NEAR-MINIMUM-TIME CONTROL OF A ROBOT MANIPULATOR

By

Tao Fan

B. Eng., Xian Jiaotong University, 1991

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF MECHANICAL ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

June 2003

Department of Mechanical Engineering
The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1Z1

Date: July 21, 2003

Abstract

This thesis deals with the problem of minimum time control of a rigid robot manipulator with point-to-point motion subject to constraints on the control inputs. Due to the nonlinear and coupled dynamics of the robot manipulator, finding minimum time strategies is algorithmically difficult and computationally very intensive, even when the dynamic equations and parameters of the manipulator are precisely known. As a result, the practical applicability of the available methods currently is very limited.

In this research, we assume the control inputs are always bang-bang and switch once. Using the Principle of Work and Energy, a simple and practical "zero-net-work" searching approach is proposed. The proposed method focuses on changes in the manipulator's kinetic energy during the time optimal motion, instead of concentrating on the system's state variables, as is usually done in conventional approaches. The "zero-net-work" method is used to develop the controllers for one-link manipulators, a 3-degree of freedom cylindrical manipulator and a two-degree of freedom revolute manipulator. The results show that if the structure of the exact minimum time control is bang-bang with a single switch, using the "zero-net-work" method we will get the exact minimum time solution. If the exact minimum time control has more than one switch, using the "zero-net-work" method we will get a near-minimum-time solution. The major advantages of the proposed method are that it does not require initial boundary value guesses and is computationally efficient.

Table of Contents

List of Tables

List of Figures

Acknowledgement

I wish to thank Dr. D. B. Cherchas for his advice and help throughout my work on this thesis. Partial financial support for the work was obtained from NSERC Research Grant A4682.

Nomenclature

$\mathbf{x}$	State vector.
$\mathbf{x}^*$	Optimal state vector.
$\mathbf{x}_{i0}$	Initial state of link i.
$\mathbf{x}_{if}$	Final state of link i.
θ	Coordinate for rotational link.
θ_i	Position of rotational link i.
$\mathbf{q}$	Joint position vector.
$\dot{\mathbf{q}}$	Joint velocity vector.
d	Coordinate for prismatic link.
$\mathbf{a}_{c,i}$	Acceleration of the center of mass of link i.
$\mathbf{a}_{e,i}$	Acceleration of the end of link i.
$\omega_{\mathbf{i}}$	Angular velocity of frame i w.r.t. frame i.
$\alpha_{\mathbf{i}}$	Angular acceleration of frame i w.r.t. frame i.
$\mathbf{g}_i$	Acceleration due to gravity(expressed in frame i).
$\mathbf{f}_i$	Force exerted by link i-1 on link i.
τ_i	Torque exerted by link i-1 on link i.
$\mathbf{R}_i^{i+1}$	Rotation matrix from frame i+1 to frame i.
$\mathbf{H}_{ij}$	Homogeneous transformation from frame i to frame j.
m_i	Mass of link i.
l_i	Length of link i.
I_i	Inertia matrix of link i about the center of mass of link i.
$\mathbf{r}_{i,ci}$	Vector from joint i to the center of mass of link i.
$\mathbf{r}_{i+1,ci}$	Vector from joint i+1 to the center of mass of link i.

$\mathbf{r}_{i,i+1}$	Vector from joint i to joint i+1.
$\mathbf{u}$	Control input vector.
$\mathbf{u}^*$	Optimal control input vector.
$\mathbf{u}_{max}$	Maximum control input vector.
τ	Joint torque vector.
τ_{max}	Maximum joint torque vector.
$\mathbf{F}_i$	Control force of link i.
J	Cost functional.
λ	Adjoint variable vector.
H	Hamiltonian function.
L	Lagrangian.
KE	Kinetic energy of the system.
PE	Potential energy of the system.
W_{Ai}	Work done by the actuator of link i.
W_{Ri}	Work done by reaction forces of link i.
W_{Gi}	Work done by gravity of link i.
$\mathbf{M}$	Manipulator inertia matrix.
$\mathbf{C}$	Coriolis-centrifugal matrix.
$\mathbf{G}$	Gravity matrix.
$\mathbf{S}$	Switching function vector.
t_0	Initial time.
t_f	Final time.
t_{sw}	Switching time.

Chapter 1

Introduction

1.1 Review of Literature

The problem of minimum time control of a robotic manipulator concerns the determination of control efforts and corresponding trajectories that will drive the end-effector of a manipulator from a given initial position to a specified desired position in minimum time while satisfying constraints e.g. the limits on the actuator efforts.

Kahn and Roth (1971) first studied this problem. Since then, the interest in this problem has increased tremendously. The problem can be divided into two categories in terms of different constraints on the manipulator motion:

a) There are constraints on the intermediate configurations of the robot arm so that the manipulator either follows a specified path or performs the movement avoiding collisions with obstacles in the work space.

b) The motion path space is obstacle-free and unconstrained between the two end points.

For the first category, some algorithms have been proposed for minimum time control of robot manipulators along a specified geometric path (Shin and Mckay 1985; Bobrow et al. 1985; Chen and Desrochers 1989; Shiller and Dubowsky 1991). Improvements in computational efficiency of the algorithm were studied by Slotine and Yang (1989). With given paths, 2n dimensional (n is the number of the links) trajectory planning problems reduce to simple 2 dimensional searching problems.

Solving the minimum time control problem for the second category represents a difficult and elusive point-to-point minimum time control problem. The approaches to this problem can be categorized into two groups:

a) Dynamic programming or space search methods.

b) Calculus of variation based methods.

The first group includes some approximation methods, such as artificial intelligence (AI) approaches (Sahar and Hollerbach 1985; Rajan 1985; Shiller and Dubowsky 1988) and nonlinear parameter optimization methods (Bobrow 1988). The basic idea is the use of an approximation of an initial feasible trajectory in conjunction with an exhaustive search to determine the unconstrained motions between two end points. In general, the algorithm for this class of approaches consists of three steps:

a) Characterize the path in some manner (e.g. spline function);

b) Given a path determine the time optimal trajectory by using existing algorithms (e.g. Bobrow's algorithm 1988; or Shin's algorithm 1985);

c) Vary the path until the minimum time path and trajectory are obtained.

Shiller and Dubowsky (1988) proposed a practical method to obtain the global time optimal motions of robotic manipulators. They extend the predefined path methods (e.g. Bobrow et al. 1985; Shin and Mckay 1985) to point-to-point problems. In their work, a set of best paths are obtained first in a global search over manipulator workspace, using graph search and hierarchical pruning techniques; and then vary the path under certain criteria to search for the shortest travel time between two given configurations.

Following the same idea, a technique was developed by Bobrow (1988). Similar to (Rajan 1985), in his work the Cartesian path along cubic B-splines and the shape of this path is varied in a manner that minimizes the travel time using a nonlinear parameter optimization algorithm. For any set of B-spline vertices (parameters to be determined) the algorithm in (Bobrow et al. 1985) was used to evaluate the minimum final time. In order to speed up convergence and improve computational efficiency for those existing optimization methods, a close initial guess for the path is necessary.

The calculus of variation based methods use the techniques of variational calculus and apply Pontryagin's maximum principle to reduce the time optimization problem to a Two Point Boundary Value Problem (TPBVP). The difficulty in finding a solution for the TPBVP problem is that state and costate variables at the initial and final times are only partially known, hence initial values must be guessed to match the correct set of final values. Since most of the robot manipulators are nonlinear systems, the numerical methods used to solve the nonlinear TPBVP were found to be highly sensitive to the initial guess of the costate values. However, it is extremely difficult to have a good guess of the initial costate values intuitively, because the costate variables do not have any simple interpretation or physical meaning. In general, time optimal control solutions are very difficult to obtained by direct solving the TPBVP.

Kahn and Roth (1971) first studied the problem. A near minimum time solution for a three-degree of freedom articulated arm was obtained by using Pontryagin's maximum principle. The major portion of this work consisted of finding a closed-loop regulation around the time optimal nominal trajectory using linearization, gravity compensation, and averaged compensation of the effects of angular speeds. The possibility of singular time optimal control was not investigated.

Weinreb and Bryson (1985) were the first to obtain substantive information about the minimum time behavior of the two-link manipulator with open initial conditions. They did not assume that the optimal controls were bang-bang, but results from Weireb's adjusted control weight (ACW) algorithm, which was based on Bryson's steepest descent algorithm (Bryson 1975), indicated that most cases tended toward bang-bang. Because the ACW program was a continuous function optimization code, it was unable to achieve sharp discontinuities in the controls, and it was computationally intensive.

Wen and Desrochers (1986) examined a sub-optimal time control algorithm which utilizes the concept of "Averaged Dynamics" (Kin and Shin 1985). It uses all available dynamics information of the current and final states to update the dynamics continually at each sampling interval. Unfortunately there is no guarantee that the system will achieve the desired final state with this method.

In Geering's paper (1986), time optimal trajectories for a cylindrical and a spherical robot, and a robot with a horizontal articulated arm with two links were obtained by using the shooting algorithm and a parameter optimization method. It was pointed out that the most important physical effects responsible for the nature of the time optimal solution are: (a) Minimization of the mass moment of inertia with respect to revolute joints; (b) Enhancement of accelerations and decelerations by exploiting reaction torques and forces.

Chen and Desrochers (1988) proved that for an n degree of freedom robot system with no path constraints on the motion, the structure of the optimal control requires that at least one of the control torques be always in saturation on every finite time interval. In other words, not all control torques are simultaneously singular (where the control variables are not bang-bang) on the time interval.

Meier and Bryson (1990) suggest a fast algorithm for a planar two-link manipulator to travel a specified distance in minimum time with initial and final position unspecified. The method reduces the computation time by forcing the control inputs to be 'bang-bang' on both regular and singular intervals, and thus makes it feasible to find minimum time solutions for various initial and final positions and various dynamic properties of a robot.

Lin (1992) proposed an efficient two-level (master and slave) parallel algorithm for solving the general time optimal problem in discrete form. However, the convergence of the algorithm was shown under the following two conditions:

a) The step size in the solution to the master problem is small enough.

b) The initial guess of final time is sufficiently close to the minimum final time.

The second condition is in general not easily satisfied since the knowledge of the final time is not known a priori.

Chen et al. (1993) develop a general procedure for computing the time optimal trajectory for robotic point-to-point motion based on perturbation and continuation methods. Both non-singular and singular cases can be handled by converting the original time optimal problem into a perturbed time optimal problem. The algorithm is composed of two phases: initialization and refinement. In the initialization phase, a Two Point Boundary Value Problem (TPBVP) resulting from the perturbed time optimal problem is solved for an appropriately chosen perturbation parameter. Then, the solution obtained from the initialization phase is refined by solving a set of Initial Value Problems (IVP) sequentially and/or in parallel until the desired solution is achieved. The method is computationally efficient since the resulting TPBVP is solved once for a large perturbation parameter and the remaining problem becomes solutions to a set of IVP sub-problems.

In Yang (1994) , a "two-step" Lyapunov approach generating near-minimum-time point-to-point motion of a robot manipulator is proposed, motivated by the physics of the minimum time motion. The method is based on the observation of the kinetic energy mechanisms in minimum time operations of robot manipulators.

In summary, it is extremely difficult, if not impossible, to obtain an exact closed-form solution to the problem of point-to-point minimum time control of robotic manipulators, due to the nonlinearity and the coupled nature of the robotic manipulator dynamics. The available numerical solutions are computationally very intensive, require good initial boundary values guesses, or are sensitive to the time-step or some other variables in the numerical procedures. Also the numerical solution is essentially an open-loop control and does not accommodate any system disturbance or parameter uncertain. The complete structure of the solution of exact minimum time motion of robotic manipulators is currently still unclear. In particular, the question of existence of time optimal controls containing singular arcs is open. As a result, the practical applicability of available methods is very limited.

