 \\ \dot{e}_2 \end{Bmatrix} \quad \textbf{(3.44)}$$

Figs. 3.7 to 3.9 show the variation of the desired θ_1, θ_2 and $\dot{\theta}_1$ along the trajectory. Since, θ_2 has a constant value along the trajectory so, $\dot{\theta}_2$ is zero. Finally, knowing the weight matrix, [W], we fed a new set of input vector not used in the training {I}, to the trained neural network and obtained the corresponding output vector, {O} of gain values. Table 3.4 gives the gain values obtained using optimal control method and the ANN method corresponding to this new set of input values. These gain values have been compared graphically in Figs. 3.10 to 3.13. These results clearly show the applicability of neural network method for on-line control of robotic manipulators. After this a trajectory control was achieved which is shown in Fig. 3.14. The results in this figure very clearly demonstrate the use of both the optimal control and ANN control method. It should be stated here that in Fig. 3.14, the initial starting point was deliberately chosen to the different from the first desired point because such a situation can be expected in real practice. Having established the ability to such situations, the initial error was not introduced in subsequent work (Figs. 3.16 - 3.22).

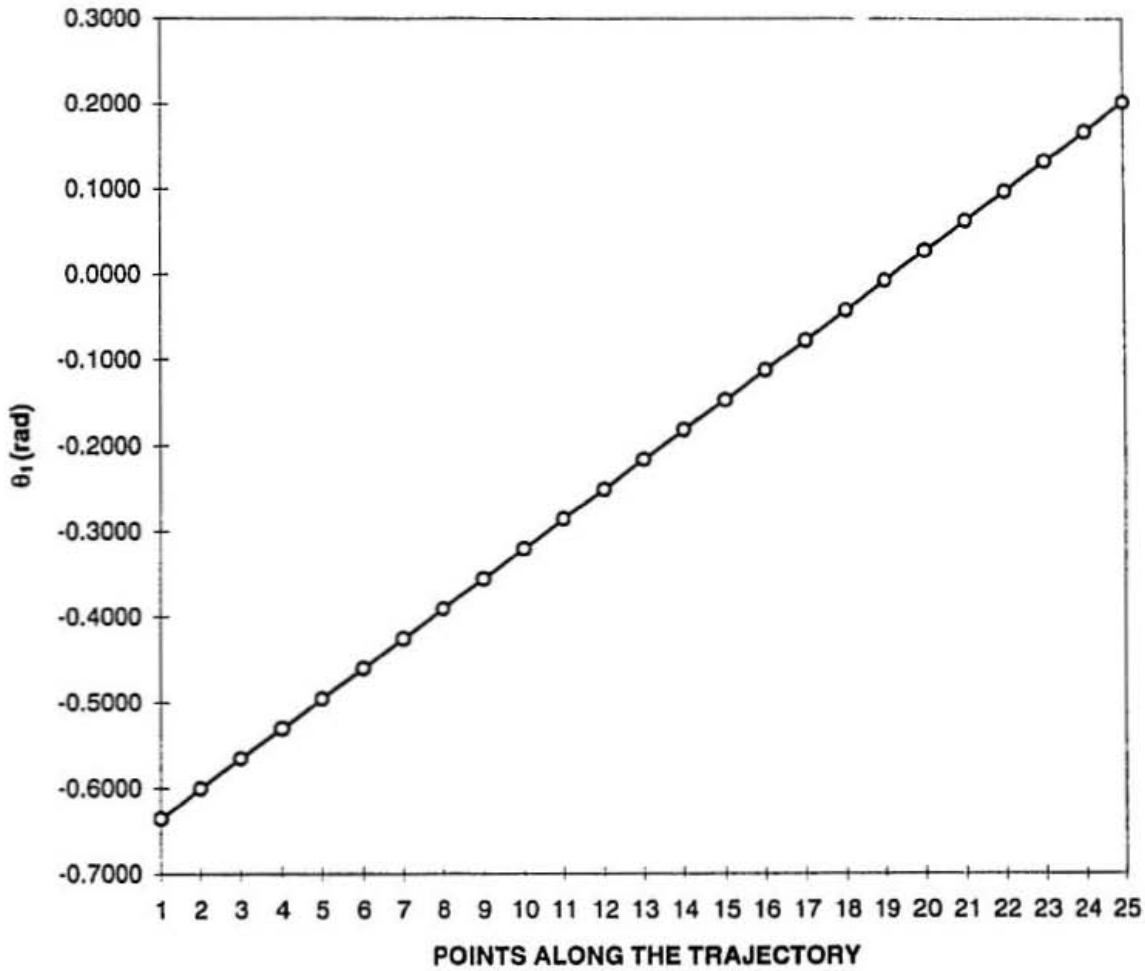

Fig. 3.7 VARIATION OF θ_1 ALONG THE TRAJECTORY

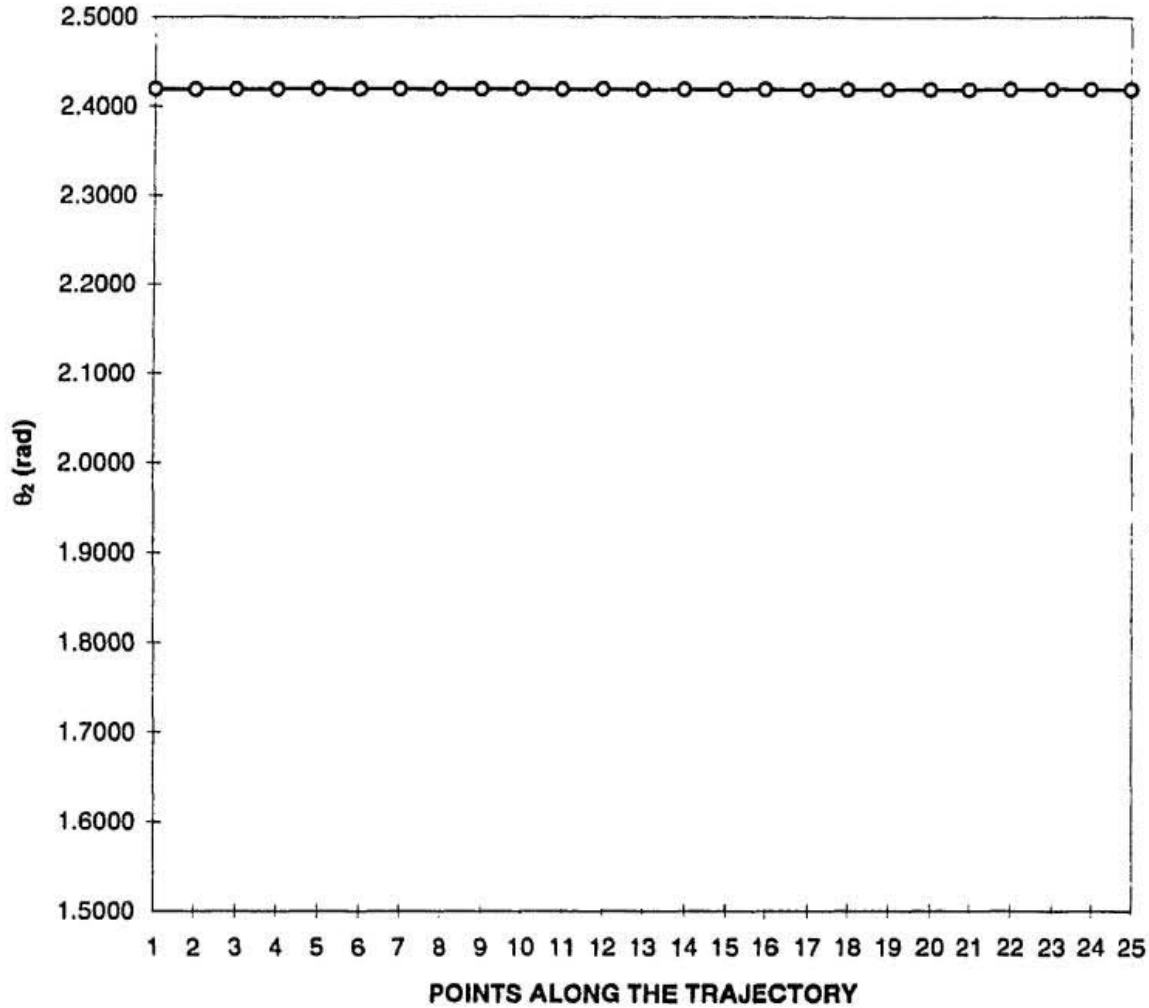

Fig. 3.8 VARIATION OF θ_2 ALONG THE TRAJECTORY

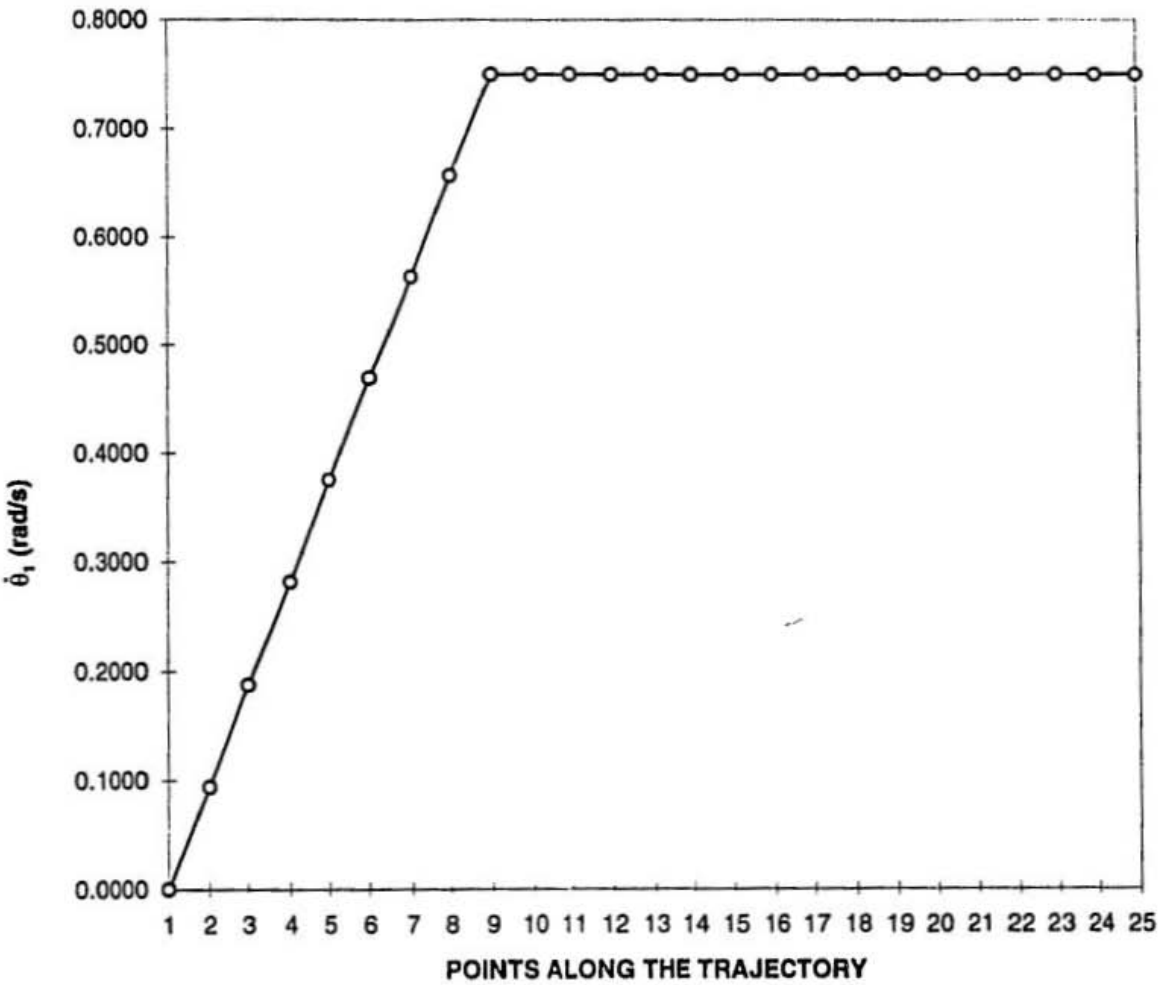

Fig. 3.9 VARIATION OF $\dot{\theta}_1$ ALONG THE TRAJECTORY

Table 3.4 COMPARISON OF GAIN VALUES OBTAINED USING OPTIMAL CONTROL (NON-LINEAR OPTIMIZATION) AND ANN METHOD

	NON-LINEAR OPTIMIZATION				LPN METHOD			
Points	Kp1	Kp2	Kv1	Kv2	Kp1	Kp2	Kv1	Kv2
2	1381.45	1325.24	1325.69	167.125	1381.45	1317.55	1320.36	167.125
3	1486.22	1358.2	1323.85	195.325	1486.28	1358.2	1323.85	195.325
4	1493.42	1361.79	1323.24	200.725	1493.42	1361.79	1323.24	200.725
5	1500.42	1366.99	1322.59	205.525	1498.69	1366.99	1322.59	205.523
6	1507.82	1371.39	1321.79	211.325	1507.82	1371.39	1321.79	211.325
7	1572.99	1424.76	1320.16	222.925	1573	1424.76	1320.16	222.925
8	1509.62	1373.99	1321.39	214.325	1509.62	1373.99	1320.88	214.325
9	1514.01	1376.98	1320.58	216.725	1514.02	1376.98	1320.58	216.725
10	1572.3	1413.86	1319.56	227.125	1572.3	1413.86	1319.56	227.125
11	1540.4	1397.77	1320.77	214.525	1540.4	1397.77	1320.77	214.525
12	1609.48	1460.95	1320.25	218.926	1609.48	1460.95	1320.25	214.41
13	1709.64	1530.91	1319.61	223.127	1709.64	1530.91	1319.61	223.127
14	1706.05	1525.71	1318.81	226.927	1706.05	1525.71	1318.81	226.927
15	1703.4	1524.32	1318.72	227.127	1703.4	1524.32	1318.72	227.127
16	1709.44	1539.11	1321.21	219.127	1709.45	1539.11	1321.21	219.127
17	1710.64	1542.11	1321.21	220.727	1710.64	1542.11	1321.21	220.727
18	1697.45	1542.72	1320.82	221.527	1697.45	1542.72	1320.82	221.527
19	1685.05	1535.12	1318.22	236.526	1685.05	1535.12	1318.22	236.526
20	1677.31	1515.18	1320.08	228.677	1677.31	1515.18	1320.08	228.677
21	1696.65	1527.71	1320.01	226.327	1696.65	1527.71	1320.01	226.327
22	1611.27	1460.54	1319.24	219.327	1611.27	1460.54	1319.24	219.327
23	1722.44	1547.7	1320.8	214.927	1722.44	1547.7	1320.8	214.927
24	1735.83	1564.3	1320.2	221.727	1735.83	1564.3	1320.2	221.727
25	1690.64	1517.91	1319.41	222.527	1690.64	1517.91	1319.41	222.527