1.2 Purpose and Scope of This Work

For most manufacturing tasks, it is desirable to move a manipulator at its highest speed to minimize the task cycle time. This project is significant in the sense that the execution time for robot tasks and industrial productivity are directly related, productivity may be increased significantly by fully exploiting the potential of existing robots. Furthermore, this is an important problem in the implementation of automatic programming for intelligent robots. The short term objective of this work is the determination of a minimum time control algorithm for robot manipulator motion. The long term objective is the reduction in execution time in the performance of industrial robot tasks and subsequent improvement in industrial productivity.

This thesis deals with time optimal point-to-point motion (obstacle-free and unconstrained motion paths between two endpoints) control problem for robot manipulators with actuator bounds. We assume the friction forces can be neglected compare with the control inputs. For simplicity, we assume the initial and final velocity of each link equal to zero. In this thesis, an approximation to the optimal control which results in a near-minimum-time control will be developed. While not producing exactly minimum time solutions, the algorithm can be easily used in practical applications to generate the near-minimum-time control inputs of robot manipulators.

Chapter 2

Problem Formulation and Solution Guidelines

2.1 Review of Conventional Variational Calculus Method

2.1.1 Optimal Control Problem for General Dynamic Systems

Suppose the plant is described by the nonlinear time-varying dynamic equation

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \tag{2.1}$$

with state vector $\mathbf{x}(t) \in R^n$ and control input vector $\mathbf{u}(t) \in R^m$.

The performance index is

$$J(\mathbf{x}, \mathbf{u}, t) = \phi(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} L(\mathbf{x}(t), \mathbf{u}(t), t)dt \tag{2.2}$$

where $[t_0, t_f]$ is the time interval of interest. The final time weighting function $\phi(\mathbf{x}(t_f), t_f)$ is a function of the final state. The weighting function $L(\mathbf{x}(t), \mathbf{u}(t), t)$ depends on the state and input at intermediate times in $[t_0, t_f]$. The performance index is selected to make the plant exhibit a desired type of performance. The optimal control problem is to find the input $\mathbf{u}^*(t)$ on the time interval $[t_0, t_f]$ that drives the plant given in Equation (2.1) along a trajectory $\mathbf{x}^*(t)$ such that the cost function in Equation (2.2) is minimized.

To solve this problem, we shall use Lagrange multipliers to adjoin the constraints given in Equation (2.1) to the performance index in Equation (2.2). Since Equation (2.1) holds at each $t \in [t_0, t_f]$, we require an associated multiplier $\lambda(t) \in R^n$, which is a function of time. If we define the Hamiltonian function as:

$$H(\mathbf{x}, \mathbf{u}, \lambda, t) = L(\mathbf{x}, \mathbf{u}, t) + \lambda^T \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \tag{2.3}$$

Using the variational calculus analysis as in Lewis (1986) to minimize J, the necessary conditions for the optimal controller are :

State equation:

$$\dot{\mathbf{x}} = \frac{\partial H}{\partial \lambda} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \tag{2.4}$$

Costate equation:

$$\dot{\lambda} = -\frac{\partial H}{\partial \mathbf{x}} \tag{2.5}$$

Boundary conditions:

$$\mathbf{x}(t_0) = \mathbf{x}_0 \tag{2.6}$$

$$(\phi_x - \lambda)^T \,|_{t_f}\, \delta\mathbf{x}(t_f) + (\phi_t + H)\,|_{t_f}\, \delta t_f = 0 \tag{2.7}$$

and if the control is unconstrained, the stationarity condition is:

$$\frac{\partial H}{\partial \mathbf{u}} = 0 \tag{2.8}$$

If the control $\mathbf{u}(t)$ is constrained to lie in an admissible region $\mathbf{u}(t) \in \mathbf{\Omega}$, which for example, might be defined by a requirement that its magnitude be less than a given value, the stationarity condition is:

$$H(\mathbf{x}^*, \mathbf{u}^*, \lambda^*, t) \leq H(\mathbf{x}^*, \mathbf{u}, \lambda^*, t), \quad \textit{for all admissible } \mathbf{u}. \tag{2.9}$$

where * denotes optimal quantities. The optimality requirement (2.9) is called *Pontryagin's maximum principle*: "The Hamiltonian must be minimized over all admissible u for optimal values of the state and costate".

The optimal control signal is:

$$\mathbf{u}^*(\mathbf{x}, t) = \arg\min_{\mathbf{u}\in\mathbf{\Omega}} H(\mathbf{x}^*, \mathbf{u}, \lambda^*, t) \tag{2.10}$$

When Hamiltonian does not depend on time explicitly, and the final time is free, we have one additional necessary condition:

$$H(\mathbf{x}^*, \mathbf{u}^*, \lambda^*, t) = 0 \tag{2.11}$$

Above additional necessary condition means that if the final time is free, Hamiltonian is equal to zero on the optimal trajectory.

Equation (2.4 - 2.7) and Equation (2.9) are necessary conditions of optimality in general, that is Pontryagin maximum principle can be applied to problems with bounded and unbounded controls. From Equation (2.10), we can see the optimal control is a feedback control law, because it depends, at time t on $\mathbf{x}(t)$, which itself depends on what happened over the whole interval from the initial time to current time.

From above we can see, the necessary conditions for optimal control of general dynamic system are a set of $2n$ first order differential equations (state and costate equations), and a set of m algebraic relations (Equation (2.10)). The set of functions to be determined includes n state variables, n costate variables λ, and m controls $\mathbf{u}$. The solution of the state and costate equations will contain 2n constants of integration. Notice that regardless of the problem specifications, the boundary conditions are always split; thus, to find an optimal trajectory, in general, a nonlinear, Two Point Boundary Value Problem (TPBVP) must be solved. Unfortunately, it is very difficult to solve the TPBVP in most case. So it is necessary to consider some more specific problems in order to get solutions in practice. An important feature of the variational approach is the form of the optimal controls can be determined. Hence, it is necessary only to consider the subset of controls having the appropriate form. This is a significant conceptual and computational advantage.

2.1.2 Exact Minimum Time Control Problem for Robot Manipulators

The Lagrange-Euler equations of motion for n-link manipulators can be written as

$$\mathbf{M}(\mathbf{q}(t))\ddot{\mathbf{q}}(t) + \mathbf{C}(\mathbf{q}(t), \dot{\mathbf{q}}(t)) + \mathbf{G}(\mathbf{q}(t)) = \boldsymbol{\tau}(t) \tag{2.12}$$

where

$\mathbf{q}(t) = n \times 1$ vector representing the joint positions of the corresponding n-link manipulator,

$\mathbf{M} = n \times n$ generalized mass inertia (positive definite real symmetric) matrix,

$\mathbf{C}(\mathbf{q}(t), \dot{\mathbf{q}}(t))$ = centrifugal, coriolis and friction term,

$\mathbf{G}(\mathbf{q}(t)) = n \times 1$ generalized gravity term (gravitational torques),

$\boldsymbol{\tau}(\mathrm{t}) = n \times 1$ vector of applied joint torques.

The state space representation of equations of motion can be formulated from the above Lagrange-Euler equations of motion. Let us define a $2n \times 1$ state vector of a manipulator as

$$\begin{aligned} \mathbf{x}^T(t) = [\mathbf{q}^T(t), \dot{\mathbf{q}}^T(t)] &= [q_1(t), \cdots, q_n(t), \dot{q}_1, \cdots, \dot{q}_n(t)] \\ &= [\mathbf{x}_1^T(t), \mathbf{x}_2^T(t)] \end{aligned} \tag{2.13}$$

where $\mathbf{x}_1(t) = \mathbf{q}(t)$ and $\mathbf{x}_2(t) = \dot{\mathbf{q}}(t)$.

Define an $n \times 1$ input vector as

$$\mathbf{u}^T(t) = [\tau_1(t), \tau_2(t), \cdots, \tau_n(t)] \tag{2.14}$$

The equations of motion can be expressed in state space representation as

$$\dot{\mathbf{x}}_1(t) = \mathbf{x}_2(t) \tag{2.15}$$

$$\dot{\mathbf{x}}_2(t) = -\mathbf{M}^{-1}(\mathbf{x}_1)[\mathbf{C}(\mathbf{x}_1, \mathbf{x}_2) + \mathbf{G}(\mathbf{x}_1)] + \mathbf{M}^{-1}\mathbf{u}(t) \tag{2.16}$$

or simply

$$\dot{\mathbf{x}}(t) = \mathbf{A}(\mathbf{x}(t)) + \mathbf{B}(\mathbf{x}(t))\mathbf{u}(t) \tag{2.17}$$

At the initial time $t = t_0$, the system is assumed to be in the initial state $\mathbf{x}(t_0) = \mathbf{x}_0$, and at the final time $t = t_f$ the system is required to be in the desired final state $\mathbf{x}(t_f) = \mathbf{x}_f$. In general, the mathematical model of a system will include certain constraints on the controls $\mathbf{u}(t)$. These constraints are expressed in terms of a constraint set Ω. A piecewise continuous control, $\mathbf{u}(t)$ satisfying the condition

$$\mathbf{u}(t) \in \Omega \quad \forall t \in [t_0, t_f] \tag{2.18}$$

is said to be as an admissible control. In this thesis, the admissible controls of the system are assumed to be bounded and satisfy the constraints,

$$|u_i| \leq (u_i)_{max} \quad \forall t \in [t_0, t_f] \tag{2.19}$$

The solution of the time optimal control problem is given by the admissible controls which minimize the cost functional

$$J = \int_{t_0}^{t_f} 1dt = t_f - t_0 \tag{2.20}$$

In order to emphasis the difference between the true minimum time control and the near-minimum-time control (NMTC), in this thesis we define the true minimum time control as Exact Minimum Time Control (EMTC). The Exact Minimum Time Control (EMTC) of robot manipulator can be stated as follows:

Given the continuous dynamical system described by Equation (2.17) with the initial state $\mathbf{x}_0$, the terminal constraint $\mathbf{x}_f$, and the control constraint set Ω; find the admissible control $\mathbf{u}(t) \in \Omega$, which transfer the state of the system from $\mathbf{x}_0$ to $\mathbf{x}_f$ in minimum time, i.e. , such that the cost functional J defined in Equation (2.20) is a minimum.

The EMTC problem can be treated as a special class of general optimal control problem presented in Section 2.1.1. For the EMTC problem, the final time t_f is free.

Define the Hamiltonian as

$$H(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}) = 1 + \boldsymbol{\lambda}^T[\mathbf{A}(\mathbf{x}) + \mathbf{B}(\mathbf{x})\mathbf{u}] \tag{2.21}$$

where vector $\boldsymbol{\lambda} \in R^{2n}$ is called the costate variable vector.