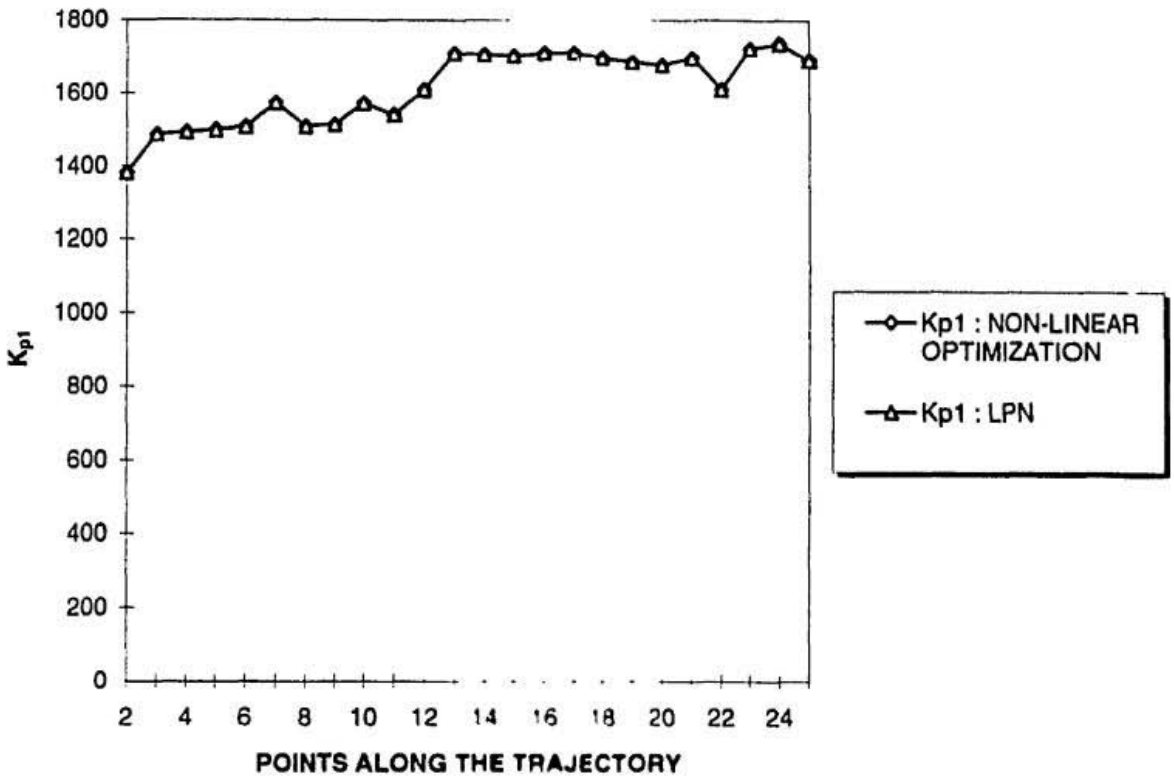

Fig. 3.10 COMPARISON OF K_{p1} VALUES OBTAINED USING OPTIMAL CONTROL AND ANN METHOD

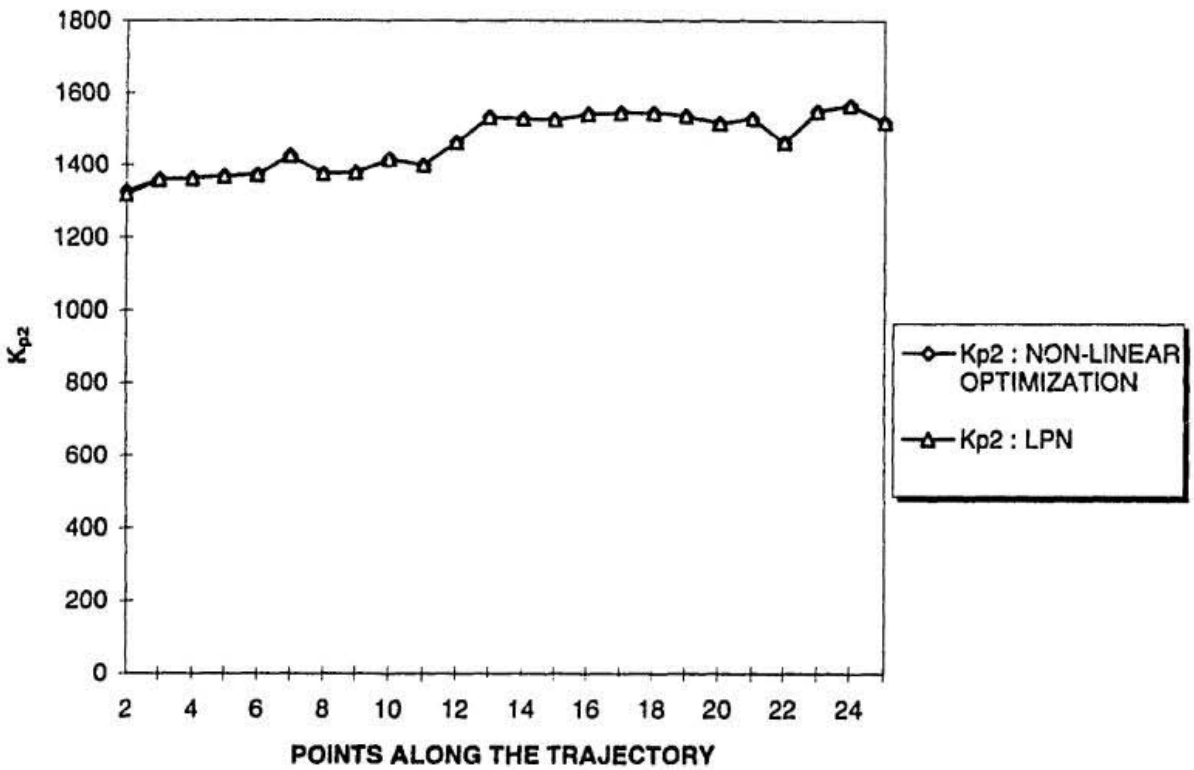

Fig. 3.11 COMPARISON OF K_{p2} VALUES OBTAINED USING OPTIMAL CONTROL AND ANN METHOD

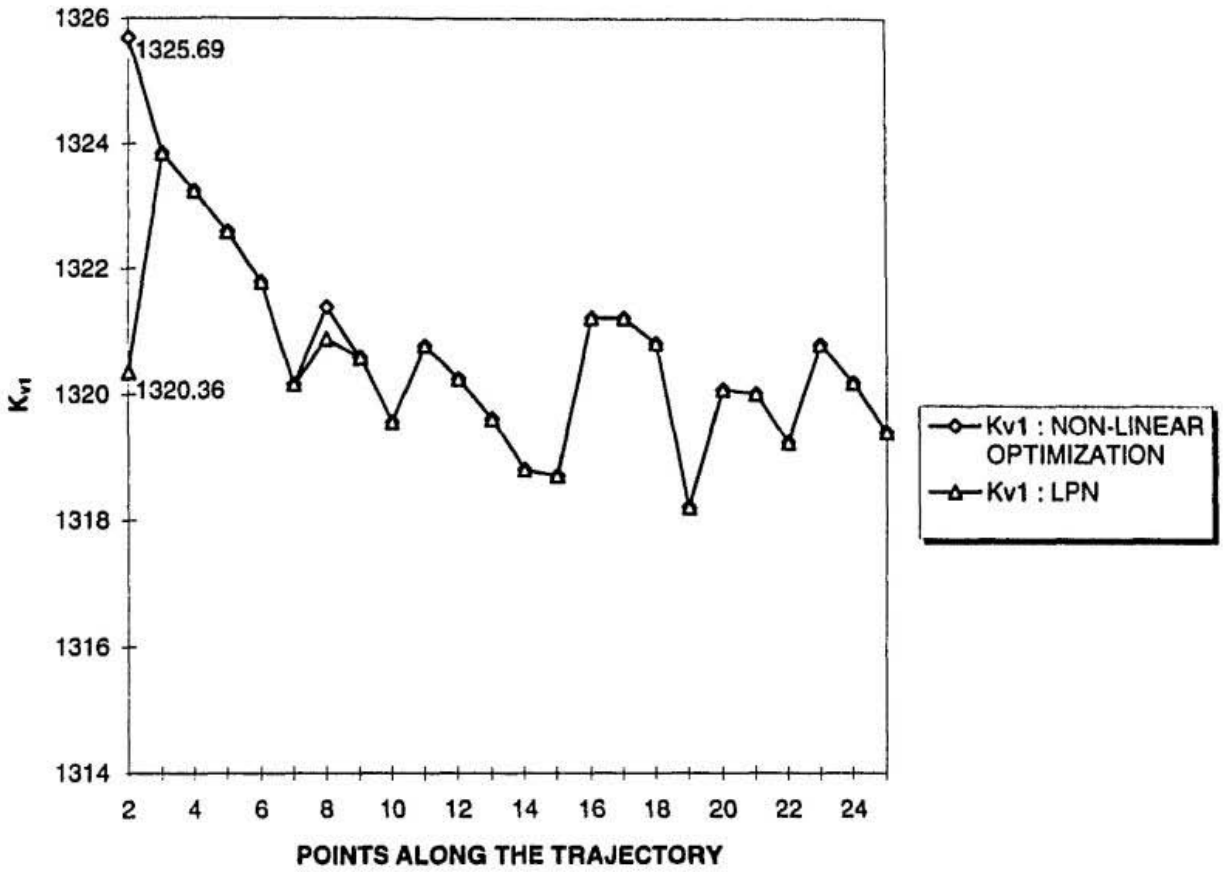

Fig. 3.12 COMPARISON OF K_{v1} VALUES OBTAINED USING OPTIMAL CONTROL AND ANN METHOD

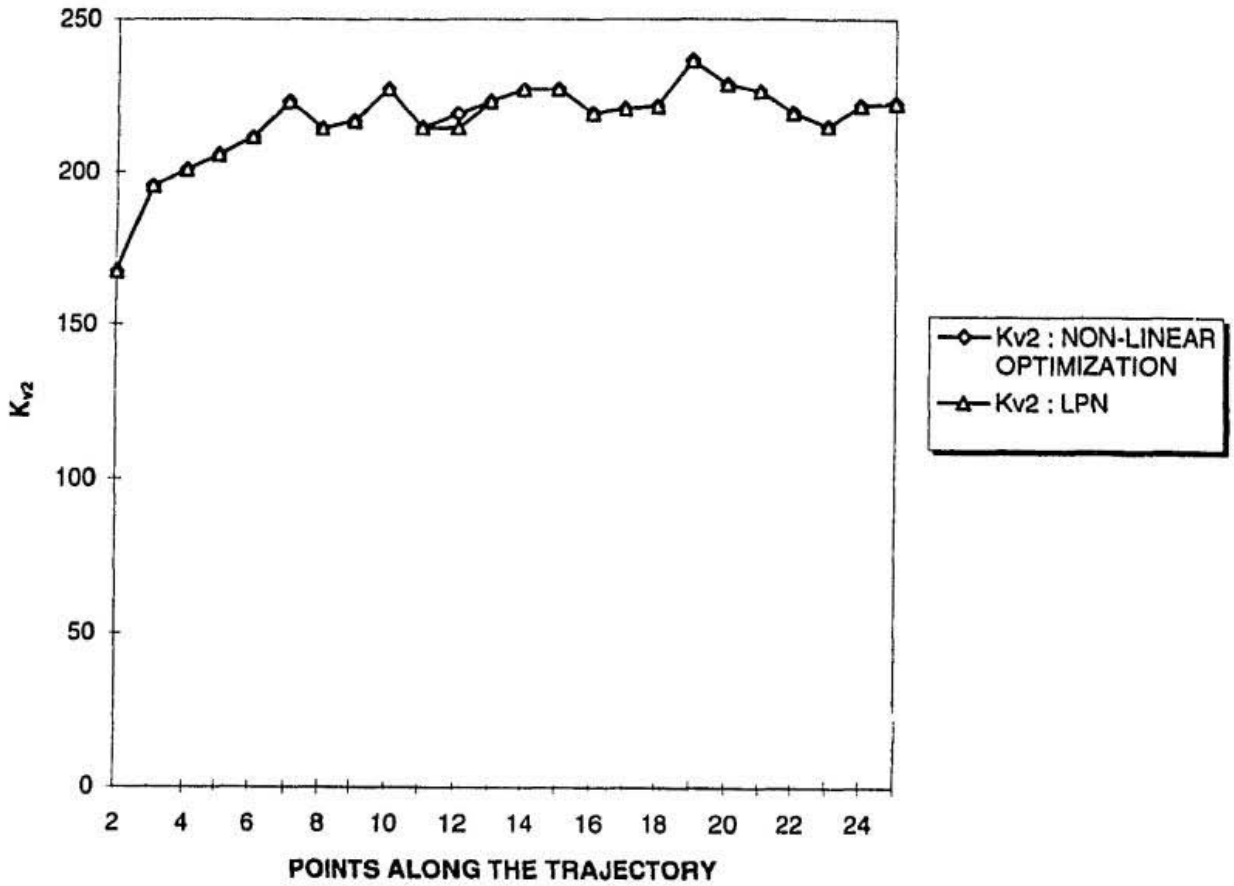

Fig. 3.13 COMPARISON OF K_{v2} VALUES OBTAINED USING OPTIMAL CONTROL AND ANN METHOD

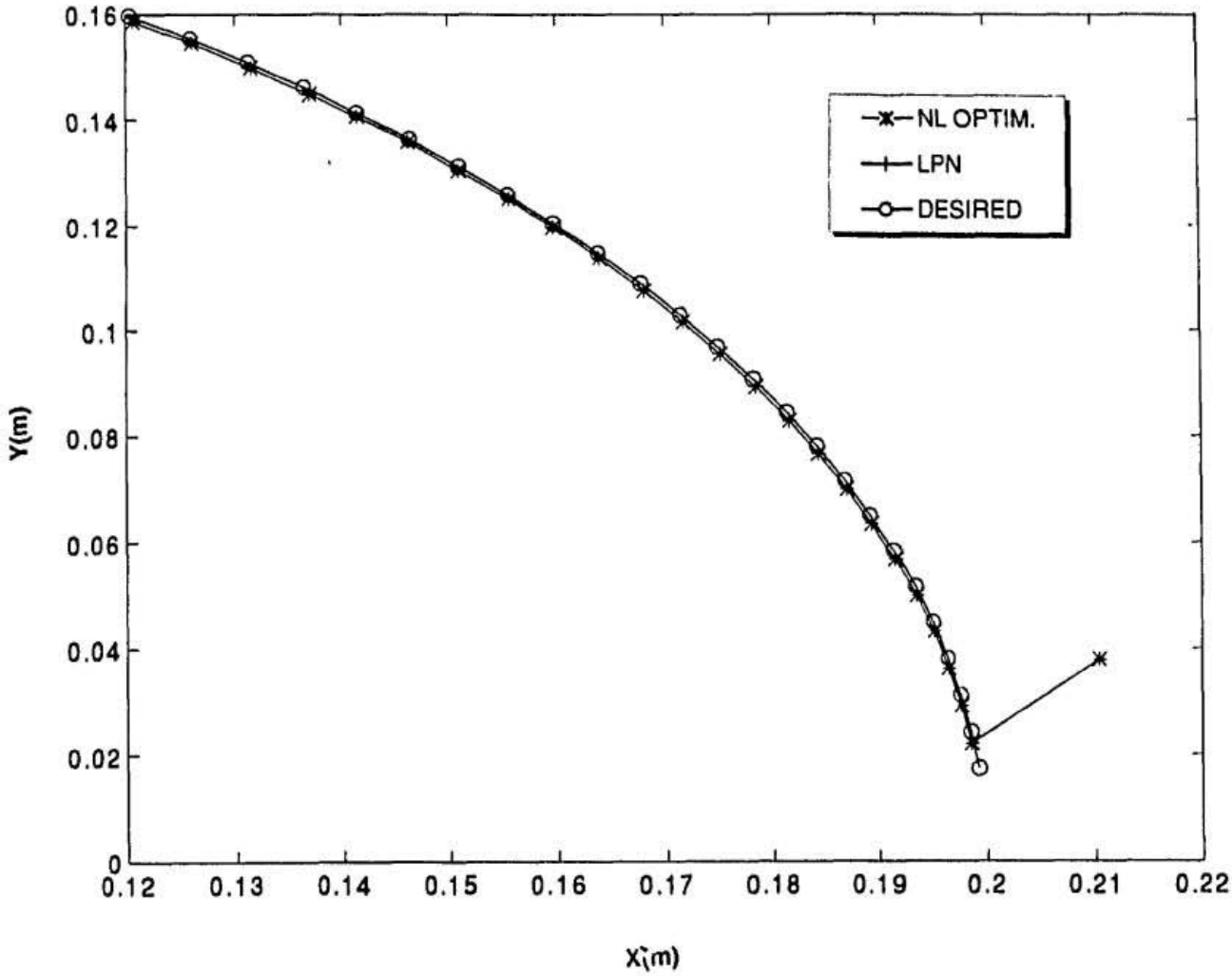

Fig. 3.14 TRAJECTORY CONTROL OBTAINED USING OPTIMAL CONTROL AND ANN METHOD

3.7 THE EFFECT OF VARIATION OF MAXIMUM TANGENTIAL VELOCITY

The effect of variation of maximum tangential velocity for the given trajectory is studied in this section. For the desired trajectory (Fig. 3.2) the maximum value of the desired tangential velocity was varied according to Table 3.5. Fig. 3.15 shows the variation of the maximum tangential velocity for some of the values according to Table 3.5. The variation of the maximum velocity values of 0.001 m/s and 0.0008466 m/s have not been shown in Fig. 3.15 but they follow the same trend. In this figure the effect of sharp and smooth change in velocity is made clear. Using these maximum tangential velocity values and the link parameters given in Table 3.2, the desired joint displacement and joint velocity values were computed. Then, following the optimal control method using a fixed starting value of θ_1, θ_2, $\dot{\theta}_1$ and $\dot{\theta}_2$ as discussed in Section 3.5, the corresponding actual joint displacement and joint velocity values are calculated. Corresponding to these velocity values the desired and actual trajectory variation is given in Figs. 3.16 to 3.22.

Figs. 3.16 to 3.22 indicate that the desired and actual (Optimal Control Method) trajectories become closer as the maximum value of the tangential velocity decreases. Also, from the figures and Table 3.5 it is obvious that the smoothing of the corner i.e. with gradual change in the velocity from accelerating to the constant velocity region, gives better results for trajectory control.

Table 3.5 VARIOUS MAXIMUM TANGENTIAL VELOCITY VALUES

S.No.	MAXIMUM TANGENTIAL VELOCITY (m/s)	MAXIMUM TANGENTIAL VELOCITY (inch/min)	PROPERTY
1	0.15	354.6	Sharp Corner
2	0.15	354.6	Smooth Corner
3	0.1	236.4	Sharp Corner
4	0.05	118.2	Sharp Corner
5	0.001	2.36	Sharp Corner
6	0.0008466	2.0	Sharp Corner
7	0.0008466	2.0	Smooth Corner