Let $\mathbf{u}^*(t)$ be an optimal control and let $\mathbf{x}^*(t)$ denote the optimal state vector and $\boldsymbol{\lambda}^*(t)$ the optimal costate variable vector. Using the techniques of variational calculus and applying Pontryagin's maximum principle (Kirk 1970), the necessary conditions for $\mathbf{u}^*(t)$ to be optimal control are:

$$\frac{d\mathbf{x}^*(t)}{dt} = \frac{\partial H(\mathbf{x}^*, \boldsymbol{\lambda}^*, \mathbf{u}^*)}{\partial \boldsymbol{\lambda}} \quad t \in [t_0, t_f] \tag{2.22}$$

$$\frac{d\boldsymbol{\lambda}(t)^*}{dt} = -\frac{\partial H(\mathbf{x}^*, \boldsymbol{\lambda}^*, \mathbf{u}^*)}{\partial \mathbf{x}} \quad t \in [t_0, t_f] \tag{2.23}$$

$$H(\mathbf{x}^*, \boldsymbol{\lambda}^*, \mathbf{u}^*) \leq H(\mathbf{x}^*, \boldsymbol{\lambda}^*, \mathbf{u}) \quad t \in [t_0, t_f] \quad \textit{for all admissible controls} \tag{2.24}$$

Equations (2.22) and Equation (2.23) are known as the canonical equations. The optimal control is the control which minimizes the Hamiltonian at every instant within the given interval. Therefore, the Exact Minimum Time Control (EMTC) law is:

$$u_i^*(t) = \begin{cases} u_{imax} & \text{if } S_i < 0 \\ -u_{imax} & \text{if } S_i > 0 \\ \text{undetermined} & \text{if } S_i = 0 \end{cases} \tag{2.25}$$

where $S_i(t)$ is the ith component of the $m \times 1$ switching functions $\mathbf{S}(t)$ defined by

$$\mathbf{S}(t) = \mathbf{B}^T \boldsymbol{\lambda} \tag{2.26}$$

When $S_i = 0$ for a period of time, the Pontryagin maximum principle is not able to define any optimal value of u_i. It is called a singular condition, and the control is referred to as singular control. Thus, for singular control, the necessary condition that $\mathbf{u}$ must minimize H provides no information about the value of the controls. If the switch function S_i is zero at individual point only, the control is nonsingular. This type of control is referred to as bang-bang control. The controls take their extremal values throughout the whole motion to minimize the maneuver time.

If H is not an explicit function of time, the additional necessary condition is: :

$$H(\mathbf{X}^*, \boldsymbol{\lambda}^*, \mathbf{u}^*) = 0 \quad t \in [t_0, t_f] \tag{2.27}$$

This means if H is not explicit function of time, for the exact minimum time trajectory, the Hamiltonian is always equal to zero.

The boundary conditions are:

$$\mathbf{x}(t_0) = \mathbf{x}_0 \tag{2.28}$$

$$\mathbf{x}(t_f) = \mathbf{x}_f \tag{2.29}$$

The set of differential equations (Equation (2.22) and (2.23)), the requirement equations (Equation (2.25) and (2.27)), and the boundary conditions (Equation (2.28) and (2.29)) completely define the EMTC problem for robot manipulator. This leads to a bang-bang exact minimum time control solution, except when a singularity occurs, i.e. $S_i(t)$ is zero for some finite time interval. As can be seen from the above optimality conditions, no boundary conditions for the costate state variables λ_i are defined. This fact leads to the problem of solving a Two Point Boundary Value Problem (TPBVP) with boundary conditions on the state $\mathbf{x}$(t) at the initial and final times but unspecified boundary values for λ , which is a major cause of the difficulties in finding exact minimum time solution for nonlinear systems such as the robot manipulators.

2.2 Guidelines for Developing a Near-Minimum-Time Algorithm

From Section 2.1, we can see that using the calculus-of-variations approach to the Exact Minimum Time Control (EMTC) problem for robot manipulators will lead to a Two Point Boundary Value Problem (TPBVP). Due to the nonlinearity of the equations of motion, a numerical solution is usually the only approach to the problem. Since the boundary conditions for the canonical adjoint system of differential equations are not known, solving the TPBVP for robot manipulators is very difficult and computationally intensive. It requires good initial guesses for the unknown parameters, and is very sensitive to the unknown initial conditions. The solution is optimal for the unique initial and final conditions, Hence, the computations for the optimal control have to be performed for each manipulator motion.

In this thesis, we will develop a new method for generating not an exact minimum time point-to-point manipulator motions, but rather a near-minimum-time solution. The near-minimum-time solution has a similar structure of the exact minimum time solution and satisfies all the boundary conditions and work-energy constraints. The major advantage of the new method is that it is computationally efficient and easy to use and does not require good initial boundary value guesses. So the method is practical for improving the efficiency of robot manipulators.

Chapter 3

Development of the Proposed Zero-Net-Work Method

3.1 Motivation

3.1.1 The Structure of the Exact Minimum Time Control Law

From Chapter 2, we have found that Pontryagin's maximum principle yields the optimal control law:

$$u_i^*(t) = \begin{cases} u_{imax} & \text{if } S_i < 0 \\ -u_{imax} & \text{if } S_i > 0 \\ \text{undetermined} & \text{if } S_i = 0 \end{cases}$$

where $\mathbf{S}(t) = \mathbf{B}^T \boldsymbol{\lambda}$ is the switching function vector.

Definition 3.1 Bang-bang Control: A control $\mathbf{u}$ is bang-bang if $u_i(t) = u_{imax}$ or $-u_{imax}$ with finite switchings for $i \in \{1, 2, ..., n\}$. For the bang-bang control, the switching functions S_i can only have finite zeros, and $u_i(t)$ switches among u_{imax} and $-u_{imax}$ at the zero of S_i.

Definition 3.2 Singular Control: A control $\mathbf{u}$ is singular if the canonical equations (Equation (2.22 - 2.23)) are satisfied and $S_i(t) = 0$ for a finite time interval $t \in [t_1, t_2]$, where $t_0 \leq t_1 < t_2 \leq t_f$ and $i \in \{1, 2, ..., n\}$.

Next we will show that the Exact Minimum Time Control (EMTC) of a robot manipulator, a singular control does not exist. That is the EMTC of a n-link robot manipulator can not be singular simultaneously over a finite time interval $[t_1, t_2]$. The equations of motion for an n-link robot manipulator are

$$\dot{\mathbf{x}}_1(t) = \mathbf{x}_2(t) \tag{3.1}$$

$$\dot{\mathbf{x}}_2(t) = \mathbf{M}^{-1}(\mathbf{x}_1)[-\mathbf{C}(\mathbf{x}_1, \mathbf{x}_2) - \mathbf{G}(\mathbf{x}_1) + \mathbf{u}(t)] \tag{3.2}$$

Let

$$\boldsymbol{\lambda}(t) = [\boldsymbol{\lambda}_a^T(t), \boldsymbol{\lambda}_b^T(t)]^T \tag{3.3}$$

where $\boldsymbol{\lambda}_a \in R^n$, $\boldsymbol{\lambda}_b \in R^n$ are the costate variables and $\boldsymbol{\lambda}_a$. $\boldsymbol{\lambda}_b$ can not be identically zero over a finite time interval.

The Hamiltonian is:

$$H = 1 + \boldsymbol{\lambda}_a^T \mathbf{x}_2(t) + \boldsymbol{\lambda}_b^T \mathbf{M}^{-1}(\mathbf{x}_1)[-\mathbf{C}(\mathbf{x}_1, \mathbf{x}_2) - \mathbf{G}(\mathbf{x}_1) + \mathbf{u}(t)] \tag{3.4}$$

The switching functions are:

$$\mathbf{S}(t) = (\mathbf{M}^{-1}(\mathbf{x}_1))^T \boldsymbol{\lambda}_b(t) \tag{3.5}$$

The necessary conditions for $\mathbf{u}^*(t)$ to be an optimal control are:

$$\frac{d\boldsymbol{\lambda}(t)}{dt} = -\frac{\partial H}{\partial \mathbf{x}} \tag{3.6}$$

Substituting Equation (3.3) into Equation (3.6) we obtain:

$$\begin{aligned} \dot{\boldsymbol{\lambda}}_a(t) &= -\frac{\partial H}{\partial \mathbf{x}_1} && (3.7) \\ \dot{\boldsymbol{\lambda}}_b(t) &= -\frac{\partial H}{\partial \mathbf{x}_2} && (3.8) \end{aligned}$$

Substituting Equation (3.4) into Equation (3.8) we obtain:

$$\dot{\boldsymbol{\lambda}}_b(t) = (\frac{\partial \mathbf{C}}{\partial \mathbf{x}_2})^T (\mathbf{M}^{-1}(\mathbf{x}_1))^T \boldsymbol{\lambda}_b(t) - \boldsymbol{\lambda}_a(t) \tag{3.9}$$

In Equation (3.9), for the finite time interval $[t_1, t_2]$ if $\boldsymbol{\lambda}_b(t) = 0$ then $\dot{\boldsymbol{\lambda}}_b(t) = 0$. Because $\boldsymbol{\lambda}_a$, $\boldsymbol{\lambda}_b$ can not be identically zero over a finite time interval, in order to satisfy Equation (3.9), we have $\boldsymbol{\lambda}_b(t) \neq 0 \quad t \in [t_1, t_2]$.

The inertia matrix $\mathbf{M}(\mathbf{x}_1)$ is always nonsingular, so from Equation (3.5) we get the switching functions:

$$\mathbf{S}(t) \neq 0 \quad t \in [t_1, t_2] \tag{3.10}$$

This means that all the controls can't be simultaneously singular over the interval $[t_1, t_2]$, where $t_1 < t_2 \leq t_f$. The structure of the EMTC of n-DOF robot system (Chen, 1989) is that at least one of the control inputs is always in saturation (bang-bang) on every finite time interval; in other words, not all control inputs are simultaneously singular on the time interval.

3.1.2 The Double Integrator Problem

Next, let us consider a simple one-link horizontal robot manipulator shown in Figure 3.1.

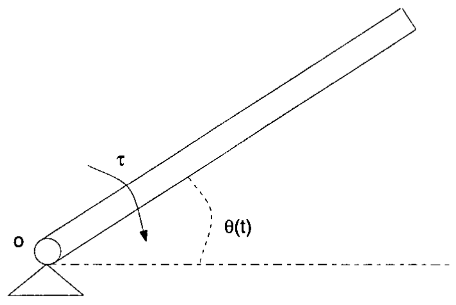

Figure 3.1: One-link-torque system

It is a second-order, linear time invariant system. The dynamic equation is:

$$I\,\ddot{\theta} = \tau \tag{3.11}$$

The absolute value of the torque τ is bounded.

$$|\tau| \leq \tau_{max}$$

The state equations are:

$$\dot{x}_1(t) = x_2(t) \tag{3.12}$$

$$\dot{x}_2(t) = u(t) \tag{3.13}$$

where

$$x_1(t) = \theta(t) \tag{3.14}$$

$$x_2(t) = \dot{\theta}(t) \tag{3.15}$$

$$u(t) = \tau(t)/I \tag{3.16}$$

Assume without loss of generality that the final states are at the origin.

The initial states

$$x_1(t_0) = \theta_0 = x_{10} \tag{3.17}$$

$$x_2(t_0) = \dot{\theta}_0 = 0 \tag{3.18}$$

and final states

$$x_1(t_f) = \theta_f = 0 \tag{3.19}$$

$$x_2(t_f) = \dot{\theta}_f = 0 \tag{3.20}$$

The control is constrained such that

$$-u_{max} \leq u(t) \leq +u_{max} \quad t_0 < t < t_f \tag{3.21}$$

where $u_{max} = \tau_{max}/I$. The point-to-point exact minimum time motion of one link manipulator with a bounded torque is well-known as a double integrator problem and is solved in the optimal control literature (Kirk,1970).