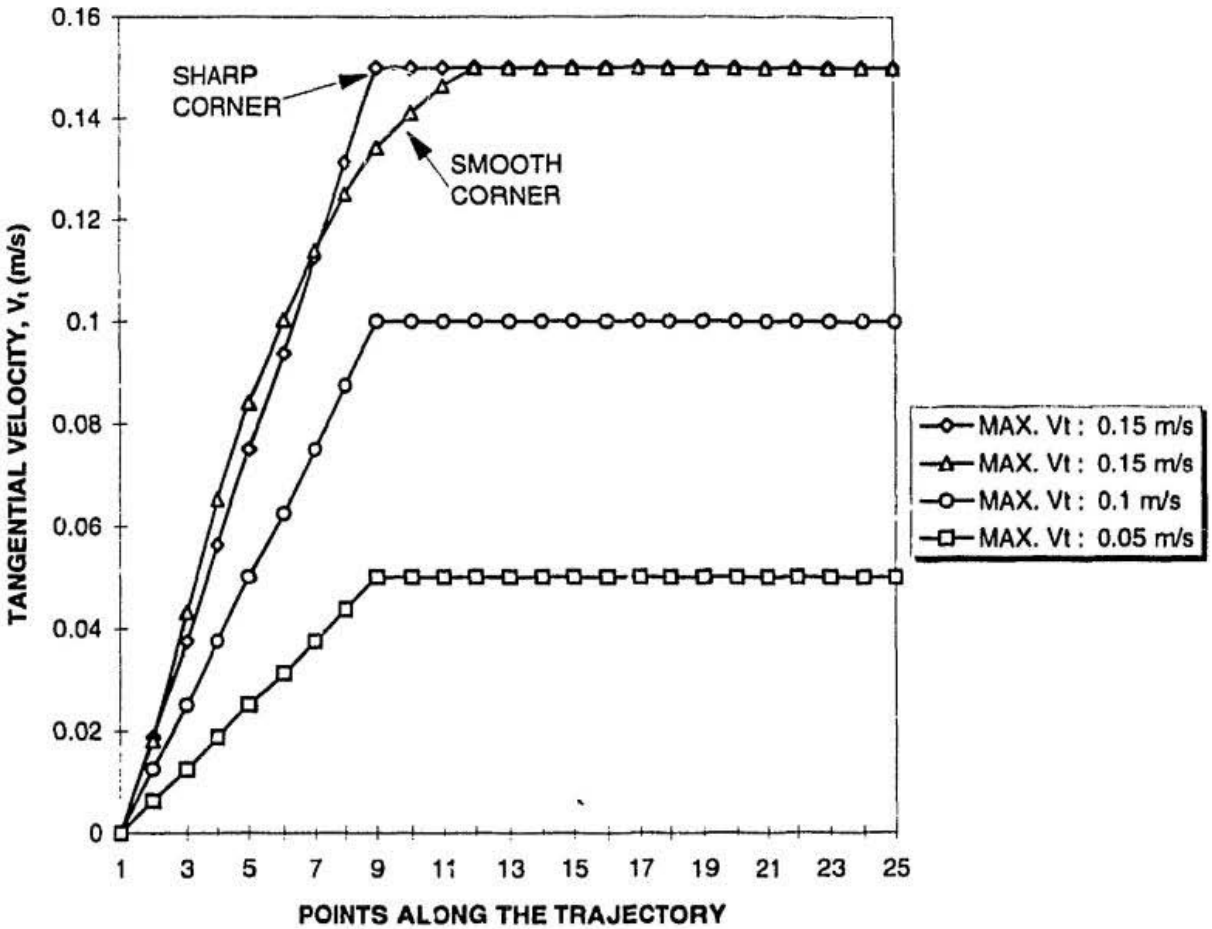

Fig. 3.15 VELOCITY VARIATION ACCORDING TO TABLE 3.5

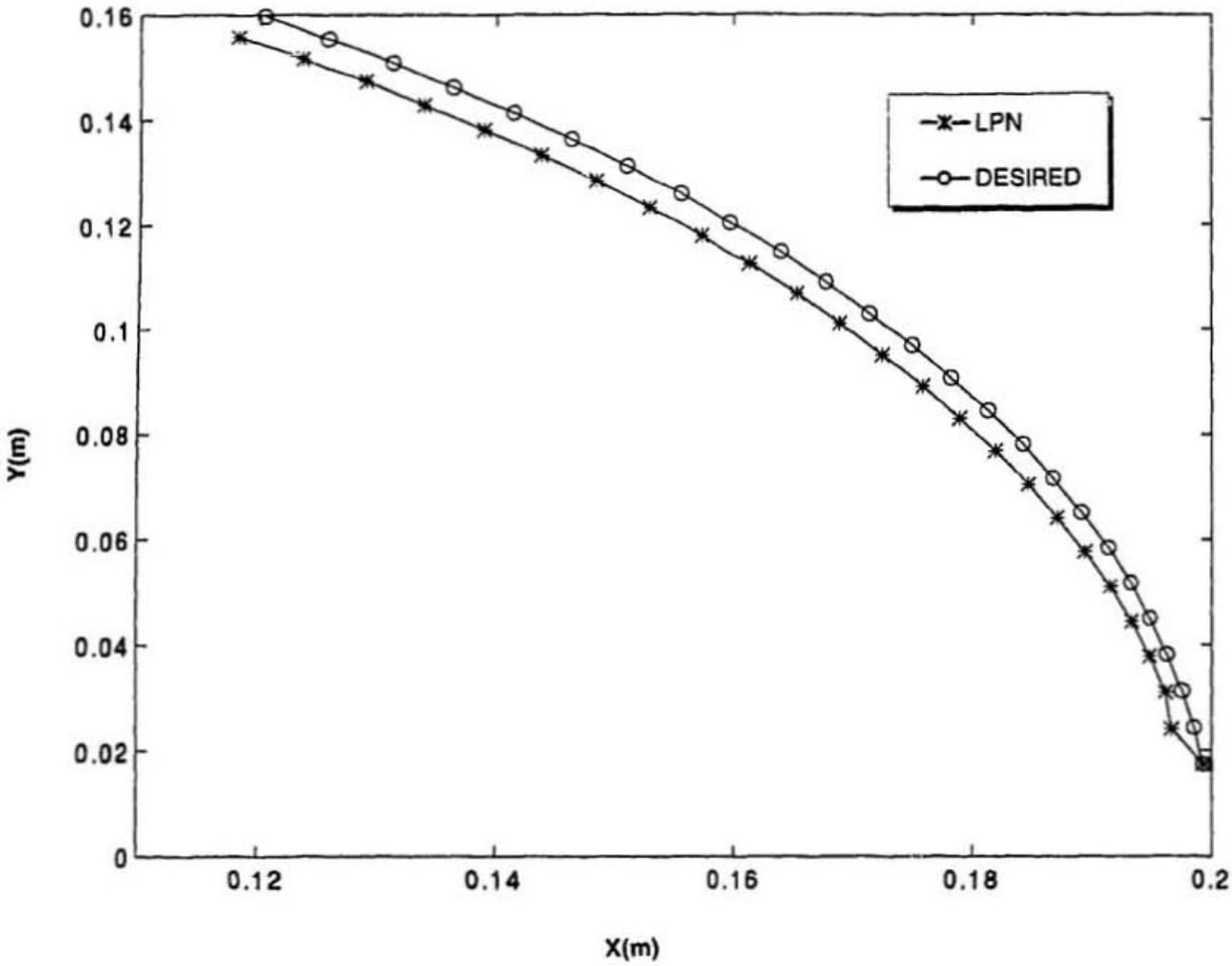

Fig. 3.16 DESIRED TRAJECTORY AND THE TRAJECTORY OBTAINED USING ANN METHOD WHEN THE MAXIMUM VELOCITY IS 0.15 m/s

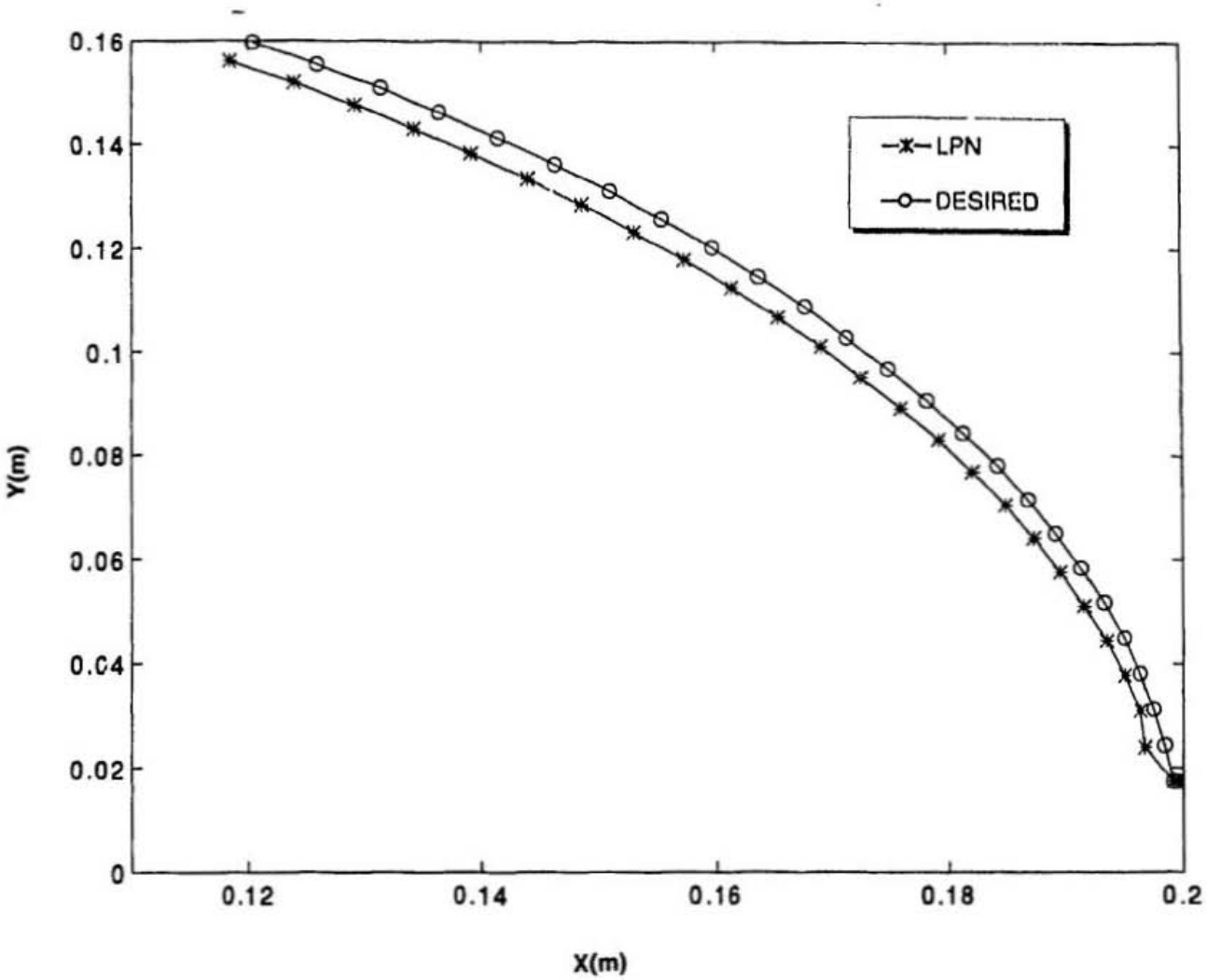

Fig. 3.17 DESIRED TRAJECTORY AND THE TRAJECTORY OBTAINED USING ANN METHOD WHEN THE MAXIMUM VELOCITY IS 0.15 m/s (SMOOTH CORNER)

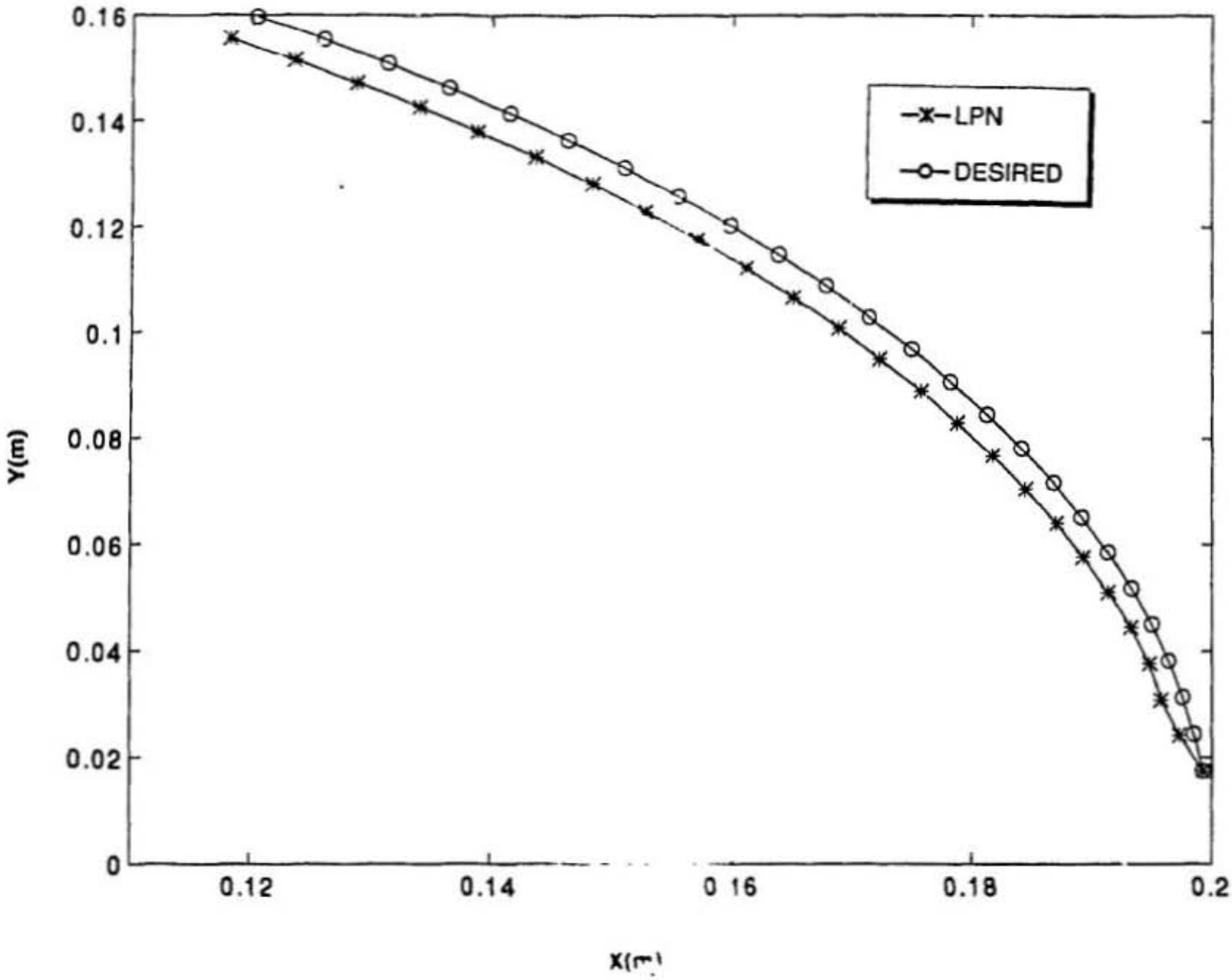

Fig. 3.18 DESIRED TRAJECTORY AND THE TRAJECTORY OBTAINED USING ANN METHOD WHEN THE MAXIMUM VELOCITY IS 0.1 m/s

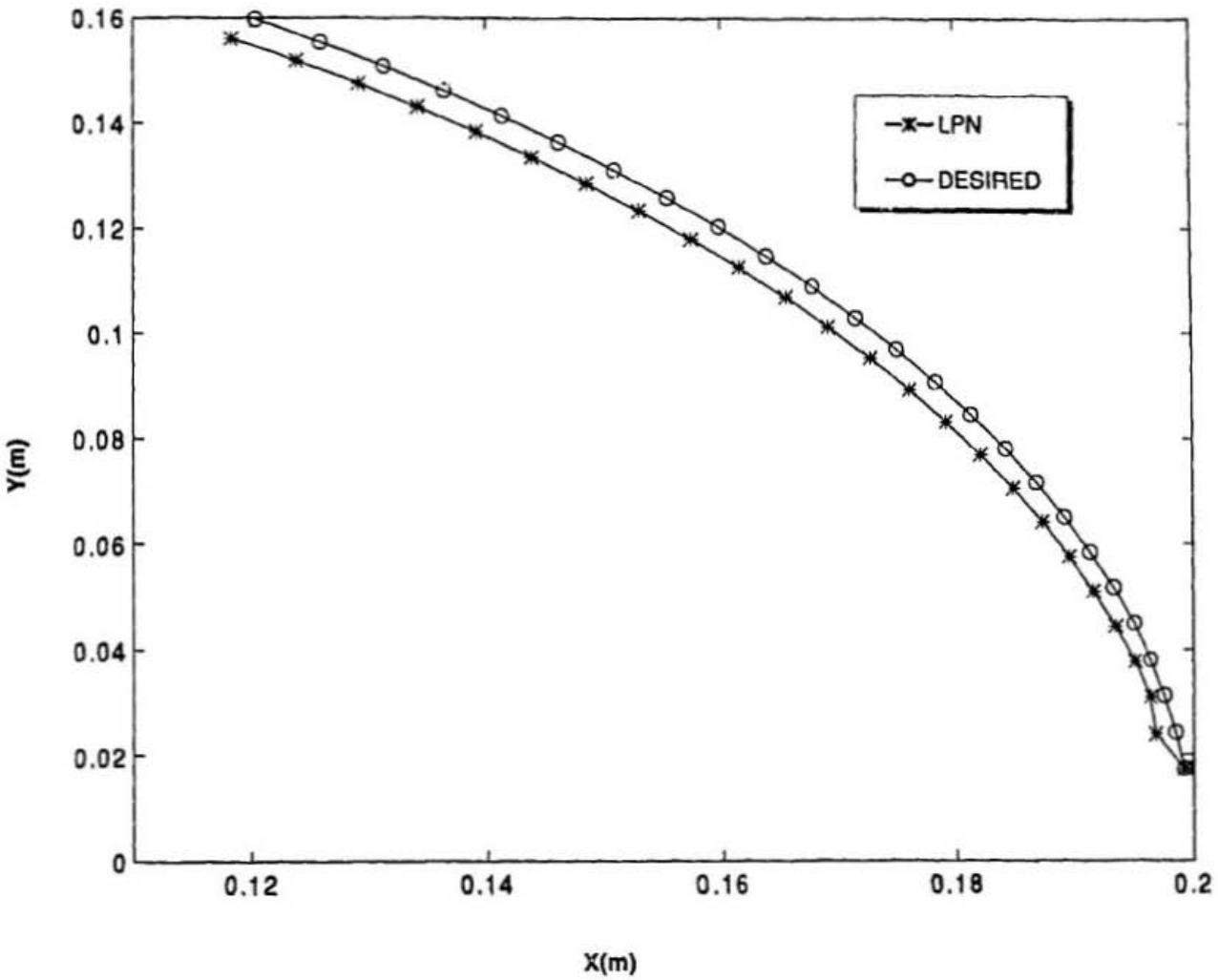

Fig. 3.19 DESIRED TRAJECTORY AND THE TRAJECTORY OBTAINED USING ANN METHOD WHEN THE MAXIMUM VELOCITY IS 0.05 m/s

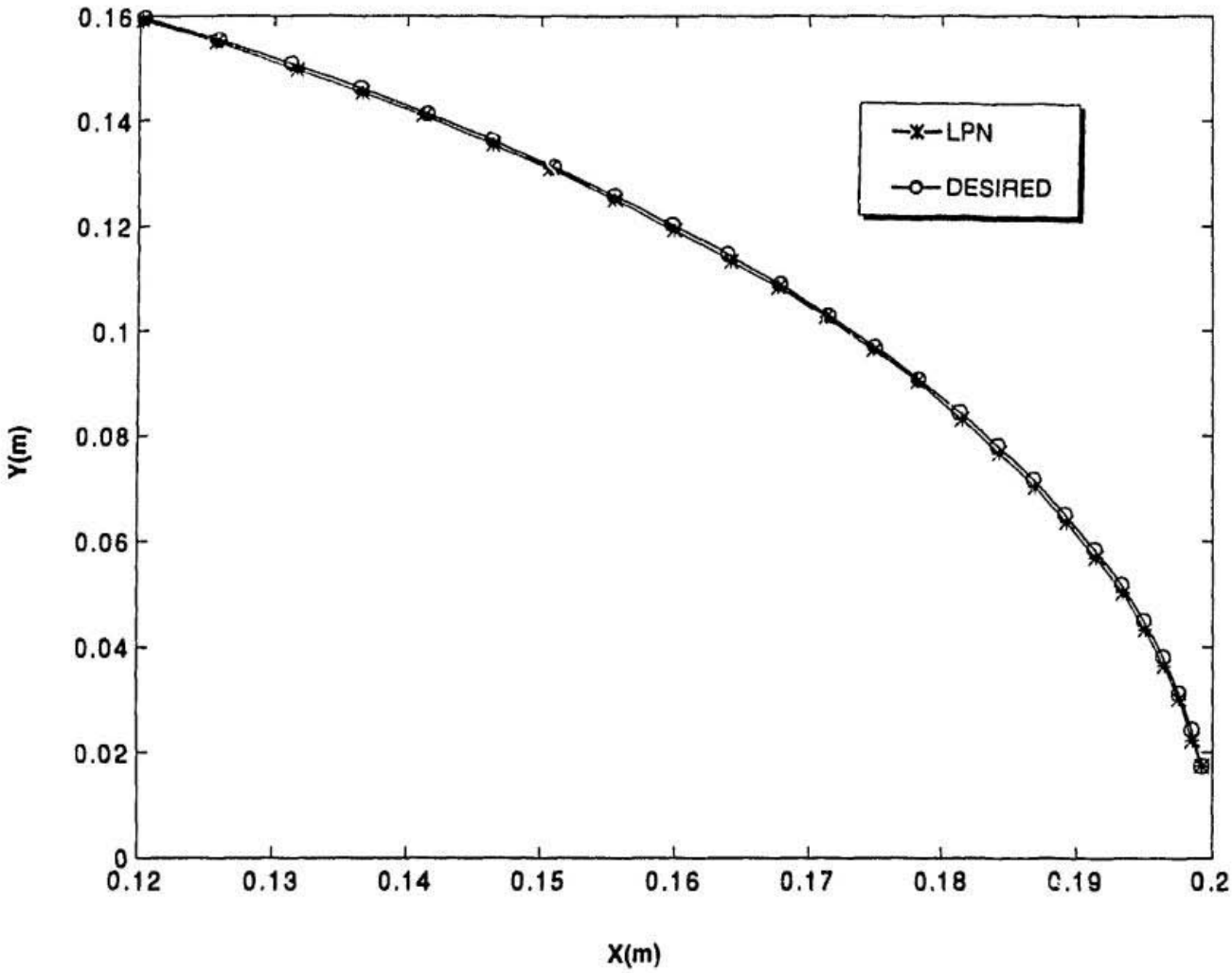

Fig. 3.20 DESIRED TRAJECTORY AND THE TRAJECTORY OBTAINED USING ANN METHOD WHEN THE MAXIMUM VELOCITY IS 0.001 m/s

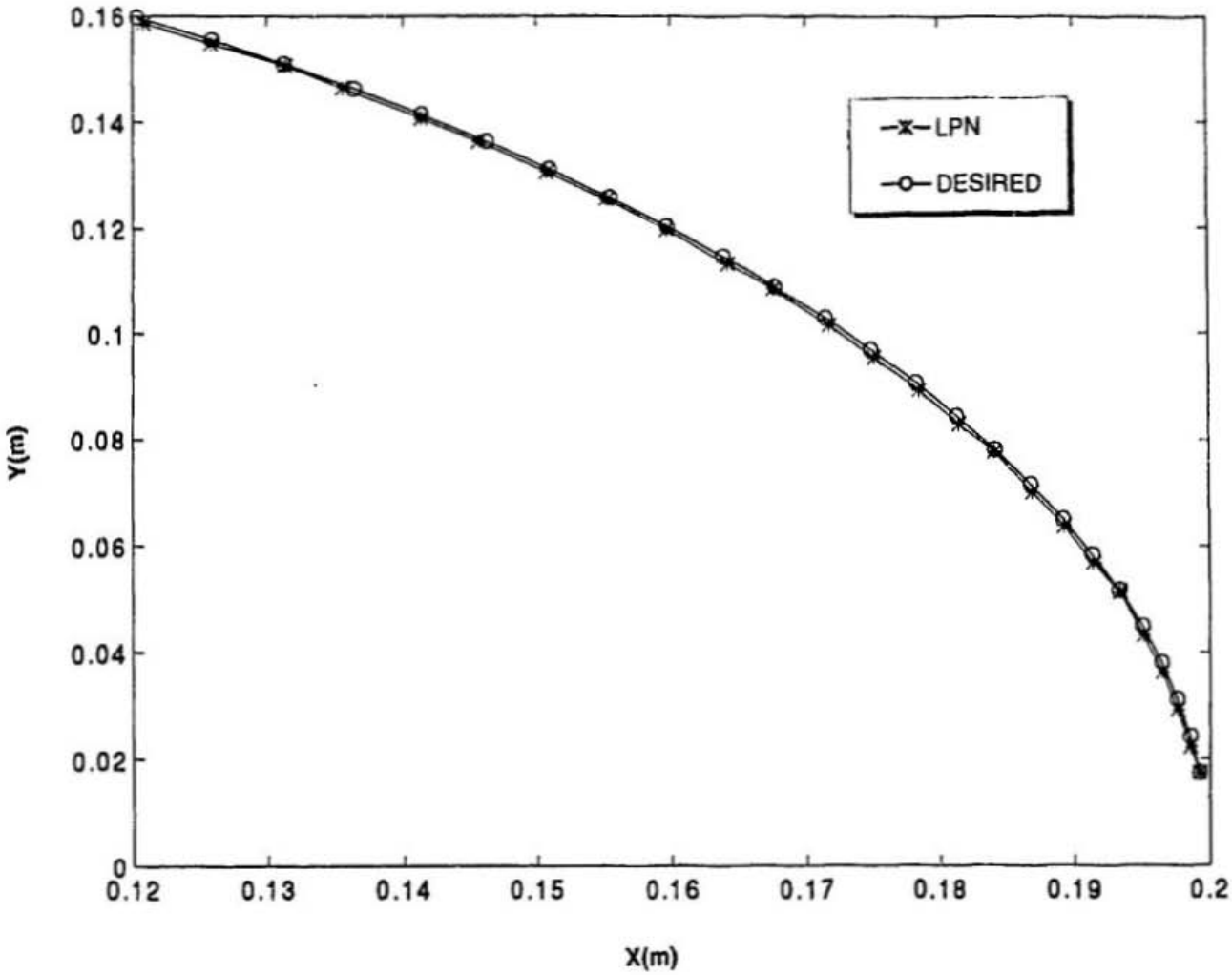

Fig. 3.21 DESIRED TRAJECTORY AND THE TRAJECTORY OBTAINED USING ANN METHOD WHEN THE MAXIMUM VELOCITY IS 2 inch/min (0.0008466 m/s)

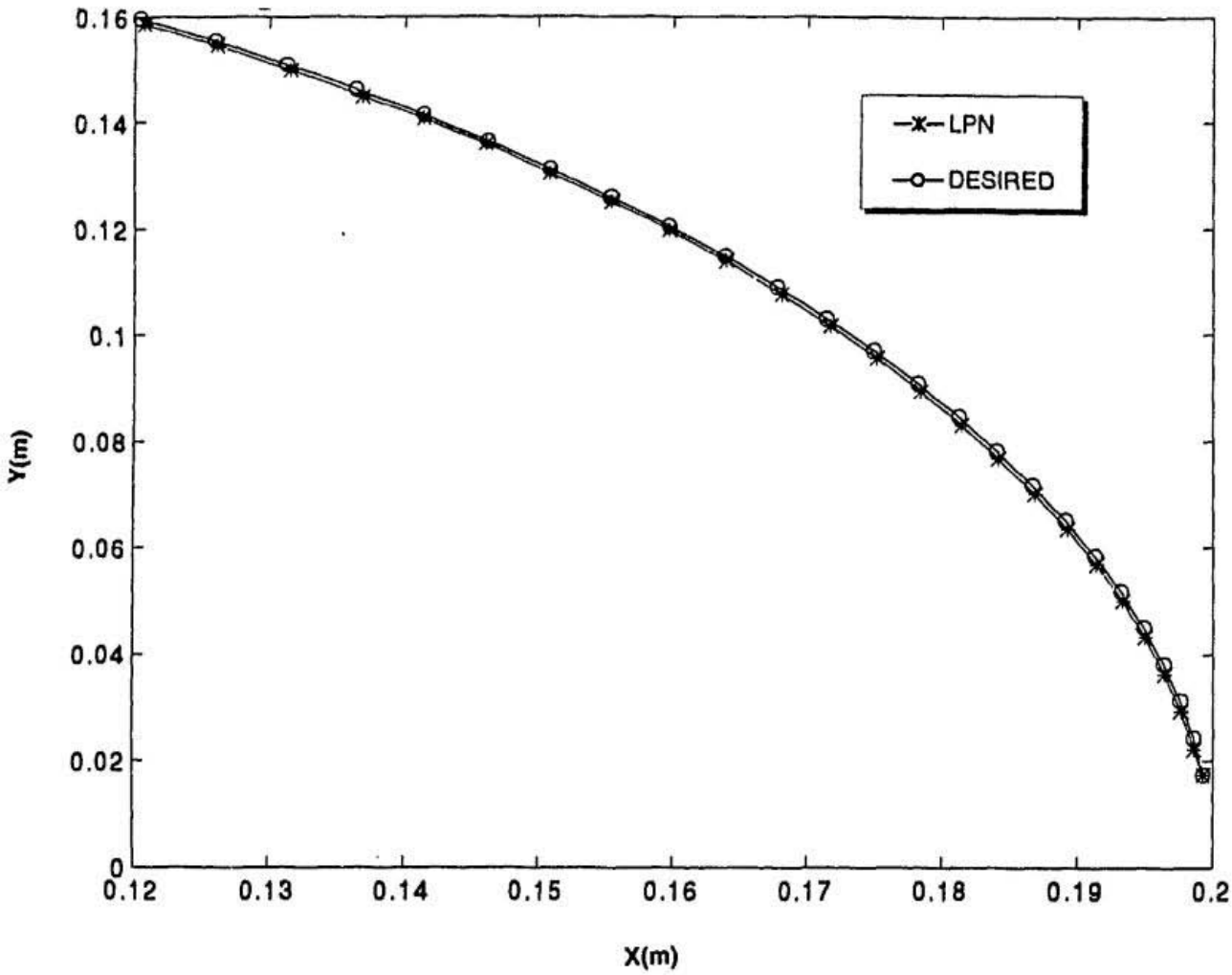

Fig. 3.22 DESIRED TRAJECTORY AND THE TRAJECTORY OBTAINED USING ANN METHOD WHEN THE MAXIMUM VELOCITY IS 2 inch/min (0.0008466 m/s : SMOOTH CORNER)

3.8 THE GENERAL CONTROL LAW

In the previous sections we have achieved the trajectory control of a two link manipulator where torque equation was given by Eq. (3.7). The computations for the position and the velocity gain values using the optimal control method were done taking the difference between the desired and actual joint position and joint velocity values in the control law. During this non-linear optimization control (Section 3.5) we had four design variables k_{p1}, k_{p2}, k_{v1} and k_{v2}.