The Hamiltonian function is

$$H = 1 + \lambda_1 x_2 + \lambda_2 u \tag{3.22}$$

Using Pontryagin's maximum principle, for $u^*(t)$ the optimal control

$$H[x^*, u^*(t), \lambda^*(t), t] \leq H[x^*, u(t), \lambda^*(t), t] \tag{3.23}$$

The costate equations are

$$\begin{cases} \dot{\lambda}_1 = 0 \\ \dot{\lambda}_2 = -\lambda_1 \end{cases}$$

Both initial and final states are fixed, the corresponding conditions on the costate variables are free. The solution to the costate equations has the form

$$\lambda_1(t) \quad = \quad k_1 \tag{3.24}$$

$$\lambda_2(t) \quad = \quad -k_1 t + k_2 \tag{3.25}$$

where k_1 and k_2 are undetermined constants. Equation (3.25) shows that $\lambda_2^*(t)$ is a linear function of time and, provided $k_1 \neq 0$, can therefore be zero at only one point in time interval $[t_0, t_1]$. This information together with the control law Equation (3.23) implies that the optimal control u^* is of the "bang-bang" form and the control input is constant expect at the switching instant. By integrating Equation (3.11) we obtain

$$\dot{\theta}(t) = \pm u_{max} t + \beta_1 \tag{3.26}$$

$$\theta(t) = \pm 0.5 u_{max} t^2 + \beta_1 t + \beta_2 \tag{3.27}$$

Let $t_0 = 0$ and t_{sw} is the switching time, t_f is the final time. We get the state equations before switch is:

$$\dot{\theta}_{bf}(t) = \pm u_{max} t \tag{3.28}$$

$$\theta_{bf}(t) = \pm 0.5 u_{max} t^2 + \theta_0 \tag{3.29}$$

The state equations after switch is:

$$\dot{\theta}_{af}(t) = \mp u_{max}(t - t_f) \tag{3.30}$$

$$\theta_{af}(t) = \mp 0.5 u_{max} t^2 \pm u_{max} t_f t \mp 0.5 u_{max} t_f^2 \tag{3.31}$$

At the switching point:

$$\dot{\theta}_{bf}(t_{sw}) = \dot{\theta}_{af}(t_{sw}) \tag{3.32}$$

$$\theta_{bf}(t_{sw}) = \theta_{af}(t_{sw}) \tag{3.33}$$

Substitute Equation(3.28) and Equation (3.30) into Equation (3.32) we get:

$$t_{sw} = 0.5 t_f \tag{3.34}$$

Substitute Equation(3.29) Equation (3.31) into Equation (3.33) and using Equation (3.34) we get:

$$t_{sw} = \sqrt{\frac{|\theta_0|}{u_{max}}} = \sqrt{\frac{|\theta_0| I}{\tau_{max}}} \tag{3.35}$$

From the above we can see for the one link robot manipulator, the final and switching time are depend on the input bounds, the initial position and the mass moment of inertia. The costate state variables $\lambda_1(t)$, $\lambda_2(t)$ can be obtained as below. Because H is not an explicit function of time, we have,

$$H(x^*(t), \lambda^*(t), u^*(t)) = 0 \quad t \in [0, t_f] \tag{3.36}$$

Substituting Equations (3.22) (3.24) (3.25) into the above equation and using the initial conditions, we obtain

$$k_2 = \frac{-1}{u^*(0)} \tag{3.37}$$

where $u^*(0)$ is the optimal control at the initial point.

At the switching point,

$$\lambda_2(t_{sw}) = -k_1 t_{sw} - \frac{1}{u^*(0)} = 0 \tag{3.38}$$

Substituting Equation (3.35) into Equation (3.38) we obtain

$$k_1 = \frac{-1}{u^*(0) t_{sw}} = \frac{-1}{u^*(0)} \sqrt{\frac{u_{max}}{|\theta_0|}} \tag{3.39}$$

So

$$\lambda_1(t) = \frac{-1}{u^*(0)} \sqrt{\frac{u_{max}}{|\theta_0|}} \tag{3.40}$$

$$\lambda_2(t) = \frac{1}{u^*(0)} \sqrt{\frac{u_{max}}{|\theta_0|}} t - \frac{1}{u^*(0)} \tag{3.41}$$

For the numerical calculations the following values are used for the one link robot manipulator: $\tau_{max} = 10N.M$, $I = 10m^2kg$, $\theta_0 = 1$.

Substituting above values into Equation (3.35) and Equation (3.34), the switching time $t_{sw} = 1s$, and the final time $t_f = 2s$.

The state equations before switching ($0 \leq t \leq 1$) are:

$$\dot{\theta}_{bf}(t) = -t \tag{3.42}$$

$$\theta_{bf}(t) = -0.5t^2 + 1 \tag{3.43}$$

The state equations after switching $(1 < t \leq 2)$ are:

$$\dot{\theta}_{af}(t) = t - 2 \tag{3.44}$$

$$\theta(t)_{af} = 0.5t^2 - 2t + 2 \tag{3.45}$$

The costate variables are:

$$\lambda_1(t) = 1 \tag{3.46}$$

$$\lambda_2(t) = 1 - t \tag{3.47}$$

Figure 3.2 shows the position and velocity trajectories. Figure 3.3 shows the applied joint torque, we can see the control is bang-bang and switches once at $t = t_{sw} = 1s$. The costate trajectories are shown in Figure 3.4. The costate variable λ_2 determines when to switch. If λ_2 changes the sign once, the optimal control is bang-bang and switches once. If the sign of λ_2 changes more than once, and is not equal to zero at finite time interval, the optimal control is still bang-bang, but will have more than one switch.

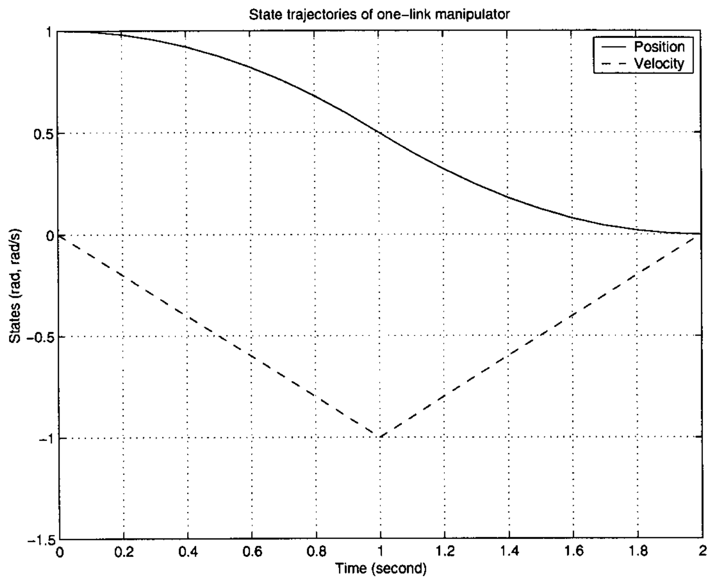

Figure 3.2: Joint position and velocity of the one-link planar manipulator

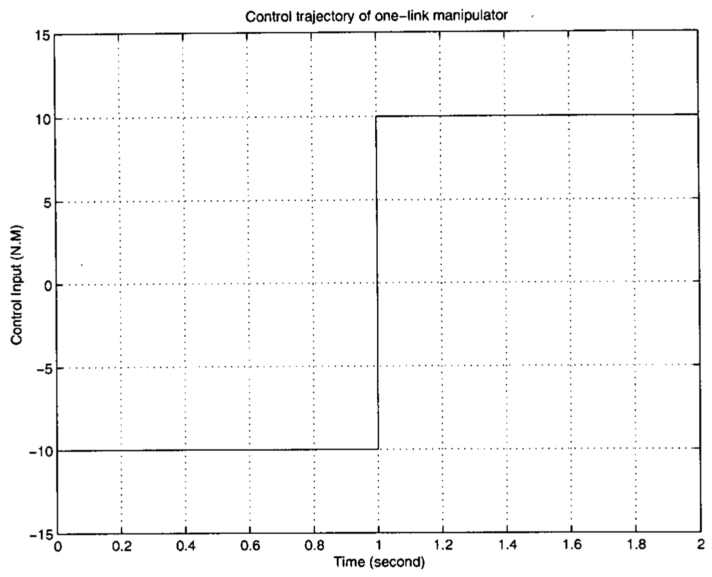

Figure 3.3: Applied joint torque of the one-link planar manipulator

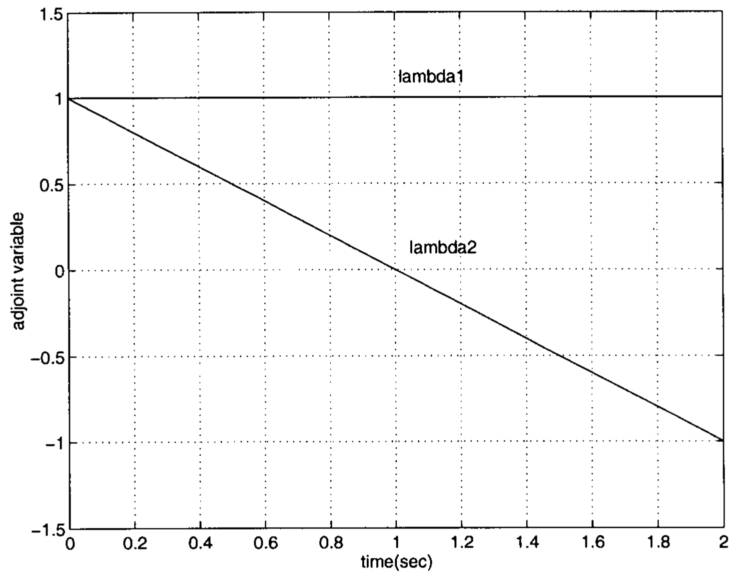

Figure 3.4: Costate trajectories of the one-link planar manipulator

3.1.3 Dynamically Decoupled Two Link Robot Manipulator

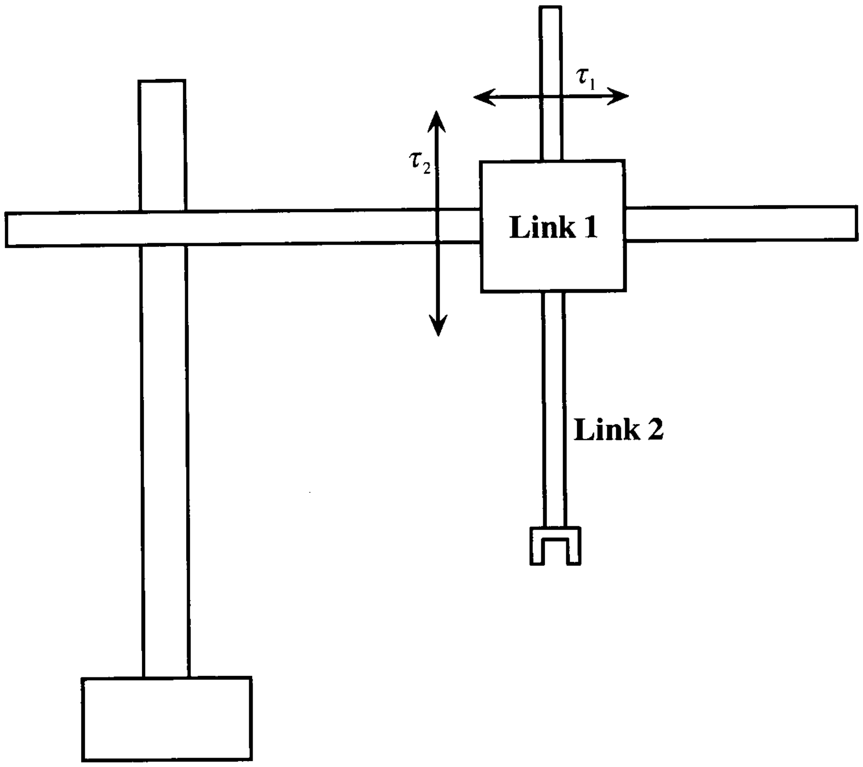

Figure 3.5: Dynamically decoupled two-link robot manipulator

Next, consider a dynamically decoupled two-link robot manipulator as shown in Figure 3.5. The dynamic equations of motion can be written as