In this section we use the general control law for the torque equation as

$$\{\tau\} = [K_p] \{e\}^{N_1} + [K_v] \{\dot{e}\}^{N_2} \qquad (3.45)$$

Here N_1 and N_2 are the exponents of the error in joint position and joint velocity values respectively. Using this as the torque equation and then following the optimal control method discussed in Section 3.5, we achieved the trajectory control and recorded the position and velocity gain values. In this general control law, while using the optimal control method, we had six design variables as k_{p1}, k_{p2}, k_{v1}, k_{v2}, N_1 and N_2. Tables 3.6 and 3.7 show optimal values for the design variables at each of the points along the trajectory. In Table 3.8 U(4) and U(6) refer to the objective function for the four and six design variable control laws respectively. Corresponding to the four and six design variable cases the optimal objective function values (Eqs. (3.11) and (3.12)) are given in Table 3.8. The bottom row of Table 3.8 shows that the sum of the objective function values at all the points for the general control law is less than that of the four design variable control law. The normalized errors in the objective function values is plotted in Fig. 3.23. The

Table 3.6 GAIN VALUES OBTAINED USING OPTIMAL CONTROL METHOD WITH 4 DESIGN VARIABLES

Points	Kp1	Kp2	Kv1	Kv2
2	1381.45	1325.24	1325.69	167.125
3	1486.22	1358.2	1323.85	195.325
4	1493.42	1361.79	1323.24	200.725
5	1500.42	1366.99	1322.59	205.525
6	1507.82	1371.39	1321.79	211.325
7	1572.99	1424.76	1320.16	222.925
8	1509.62	1373.99	1321.39	214.325
9	1514.01	1376.98	1320.58	216.725
10	1572.3	1413.86	1319.56	227.125
11	1540.4	1397.77	1320.77	214.525
12	1609.48	1460.95	1320.25	218.926
13	1709.64	1530.91	1319.61	223.127
14	1706.05	1525.71	1318.81	226.927
15	1703.4	1524.32	1318.72	227.127
16	1709.44	1539.11	1321.21	219.127
17	1710.64	1542.11	1321.21	220.727
18	1697.45	1542.72	1320.82	221.527
19	1685.05	1535.12	1318.22	236.526
20	1677.31	1515.18	1320.08	228.677
21	1696.65	1527.71	1320.01	226.327
22	1611.27	1460.54	1319.24	219.327
23	1722.44	1547.7	1320.8	214.927
24	1735.83	1564.3	1320.2	221.727
25	1690.64	1517.91	1319.41	222.527

Table 3.7 GAIN VALUES OBTAINED USING OPTIMAL CONTROL METHOD FOR THE GENERAL CONTROL LAW

Points	Kp1	Kp2	Kv1	Kv2	N1	N2
2	1388.4030	1312.9910	1327.3880	141.8002	2.0354	2.0342
3	1609.9450	1378.4970	1324.8940	191.3996	2.0982	2.0808
4	1435.9910	1333.5750	1326.5690	156.8000	2.0582	2.0542
5	1438.7900	1335.9740	1326.5680	158.8001	2.0602	2.0560
6	1452.5840	1343.7680	1325.5620	167.0000	2.0674	2.0632
7	1455.5830	1346.1670	1325.5610	168.0000	2.0694	2.0650
8	1459.3810	1348.9650	1325.1590	170.2001	2.0710	2.0666
9	1489.5730	1374.1560	1323.3480	190.7998	2.0846	2.0796
10	1492.5720	1377.1550	1323.1480	193.3998	2.0872	2.0822
11	1528.5590	1402.5400	1322.3330	201.1998	2.1046	2.0974
12	1530.9580	1404.7390	1322.1320	201.9998	2.1064	2.0994
13	1564.7430	1431.1250	1320.7160	213.9998	2.1308	2.1202
14	1567.1420	1433.1230	1320.5150	214.7998	2.1326	2.1220
15	1569.9410	1435.4220	1320.3130	214.7999	2.1344	2.1238
16	1649.1150	1485.7940	1320.8850	218.0000	2.1808	2.1658
17	1655.1120	1504.7920	1321.2830	222.9999	2.1872	2.1726
18	1648.5120	1499.7930	1320.6840	221.7999	2.1886	2.1734
19	1634.5190	1485.7980	1317.6900	235.3997	2.1808	2.1662
20	1643.9140	1487.3940	1319.0860	229.3998	2.1858	2.1714
21	1638.3170	1492.9970	1319.2880	229.7998	2.1808	2.1674
22	1672.9070	1514.9870	1320.0770	227.1998	2.1972	2.1866
23	1617.5220	1482.2030	1319.6930	215.0000	2.1654	2.1504
24	1618.7210	1483.4020	1319.6920	216.2000	2.1662	2.1520
25	1622.7190	1486.6500	1319.0400	220.3003	2.1694	2.1544

Table 3.8 COMPARISON OF OBJECTIVE FUNCTION VALUES OBTAINED USING 4 DESIGN VARIABLE AND GENERAL CONTROL LAW

POINTS	U(4)	U(6)	U(6)-U(4)
2	11545.81	11462.97	-82.84
3	9083.952	2926145	2917061.048
4	8271.521	8915.702	644.181
5	7605.531	8228.428	622.897
6	7084.709	7595.758	511.049
7	6176.813	7174.384	997.571
8	6420.974	6851.461	430.487
9	6201.063	6380.96	179.897
10	708232.3	6198.987	-702033.313
11	5759.045	5826.917	67.872
12	5237.813	5751.221	513.408
13	4696.467	5584.335	887.868
14	4774.192	5609.491	835.299
15	4813.048	5617.618	804.57
16	6877855	4038835	-2839020
17	9250850	1.28E+07	3549150
18	7751353	7563552	-187801
19	7670226	2700556	-4969670
20	1.22E+07	5869258	-6330742
21	9153188	8427011	-726177
22	5065.401	1.16E+07	11594934.6
23	3738899	4905.289	-3733993.711
24	6375608	4779.574	-6370828.426
25	2638160	4649.015	-2633510.985
SUM	**66457108**	**56030889**	**-10426218.53**

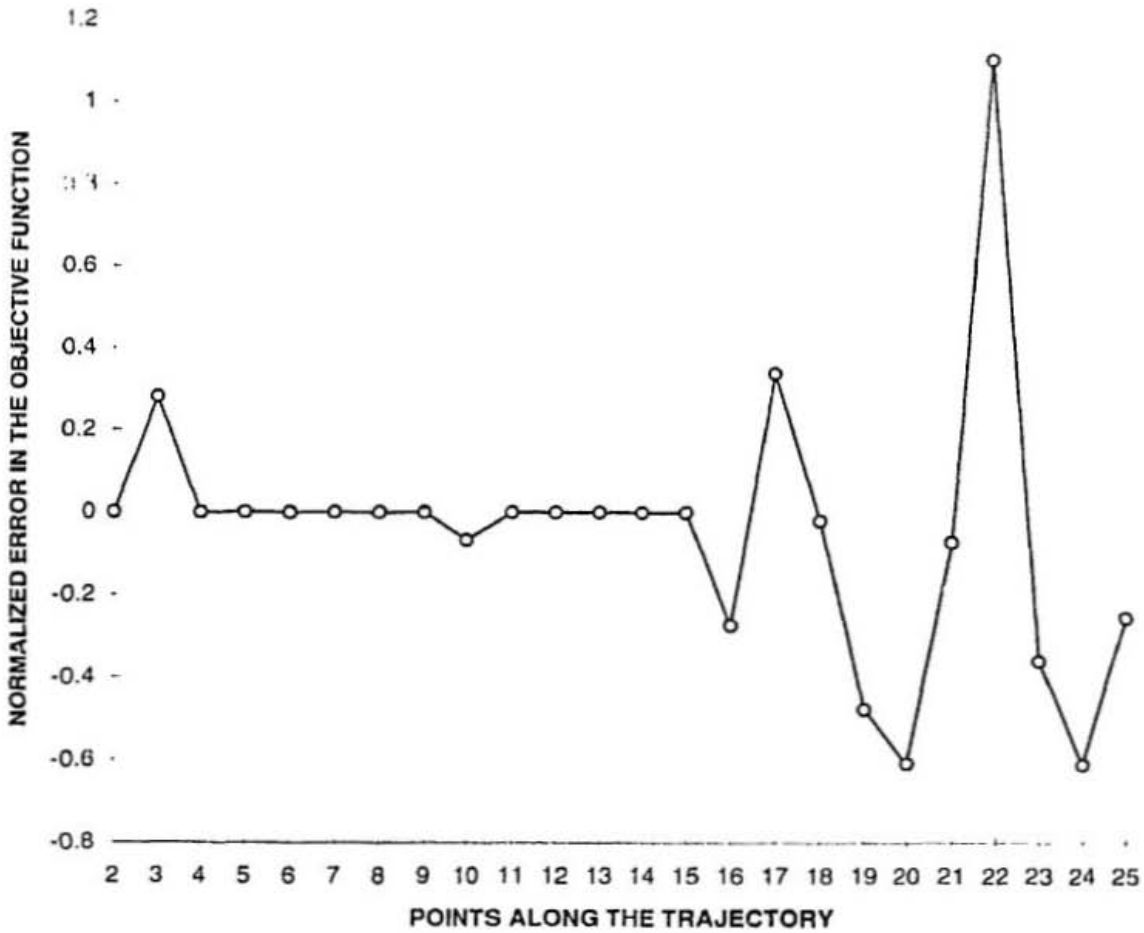

Fig. 3.23 VARIATION OF THE NORMALIZED ERROR BETWEEN THE OBJECTIVE FUNCTIONS

curve is obtained by dividing each of the numbers in the fourth column of Table 3.8 by the number in its bottom row. The percentage improvement is defined as

$$\frac{\Sigma\{U_i(\ddot{o}) - U_i(4)\}}{\Sigma U_i(4)} \times 100 \qquad \textbf{(3.46)}$$

Referring to Table 3.8, the improvement of 15.68% was achieved in the objective function when using the general control law.

3.9 CONCLUSIONS

In this chapter the various control issues of a planar two link manipulator have been discussed. The optimal control method (non-linear optimization) was used to achieve the trajectory control and compute the position and velocity gain values off-line. The training sets for the ANN method were generated using the optimal control method. Also, the on-line trajectory control of the same two link manipulator was achieved using the LPN (ANN) method. The results show that the LPN method can be successfully used for the on-line trajectory control. The effect of the variation of the maximum tangential velocity along the trajectory was also studied and it was found that better control is achieved if the maximum velocity is lower. The general control law for the trajectory yields better results.

The optimal control method was done off-line, and it has possibility of numerous computations. On the other hand, in the ANN method, the weights were obtained on the off-line basis, but the gain values were obtained on-line, which were approximate. Thus, the optimal control method yields accurate results but cannot be used on-line due to the computational requirements.

CHAPTER 4

CONCLUSIONS AND RECOMMENDATIONS

4.1 DISCUSSION AND CONCLUSIONS

The objective of this work was to estimate the dynamic parameters and the on-line trajectory control of the planar two link manipulator. This objective was achieved through the use of an ANN method called Linear Programming Neuro (LPN) method. The identification of dynamic parameters was carried out by using a link of regular geometry and associated inertia values. The inertia matrix of this link was then transformed to another set of axes to obtain a full matrix which a link having complicated mass distribution is normally expected to have in actual practice. These matrices were then used to train the ANN. After, the training, the ANN was used to predict the dynamic parameters of the unknown manipulator. In this thesis the various control issues of a planar two link manipulator are also discussed. The optimal control method (non-linear optimization) was used to achieve the trajectory control and compute the position and velocity gain values, off-line. Also the on-line trajectory control of the same two link manipulator was achieved using the LPN (ANN) method. The results show that the LPN method can be successfully used for the on-line trajectory control. The effects of the variation of the maximum tangential velocity and the general control law were also studied

in relation to the control aspect of the manipulator.

Based on this work, the following conclusions are drawn:

1. Neural networks can be used for the identification of dynamic parameters of robotic manipulators.
2. Corresponding to the estimated dynamic parameters and their desired values the input values, forces and torques, were recomputed. From the present work it was shown that as the end effector is made to travel along the trajectory, the errors in forces and torques are very small. The maximum error was less that 2% for all the points having significant and comparable torque values.
3. One can achieve very accurate trajectory control using the non-linear optimal control method.
4. The ANN method was used to compute the weight matrices for all the points along the trajectory by the training sets obtained from the optimal control method. These weight matrices can then be used for the on-line trajectory control of the manipulator.
5. One can also achieve sufficiently accurate on-line trajectory control using the ANN method.
6. The accuracy of the control results improves as the maximum trajectory velocity is decreased.
7. The use of general control law (six design variables) yields better control results as compared to the control law using four design variables. An

improvement of 15.68% was achieved in the objective function when using the general control law over the four variable control law.

4.2 RECOMMENDATIONS FOR FUTURE WORK

Future research work can be pursued on the following topics:

1. LP-neuro method can be extended to many applications in the on-line control of robotic manipulators.
2. Efficient techniques like Karmarkar's algorithm can be applied in Linear Programming for faster convergence to problems involving a large number of design variables.
3. It may be possible that the Optimal Control Method can be used in the on-line control of manipulators having more degrees of freedom if computations are done on parallel processors and faster computers.
4. The neural network method can also be used to solve problems involving friction and other uncertainties in the trajectories of robotic manipulators.

REFERENCES

Aggarwal, R., Sharan, A.M., Balasubramanian, R., (1995), "Trajectory Control of Robotic Manipulators using Artificial Neural Network Method", *accepted for 1995 WNN/WCI Conference, San Antonio*, September 22-28, 1995.

Akio, I., Takeshi, F., and Shigeru, O., (1992), "A Neural Network Compensator for Uncertainties of Robotic Manipulators", *IEEE Transactions on Industrial Electronics*, Vol.39, No.6, pp.565-569.

An, C.H., Atkenson, C.G., Hollerbatch, J.M., (1986), "Estimation of Inertial Parameters of Rigid Body Links of Manipulators", *IEEE Conference on Robotics and automation.*

Asada, H., (1990), "Teaching and Learning of Compliance Using Neural Nets: Representation and Generation of Nonlinear Compliance", *Proc. IEEE Int. Conf. on Robotics and Automation"*, Vol.2, pp.1237-1244.

Atkenson, C.G., An, C.H., Hollerbach, J.M., (1986), "Estimation of Inertial Parameters of Manipulator Loads and Links", *International Journal of Robotics Research*, Vol. 5 No. 3 pp. 101-119.

Balasubramanian, R., (1993), "Application of Neural Networks in Robotic Control and Design of Mechanisms", Memorial University of Newfoundland, Canada, *Master's Thesis.*

Balasubramanian, R., and Sharan, A.M., (1993), "LP-Neuro Method - A New Approach for Solving Neural Network Problems", *Proceedings of the Fifth Workshop on Neural Networks*, WNN93/FNN93, San Francisco.

Barmann, F., and Biegler-Konig, F., (1992), "On a Class of Efficient Learning Algorithms for Neural Networks", *Neural Networks*, Vol.5, pp.139-142.

Craig, J.J., (1994), "*Introduction to Robotics Mechanics and Control - Second Edition*", Addison-Wesley Publishing Company, New York.

D'Souza, A., and Garg, V.K., (1988), "*Advanced Dynamics-Modelling and Analysis*", Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Dubowsky, S., and Cheah, H., (1989), "A Method for estimating the Mass Properties of a Manipulator by Measuring the Reaction Moments at its base", *Proceedings of IEEE International Conference on Robotics and Automation*, Washington D.C.

Fukuda, T., Shibata, T., Kosuge, K., (1991), "Neuromorphic Sensing and Control - Applications to Position, Force and Impact Control for Robotic Manipulators", *Proc. of the 30th Conf. on Decision and Control*, Brighton, England, pp.162-167.

Fukuda, T., and Shibata, T., (1992), "Theory and Applications of Neural Networks for Industrial Control Systems", *IEEE Transactions on Industrial Electronics*, Vol.39, No.6, pp.472-487.

Greenwood, D.T., (1970), "*Principles of Dynamics*", Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Grossberg, G., (1982), *Studies of Mind and Brain: Neural Principles of Learning, Perception, Development, Cognition and Motor Control*, Boston, MA.

Gu, Y. and Chan, J.W.M., (1989), "On design of Non-linear Robotic Control System with Neural Networks", *Proc. IEEE Int. Conf. on Systems, Man and Cybernetics*, Cambridge, Massachusetts, pp.200-205.

Gulati, S., Barhen, J., and Iyengar, S.S., (1990), "Computational Neural Learning Formalisms for Manipulator Inverse Kinematics", *NASA Conference on Space Telerobotics*, Washington, Vol.1, pp.333-342.

Helferty, J.J., and Biswas, S., (1990), "Neuromorphic Control as a Self-Tuning Regulator", *Proc. 5th IEEE Int. Symp. on Intelligent Control*, Philadelphia, Pennsylvania, pp.506-511.

Hertz, J., and Krogh, A., (1991), "*Introduction to the Theory of Neural Computation*", Addison-Wesley Publishing Company, New York.

Hopfield, J.J., (1982), "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", *Proc. Nat. Acad. Sci.*, U.S.A, Vol.79, pp.2554-2558.

Jamshidi, M., Horne, B., and Vadiee, N., (1990), "A Neural Network-Based Controller for a Two-Link Robot", *Proc. 29th Conference on Decision and Control*, Honolulu, Hawaii, pp.3256-3257.

Karakasoglu, A. and Sundareshan, M.K., (1990), "Decentralized Variable Structure Control of Robotic Manipulators: Neural Computational Algorithms", *Proc. 29th Conference on Decision and control,* Honolulu, Hawaii, pp.3258-3259.

Kawasaki, H., Nishimura, K., (1986), "Parameter Identification of Mechanical Manipulators", *SICE Transactions* Vol. 22, No. 1, pp. 76-83.

Kawato, M., (1990), "Computational Schemes and Neural Network Models for Formation and Control of Multi-joint Arm Trajectory", *Neural Networks for Control*, The MIT Press, London, pp.197-228.

Khalil, W. and Bennis, F., April (1995), "Symbolic Calculation of the Base Inertial Parameters of Closed-Loop Robots", *The International Journal of Robotics Research*, Vol.14, No.2, pp.112-128.

Khosla, P., Kanade, T., (1985), "Parameter Identification of Robot Dynamics", *IEEE Conference Decision and Control*, Fort Lauderdale.

Kohonen, T., (1988), "An Introduction to Neural Computing", *Neural Networks*, Vol.1, No.1, pp.3-16.

Kulkarni, A.D., (1991), "Solving Ill-Posed Problems with Artificial Neural Networks", *Neural Networks*, Vol.4, pp.477-484.

Li, W., and Slotine, E., (1988), "On-Line Parameter Estimation for Robot Manipulators", *IEEE International Conference on Systems, Man and Cybernetics*, Peking, China.

Mayeda, H., Yoshida, K., Osuka, K., (1989), "A New Identification Method for Serial Manipulator Arms", *Proceedings 9th IFAC World Congress*, pp. 2429-2434.

Narendra, K.S., and Parthasarathy, K., March (1990), "Identification and Control of Dynamical Systems using Neural Networks", *IEEE Transactions on Neural Networks*, Vol.1, No.1, pp.4-27.

Nguyen, L., Patel, R.V., and Khorasani, K., (1990), "Neural Network Architectures for the Forward Kinematics Problem in Robotics", *Proc. Joint IEEE International Neural Network Conf.*, Sandiego, California , pp.393-399.

Ozaki, T., Suzuki, T., Furuhashi, T., Okuma, S., Uchikawa, Y., June (1991), "Trajectory Control of Robotic Manipulators Using Neural Networks", *IEEE Transactions on Industrial Electronics*, Vol. 38, No.3.

Rothbart, Harold A., (1973), "*Mechanical Design and Systems Handbook*", McGraw-Hill Book Company, New York.

Rumelhart, D.E., and McClelland, J.L., (1986), *Parallel Distributed Processing*, MIT Press, Massachusetts, Vols.1 and 2.

Sharan, A.M., and Aggarwal, R., "The Estimation of Dynamic Parameters of a Robotic Manipulator by Neural Network Method", *accepted for 1994 WNN/FNN Conference, Washington D.C.*, December 7-10, 1994.

Shoham, M., Li, C.J., Hacham, Y., Kreindler, E., (1992), "Neural Network Control of Robot Arms", *Annals of the CIRP* Vol. 41, pp.407-410.

Siddall, J.N., (1982) "*Optimal Engineering Design - Principles and Applications*", Marcel Dekker, Inc. New York and Basel.

Spong, M.W., and Vidyasagar, M., (1989), "*Robot Dynamics and Control*", John Wiley and Sons.

Tabary, G., and Salaun, I., 1992, "Control of a Redundant Articulated System by Neural Networks", *Neural Networks*, Vol.5, pp.305-311.

Yamamura, A.A., Sideris, A., Ji, C., and Psaltis, D., 1990, "Neural Network Control of a Two-Link Manipulator", *Proc. 29th Conf. on Decision and Control*, Honolulu, Hawaii, pp.3303-3307.

Yuh, J., 1992, "On Neural Net Controllers for Robotic Manipulators", *Proc. 4th Int. Symp. on Robotics and Manufacturing*, pp.757-762.

Zurada, Jacek M., (1992), "*Introduction to Artificial Neural Systems*", West Publishing Company, New York.

APPENDIX A

INERTIA TENSOR

The inertia tensor describes the mass properties of a body with respect to rotations about the centroid. Fig. A.1 shows a rigid body with an attached frame. The inertia tensor relative to frame {A}, attached to the rigid body, is expressed in the 3 x 3 symmetric matrix form as:

$$ {}^{A}[I] = \begin{bmatrix} I_{xx} & -I_{yy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix} \qquad \textbf{(A.1)} $$

where the scalar elements are given by

$$ I_{xx} = \iiint_V (y^2+z^2)\, \rho\, dv \qquad \textbf{(A.2)} $$

$$ I_{yy} = \iiint_V (x^2+z^2)\, \rho\, dv \qquad \textbf{(A.3)} $$

$$ I_{zz} = \iiint_V (x^2+y^2)\, \rho\, dv \qquad \textbf{(A.4)} $$

$$ I_{xy} = \iiint_V xy\, \rho\, dv \qquad \textbf{(A.5)} $$

$$ I_{xz} = \iiint_V xz\, \rho\, dv \qquad \textbf{(A.6)} $$

$$ I_{yz} = \iiint_V yz\, \rho\, dv \qquad \textbf{(A.7)} $$

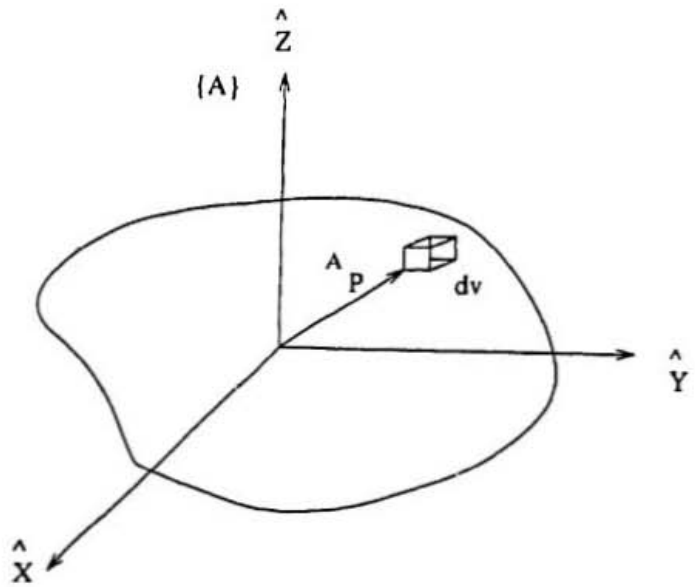

Fig. A.1 RIGID BODY WITH AN ATTACHED FRAME

where the rigid body is composed of differential volume elements, dv, containing material of density ρ. Each volume element is located with a vector, ${}^{A}P = [x\ y\ z]^{T}$ as shown in Fig. A.1. The elements I_{xx}, I_{yy}, and I_{zz} are called the mass moments of inertia. The elements with mixed indices are called mass products of inertia.

APPENDIX B

FORMULAS FOR ROTATION ABOUT THE PRINCIPAL AXES BY θ

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad \textbf{(B.1)}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad \textbf{(B.2)}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(B.3)}$$

APPENDIX C

ITERATIVE NEWTON-EULER DYNAMIC ALGORITHM

The Iterative Newton-Euler Dynamics Algorithm is for computing the torques that correspond to a given trajectory of a manipulator. Assuming that we know the position, velocity and acceleration of the joints, $(\theta, \dot{\theta}, \ddot{\theta})$. With this knowledge and with the knowledge of the kinematics and mass distribution information of the robot, we can calculate the joint torques required to cause this motion.

OUTWARD ITERATIONS TO COMPUTE VELOCITIES AND ACCELERATIONS

In order to compute inertial forces acting on the links it is necessary to compute the rotational velocity and linear and rotational acceleration of the center of mass of each link of the manipulator at any given instant. These computations are done in an iterative nature starting with link 1 and moving successively, link by link, outward to link n.

The rotational velocity for joint i+1 is given by

$$^{i+1}\omega_{i+1} = {}^{i+1}_{i}R\ {}^{i}\omega_{i} + \dot{\theta}_{i+1}\,{}^{i+1}\hat{Z}_{i+1} \qquad \text{(C.1)}$$

The equation for transforming angular acceleration for one link to the next is given as,

$$^{i+1}\dot{\omega}_{i+1} = {}^{i+1}_{i}R\ {}^{i}\dot{\omega}_{i} + {}^{i+1}_{i}R\ {}^{i}\omega_{i} \times \dot{\theta}_{i+1}\,{}^{i+1}\hat{Z}_{i+1} + \ddot{\theta}_{i+1}\,{}^{i+1}\hat{Z}_{i+1} \qquad \text{(C.2)}$$

When joint i + 1 is prismatic, this simplifies to

$$^{i+1}\dot{\omega}_{i+1} = {}^{i+1}_{i}R\ {}^{i}\dot{\omega}_{i} \qquad \text{(C.3)}$$

The linear acceleration of each link frame origin is obtained by:

$$^{i+1}\dot{v}_{i+1} = {}^{i+1}_{i}R\left[{}^{i}\dot{\omega}_i \times {}^{i}P_{i+1} + {}^{i}\omega_i \times \left({}^{i}\omega_i \times {}^{i}P_{i+1} \right) + {}^{i}\dot{v}_i \right] \quad \text{(C.4)}$$

which, for prismatic joint i + 1, becomes

$$^{i+1}\dot{v}_{i+1} = {}^{i+1}_{i}R\left({}^{i}\dot{\omega}_i \times {}^{i}P_{i+1} + {}^{i}\omega_i \times \left({}^{i}\omega_i \times {}^{i}P_{i+1} \right) + {}^{i}\dot{v}_i \right) + 2\,{}^{i+1}\omega_{i+1} \times \dot{d}_{i+1}\,{}^{i+1}\hat{Z}_{i+1} + \ddot{d}_{i+1}\,{}^{i+1}\hat{Z}_{i+1} \quad \text{(C.5)}$$

We also will need the linear acceleration of the center of mass of each link, which also can be found as:

$$^{i}\dot{v}_{C_i} = {}^{i}\dot{\omega}_i \times {}^{i}P_{C_i} + {}^{i}\omega_i \times \left({}^{i}\omega_i \times {}^{i}P_{C_i} \right) + {}^{i}\dot{v}_i \quad \text{(C.6)}$$

where we imagine a frame, $\{C_i\}$, attached to each link with its origin located at the center of mass of the link, and with the same orientation as the link frame, {i}. Eq. (C.6) doesn't involve joint motion at all, and so is valid for joint i + 1 revolute or prismatic.

THE FORCE AND TORQUE ACTING ON A LINK

Having computed the linear and angular accelerations of the mass center of each link, we can apply the Newton-Euler Equations to compute the inertial force and torque acting at the center of mass of each link. Thus we have

$$F_i = m\dot{v}_{C_i} \quad \text{(C.7)}$$

$$N_i = {}^{C_i}I\,\dot{\omega}_i + \omega_i \times {}^{C_i}I\,\omega_i \quad \text{(C.8)}$$

where $\{C_i\}$ has its origin at the center of mass of the link, and has the same orientation as the link frame, {i}.

INWARD ITERATIONS TO COMPUTE FORCES AND TORQUES

Having computed the forces and torques acting on each link, it now remains to calculate the joint torques which will result in these net forces and torques being applied to each link.

We can do this by writing a force balance and moment balance equation based on a free body diagram of a typical link (Fig. C.1). Each link has forces and torques exerted on it by its neighbours, and in addition experiences an inertial force and torque. Defining:

f_i = force exerted on link i by link i - 1,

n_i = torque exerted on link i by link i - 1.

By summing forces acting on link i we arrive at a force balance relationship,

$$^iF_i = {}^if_i - {}^i_{i+1}R\,{}^{i+1}f_{i+1} \qquad \textbf{(C.9)}$$

By summing torques about the center of mass and setting them equal to zero we arrive at torque balance equation:

$$^iN_i = {}^in_i - {}^in_{i+1} + \left(-{}^iP_{C_i}\right) \times {}^if_i - \left({}^iP_{i+1} - {}^iP_{C_i}\right) \times {}^if_{i+1} \qquad \textbf{(C.10)}$$

Using the result from the force balance relation Eq. (C.9) and adding a few rotation matrices we can write Eq. (C.10) as

$$^iN_i = {}^in_i - {}^i_{i+1}R\,{}^{i+1}n_{i+1} - {}^iP_{C_i} \times {}^iF_i - {}^iP_{i+1} \times {}^i_{i+1}R\,{}^{i+1}f_{i+1} \qquad \textbf{(C.11)}$$

Finally, we can rearrange the force and torque equations so that they appear as iterative relationships from higher-numbered neighbour to lower-numbered neighbour.

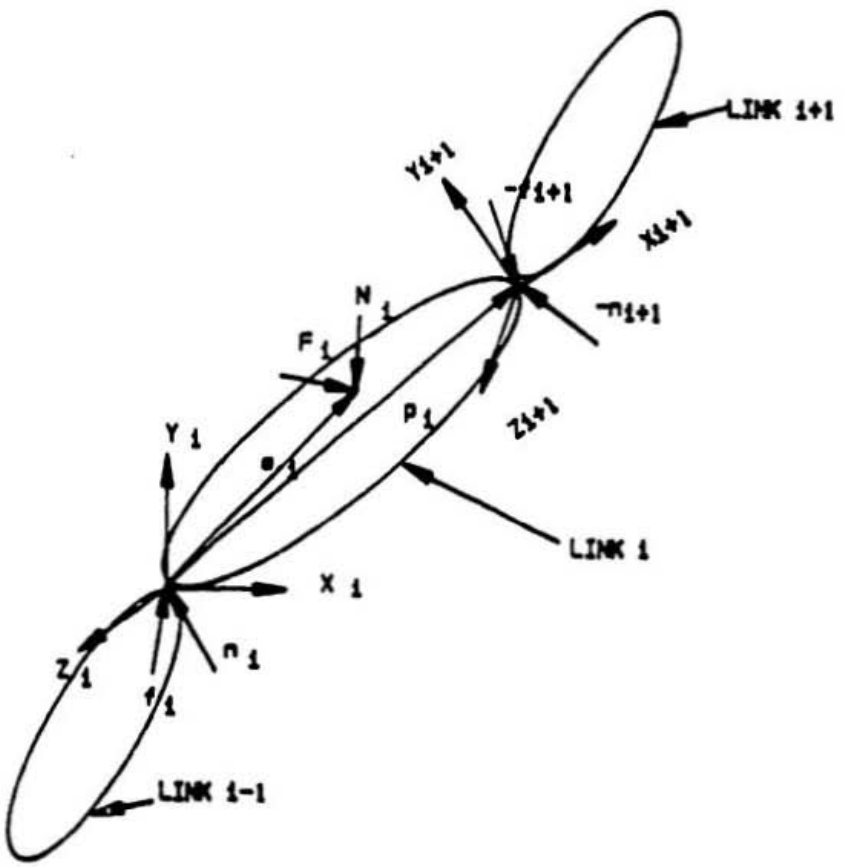

Fig. C.1 FREE BODY DIAGRAM OF A LINK

$$ {}^{i}f_i = {}^{i}_{i+1}R\,{}^{i+1}f_{i+1} + {}^{i}F_i \qquad \text{(C.12)} $$

$$ {}^{i}n_i = {}^{i}N_i + {}^{i}_{i+1}R\,{}^{i+1}n_{i+1} + {}^{i}P_{C_i} \times {}^{i}F_i + {}^{i}P_{i+1} \times {}^{i}_{i+1}R\,{}^{i+1}f_{i+1} \qquad \text{(C.13)} $$

These equations are evaluated link by link starting from link n and working inward toward the base of the robot. These inward force iterations are analogous to the static force iterations, except that inertial forces and torques are now considered at each link.

As in the static case, the required joint torques are found by taking the $\hat{Z}$ component of the torque applied by one link on its neighbour:

$$ \tau_i = {}^{i}n_i^{T}\,{}^{i}\hat{Z}_i \qquad \text{(C.14)} $$

For joint i + 1 prismatic, we use

$$ \tau_i = {}^{i}f_i^{T}\,{}^{i}\hat{Z}_i \qquad \text{(C.15)} $$

where we have used the symbol τ for a linear actuator force.

Note that for a robot moving in free space, ${}^{N+1}f_{N+1}$ and ${}^{N+1}n_{N+1}$ are set equal to zero, and so the first application of the equations for link n is very simple. If the robot is contacting the environment, the forces and torques due to this contact may be included in the force balance by having non-zero ${}^{N+1}f_{N+1}$ and ${}^{N+1}n_{N+1}$. These equations are summarized in Table 2.1 for the case of all joints rotational.

APPENDIX D

PROGRAM LISTINGS

D.1 Trajectory Generation

The trajectory generation of the planar two link manipulator is done as:

1. This program calculates the cartesian displacement, velocity and acceleration values for the 25 points along the trajectory for the planar two link manipulator knowing the radial velocity, radial acceleration and tangential acceleration values (D.1.1).
2. This program uses the inverse kinematic calculations to calculate the joint displacement, velocity and acceleration values for the 25 points along the trajectory (D.1.2).

These programs are written in FORTRAN. The link parameters are known for the planar two link manipulator. For all these programs OPTIVAR Designer's optimization subroutines [Siddall, 1982] are widely used.

D.1.1 Generation of Cartesian displacement, velocity and acceleration values

```
      PARAMETER (PI = 3.1415927)
      DIMENSION R(2,2),VTAN(2,1),VEL(2,1),RACC(25),TACC(25)
      DIMENSION ACC(2,1),ATAN(2,1),BTAN(2,1)
      OPEN(UNIT = 1, FILE = 'vtan.dat', STATUS = 'OLD')
      OPEN(UNIT = 2, FILE = 'atan.dat', STATUS = 'OLD')
      OPEN(UNIT = 3, FILE = 'values.out', STATUS = 'OLD')
      OPEN(UNIT = 4, FILE = 'circ.dat', STATUS = 'OLD')

      RAD = 0.2
      WRITE(*,*) 'R(m)      Th(deg)    Th(Rad)          X          Y'
      DO 5 TH = 5.0,53.0,2.0
      THETA = TH * (PI/180.0)
      X = RAD*COSD(TH)
      Y = RAD*SIND(TH)
      WRITE(*,*) RAD,TH,THETA, X,Y
      WRITE(4,*) RAD,TH,THETA, X,Y
      WRITE(3,*) X,Y
5     CONTINUE

      WRITE(*,*)
      WRITE(*,*) 'XDOT          YDOT    RADIAL VEL.  TANGENTIAL VEL.'

      DO 10 TH = 5.0,53.0,2.0
      READ(1,*) (VTAN(II,1),II=1,2)
      R(1,1) = COSD(TH)
      R(1,2) = -SIND(TH)
      R(2,1) = SIND(TH)
      R(2,2) = COSD(TH)
      CALL MATMUL(R,VTAN,VEL,2,2,1)
      WRITE(*,*) (VEL(J,1),J=1,2), (VTAN(J,1),J=1,2)
      WRITE(3,*) (VEL(J,1),J=1,2)
10    CONTINUE

      CALL CALRACC(RACC)
      CALL CALTACC(TACC)

      DO 15 I = 1,25
      ATAN(1,1) = RACC(I)
      ATAN(2,1) = TACC(I)
      WRITE(2,*) (ATAN(J,1),J=1,2)
15    CONTINUE
      REWIND(2)

      WRITE(*,*) 'XDDOT         YDDOT    RADIAL ACC.  TANGENTIAL ACC.'
      DO 17 TH = 5.0,53.0,2.0
```

```
      READ(2,*) (BTAN(J,1),J=1,2)
      R(1,1) = COSD(TH)
      R(1,2) = -SIND(TH)
      R(2,1) = SIND(TH)
      R(2,2) = COSD(TH)
      CALL MATMUL(R,BTAN,ACC,2,2,1)
      WRITE(*,*) (ACC(J,1),J=1,2), (BTAN(J,1),J=1,2)
      WRITE(3,*) (ACC(J,1),J=1,2)
17    CONTINUE

      CLOSE(1)
      CLOSE(2)
      CLOSE(3)
      CLOSE(4)
      STOP
      END

C     ***** Calculates the radial acceleration **************
      SUBROUTINE CALRACC(RACC)
      DIMENSION RACC(25),V(25),JUNK(2,1)
      COMMON V
      RAD = 0.2
      DO 30 I = 1,25
      READ(1,*) V(I)
      RACC(I) = ((V(I))**2.0)/RAD
30    CONTINUE
      RETURN
      END

C     ***** Calculates the tangential acceleration **************
      SUBROUTINE CALTACC(TACC)
      DIMENSION TACC(25),V(25)
      COMMON V

C     The number 3.49065e-3 is 's' i.e. r.d_theta
C     s = 0.2 * pi/180*(2 deg)

      DO 40 I = 1,24
      TACC(I) = (V(I+1)**2.0-V(I)**2.0)/(2.0*6.981317e-03)
40    CONTINUE
      TACC(25) = 0.0
      RETURN
      END

C     ********** Multiplication of two matrices ************
      SUBROUTINE MATMUL(A,B,C,I,J,K)
      DIMENSION A(I,J),B(J,K),C(I,K)
```

```
      DO 88 II=1,I
      DO 88 KK=1,K
      SUM=0.
      DO 77 JJ=1,J
      SUM=SUM+A(II,JJ)*B(JJ,KK)
77    CONTINUE
      C(II,KK)=SUM
88    CONTINUE
      RETURN
      END
```