$$\begin{bmatrix} m_1 + m_2 & 0 \\ 0 & m_2 \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix}$$

where m_1, m_2 are the masses of each link. Solving the optimal control problem for this case, the exact minimum times are:

$$t_{f1} = 2t_{sw1} = 2\sqrt{\frac{|q_{10}|(m_1 + m_2)}{\tau_{1max}}} \tag{3.48}$$

$$t_{f2} = 2t_{sw2} = 2\sqrt{\frac{|q_{20}|m_2}{\tau_{2max}}} \tag{3.49}$$

where q_{10}, q_{20} are the initial position for each link, and t_{sw1} t_{sw2} are the switching times. In this case, the final time of the exact minimum time motion can be different for each link, depending on the remaining trajectory travel required, the input bound and the mass. One link can reach to the final position earlier than the other and wait until the other one reaches the final position. If the two links are dynamically coupled, than they will influence each other. Because the structure of the EMTC is such that at least one of the control inputs is always bang-bang on every finite time interval, the exact minimum time solution will be such that one control input is bang-bang and the other control input varies within the bound in such a way that simultaneous reaching of final time be guaranteed. If the control input, which is singular and therefore not bang-bang, is treated to be bang-bang, then it forces one link to reach the final position faster than the other one, and the slower final time sets an upper bound on the final time of the system.

3.1.4 Work Energy Behavior

Next we will investigate the work energy behavior of the Exact Minimum Time Control (EMTC) motion. Consider the robot manipulator moving from position $\mathbf{P}_1$ to position $\mathbf{P}_2$. The Work Energy Principle for each individual link states that: The net work done by all the forces/torques is equal to the change in kinetic energy, i.e.

$$T_{i2} - T_{i1} = Work_i \tag{3.50}$$

where

T_{i1}, T_{i2} = initial and final values of the kinetic energy of link i,

$Work_i$ = work done by all the forces/torques acting on link i.

Because the initial and final values of the kinetic energy of each link are equal to zero, so for link i,

$$Work_i = W_{Ai} + W_{Ri} + W_{Gi} = 0 \tag{3.51}$$

where

W_{Ai} = work done by the actuator,

W_{Ri} = work done by the reaction forces/torques,

W_{Gi} = work done by gravity.

So the work energy behavior of the exact minimum time motion is such that, for each individual link, the work done by all forces/torques is equal to zero.

3.2 Zero-Net-Work Method

The dynamic response of Exact Minimum Time Control (EMTC) can be interpreted in the physical sense as follows: During the exact minimum time motion, the kinetic energy is initially increased as quickly as possible and then decreased after the switching point while satisfying the control constraints ($|\tau| \leq \tau_{max}$) and boundary conditions. For each individual link, the work done by all forces acting on the link is equal to the change in kinetic energy of the link. If the robot system is total decoupled, then the control is bang-bang and there is only one switch per link. We can treat each link as a one-link robot system, and the problem is a double integrator problem. Unfortunately, most robot systems are nonlinear and coupled systems, and the links can influence each other, so the optimal motion is such that at least one control input is bang-bang, while the other control inputs vary within the given bound in a way to guarantee that the robot will reach the final state. In summary the basic structure of the EMTC is bang-bang, and if there is dynamic coupling of the links, the optimal control will still be bang-bang, but will switch more than once, and for some links the control input may vary within the given bound in order to satisfied the boundary conditions.

The basic idea of the zero-net-work method is that we assume, the control inputs are bang-bang (This will keep the basic structure of the EMTC). Also, for each link, we assume the control input switches only once (This will reduce the computation time and increase the efficiency of the algorithm). If we can find the switching point for each link that satisfies all the constraints (boundary conditions, work energy constraints, input bound constraints), then we can obtain a solution which is close to the EMTC solutions. Using the zero-net-work method, we can have a Near-Minimum-Time Control (NMTC) solution. Because we assume the control input is bang-bang, that is for each link the control input $u_i = \pm u_{imax}$, the control constraints $|u_i| \leq u_{imax}$ are always satisfied. We assume the initial velocities and final velocities are equal to zero and from the work energy behavior of the exact minimum time motion, we can say that if we find the switching points for each link such that work done by all the forces/torques for each link $Work_i = 0$, then the velocity constraints can be satisfied. So the basic task for zero-net-work method is to search for the switching points for each link so that the work done on each individual link is equal to zero. Since in the zero-net-work method, we assume only one switch, the switching point for each link is very important.

Next we will investigate the influence of different switching point for a single link. Figure 3.6 shows a planar one-link robot. θ_{sw} is the angle at the switching point. For different values of θ_{sw} there are following three cases:

Case 1. $\theta_{sw} = \theta_{sw}^{ZNW}$

θ_{sw} is equal to the switching point of the zero-net-work (ZNW) switching point θ_{sw}^{ZNW} . The robot stops exactly at the final position. $Work = 0$ at the final position. The free body diagram of this case is shown in Figure 3.6. The position and velocity trajectories are shown in Figure 3.7.

Case 2. $\theta_{sw} > \theta_{sw}^{ZNW}$

In this case the switching point is greater than the zero-net-work switching point. This means the robot arm switches earlier than it should be. The free body diagram of this case is shown in Figure 3.8 . When the velocity equals to zero, the robot hasn't reached the final position yet as shown in Figure (3.8). In this case $Work \neq 0$ at the final position. The position and velocity trajectories are shown in Figure 3.9.

Case 3. $\theta_{sw} < \theta_{sw}^{ZNW}$

In this case the switching point is smaller than the zero-net-work switching point. This means the robot arm switches too late than it should be. The free body diagram of this case is shown in Figure 3.10. When the robot reach the final position, the velocity is not equal to zero, so it can't stop at the final position as shown in Figure (3.11). In this case $Work \neq 0$ at the final position. The position and velocity trajectories are shown in Figure 3.11.

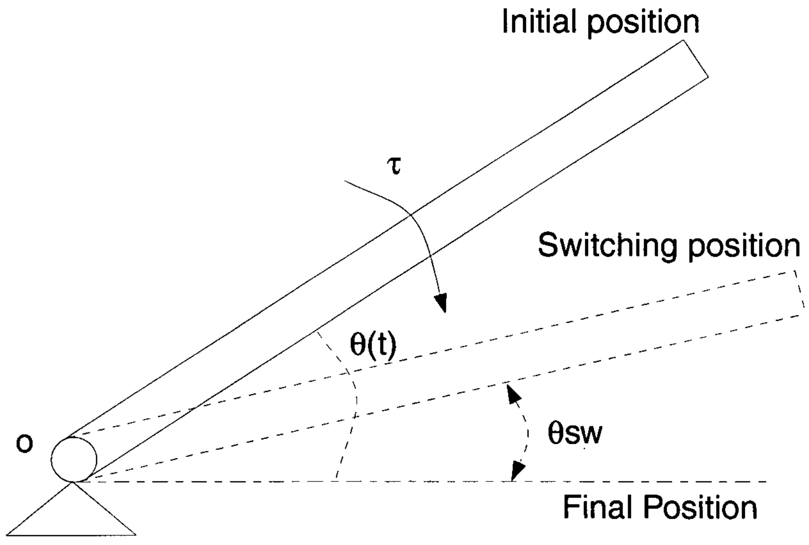

Figure 3.6: Free body diagram of one-link robot for case 1

For the zero-net-work method, we set the initial value of the switching point smaller than the zero-net-work switching point. If the searching step size is small enough, case 2 will not happen. We can always control the robot reach the final position. If the switching position is not equal to the zero-net-work switching position, at the final position the work energy constraint can't be satisfied and $Work_i > 0$. If we assume the initial switching position for each link equals to zero, and then increase the values of the switching position step by step until we get for each link $Work_i = 0$, then we find the zero-net-work near-minimum-time solution for the robot.

From the work energy behavior of the robot we obtain for link i, the total work done by all the forces:

$$Work_i = W_{Ai} + W_{Ri} + W_{Gi} \tag{3.52}$$

Because we assume the control inputs switch once so the work done by the actuator,

$$\begin{aligned} W_{Ai} &= W^b_{Ai} + W^a_{Ai} && (3.53) \\ &= -\tau_{max}(\theta_{sw} - \theta_0) + \tau_{max}(\theta_f - \theta_{sw}) && (3.54) \end{aligned}$$