D.1.2 Generation of Joint displacement, velocity and acceleration values

```
      PARAMETER (PI = 3.141593)
      DIMENSION VEL(2,1),ACC(2,1),VELJ(2,1),ACCJ(2,1),R1(3,3),R2(3,3)

      OPEN(UNIT = 22, FILE = 'inf.dat',STATUS = 'OLD')
      OPEN(UNIT = 32, FILE = 'desired.out', STATUS = 'OLD')
      OPEN(UNIT = 33, FILE = 'all_joint.out', STATUS = 'OLD')

      READ(22,*) AM1, AM2, AL1, AL2, B, C

      DO 10 JLOOP = 1,25
      READ(22,*) X, Y, (VEL(I,1),I=1,2), (ACC(I,1),I=1,2)
C     WRITE(*,*) X,Y,(VEL(I,1),I=1,2),(ACC(I,1),I=1,2)
      CALL INV(AL1,AL2,X,Y,VEL,ACC,TH1,TH2,VELJ,ACCJ,R1,R2)

      WRITE(*,*) 'JOINT ANGLES (degrees)'
      WRITE(*,*) TH1, TH2
      WRITE(*,*) 'JOINT VELOCITIES'
      WRITE(*,*) (VELJ(II,1),II=1,2)
      WRITE(*,*) 'JOINT ACCELERATIONS'
      WRITE(*,*) (ACCJ(II,1),II=1,2)

      TH_RAD1 = (PI/180.0) * TH1
      TH_RAD2 = (PI/180.0) * TH2
      WRITE(32,*) TH_RAD1,TH_RAD2,(VELJ(II,1),II=1,2)
      WRITE(33,*) TH_RAD1,TH_RAD2,(VELJ(II,1),II=1,2),
     1 (ACCJ(II,1),II=1,2)

10    CONTINUE

      CLOSE(22)
      CLOSE(32)
      CLOSE(33)
      STOP
```

```
      END

C
C     Inverse kinematics of a two-link planar manipulator
C

      SUBROUTINE INV(AL1,AL2,X,Y,VEL,ACC,TH1,TH2,VELJ,ACCJ,R1,R2)
      DIMENSION AM(2,2),AMINV(2,2),VEL(2,1),VELJ(2,1),ACC(2,1),ACCJ(2,1)
      DIMENSION BM(2,2),BMINV(2,2),VINT1(2,1),VINT2(2,1),R1(3,3),R2(3,3)

      C2 = (X**2.0 + Y**2.0 - AL1**2.0 - AL2**2.0)/(2.0*AL1*AL2)
      S2 = SQRT(1.0-C2**2.0)
      THETA2 = ATAN2(S2,C2)
      TH2 = ATAN2D(S2,C2)
      AK1 = AL1 + (AL2 * C2)
      AK2 = AL2 * S2
      THETA1 = ATAN2(Y,X) - ATAN2(AK2,AK1)
      TH1 = ATAN2D(Y,X) - ATAN2D(AK2,AK1)
      S1 = SIN(THETA1)
      C1 = COS(THETA1)
      S12 = SIN(THETA1 + THETA2)
      C12 = COS(THETA1 + THETA2)
      AM(1,1) = -AL1*S1-AL2*S12
      AM(1,2) = -AL2*S12
      AM(2,1) = AL1*C1 + AL2*C12
      AM(2,2) = AL2*C12
      DETER1 = AM(1,1)*AM(2,2) - AM(1,2)*AM(2,1)
      AMINV(1,1) = AM(2,2)/DETER1
      AMINV(2,1) = -(AM(2,1)/DETER1)
      AMINV(1,2) = -(AM(1,2)/DETER1)
      AMINV(2,2) = AM(1,1)/DETER1
      CALL MATMUL(AMINV,VEL,VELJ,2,2,1)

      VELJ1 = VELJ(1,1)
      VELJ2 = VELJ(2,1)
      BM(1,1) = -AL1*C1*VELJ1-AL2*C12*(VELJ1+VELJ2)
      BM(1,2) = -AL2*C12*(VELJ1+VELJ2)
      BM(2,1) = -AL1*S1*VELJ1-AL2*S12*(VELJ1+VELJ2)
      BM(2,2) = -AL2*S12*(VELJ1+VELJ2)
      CALL MATMUL(BM,VELJ,VINT1,2,2,1)
      DO 45 I = 1,2
      VINT2(I,1) = ACC(I,1) - VINT1(I,1)
45    CONTINUE
      CALL MATMUL(AMINV,VINT2,ACCJ,2,2,1)
      CALL RMAT(THETA1,R1)
      CALL RMAT(THETA2,R2)
      RETURN
```

```
      END

C     ********** Multiplication of two matrices ***********
      SUBROUTINE MATMUL(A,B,C,I,J,K)
      DIMENSION A(I,J),B(J,K),C(I,K)
      DO 88 II=1,I
      DO 88 KK=1,K
      SUM=0.
      DO 77 JJ=1,J
      SUM=SUM+A(II,JJ)*B(JJ,KK)
77    CONTINUE
      C(II,KK)=SUM
88    CONTINUE
      RETURN
      END

C     ****************** Rotation matrix **************
      SUBROUTINE RMAT(THETA,R)
      DIMENSION R(3,3)
      R(1,1) = COS(THETA)
      R(1,2) = -SIN(THETA)
      R(1,3) = 0.0
      R(2,1) = SIN(THETA)
      R(2,2) = COS(THETA)
      R(2,3) = 0.0
      R(3,1) = 0.0
      R(3,2) = 0.0
      R(3,3) = 1.0
      RETURN
      END
```

D.2 Generation of Training Sets for Parameter Estimation

1. This program generates the input and output training vectors (Table 2.2) for the training of the Artificial Neural Network. This program makes use of the Iterative Newton-Euler Dynamics algorithm for the calculation of the force, torque and the inertia matrix values (D.2.1).

D.2.1 Generation of training sets for parameter estimation problem

```
DIMENSION ALPHA(10), BETA(10), GAMMA(10)

DIMENSION AIFIN1(3,3),AIFIN2(3,3),PCFIN1(3,1),PCFIN2(3,1)

DIMENSION VEL(2,1),ACC(2,1),VELJ(2,1),ACCJ(2,1),R1(3,3),R2(3,3)

CALL ABG(ALPHA, BETA, GAMMA)
OPEN(UNIT = 22, FILE = 'inf.dat',STATUS = 'OLD')
READ(22,*) AM1, AM2, AL1, AL2, B, C

DO 5 ILOOP = 1,10
CALL IMAT(AM1,AL1,B,C,ALPHA,BETA,GAMMA,AIFIN1,PCFIN1,ILOOP)
DO II = 1,3
WRITE(*,*) (AIFIN1(II,JJ),JJ=1,3)
END DO
DO II = 1,3
WRITE(*,*) (PCFIN1(II,1))
END DO

CALL IMAT(AM2,AL2,B,C,ALPHA,BETA,GAMMA,AIFIN2,PCFIN2,ILOOP)
DO II = 1,3
WRITE(*,*) (AIFIN2(II,JJ),JJ=1,3)
END DO
DO II = 1,3
WRITE(*,*) (PCFIN2(II,1))
END DO

DO 10 JLOOP = 1,1
READ(22,*) X, Y, (VEL(I,1),I=1,2), (ACC(I,1),I=1,2)
CALL INV(AL1,AL2,X,Y,VEL,ACC,TH1,TH2,VELJ,ACCJ,R1,R2)

WRITE(*,*) TH1
WRITE(*,*) TH2
DO II = 1,2
WRITE(*,*) (VELJ(II,1))
END DO
DO II = 1,2
WRITE(*,*) (ACCJ(II,1))
END DO
DO II = 1,3
WRITE(*,*) (R1(II,JJ),JJ=1,3)
END DO
DO II = 1,3
WRITE(*,*) (R2(II,JJ),JJ=1,3)
END DO
```

```
      CALL DYN(AM1,AM2,VELJ,ACCJ,AIFIN1,AIFIN2,PCFIN1,PCFIN2,R1,R2)

10    CONTINUE
5     CONTINUE

      CLOSE(22)
      STOP
      END

C     ********* Variation of the values of alpha, beta and gamma **********
      SUBROUTINE ABG(ALPHA, BETA, GAMMA)
      DIMENSION ALPHA(10), BETA(10), GAMMA(10)
      ALPHA(1) = 5.0
      BETA(1) = 5.0
      GAMMA(1) = 5.0
      DO 11 I = 1,9
      ALPHA(I+1) = ALPHA(1) + 2.0 * I
      BETA(I+1) = BETA(1) + 2.0 * I
      GAMMA(I+1) = GAMMA(1) + 2.0 * I
11    CONTINUE
      RETURN
      END

C     ******** Generation of inertia matrix ********************
      SUBROUTINE IMAT(AM,AL,B,C,ALPHA,BETA,GAMMA,AIFIN,PCFIN,ILOOP)
      DIMENSION ALPHA(10), BETA(10), GAMMA(10)
      DIMENSION AIC(3,3),PC(3,1),PCT(1,3),A1(1,1),C1(3,3),AIA(3,3),AIFIN(3,3)
      DIMENSION AI(3,3),B1(3,3),D1(3,3),E1(3,3),CM(3,3),CMT(3,3),F1(3,3)
      DIMENSION PCFIN(3,1)

      XC = AL/2.0
      YC = 0.0
      ZC = 0.0
      AIC(1,1) = (1.0/12.0)*AM*(B**2.0+C**2.0)
      AIC(1,2) = 0.0
      AIC(1,3) = 0.0
      AIC(2,1) = 0.0
      AIC(2,2) = (1.0/12.0)*AM*(C**2.0+AL**2.0)
      AIC(2,3) = 0.0
      AIC(3,1) = 0.0
      AIC(3,2) = 0.0
      AIC(3,3) = (1.0/12.0)*AM*(AL**2.0+B**2.0)
      PC(1,1) = XC
      PC(2,1) = YC
      PC(3,1) = ZC

      AI(1,1) = 1.0
```

```
      AI(1,2) = 0.0
      AI(1,3) = 0.0
      AI(2,1) = 0.0
      AI(2,2) = 1.0
      AI(2,3) = 0.0
      AI(3,1) = 0.0
      AI(3,2) = 0.0
      AI(3,3) = 1.0
      CALL TRANS(PC,PCT,3,1)
      CALL MATMUL(PCT,PC,A1,1,3,1)
      SC = A1(1,1)
      CALL SCALARMUL(AI,SC,B1,3,3)
      CALL MATMUL(PC,PCT,C1,3,1,3)
      CALL MATSUB(B1,C1,D1,3,3)
      CALL SCALARMUL(D1,AM,E1,3,3)
      CALL MATADD(AIC,E1,AIA,3,3)
      CM(1,1) = COSD(ALPHA(ILOOP))*COSD(BETA(ILOOP))
      CM(1,2) = COSD(ALPHA(ILOOP))*SIND(BETA(ILOOP))*SIND(GAMMA(ILOOP))-
     1  SIND(ALPHA(ILOOP))*COSD(GAMMA(ILOOP))
      CM(1,3) = COSD(ALPHA(ILOOP))*SIND(BETA(ILOOP))*COSD(GAMMA(ILOOP))+
     1  SIND(ALPHA(ILOOP))*SIND(GAMMA(ILOOP))
      CM(2,1) = SIND(ALPHA(ILOOP))*COSD(BETA(ILOOP))
      CM(2,2) = SIND(ALPHA(ILOOP))*SIND(BETA(ILOOP))*SIND(GAMMA(ILOOP))+
     1  COSD(ALPHA(ILOOP))*COSD(GAMMA(ILOOP))
      CM(2,3) = SIND(ALPHA(ILOOP))*SIND(BETA(ILOOP))*COSD(GAMMA(ILOOP))-
     1  COSD(ALPHA(ILOOP))*SIND(GAMMA(ILOOP))
      CM(3,1) = -SIND(BETA(ILOOP))
      CM(3,2) = COSD(BETA(ILOOP))*SIND(GAMMA(ILOOP))
      CM(3,3) = COSD(BETA(ILOOP))*COSD(GAMMA(ILOOP))
      CALL TRANS(CM,CMT,3,3)
      CALL MATMUL(CMT,AIA,F1,3,3,3)
      CALL MATMUL(F1,CM,AIFIN,3,3,3)
      CALL MATMUL(CMT,PC,PCFIN,3,3,1)
      RETURN
      END

C     Inverse kinematics of planar two link manipulator

      SUBROUTINE INV(AL1,AL2,X,Y,VEL,ACC,TH1,TH2,VELJ,ACCJ,R1,R2)
      DIMENSION AM(2,2),AMINV(2,2),VEL(2,1),VELJ(2,1),ACC(2,1),ACCJ(2,1)
      DIMENSION BM(2,2),BMINV(2,2),VINT1(2,1),VINT2(2,1),R1(3,3),R2(3,3)
      C2 = (X**2.0 + Y**2.0 - AL1**2.0 - AL2**2.0)/(2.0*AL1*AL2)
      S2 = SQRT(1.0-C2**2.0)
      THETA2 = ATAN2(S2,C2)
      TH2 = ATAN2D(S2,C2)
      AK1 = AL1 + (AL2 * C2)
```

```
      AK2 = AL2 * S2
      THETA1 = ATAN2(Y,X) - ATAN2(AK2,AK1)
      TH1 = ATAN2D(Y,X) - ATAN2D(AK2,AK1)
      S1 = SIN(THETA1)
      C1 = COS(THETA1)
      S12 = SIN(THETA1 + THETA2)
      C12 = COS(THETA1 + THETA2)
      AM(1,1) = -AL1*S1-AL2*S12
      AM(1,2) = -AL2*S12
      AM(2,1) = AL1*C1 + AL2*C12
      AM(2,2) = AL2*C12
      DETER1 = AM(1,1)*AM(2,2) - AM(1,2)*AM(2,1)
      AMINV(1,1) = AM(2,2)/DETER1
      AMINV(2,1) = -(AM(2,1)/DETER1)
      AMINV(1,2) = -(AM(1,2)/DETER1)
      AMINV(2,2) = AM(1,1)/DETER1
      CALL MATMUL(AMINV,VEL,VELJ,2,2,1)

      VELJ1 = VELJ(1,1)
      VELJ2 = VELJ(2,1)
      BM(1,1) = -AL1*C1*VELJ1-AL2*C12*(VELJ1+VELJ2)
      BM(1,2) = -AL2*C12*(VELJ1+VELJ2)
      BM(2,1) = -AL1*S1*VELJ1-AL2*S12*(VELJ1+VELJ2)
      BM(2,2) = -AL2*S12*(VELJ1+VELJ2)
      CALL MATMUL(BM,VELJ,VINT1,2,2,1)
      DO 45 I = 1,2
      VINT2(I,1) = ACC(I,1) - VINT1(I,1)
45    CONTINUE
      CALL MATMUL(AMINV,VINT2,ACCJ,2,2,1)
      CALL RMAT(THETA1,R1)
      CALL RMAT(THETA2,R2)
      RETURN
      END

C     Iterative Newton-Euler dynamics algorithm

      DYN(SMASS1,SMASS2,VELJ,ACCJ,XMAT1,XMAT2,PCFIN1,PCFIN2,R1,R2)
      DIMENSION R1(3,3),R2(3,3),RT(3,3),XMAT(3,3)
      DIMENSION OMEGA(3),OMEGAI(3),ALPHA(3),ALPHAI(3),A(3),P(3)
      DIMENSION X4(3),AI(3),OMEGAI1(3),ST1(3),ST2(3),ST3(3)
      DIMENSION QDTDTV(3),QDTV(3),X1(3),X2(3),X3(3)
      DIMENSION SI1(3),SI2(3),Y1(3),Y2(3),X(3),ALPHAI1(3)
      DIMENSION Y4(3),ACI(3),FI(3),Z1(3),Z2(3),Z3(3),XNI(3),Y3(3)
      DIMENSION RMAT(3,3),FO(3),FORCE(3),TT(3),FORI(3),RNN(3)
      DIMENSION TOR(3),PC(3),POS(3),E1(3),E2(3),E3(3),E4(3)
      DIMENSION E5(3),E6(3),TORQ(3),RESF(3),CHECN(3)
      DIMENSION VELJ(2,1),ACCJ(2,1)
```

```
      DIMENSION PCFIN1(3,1), PCFIN2(3,1)

      DO J = 1,3
      SI1(J) = PCFIN1(J,1)
      END DO
      DO J = 1,3
      SI2(J) = PCFIN2(J,1)
      END DO

      WRITE(*,*)
      WRITE(*,*)'FORWARD RECURSION FOR A TWO LINK MANIPULATOR'
      WRITE(*,*)'THE JOINT IS ROTATIONAL'
      WRITE(*,*)

      WRITE(*,*)'FORWARD RECURSION FOR LINK # 1'
      WRITE(*,*) 'STEP 1: '

      OPEN(UNIT = 1, FILE = 'fr1.dat', STATUS = 'OLD')
      OPEN(UNIT = 3, FILE = 'frint.dat', STATUS = 'OLD')
      OPEN(UNIT = 9, FILE = 'l1int.dat', STATUS = 'OLD')

      DO 20 I = 1,3
      DO 20 J = 1,3
20    RT(I,J) = R1(J,I)

C     ******** OMEGA[I-1] VECTOR  ************
      DO 30 I = 1,3
      READ(1,*) OMEGA(I)
30    CONTINUE

      CALL MATMUL(RT,OMEGA,OMEGAI1,3,3,1)

      QDTDT1 = ACCJ(1,1)
      QDT1 = VELJ(1,1)
      QDTDTV(1) = 0.
      QDTDTV(2) = 0.
      QDTDTV(3) = QDTDT1
      QDTV(1) = 0.
      QDTV(2) = 0.
      QDTV(3) = QDT1
      CALL VECSUM(OMEGAI1,QDTV,OMEGAI)
      WRITE(*,*) 'ANGULAR VELOCITY OMEGA[I] IS '
      WRITE(*,*) (OMEGAI(J),J=1,3)

      DO 40 J = 1,3
      WRITE(3,*) OMEGAI(J)
40    CONTINUE
```

```
      WRITE(*,*) 'STEP 2:'
      WRITE(*,*)

C     ******** OMEGADOT[I-1] VECTOR **********
      DO 50 I = 1,3
      READ(1,*) ALPHA(I)
50    CONTINUE

      CALL MATMUL(RT,ALPHA,ALPHAI1,3,3,1)
      CALL MATMUL(RT,OMEGA,ST1,3,3,1)
      CALL VECCROSS(ST1,QDTV,ST2)
      CALL VECSUM(ST2,QDTDTV,ST3)
      CALL VECSUM(ALPHAI1,ST3,ALPHAI)
      WRITE(*,*) ' ANGULAR ACCELERATION OMEGADOT[I] IS '
      WRITE(*,*) (ALPHAI(J), J = 1,3)
      WRITE(*,*)

      DO 60 J = 1,3
      WRITE(3,*) ALPHAI(J)
60    CONTINUE

      WRITE(*,*) '  STEP 3: '
      WRITE(*,*)

C     ********** VDOT[I-1] VECTOR ***********
      DO 70 I = 1,3
      READ(1,*) A(I)
70    CONTINUE

C     *********** P[I-1] VECTOR ***********
      DO 80 I = 1,3
      READ(1,*) P(I)
80    CONTINUE

      CALL VECCROSS(OMEGA,P,X)
      CALL VECCROSS(OMEGA,X,X1)
      CALL VECCROSS(ALPHA,P,X2)
      CALL VECSUM(X1,X2,X3)
      CALL VECSUM(X3,A,X4)
      CALL MATMUL(RT,X4,AI,3,3,1)
      WRITE(*,*) ' THE VDOT[I] VECTOR IS : '
      WRITE(*,*) (AI(I),I = 1,3)
      WRITE(*,*)

      DO 90 J = 1,3
      WRITE(3,*) AI(J)
90    CONTINUE
```

```
      WRITE(*,*)' STEP 4:  '
      WRITE(*,*)

C     ******** PC[I] VECTOR ***********
      CALL VECCROSS(OMEGAI,SI1,Y1)
      CALL VECCROSS(OMEGAI,Y1,Y2)
      CALL VECCROSS(ALPHAI,SI1,Y3)
      CALL VECSUM(Y2,Y3,Y4)
      CALL VECSUM(AI,Y4,ACI)
      WRITE(*,*) ' THE VDOT[CI] VECTOR IS '
      WRITE(*,*)(ACI(J), J = 1,3)
      WRITE(*,*)
      WRITE(*,*) '  STEP 5 '
      WRITE(*,*)
      DO 110 I = 1,3
110   FI(I) = SMASS1  * ACI(I)
      WRITE(*,*) ' THE FORCE VECTOR IS : '
      WRITE(*,*) (FI(I), I = 1,3)

      DO 111 I = 1,3
      WRITE(9,*) FI(I)
111   CONTINUE

      WRITE(*,*) ' STEP 6 '
      WRITE(*,*)

C     ******* I MATRIX [3X3] ***********
      CALL MATMUL(XMAT1,OMEGAI,Z1,3,3,1)
      CALL VECCROSS(OMEGAI,Z1,Z2)
      CALL MATMUL(XMAT1,ALPHAI,Z3,3,3,1)
      CALL VECSUM(Z2,Z3,XNI)
      WRITE(*,*) ' THE TORQUE VECTOR IS : '
      WRITE(*,*) (XNI(J), J = 1,3)
      WRITE(*,*)

      DO 121 II = 1,3
      WRITE(9,*) XNI(II)
121   CONTINUE

      DO 122 II = 1,3
      WRITE(9,*) SI1(II)
122   CONTINUE

      CLOSE(1)
      CLOSE(3)
      CLOSE(9)
```

```
      OPEN(UNIT = 2, FILE = 'fr2.dat', STATUS = 'OLD')
      OPEN(UNIT = 3, FILE = 'frint.dat', STATUS = 'OLD')
      OPEN(UNIT = 4, FILE = 'I2int.dat', STATUS = 'OLD')
      WRITE(*,*)
      WRITE(*,*)'FORWARD RECURSION FOR LINK # 2'
      WRITE(*,*) 'STEP 1: '

C     ***** ROTATIONAL MATRIX [R] NOT ITS TRANSPOSE *******
      DO 140 I = 1,3
      DO 140 J = 1,3
140   RT(I,J) = R2(J,I)

C     ******** OMEGA[I-1] VECTOR  ************
      DO 150 I = 1,3
      READ(3,*) OMEGA(I)
150   CONTINUE

      CALL MATMUL(RT,OMEGA,OMEGAI1,3,3,1)
      QDTDT2 = ACCJ(2,1)
      QDT2 = VELJ(2,1)
      QDTDTV(1) = 0.
      QDTDTV(2) = 0.
      QDTDTV(3) = QDTDT2
      QDTV(1) = 0.
      QDTV(2) = 0.
      QDTV(3) = QDT2
      CALL VECSUM(OMEGAI1,QDTV,OMEGAI)
      WRITE(*,*) 'ANGULAR VELOCITY OMEGA[I] IS '
      WRITE(*,*) (OMEGAI(J),J=1,3)

      WRITE(*,*) 'STEP 2:'
      WRITE(*,*)

C     ******** OMEGADOT[I-1] VECTOR **********
      DO 160 I = 1,3
      READ(3,*) ALPHA(I)
160   CONTINUE

      CALL MATMUL(RT,ALPHA,ALPHAI1,3,3,1)
      CALL MATMUL(RT,OMEGA,ST1,3,3,1)
      CALL VECCROSS(ST1,QDTV,ST2)
      CALL VECSUM(ST2,QDTDTV,ST3)
      CALL VECSUM(ALPHAI1,ST3,ALPHAI)
      WRITE(*,*) ' ANGULAR ACCELERATION OMEGADOT[I] IS '
      WRITE(*,*) (ALPHAI(J), J = 1,3)
      WRITE(*,*)
```

```
      WRITE(*,*) '  STEP 3: '
      WRITE(*,*)

C     ********** VDOT[I-1] VECTOR ***********
      DO 170 I = 1,3
      READ(3,*) A(I)
170   CONTINUE

C     ********** P[I-1] VECTOR ***********
      DO 180 I = 1,3
      READ(2,*) P(I)
180   CONTINUE

      CALL VECCROSS(OMEGA,P,X)
      CALL VECCROSS(OMEGA,X,X1)
      CALL VECCROSS(ALPHA,P,X2)
      CALL VECSUM(X1,X2,X3)
      CALL VECSUM(X3,A,X4)
      CALL MATMUL(RT,X4,AI,3,3,1)
      WRITE(*,*) ' THE VDOT[I] VECTOR IS : '
      WRITE(*,*) (AI(I),I = 1,3)
      WRITE(*,*)

      WRITE(*,*)' STEP 4:  '
      WRITE(*,*)

C     ******** PC[I] VECTOR ***********
      CALL VECCROSS(OMEGAI,SI2,Y1)
      CALL VECCROSS(OMEGAI,Y1,Y2)
      CALL VECCROSS(ALPHAI,SI2,Y3)
      CALL VECSUM(Y2,Y3,Y4)
      CALL VECSUM(AI,Y4,ACI)
      WRITE(*,*) ' THE VC  .T[CI] VECTOR IS '
      WRITE(*,*)(ACI(J), J = 1,3)
      WRITE(*,*)
      WRITE(*,*) '  STEP 5 '
      WRITE(*,*)
      DO 200 I = 1,3
200   FI(I) = SMASS2  * ACI(I)
      WRITE(*,*) ' THE FORCE VECTOR IS  '
      WRITE(*,*) (FI(I), I = 1,3)

      DO 201 I = 1,3
      WRITE(4,*) FI(I)
201   CONTINUE

      WRITE(*,*) ' STEP 6 '
```

```
      WRITE(*,*)

C     ******* I MATRIX [3X3] ***********
      CALL MATMUL(XMAT2,OMEGAI,Z1,3,3,1)
      CALL VECCROSS(OMEGAI,Z1,Z2)
      CALL MATMUL(XMAT2,ALPHAI,Z3,3,3,1)
      CALL VECSUM(Z2,Z3,XNI)
      WRITE(*,*) ' THE TORQUE VECTOR IS : '
      WRITE(*,*) (XNI(J), J = 1,3)
      WRITE(*,*)

      DO 211 I = 1,3
      WRITE(4,*) XNI(I)
211   CONTINUE

      DO 212 I = 1,3
      WRITE(4,*) SI2(I)
212   CONTINUE

      CLOSE(2)
      CLOSE(3)
      CLOSE(4)

      WRITE(*,*)
      WRITE(*,*) 'FORWARD RECURSION IS OVER'
      WRITE(*,*) 'BACKWARD RECURSION STARTS HERE'
      WRITE(*,*)
      WRITE(*,*) 'BACKWARD RECURSION FOR LINK # 2'
      WRITE(*,*)
      OPEN(UNIT = 4, FILE = 'I2int.dat', STATUS = 'OLD')
      OPEN(UNIT = 10, FILE = 'br1.dat', STATUS = 'OLD')
      OPEN(UNIT = 8, FILE = 'brint.dat', STATUS = 'OLD')
      WRITE(*,*) 'STEP 7: '
      WRITE(*,*)

C     ********* [R] MATRIX FOR I TO I+1 ***********
      DO 220 I = 1,3
      READ(10,*) (RMAT(I,J),J=1,3)
220   CONTINUE

C     ********** VECTOR f(I+1) **************
      DO 230 I = 1,3
      READ(10,*) fo(I)
230   CONTINUE

C     ********** VECTOR F OF LINK I **********
      DO 240 I = 1,3
```

```
      READ(4,*) FORCE(I)
240   CONTINUE

      CALL MATMUL(RMAT,FO,TT,3,3,1)
      CALL VECSUM(TT,FORCE,FORI)
      WRITE(*,*)' f VECTOR OF LINK I:'
      WRITE(*,*)(FORI(I),I=1,3)
      WRITE(*,*)

      DO 241 I = 1,3
      WRITE(8,*) FORI(I)
241   CONTINUE

      WRITE(*,*)'CROSS-CHECKING OF FORCE RESULTS'
      WRITE(*,*)' eqn (6.   ). '
      WRITE(*,*)' L.H.S - R.H.S = {0 0 0}'
      DO 2344 IMK = 1,3
      RESF(IMK) = FORI(IMK) - TT(IMK) - FORCE(IMK)
2344  CONTINUE
      WRITE(*,*)'CORRECT IF RESIDUE VECTOR IS ZERO'
      WRITE(*,*)
      WRITE(*,*)'PRINTING RESIDUE VECTOR BELOW:'
      WRITE(*,*)(RESF(JMK),JMK = 1,3)
      WRITE(*,*)'STEP 8'
      WRITE(*,*)

C     ********* N(I) VECTOR ***********
      DO 250 I = 1,3
      READ(4,*) RNN(I)
250   CONTINUE

C     ********* n(I+1) VECTOR ***********
      DO 260 I = 1,3
      READ(10,*) TOR(I)
260   CONTINUE

C     ********** PC(I) VECTOR ***********
      DO 270 I = 1,3
      READ(4,*) PC(I)
270   CONTINUE

C     ********** P(I) VECTOR ***********
      DO 280 I = 1,3
      READ(10,*) POS(I)
280   CONTINUE

      CALL MATMUL(RMAT,FO,E1,3,3,1)
```

```
      CALL VECCROSS(POS,E1,E2)
      CALL VECCROSS(PC,FORCE,E3)
      CALL MATMUL(RMAT,TOR,E4,3,3,1)
      CALL VECSUM(RNN,E4,E5)
      CALL VECSUM(E5,E3,E6)
      CALL VECSUM(E6,E2,TORQ)
      WRITE(*,*)'n(I) vector is ?'
      WRITE(*,*)(TORQ(I),I=1,3)
      WRITE(*,*)

      DO 281 I = 1,3
      WRITE(8,*) TORQ(I)
281   CONTINUE

      WRITE(*,*)'CROSS-CHECKING OF TORQUE RESULTS'
      WRITE(*,*)'  eqn (6.  )'
      WRITE(*,*)' L.H.S - R.H.S = { 0 0 0}'
      WRITE(*,*)'CORRECT IF RESIDUE IS ZERO'
      WRITE(*,*)
      WRITE(*,*)'PRINTING RESIDUE VALUE BELOW'
      DO 5746 IUK = 1,3
      CHECN(IUK) = TORQ(IUK)-RNN(IUK)-E4(IUK)-E3(IUK)-E2(IUK)
5746  CONTINUE
      WRITE(*,*)(CHECN(JUK),JUK=1,3)
      WRITE(*,*)
      CLOSE(4)
      CLOSE(10)
      CLOSE(8)

      WRITE(*,*)
      WRITE(*,*) 'BACKWARD RECURSION FOR LINK # 1'
      WRITE(*,*)
      OPEN(UNIT = 9, FILE = 'l1int.dat', STATUS = 'OLD')
      OPEN(UNIT = 7, FILE = 'br2.dat', STATUS = 'OLD')
      OPEN(UNIT = 8, FILE = 'brint.dat', STATUS = 'OLD')
      WRITE(*,*) 'STEP 7: '
      WRITE(*,*)

C     ********** VECTOR f(I+1) **************
      DO 300 I = 1,3
      READ(8,*) fo(I)
300   CONTINUE

C     ********** VECTOR F OF LINK I **********
      DO 310 I = 1,3
      READ(9,*) FORCE(I)
310   CONTINUE
```

```
      CALL MATMUL(R2,FO,TT,3,3,1)
      CALL VECSUM(TT,FORCE,FORI)
      WRITE(*,*)' f VECTOR OF LINK I:'
      WRITE(*,*)(FORI(I),I=1,3)
      WRITE(*,*)
      WRITE(*,*)'CROSS-CHECKING OF FORCE RESULTS'
      WRITE(*,*)' eqn (6.   ). '
      WRITE(*,*)' L.H.S - R.H.S = {0 0 0}'
      DO 320 IMK = 1,3
      RESF(IMK) = FORI(IMK) - TT(IMK) - FORCE(IMK)
320   CONTINUE
      WRITE(*,*)'CORRECT IF RESIDUE VECTOR IS ZERO'
      WRITE(*,*)
      WRITE(*,*)'PRINTING RESIDUE VECTOR BELOW:'
      WRITE(*,*)(RESF(JMK),JMK = 1,3)
      WRITE(*,*)'STEP 8'
      WRITE(*,*)

C     ********* N(I) VECTOR ***********
      DO 330 I = 1,3
      READ(9,*) RNN(I)
330   CONTINUE

C     ********* n(I+1) VECTOR ***********
      DO 340 I = 1,3
      READ(8,*) TOR(I)
340   CONTINUE

C     ********** PC(I) VECTOR **********
      DO 350 I = 1,3
      READ(9,*) PC(I)
350   CONTINUE

C     ********** P(I) VECTOR **********
      DO 360 I = 1,3
      READ(7,*) POS(I)
360   CONTINUE

      CALL MATMUL(R2,FO,E1,3,3,1)
      CALL VECCROSS(POS,E1,E2)
      CALL VECCROSS(PC,FORCE,E3)
      CALL MATMUL(R2,TOR,E4,3,3,1)
      CALL VECSUM(RNN,E4,E5)
      CALL VECSUM(E5,E3,E6)
      CALL VECSUM(E6,E2,TORQ)
      WRITE(*,*)'n(I) vector is ?'
      WRITE(*,*)(TORQ(I),I=1,3)
```

```
      WRITE(*,*)
      WRITE(*,*)'CROSS-CHECKING OF TORQUE RESULTS'
      WRITE(*,*)'  eqn (6.  )'
      WRITE(*,*)' L.H.S - R.H.S = { 0 0 0}'
      WRITE(*,*)'CORRECT IF RESIDUE IS ZERO'
      WRITE(*,*)
      WRITE(*,*)'PRINTING RESIDUE VALUE BELOW'
      DO 370 IUK = 1,3
      CHECN(IUK) = TORQ(IUK)-RNN(IUK)-E4(IUK)-E3(IUK)-E2(IUK)
370   CONTINUE
      WRITE(*,*)(CHECN(JUK),JUK=1,3)
      WRITE(*,*)
      CLOSE(7)
      CLOSE(8)
      CLOSE(9)
      RETURN
      END

C     ********** Multiplication of two matrices ***********
      SUBROUTINE MATMUL(A,B,C,I,J,K)
      DIMENSION A(I,J),B(J,K),C(I,K)
      DO 88 II=1,I
      DO 88 KK=1,K
      SUM=0.
      DO 77 JJ=1,J
      SUM=SUM+A(II,JJ)*B(JJ,KK)
77    CONTINUE
      C(II,KK)=SUM
88    CONTINUE
      RETURN
      END

C     ************** Addition of two vectors **************
      SUBROUTINE VECSUM(A,B,C)
      DIMENSION A(3),B(3),C(3)
      DO 99 I = 1,3
99    C(I) = A(I) + B(I)
      RETURN
      END

C     ***************** Cross product of two vectors ***************
      SUBROUTINE VECCROSS(A,B,C)
      DIMENSION A(3),B(3),C(3)
      C(1)=A(2)*B(3)-A(3)*B(2)
      C(2)=A(3)*B(1)-A(1)*B(3)
      C(3)=A(1)*B(2)-A(2)*B(1)
      RETURN
```

```
      END

C     ********** Rotation matrix ***************
      SUBROUTINE RMAT(THETA,R)
      DIMENSION R(3,3)
      R(1,1) = COS(THETA)
      R(1,2) = -SIN(THETA)
      R(1,3) = 0.0
      R(2,1) = SIN(THETA)
      R(2,2) = COS(THETA)
      R(2,3) = 0.0
      R(3,1) = 0.0
      R(3,2) = 0.0
      R(3,3) = 1.0
      RETURN
      END

C     ************ Transpose of a matrix ****************
      SUBROUTINE TRANS(A,B,II,JJ)
      DIMENSION A(II,JJ), B(JJ,II)
      DO 44 ROW = 1,II
      DO 44 COL = 1,JJ
     B(COL,ROW)=A(ROW,COL)
44    CONTINUE
      RETURN
      END

C     ******** Multiplication of a matrix by a scalar *******
      SUBROUTINE SCALARMUL(A,B,C,I,J)
      DIMENSION A(I,J),C(I,J)
      DO 99 II = 1,I
      DO 99 JJ = 1,J
      C(II,JJ) = B*A(II,JJ)
99    CONTINUE
      RETURN
      END

C     ********** Subtraction of two matrices ***************
      SUBROUTINE MATSUB(A,B,C,I,J)
      DIMENSION A(I,J),B(I,J),C(I,J)
      DO 66 II = 1,I
      DO 66 JJ = 1,J
      C(II,JJ) = A(II,JJ) - B(II,JJ)
66    CONTINUE
      RETURN
      END
```

```
C       ********** Addition of two matrices **************
        SUBROUTINE MATADD(A,B,C,I,J)
        DIMENSION A(I,J),B(I,J),C(I,J)
        DO 55 II = 1,I
        DO 55 JJ = 1,J
        C(II,JJ) = A(II,JJ) + B(II,JJ)
55      CONTINUE
        RETURN
        END
```

D.3 Control of a planar two link robotic manipulator using Optimal Control Method

1. This program is used to achieve the trajectory control and obtain the position and velocity gain values on the off-line basis using the Hookes and Jeeves non-linear optimization method. It makes use of the fourth order Runge-Kutta method (D.3.1).

D.3.1 Optimal control method (Non-linear optimization)

```
      DIMENSION XX(4)
      DIMENSION XSTRT(4),RMAX(4),RMIN(4),PHI(2),PSI(2),W(7000)
      DIMENSION RX(4),TH_DE1(25),TH_DE2(25),THD_DE1(25),THD_DE2(25)
      COMMON /LINKD/ RL1,RL2
      COMMON /RMASS/ RM1,RM2
      COMMON /DESIRED/ TH_DE1,TH_DE2,THD_DE1,THD_DE2
      COMMON /RK4C/ T,NEQ,DT,XX
      COMMON /SEEKC/ IDATA,IPRINT,NSHOT,NTEST,MAXM,F,G,TOL,
     1 ZERO,R,REDUCE
      COMMON ILOOP

      OPEN (UNIT = 11, FILE = 'desired.dat', STATUS = 'OLD')
      OPEN (UNIT = 12, FILE = 'result.out', STATUS = 'OLD')
      OPEN (UNIT = 13, FILE = 'actual.out', STATUS = 'OLD')
      OPEN (UNIT = 14, FILE = 'err_th.out', STATUS = 'OLD')
      OPEN (UNIT = 15, FILE = 'err_thd.out', STATUS = 'OLD')
      OPEN (UNIT = 16, FILE = 'kpkv.out', STATUS = 'OLD')
      OPEN (UNIT = 17, FILE = 'xstart.dat', STATUS = 'OLD')
      OPEN (UNIT = 18, FILE = 'inp.out', STATUS = 'OLD')

C     ****** Other variables ********
      RL1 = 0.3
      RL2 = 0.2
      RM1 = 4.0
      RM2 = 3.0
      T = 0.0
      XX(1) = -0.6354678
      XX(2) = 2.418859
      XX(3) = 0.0
      XX(4) = 0.0
      NEQ = 4
      DT = 0.001

C     ***** Optimization variables ****
      IDATA = 0
      IPRINT = 0
      N = 4
      F = 0.0001
      G = 0.0001
      NCONS = 0
      NEQUS = 0
      NPENAL = 5
      MAXM = 25000
      DATA RMAX/1.0E3,1.0E3,1.0E3,1.0E3/
      DATA RMIN/-1.0E3,-1.0E3,-1.0E3,-1.0E3/
```

```
C       DATA XSTRT/150.0,150.0,150.0,150.0/

        DO 10 IP = 1,25
        READ(11,*) TH_DE1(IP),TH_DE2(IP),THD_DE1(IP),THD_DE2(IP)
        WRITE(*,*) TH_DE1(IP),TH_DE2(IP),THD_DE1(IP),THD_DE2(IP)
10      CONTINUE

        DO 20 IN = 1,1
        READ(17,*) (XSTRT(J),J=1,4)
        WRITE(*,*) (XSTRT(J),J=1,4)

C       ***** Taking care of point 1 on the trajectory *********
        WRITE(13,*) -0.6354678,2.418859,0.0,0.0
        WRITE(14,*) 0.0,0.0
        WRITE(15,*) 0.0,0.0
        WRITE(16,*) (XSTRT(I),I=1,4)
        WRITE(18,*) -0.6354678,2.418859,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
C       ***********************************************************

        DO 30 ILOOP = 1,24
        CALL SEEK(N,NCONS,NEQUS,NPENAL,RMAX,RMIN,XSTRT,RX,U,PHI,PSI,
     1  NVIOL,W)
        CALL ANSWER(U,RX,PHI,PSI,N,NCONS,NEQUS)
        WRITE(12,*) ILOOP
        WRITE(12,*) U
        WRITE(12,*) (RX(I),I=1,4)
        WRITE(12,*) (XX(I),I=1,4)
        WRITE(13,*) (XX(I),I=1,4)
        WRITE(14,*) (TH_DE1(ILOOP+1)-XX(1)),(TH_DE2(ILOOP+1)-XX(2))
        WRITE(15,*) (THD_DE1(ILOOP+1)-XX(3)),(THD_DE2(ILOOP+1)-XX(4))
        WRITE(16,*) (RX(I),I=1,4)
        WRITE(18,*) (XX(I),I=1,4),(TH_DE1(ILOOP+1)-XX(1)),
     1  (TH_DE2(ILOOP+1)-XX(2)),(THD_DE1(ILOOP+1)-XX(3)),
     1  (THD_DE2(ILOOP+1)-XX(4))
30      CONTINUE
20      CONTINUE

        CLOSE(11)
        CLOSE(12)
        CLOSE(13)
        CLOSE(14)
        CLOSE(15)
        CLOSE(16)
        CLOSE(17)
        CLOSE(18)
        STOP
        END
```

```
C       ********* Definition of objective function *********
        SUBROUTINE UREAL(RX,U)
        DIMENSION XX(4),F(4),YI(4),YJ(4),YK(4),YL(4),UU(4)
        DIMENSION RX(1),TH_DE1(25),TH_DE2(25),THD_DE1(25),THD_DE2(25)
        COMMON /DESIRED/ TH_DE1,TH_DE2,THD_DE1,THD_DE2
        COMMON /RK4C/ T,NEQ,DT,XX
        COMMON ILOOP

        WRITE(*,*) (XX(I),I=1,4)
        DO 50 LOOP = 1,1000
        CALL RK4(T,DT,NEQ,XX,F,YI,YJ,YK,YL,UU,RX,ILOOP)
50      CONTINUE

        QW1 = ((TH_DE1(ILOOP+1) - XX(1))*10.0)**2
        QW2 = ((TH_DE2(ILOOP+1) - XX(2))*10.0)**2
        QW3 = (THD_DE1(ILOOP+1) - XX(3))**2
        QW4 = (THD_DE2(ILOOP+1) - XX(4))**2
        U = (QW1+QW2+QW3+QW4)*1.0e6
C       EDISP = 100.0*((TH_DE1(ILOOP+1)-XX(1))**2+(TH_DE2(ILOOP+1)-XX(2))**2)
C       EVEL = 100.0*((THD_DE1(ILOOP+1)-XX(3))**2+(THD_DE2(ILOOP+1)-XX(4))**2)
C       U = (EDISP + EVEL)*1.0e5
        RETURN
        END

C       ****** Fourth order Runge-Kutta method ***********
        SUBROUTINE RK4(T,DT,N,XX,F,XI,XJ,XK,XL,UU,RX,ILOOP)
        DIMENSION XI(N),XJ(N),XK(N),XL(N),UU(N),XX(N),F(N)
        DIMENSION RX(4)
        DO 10 I = 1,N
10      UU(I) = XX(I)
        CALL FUN(XX,F,N,T,RX,ILOOP)
        DO 20 I = 1,N
        XI(I) = F(I) * DT
20      XX(I) = UU(I) + XI(I)/2.0
        T = T+DT/2.0
        CALL FUN(XX,F,N,T,RX,ILOOP)
        DO 30 I = 1,N
        XJ(I) = F(I)*DT
30      XX(I) = UU(I)+XJ(I)/2.0
        CALL FUN(XX,F,N,T,RX,ILOOP)
        DO 40 I = 1,N
        XK(I) = F(I)*DT
40      XX(I) = UU(I)+XK(I)
        T = T+DT/2.0
        CALL FUN(XX,F,N,T,RX,ILOOP)
        DO 50 I = 1,N
        XL(I) = F(I)*DT
```

```
50      XX(I) = UU(I)+(XI(I)+2.0*XJ(I)+2.0*XK(I)+XL(I))/6.0
        RETURN
        END

C       ******* Function definition for the Runge-Kutta method **********
        SUBROUTINE FUN(XX,F,N,T,RX,ILOOP)
        DIMENSION XX(N),F(N)
        DIMENSION TORQ(2),RX(4)
        DIMENSION QM(2,2),QV(2),QG(2),QT1(2),QT2(2)
        DIMENSION THDDTI(2),QMI(2,2)
        DIMENSION TH_DE1(25),TH_DE2(25),THD_DE1(25),THD_DE2(25)
        COMMON /LINKD/ RL1,RL2
        COMMON /RMASS/ RM1,RM2
        COMMON /DESIRED/ TH_DE1,TH_DE2,THD_DE1,THD_DE2
C       WRITE(*,*) (RX(I),I=1,4)
        GH = 9.81
        TH_CP1 = XX(1)
        TH_CP2 = XX(2)
        THD_CP1 = XX(3)
        THD_CP2 = XX(4)

        TORQ(1) =  RX(1)*(TH_DE1(ILOOP+1) - TH_CP1) +
     1  RX(3)*(THD_DE1(ILOOP+1) - THD_CP1)
        TORQ(2) =  RX(2)*(TH_DE2(ILOOP+1) - TH_CP2) +
     1  RX(4)*(THD_DE2(ILOOP+1) - THD_CP2)

        QM(1,1) = RL2**2*RM2 + 2.0*RL1*RL2*RM2*COS(TH_CP2)
     1  +  RL1**2*(RM1 + RM2)
        QM(1,2) = RL2**2*RM2 + RL1*RL2*RM2*COS(TH_CP2)
        QM(2,1) = RL2**2*RM2 + RL1*RL2*RM2*COS(TH_CP2)
        QM(2,2) = RL2**2*RM2
        QV(1) = -RM2*RL1*RL2*SIN(TH_CP2)*THD_CP2**2 -
     1  2.0*RM2*RL1*RL2*SIN(TH_CP2)*THD_CP1*THD_CP2
        QV(2) = RM2*RL1*RL2*SIN(TH_CP2)*THD_CP1**2
        QG(1) = RM2*RL2*GH*COS(TH_CP1+TH_CP2) +
     1  (RM1 + RM2)*RL1*GH*COS(TH_CP1)
        QG(2) = RM2*RL2*GH*COS(TH_CP1 + TH_CP2)
        CALL QMINV(QM,QMI)
        CALL VECADD(QV,QG,QT1,2)
        CALL VECSUB(TORQ,QT1,QT2,2)
        CALL RMATVEC(QMI,QT2,THDDTI,2,2)
        F(1) = XX(3)
        F(2) = XX(4)
        F(3) = THDDTI(1)
        F(4) = THDDTI(2)
        RETURN
        END
```

```
C       ***** Subroutine to find the inverse of a 2 x 2 matrix ******
        SUBROUTINE QMINV(QM,QMI)
        DIMENSION QM(2,2),QMI(2,2)
        DO 410 I = 1,2
        DO 410 J = 1,2
        QMI(I,J) = 0.0
410     CONTINUE
        QMP = QM(1,1)*QM(2,2) - QM(1,2)*QM(2,1)
        QMI(1,1) = QM(2,2)/QMP
        QMI(1,2) = -QM(1,2)/QMP
        QMI(2,1) = -QM(2,1)/QMP
        QMI(2,2) = QM(1,1)/QMP
        RETURN
        END

C       ***** Addition of two vectors ******
        SUBROUTINE VECADD(A,B,C,M)
        DIMENSION A(M),B(M),C(M)
        DO 4256 I = 1,M
        C(I) = A(I) + B(I)
4256    CONTINUE
        RETURN
        END

C       ***** Subtraction the two vectors ******
        SUBROUTINE VECSUB(A,B,C,M)
        DIMENSION A(M),B(M),C(M)
        DO 4169 I = 1,M
        C(I) = A(I) - B(I)
4169    CONTINUE
        RETURN
        END

C       ***** Multiply a matrix with a vector ******
        SUBROUTINE RMATVEC(A,B,C,M,N)
        DIMENSION A(M,N),B(N),C(M)
        DO 4430 I = 1,M
        C(I) = 0.0
        DO 4440 J = 1,N
        C(I) = C(I) + A(I,J)*B(J)
4440    CONTINUE
4430    CONTINUE
        RETURN
        END

C       ******* Inequality constraints ********
        SUBROUTINE CONST(RX,NCONS,PHI)
```

```
      DIMENSION RX(1),PHI(1)
      RETURN
      END

C     ******* Equality constraints ********
      SUBROUTINE EQUAL(RX,PSI,NEQUS)
      DIMENSION RX(1),PSI(1)
      RETURN
      END
```