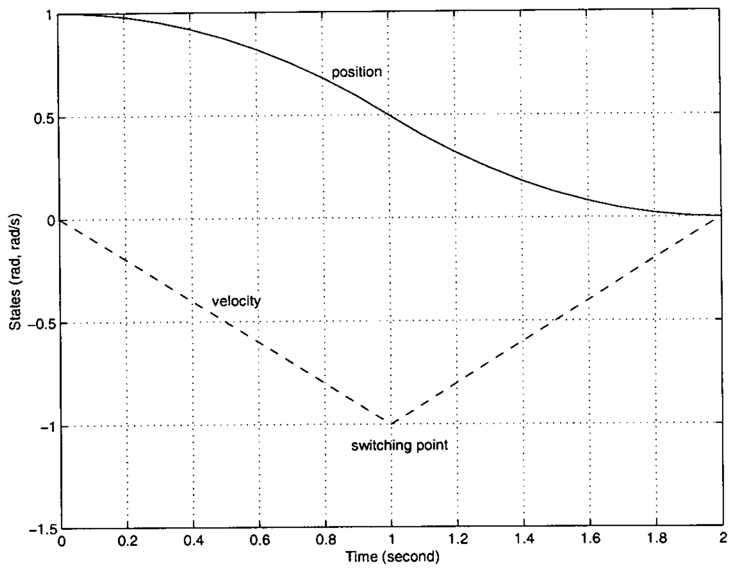

Figure 3.7: Position and velocity trajectories of case 1

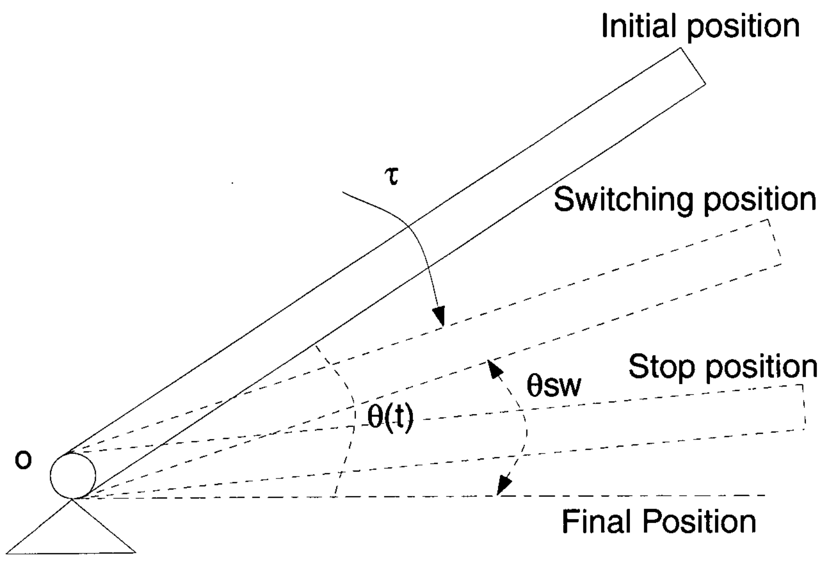

Figure 3.8: Free body diagram of case 2

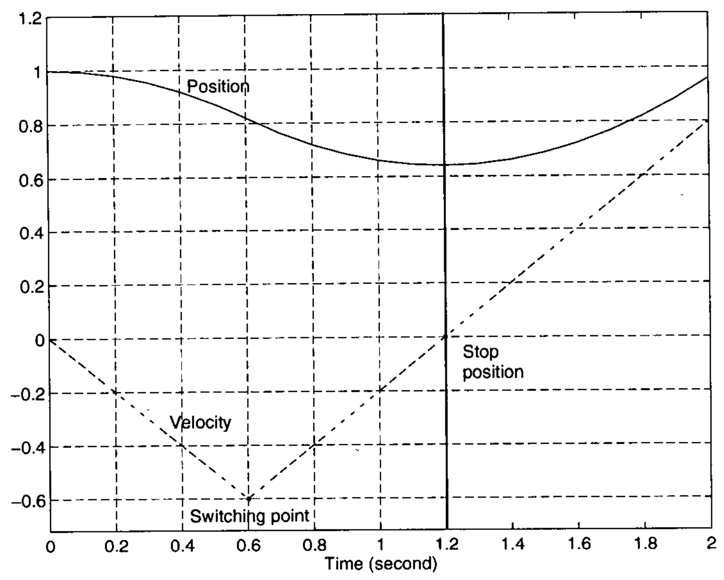

Figure 3.9: Position and velocity trajectories of case 2

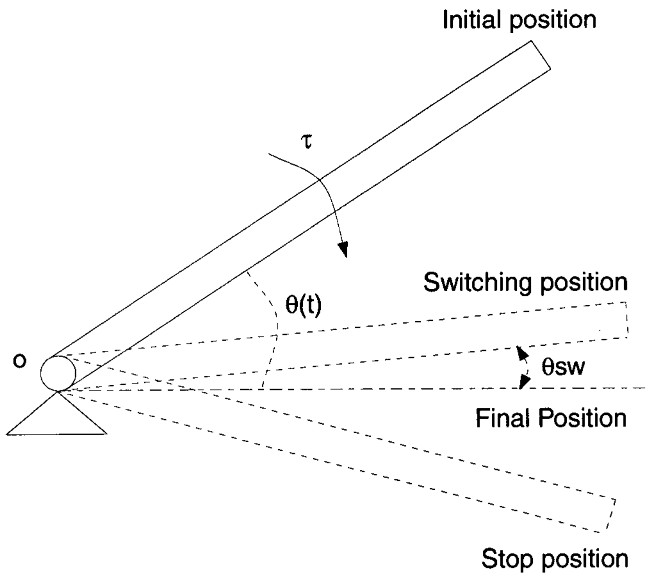

Figure 3.10: Free body diagram of case 3

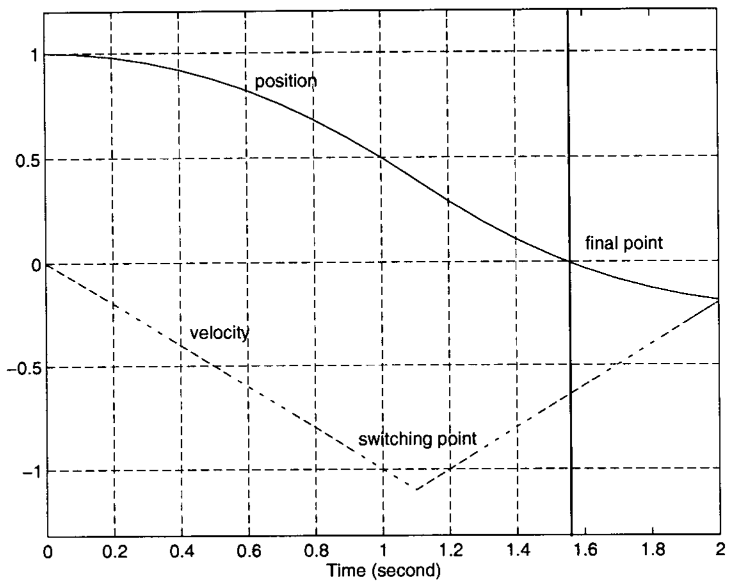

Figure 3.11: Position and velocity trajectories of case 3

where W^b_{Ai} is the work done by the actuator before the switching point,

W^a_{Ai} is the work done by the actuator after the switching point,

θ_0, θ_f are the initial and final position of link i, respectively,

θ_{sw} is the switching position. From above equations we get,

$$Work_i = -2\tau_{max}\theta_{sw} + \tau_{max}\theta_0 + W_{Ri} + W_{Gi} \tag{3.55}$$

The work done by the reaction and gravity forces can be found by using Newton-Euler formulation. Because we know the switching positions, so we know the control inputs, we can then integrate the dynamic equations of the system. We can get the vectors $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$, which are the position, velocity and acceleration of the robot. By using Newton-Euler formulation we can calculate the work done by all the forces for each link.

Next we will develop the general Newton-Euler formulation of n-link revolute manipulator.

Newton's Second Law: The rate of change of the linear momentum equals the total force applied to the body.

$$\mathbf{f} = \frac{d\mathbf{P}}{dt} = \frac{d(m\mathbf{v})}{dt} \tag{3.56}$$

where $\mathbf{P}$ is the linear momentum.

Euler's Angular Momentum Law: The rate of change of the angular momentum equals the total torque applied to the body.

$$\tau = \frac{d\mathbf{H}}{dt} = \frac{d(I\boldsymbol{\omega})}{dt} \tag{3.57}$$

where **H** is the angular momentum.

In the Newton-Euler formulation we treat each link of the manipulator separately. Newton-Euler approach is a recursive method for computing joint forces and torques. We first step forward through the chain of links to compute the kinematic parameters of the links (velocity, angular velocity, linear & angular acceleration), then step backwards through the links and using Newton's second law and Euler's angular momentum law to computer the joint forces and torques. Each link of the manipulator is coupled to other links, by using forward-backward recursion we can determine all the coupling forces and torques. Thus through Newton-Euler approach, we can calculate all the forces and torques acting on each link. Also the closed-form dynamic equations of the system can also be developed.

We first choose frame $F_0, \ldots, F_n$ using Denavit-Hartenberg (D-H) convention. Frame F_0 is an inertial frame, and frame F_i is rigidly attached to the end point of link i. The following vectors are all defined in frame F_i:

$\mathbf{a}_{c,i}$ = the acceleration of the center of mass of link i.

$\mathbf{a}_{e,i}$ = the acceleration of the end of link i (i.e. joint i+1).

$\boldsymbol{\omega}_i$ = the angular velocity of frame F_i w.r.t. frame F_0.

$\boldsymbol{\alpha}_i$ = the angular acceleration of frame F_i w.r.t. frame F_0.

$\mathbf{g}_i$ = the acceleration due to gravity (expressed in frame F_i).

$\mathbf{f}_i$ = the force exerted by link i-1 on link i.

$\boldsymbol{\tau}_i$ = the torque exerted by link i-1 on link i.

$\mathbf{R}_i^{i+1}$ = the rotation matrix from frame F_{i+1} to frame F_i.

m_i = the mass of link i.

I_i = the inertia matrix of link i about a frame parallel to frame F_i whose origin is at the center of mass of link i.

$\mathbf{r}_{i,ci}$ = the vector from joint i to the center of mass of link i.

$\mathbf{r}_{i+1,ci}$ = the vector from joint i+1 to the center of mass of link i.

$\mathbf{r}_{i,i+1}$ = the vector from joint i to joint i+1.

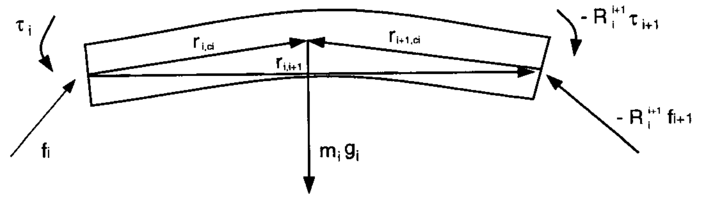

Figure 3.12: Forces and moments on link i

Now consider the free body diagram shown in Figure (3.12), this shows link i together with all forces and torques acting on it. Applying Newton's second law, we have the force balance equation for link i:

$$\mathbf{f}_i = \mathbf{R}_i^{i+1}\mathbf{f}_{i+1} + m_i\mathbf{a}_{c,i} - m_i\mathbf{g}_i \tag{3.58}$$

Applying Euler's law about center of mass, the moment equation of link i is:

$$\boldsymbol{\tau}_i = \mathbf{R}_i^{i+1}\boldsymbol{\tau}_{i+1} - \mathbf{f}_i \times \mathbf{r}_{i,ci} + (\mathbf{R}_i^{i+1}\mathbf{f}_{i+1}) \times \mathbf{r}_{i+1,ci} + I_i\boldsymbol{\alpha}_i + \boldsymbol{\omega}_i \times (I_i\boldsymbol{\omega}_i) \tag{3.59}$$

We can now state the Newton-Euler formulation as follows.

Forward recursion:

Start with the initial conditions

$$\boldsymbol{\omega}_0 = 0,\ \boldsymbol{\alpha}_0 = 0,\ \mathbf{a}_{c,0} = 0,\ \mathbf{a}_{e,0} = 0 \tag{3.60}$$

and solve following equations to compute $\boldsymbol{\omega}_i$, $\boldsymbol{\alpha}_i$, and $\mathbf{a}_{c,i}$ for i increasing from i to n.

$$\boldsymbol{\omega}_i = (\mathbf{R}_{i-1}^i)^T\boldsymbol{\omega}_{i-1} + \mathbf{b}_i\dot{q}_i \tag{3.61}$$

$$\boldsymbol{\alpha}_i = (\mathbf{R}_{i-1}^i)^T\boldsymbol{\alpha}_{i-1} + \mathbf{b}_i\ddot{q}_i + \boldsymbol{\omega}_i \times \mathbf{b}_i\dot{q}_i \tag{3.62}$$

$$\mathbf{a}_{e,i} = (\mathbf{R}_{i-1}^i)^T\mathbf{a}_{e,i-1} + \dot{\boldsymbol{\omega}}_i \times \mathbf{r}_{i,i+1} + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \mathbf{r}_{i,i+1}) \tag{3.63}$$

$$\mathbf{a}_{c,i} = (\mathbf{R}_{i-1}^i)^T\mathbf{a}_{e,i-1} + \dot{\boldsymbol{\omega}}_i \times \mathbf{r}_{i,ci} + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \mathbf{r}_{i,ci}) \tag{3.64}$$

where $\mathbf{b}_i = (\mathbf{R}_0^i)^T \mathbf{z}_{i-1}$ is the axis of rotation of joint i expressed if frame F_i.

Backward recursion:

Start with the terminal conditions $\mathbf{f}_{n+1}$, $\boldsymbol{\tau}_{n+1}$ and use Equation (3.58) Equation (3.59) to compute $\mathbf{f}_i$ and $\boldsymbol{\tau}_i$ for i decrease from n to 1.

By using above forward and backward procedure we can find all the forces and torques acting on link i. The work done by all the forces and torques acting on link i can then be calculated.

The overall scheme of the proposed zero-net-work method is shown in Figure 3.13. We first set the initial switching position for each link to zero, that is the link torque does not switch until the link reaches the final position. By doing this, case 2 will never happen for each link, that means we can always control the robot to reach the final position. But case 3 may happen, that is the robot can reach the final position but the work done by all the forces for each link is not equal to zero at the final position.

The control input can be found based on the switching point $\mathbf{q}_{sw}$,

$$\mathbf{q} \geq \mathbf{q}_{sw}, \quad \tau = -\tau_{max} \tag{3.65}$$

$$\mathbf{q} < \mathbf{q}_{sw}, \quad \tau = +\tau_{max} \tag{3.66}$$

After we find the control input, we can integrate the dynamic equations of the system by one time step. From that we know the position, velocity and acceleration of each link, by using Equation (3.58) and Equation (3.59), and we can calculate all the forces/torques act on each link. So the work done by all the forces/torques for each link can be calculated. If the robot reaches the final position and $Work > 0$, this means the initial value of switching position $\mathbf{q}_{sw}$ is too small, we then increase $\mathbf{q}_{sw}$ by one step $\mathbf{q}_{sw} = \mathbf{q}_{sw} + \mathbf{q}_{swstep}$, recalculate the work, and continue to iterate on q_{sw} until $Work = 0$. After we find the zero-net-work switching positions for each link. They can then be used online to control the robot manipulator. This will result a near-minimum-time control law.

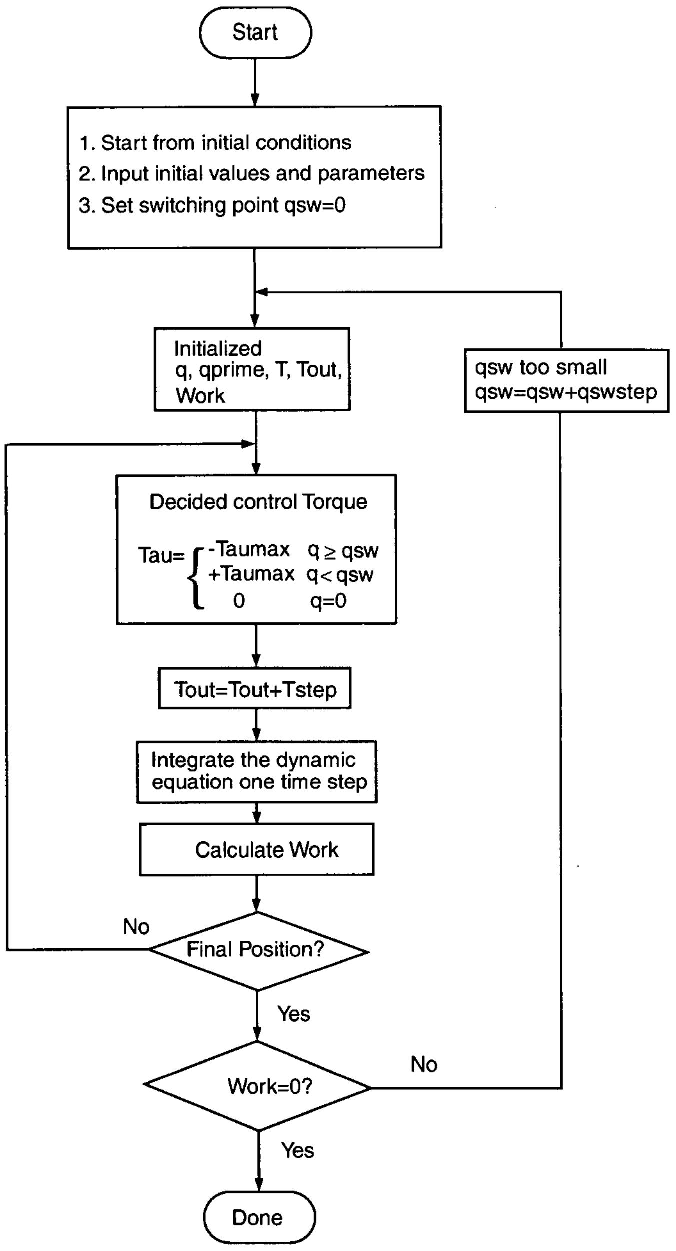
Start
1. Start from initial conditions
2. Input initial values and parameters
3. Set switching point qsw=0
Initialized q, qprime, T, Tout, Work
qsw too small qsw=qsw+qswstep
Decided control Torque
Tau= -Taumax q ≥ qsw; +Taumax q < qsw; 0 q=0
Tout=Tout+Tstep
Integrate the dynamic equation one time step
Calculate Work
No
Final Position?
Yes
Work=0?
No
Yes
Done

Figure 3.13: Overall scheme of the zero-net-work method

Chapter 4

Development of the Zero-Net-Work Controller for One-Link Manipulators

4.1 One-Link Horizontal Manipulator

Next we will investigate the control of a one-link horizontal manipulator. First we will use the traditional method to obtain the exact minimum time solution of the manipulator, and then use the proposed zero-net-work method to solve the problem. We will compare the zero-net-work solution with the true minimum time solution using a computer simulation.

4.1.1 Dynamic Model

The one-link horizontal manipulator in Chapter 3 which is shown in Figure 3.1 will be used.

4.1.2 Exact Minimum Time Solution

The Exact Minimum Time Control (EMTC) of the one-link horizontal robot is a double integrator problem. In Chapter 3, we integrate the dynamic equation of the system and find t_f for the exact minimum time solution as,

$$t_f = 2t_{sw} = 2\sqrt{\frac{|\theta_0|I}{\tau_{max}}} \tag{4.1}$$

Where t_f is the final time and t_{sw} is the switching time, and θ_0 is the initial joint position. When $t \leq t_{sw}$ the control input is $-\tau_{max}$, when $t > t_{sw}$ the control input is $+\tau_{max}$.

4.1.3 Zero-Net-Work Solution

Next we will use the zero-net-work method to control the one-link horizontal manipulator. First we assume the control input is bang-bang and has one switch. The torque before the switching point θ_{sw} is $\tau_b = -|\tau_{max}|$, the torque after the switching point is $\tau_a = +|\tau_{max}|$. The work before the switching point is:

$$\begin{aligned} W_b &= \int_{\theta_0}^{\theta_{sw}} \tau_b d\theta \\ &= \tau_b(\theta_{sw} - \theta_0) \\ &= -|\tau_{max}|(\theta_{sw} - \theta_0) \end{aligned} \tag{4.2}$$

The work after the switching point is:

$$\begin{aligned} W_a &= \int_{\theta_{sw}}^{\theta_f} \tau_a d\theta \\ &= \tau_a(\theta_f - \theta sw) \\ &= -|\tau_{max}|\theta_{sw} \end{aligned} \tag{4.3}$$

The work energy equation of the system is:

$$Work = W_b + W_a = 0 \tag{4.4}$$

Substituting Equation (4.2) and Equation (4.3) into Equation (4.4) we obtain

$$Work = -2|\tau_{max}|\theta_{sw} + |\tau_{max}|\theta_0 = 0 \tag{4.5}$$

So the switching position:

$$\theta_{sw} = \frac{\theta_0}{2} \tag{4.6}$$

From above we can see , for one link robot manipulator, using zero-net-work method the switching point can be easily found. Base on this we can design the zero-net-work controller for the one-link horizontal manipulator as follow: We use the position sensor to get the position of the robot arm at every moment. When $\theta(t) < \theta_{sw}$ the control input is $+\tau_{max}$; when $\theta(t) \geq \theta_{sw}$ the control input is $-\tau_{max}$. Figure 4.1 is the block diagram of the zero-net-work control system.

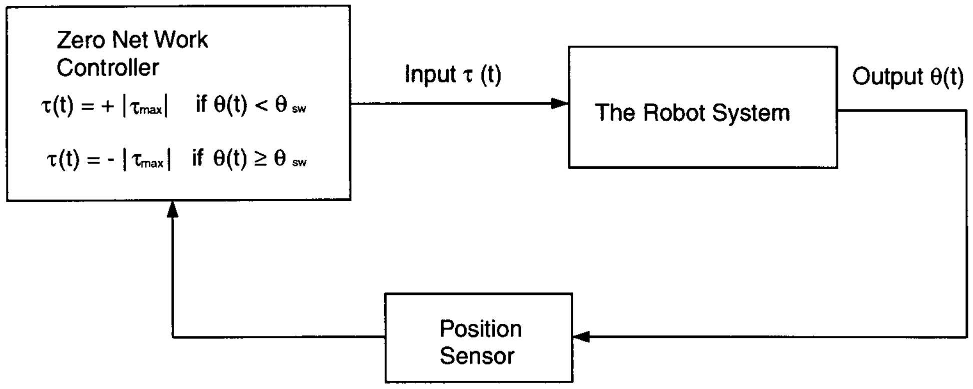

Figure 4.1: Block diagram of the zero-net-work control system

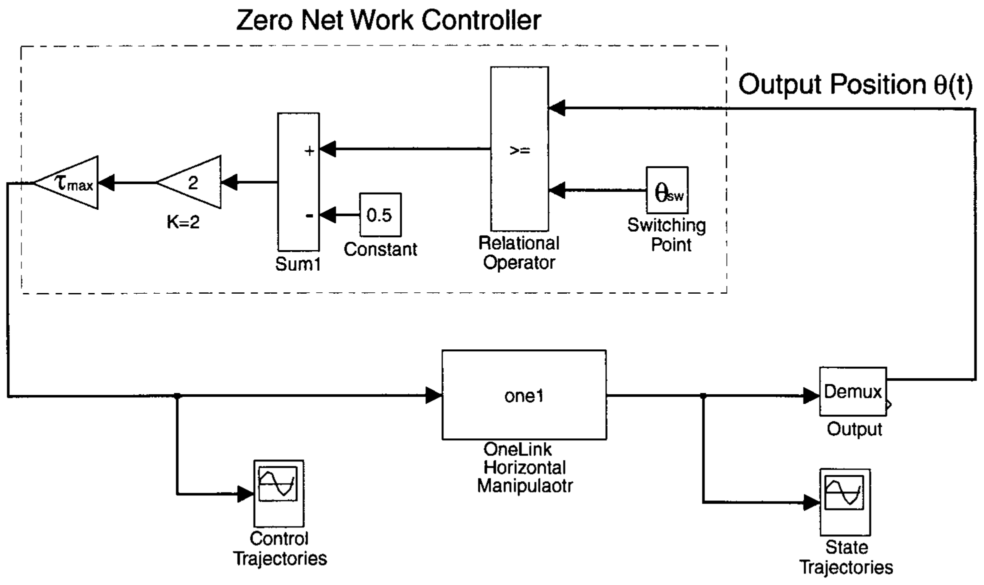

Figure 4.2: Simulation block diagram of the horizontal robot

4.1.4 Computer Simulation

Next we will show the simulation results of the zero-net-work control. We use the same parameters for the manipulator as in Chapter 3 , and use the Simulink for the simulation. Figure 4.2 is the block diagram of the simulation. The system's s-function "one1.m" can be found in the Appendix A.1 . The simulation results for two different motion configurations are presented in the follow:

Simulation 1:

Move the arm from initial state (1, 0) to final state (0,0). Using the zero-net-work method, we obtain the switching position for this case is:

$$\theta_{sw} = \frac{\theta_0}{2} = 0.5 \tag{4.7}$$

Figure 4.3 shows the position and velocity trajectories. Figure 4.4 shows the applied joint torque. The simulation results show that the final time of the zero-net-work solution is $t_f = 2 second$. The exact minimum time solution for this case is:

$$t_f = 2t_{sw} = 2\sqrt{\frac{|\theta_0| I}{\tau_{max}}} = 2 second \tag{4.8}$$

So for this case using the zero-net-work method we obtain the exact minimum time solution.