D.4 Control of a planar two link robotic manipulator using Artificial Neural Network Method (LPN Method)

1. Several sets of input and output vectors are computed using the optimal control method. This program uses the LPN method and does the training in such a way that different weight matrices are generated for all the 25 points along the trajectory. Then the weight matrix [W] is used on-line to evaluate the output parameters (gain values) corresponding to unknown set of input values. The following program depicts the generation of the weight matrix [W] for one point along the trajectory (D.4.1).

D.4.1 A LP approach for neural networks

```
C     ********************************************************
C     YW = N + 4*(N + NCONS + NEQUS)
C       WHERE N     = NUMBER OF DESIGN VARAIBLES = 1
C             NCONS = NUMBER OF INEQUALITY CONSTRAINTS = 1
C           NEQUS = NUMBER OF EQUALITY CONSTRAINTS = 1
C     YW = 1 + 4*(1+1+1) = 13
C              L.P.
C     ZW = M*(5+M)
C       - WHERE M IS NUMBER OF CONSTRAINING EQUATIONS = ROWS
C                 WHICH IS DEFINED BELOW
C     ********************************************************
C              ZW = 36*(5+36) = 1476
C     ********************************************************
C          DIMENSIONS FOR [A] MATRIX
C          ROWS =  (NO.OF OUTPUT)*TOTAL.NO.OF TRIALS
C                  =  4 * (9) = 36
C          COLUMNS = (NO.OF INPUT*2 + 4)*NO.OF OUTPUT
C                          = (8*2 + 4)*4 = 80
C     ********************************************************

          DIMENSION YX(1),YXSTRT(1),RMAX(1),RMIN(1),PHI(1),PSI(1)
          DIMENSION YW(50),ZX(80),ZA(36,80),ZB(36),ZC(80)
          DIMENSION ZAP(36,80),ZCP(80),ZBP(36),ZW(1476)
          DIMENSION BBW(4,8),BBC(4),BBD(4),ZTOB(8)
          COMMON
     1  /SEEKC/IDATA,IPRINT,NSHOT,NTEST,MAXM,F,G,TOL,ZERO,
     2   R,REDUCE
          COMMON /BL1/ZC,ZX,ZW,ZAP,ZCP,ZBP
          COMMON /BL3/ZB
          COMMON /BL2/MZ,NZ,NUTS,NT,NI,NOUT,NOI
          COMMON /FAL/ZTOB
          COMMON /ANT/ZA
          OPEN(2, FILE = 'scout.dat', STATUS = 'OLD')
          OPEN(3, FILE = 'clpga.out', STATUS = 'OLD')
          OPEN(14, FILE = 'lin.dat', STATUS = 'OLD')
          READ(14,*)MZ
          READ(14,*)NZ
          READ(14,*)NUTS
          READ(14,*)NT,NI
          READ(14,*)NOUT,NOI
          MAXM = 10000
          CALL ALPG
     1    WRITE(*,*)
          DO 196 I = 1,MZ
          READ(2,*)ZB(I)
```

```
196   CONTINUE
      READ(3,*)(ZC(I),I = 1,NZ)
      F = 0.001
      G = 0.01
      NY = 1
      NCONS = 0
      NEQUS = 0
      NPENAL = 3
      IDATA = 0
      DATA RMAX/0.10/
      DATA RMIN/-0.10/
      DATA YXSTRT/0.0013427/
      NOISE = 1
      CALL
     1 SEEK(NY,NCONS,NEQUS,NPENAL,RMAX,RMIN,YXSTRT,YX,YU,PHI,
     2 PSI,NVIOL,YW)
      CALL ANSWER(YU,YX,PHI,PSI,NY,NCONS,NEQUS)
      WRITE(*,*)'DO YOU WISH TO CONTINUE ?'
      WRITE(*,*)'1:  YES '
      WRITE(*,*)'2: NO '
      READ(*,*)NDEC
      IF(NDEC.EQ.1) THEN
      GO TO 1
      ELSE
      END IF
      STOP
      END

      SUBROUTINE UREAL(YX,YU)
      DIMENSION YX(1),YXSTRT(1),RMAX(1),RMIN(1),PHI(1),PSI(1)
      DIMENSION YW(50),ZX(80),ZA(36,80),ZB(36),ZC(80)
      DIMENSION ZAP(36,80),ZBP(36),ZCP(80)
      DIMENSION ZW(1476),ZTOB(8)
      DIMENSION BBW(4,8),BBC(4),BBD(4),D(2)
      DIMENSION SAK(4),RT3(4),ZZTOB(4)
      DIMENSION RAJ1(8),RAJ2(4)

      COMMON /BL1/ZC,ZX,ZW,ZAP,ZCP,ZBP
      COMMON /BL3/ZB
      COMMON /BL2/MZ,NZ,NUTS,NT,NI,NOUT,NOI
      COMMON /SIMPLEC/NSTOP,IDATA,NNDEX
      COMMON /PAKS/BBW,BBC,BBD
      COMMON /FAL/ZTOB
      COMMON /ANT/ZA
      COMMON /DEPUT/D,RMX,RMN
      OPEN(15, FILE = 'scinp.dat', STATUS = 'OLD')
      OPEN(9, FILE =  'scout.dat', STATUS = 'OLD')
```

```
      OPEN(19, FILE = 'perc.mat', STATUS = 'OLD')
      OPEN(18, FILE = 'obj.mat', STATUS = 'OLD')
      OPEN(21, FILE = 'myinp.dat', STATUS = 'OLD')
      OPEN(22, FILE = 'myout.dat', STATUS = 'OLD')
      OPEN(23, FILE = 'compare.out', STATUS = 'OLD')
      IDATA = 0
      NSTOP = 3000
      NNDEX = 1
      DO 20 I = 1,MZ
      KR = 1
      DO 30 J = 1,NZ
      IF(J.GT.(20*KR)) THEN
      KR = KR + 1
      END IF
      IF((J.GE.(1+20*(KR-1))).AND.(J.LT.(17+20*(KR-1)))) THEN
      ZAP(I,J) = YX(1)*ZA(I,J)
      ELSE
      ZAP(I,J) = ZA(I,J)
      END IF
30    CONTINUE
20    CONTINUE
      SQM = 0.0
      DO 40 I = 1,MZ
      ZBP(I) = YX(1)*ZB(I)
      SQM = SQM + ZB(I)
40    CONTINUE
      KR = 1
      DO 50 J = 1,NZ
      IF(J.GT.(20*KR)) THEN
      KR = KR + 1
      END IF
      IF((J.GE.(1 + 20*(KR-1))).AND.(J.LT.(17+20*(KR-1)))) THEN
      ZCP(J) = YX(1)*ZC(J)
      ELSE
      ZCP(J) = ZC(J)
      END IF
50    CONTINUE
      CALL SIMPLE(NZ,MZ,ZAP,ZBP,ZCP,ZX,ZU,ZW)
      XX = 0.0
      DO 501 J = 1,NZ
      XX = XX + ZCP(J)*ZX(J)
501   CONTINUE
      YYU = SQM - XX
      WRITE(*,*)'L.P. OBJECTIVE FUNCTION IS',YYU
C     CALL WASS(ZX,BBW,BBC,BBD)
      YT1 = 0.0
      YT2 = 0.0
```

```
      YT3 = 0.0
      DO 55 NM = 1,16,2
      DO 60 MM = NM,80,20
      YT1 = YX(1)*(ZC(NM)*ZX(MM)) + YT1
60    CONTINUE
55    CONTINUE
      DO 65 NM = 2,16,2
8     DO 70 MM = NM,80,20
      YT2 = YX(1)*(ZC(NM)*ZX(MM)) + YT2
70    CONTINUE
65    CONTINUE
      DO 75 NM = 17,20
      DO 80 MM = NM,80,20
      YT3 = (ZC(NM)*ZX(MM)) + YT3
80    CONTINUE
75    CONTINUE
      YT4 = YX(1)*SQM
C     CALL RMIXD(BBW,BBC,BBD,YX,ZTOB,RT3)
      YU = (YT1 + YT2 + YT3 - YT4)**4
      WRITE(*,*)'SLOPE IS',YX(1)
      WRITE(*,*)'SEEK OBJECTIVE FUNCTION IS',YU
      IF (NSW.EQ.1) THEN
      STOP
      END IF
      IF(YU.LT.(6.0E-6)) THEN
      DO 881 ISK = 1,9
      DO 884 I = 1,8
      READ(15,*)ZTOB(I)
884   CONTINUE
      DO 886 I = 1,4
      READ(9,*)ZZTOB(I)
886   CONTINUE

      DO 887 I = 1,8
      READ(21,*) RAJ1(I)
887   CONTINUE
      DO 888 I = 1,4
      READ(22,*) RAJ2(I)
888   CONTINUE

      CALL WASS(ZX,BBW,BBC,BBD)
      CALL RMIXD(BBW,BBC,BBD,YX,RAJ1,RT3)
      DO 923 JK = 1,4
      WRITE(23,*)RAJ2(JK),RT3(JK)
      WRITE(18,*)RT3(JK)
923   CONTINUE
881   CONTINUE
```

```
      NSW = 1
      ELSE
      END IF
      RETURN
      END

      SUBROUTINE CONST(YX,NCONS,PHI)
      DIMENSION PHI(1),D(2),YX(1)
      COMMON /DEPUT/D,RMX,RMN
      NOISE = NOISE + 1
      RETURN
      END

      SUBROUTINE EQUAL(YX,PSI,NEQUS)
      DIMENSION YX(1),PSI(1)
      RETURN
      END

      SUBROUTINE RMATMUL(A,B,C,M,N)
      DIMENSION A(M,N),B(N),C(M)
      DO 4430 I = 1,M
      C(I) = 0.0
      DO 4440 J = 1,N
      C(I) = C(I) + A(I,J)*B(J)
4440  CONTINUE
4430  CONTINUE
      RETURN
      END

C     **** Generation of [A] matrix for linear programming *******
      SUBROUTINE ALPG
      DIMENSION X(8),ZA(36,80)
      COMMON /ANT/ZA
      OPEN(17, FILE = 'scinp.dat', STATUS = 'OLD')
      DO 1050 NI = 1,36,4
      KNI = NI
      NS = 1
      NT = 16
      DO 714 I = 1,8
      READ(17,*)X(I)
714   CONTINUE
780   CONTINUE
      IF(NT.GT.80) GOTO 7100
      MT = 1
      DO 710 I = NS,NT,2
      ZA(KNI,I) = X(MT)
      MT = MT + 1
```

```
710   CONTINUE
      MT = 1
      DO 720 I = NS+1,NT,2
      ZA(KNI,I) = -X(MT)
      MT = MT + 1
720   CONTINUE
      DO 730 I = NT+1,NT+4
      ZA(KNI,I) = 1.0
730   CONTINUE
      DO 740 I = NT+2,NT+3
      ZA(KNI,I) = -1.0
740   CONTINUE
      KNI = KNI + 1
      NS = NS + 20
      NT = NT + 20
      GO TO 780
7100  CONTINUE
1050  CONTINUE
      RETURN
      END

      SUBROUTINE WASS(ZX,BBW,BBC,BBD)
      DIMENSION ZX(80),BBW(4,8),BBC(4),BBD(4)
      ND = 1
      RAMYA = 0.0
      DO 432 I = 1,4
      DO 433 J = 1,8
      RAMYA = ZX(ND) - ZX(ND + 1)
      BBW(I,J) = RAMYA
      ND = ND + 2
433   CONTINUE
      BBC(I) = ZX(ND) - ZX(ND + 1)
      ND = ND + 2
      BBD(I) = ZX(ND) - ZX(ND + 1)
      ND = ND + 2
432   CONTINUE
      RETURN
      END

      SUBROUTINE RMIXD(BBW,BBC,BBD,YX,ZTOB,RT3)
      DIMENSION YX(1),CBC(4),CBD(4),RT1(4),RT2(4),RT3(4)
      DIMENSION ZTOB(8),BBW(4,8),BBC(4),BBD(4)
      DO 2339 I = 1,4
      CBC(I) = 0.0
      CBD(I) = 0.0
      RT1(I) = 0.0
      RT2(I) = 0.0
```

```
      RT3(I) = 0.0
2339  CONTINUE
      CALL RMATMUL(BBW,ZTOB,RT1,4,8)
      DO 2340 I = 1,4
      CBC(I) = (1.0/YX(1))*BBC(I)
      CBD(I) = (1.0/YX(1))*BBD(I)
2340  CONTINUE
      CALL RMATADD(RT1,CBC,RT2,4)
      CALL RMATSUB(RT2,CBD,RT3,4)
      WRITE(18,*)(RT3(JK),JK=1,4)
      RETURN
      END

C     ******* Addition of twc vectors ******
      SUBROUTINE RMATADD(A,B,C,M)
      DIMENSION A(M),B(M),C(M)
      DO 4256 I = 1,M
      C(I) = A(I) + B(I)
4256  CONTINUE
      RETURN
      END

C     ******* Subtraction of two vectors ******
      SUBROUTINE RMATSUB(A,B,C,M)
      DIMENSION A(M),B(M),C(M)
      DO 4169 I = 1,M
      C(I) = A(I) - B(I)
4169  CONTINUE
      RETURN
      END

C     ***********************************************
C     GENERATION OF [C] MATRIX FOR LINEAR PROGRAMMING
C     ***********************************************
      DIMENSION X1(8),X2(8),X3(8),X4(8),X5(8)
      DIMENSION X6(8),X7(8),X8(8),X9(8)
      DIMENSION A(1,80),Y(8)
      OPEN(1, FILE = 'scinp.dat', STATUS = 'OLD')
      OPEN(2, FILE = 'clpga.out', STATUS = 'OLD')
      DO 14 I = 1,8
      READ(1,*)X1(I)
14    CONTINUE
      DO 15 I = 1,8
      READ(1,*)X2(I)
15    CONTINUE
      DO 16 I = 1,8
      READ(1,*)X3(I)
```

```
16      CONTINUE
        DO 17 I = 1,8
        READ(1,*)X4(I)
17      CONTINUE
        DO 18 I = 1,8
        READ(1,*)X5(I)
18      CONTINUE
        DO 19 I = 1,8
        READ(1,*)X6(I)
19      CONTINUE
        DO 20 I = 1,8
        READ(1,*)X7(I)
20      CONTINUE
        DO 21 I = 1,8
        READ(1,*)X8(I)
21      CONTINUE
        DO 22 I = 1,8
        READ(1,*)X9(I)
22      CONTINUE
        DO 44 K = 1,8
        Y(K) = X1(K)+X2(K)+X3(K)+X4(K)+X5(K)+X6(K)+X7(K)
     1  + X8(K) + X9(K)
44      CONTINUE
        NS = 1
        NT = 16
80      CONTINUE
        IF(NT.GT.80) GOTO 100
        MT = 1
        DO 310 I = NS,NT,2
        A(1,I) = Y(MT)
        MT = MT + 1
310     CONTINUE
        MT = 1
        DO 320 I = NS+1,NT,2
        A(1,I) = -Y(MT)
        MT = MT + 1
320     CONTINUE
        DO 330 I = NT+1,NT+4
        A(1,I) = 9.0
330     CONTINUE
        DO 340 I = NT+2,NT+3
        A(1,I) = -9.0
340     CONTINUE
        NS = NS + 20
        NT = NT + 20
        GO TO 80
100     CONTINUE
```

```
1050  CONTINUE
      WRITE(2,*)(A(1,J),J = 1,80)
      STOP
      END
```