Simulation 2:

Move the arm from initial state $(\pi/2, 0)$ to final state (0,0).

The switching position for this case is:

$$\theta_{sw} = \frac{\theta_0}{2} = \pi/4 \tag{4.9}$$

Figure 4.5 shows the position and velocity trajectories. Figure 4.6 shows the applied joint torque. The simulation results show that the final time of the zero-net-work solution is $t_f =$ 2.51 *second*. The exact minimum time solution for this case is:

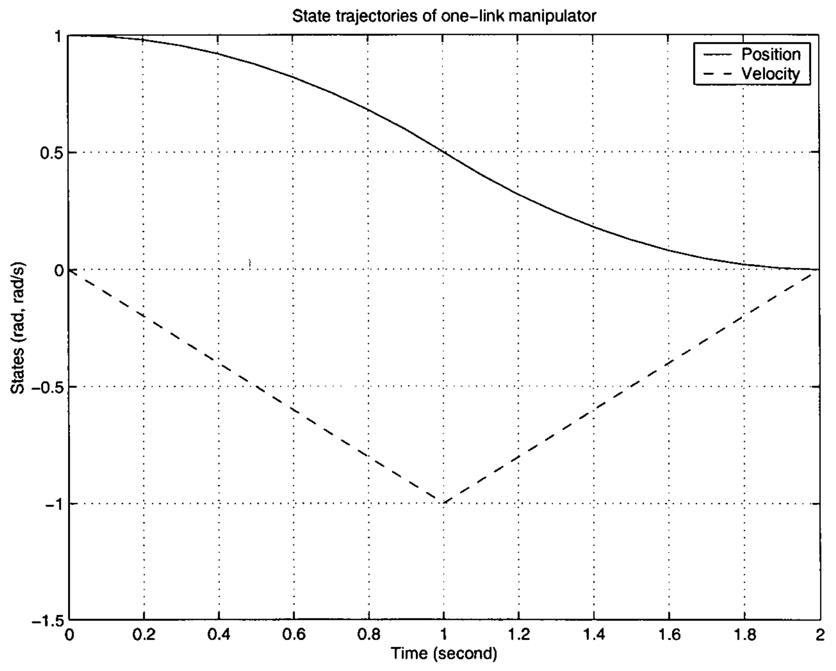

Figure 4.3: State trajectories of zero-net-work control simulation 1

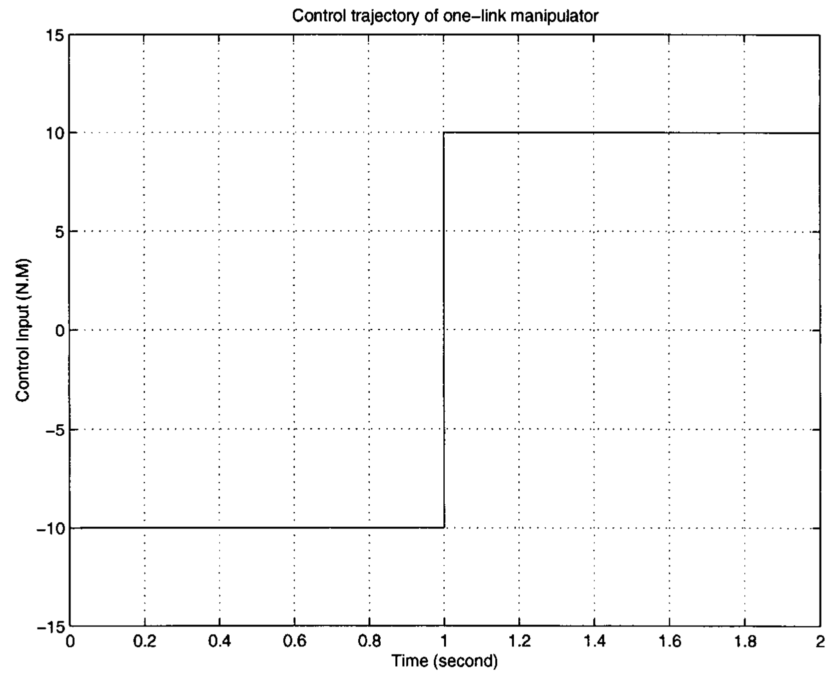

Figure 4.4: Applied joint torque of zero-net-work control simulation 1

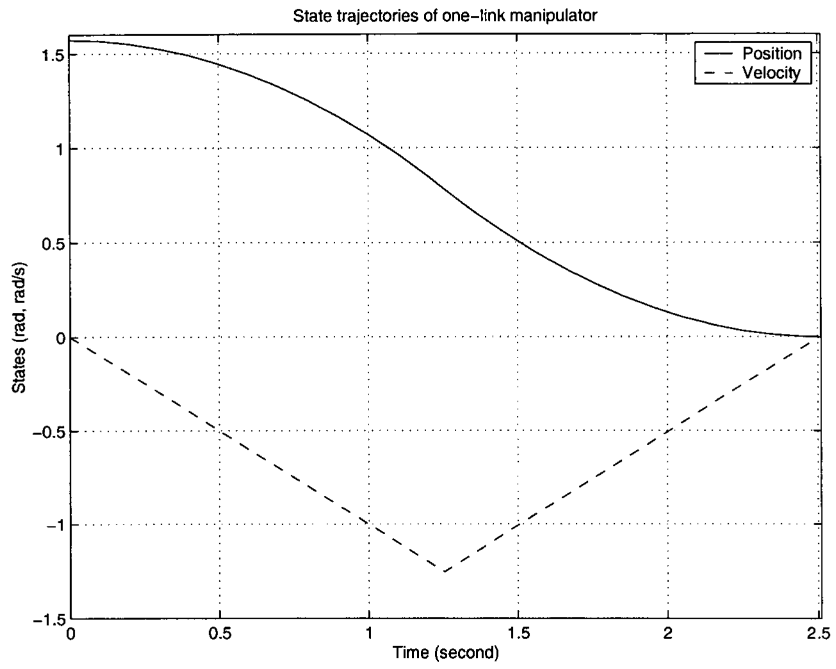

Figure 4.5: State trajectories of zero-net-work control simulation 2

$$t_f = 2t_{sw} = 2\sqrt{\frac{|\theta_0|I}{\tau_{max}}} = 2.51 \ second \tag{4.10}$$

So for this case using the zero-net-work method we can also obtain the exact minimum time solution.

4.1.5 Summary

From above we can see, for one-link horizontal manipulator, the zero-net-work method can obtain the exact minimum time solution. The zero-net-work method is very simple for this one link manipulator and the switching position can be easily found based on the initial position.

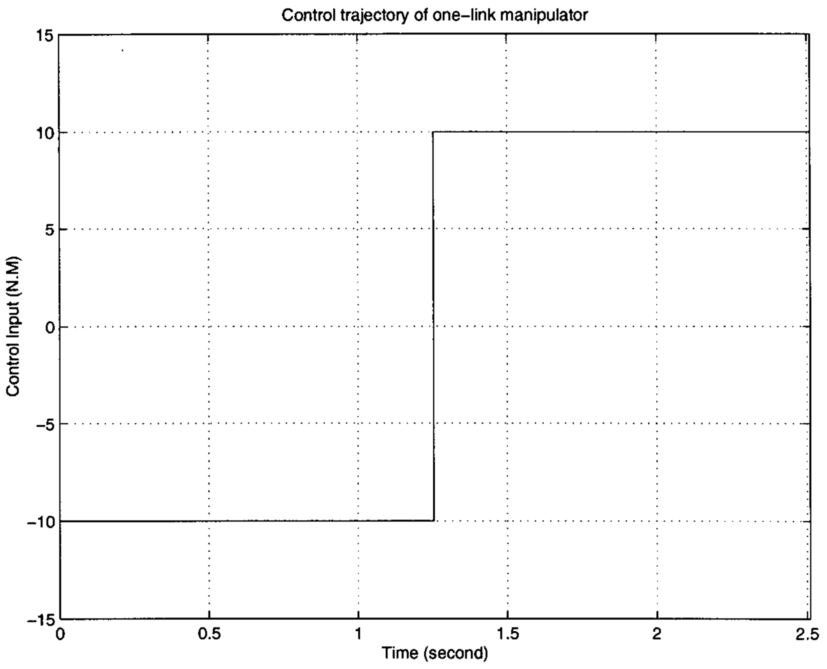

Figure 4.6: Applied joint torque of zero-net-work control simulation 2

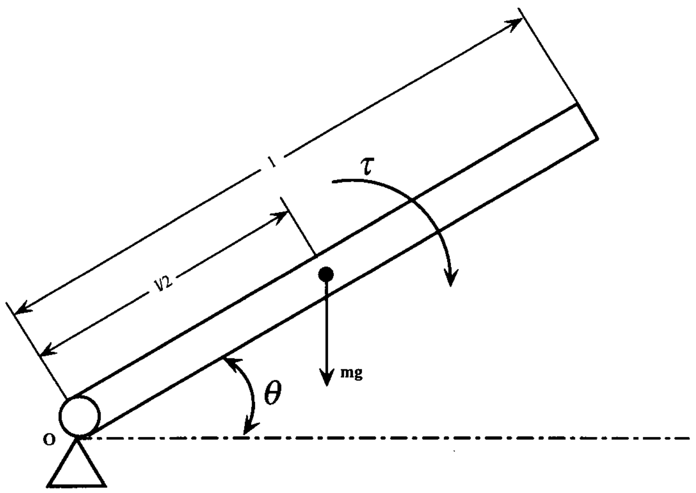

Figure 4.7: One-link vertical manipulator

4.2 One-Link Vertical Manipulator

4.2.1 Dynamic Model

Next we will consider a one-link vertical manipulator as shown in Figure 4.7. Assume gravity acts vertically downward. The equation of motion of the robot can be obtained using the Euler-Lagrange equation.

The kinetic energy of the system is:

$$KE = \frac{1}{2}I\dot{\theta}^2 \tag{4.11}$$

where I is the moment of inertial about the joint O.

The potential energy of the system:

$$PE = \frac{1}{2}mglsin\theta \tag{4.12}$$

The Lagrangian L is given by:

$$\begin{aligned} L &= KE - PE \\ &= \frac{1}{2}I\dot{\theta}^2 - \frac{1}{2}mglsin\theta \end{aligned} \tag{4.13}$$

Substituting above expression into Euler-Lagrange equation yields the equation of motion:

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{\theta}}) - \frac{\partial L}{\partial \theta} = I\ddot{\theta} + mgl/2cos\theta = \tau \tag{4.14}$$

The absolute value of the control torque τ is bounded:

$$|\tau| \leq \tau_{max} \tag{4.15}$$

The state space equations are:

$$\dot{x}_1(t) = x_2(t) \tag{4.16}$$

$$\dot{x}_2(t) = \frac{\tau(t)}{I} - \frac{mglcos(x_1(t))}{2I} \tag{4.17}$$

where

$$x_1 = \theta(t) \tag{4.18}$$

$$x_2 = \dot{\theta}(t) \tag{4.19}$$

$$|\tau| \leq \tau_{max} \tag{4.20}$$

4.2.2 The Exact Minimum Time Solution

The initial conditions of the robot are:

$$x_1(0) = x_{10} \tag{4.21}$$

$$x_2(0) = 0 \tag{4.22}$$

The final conditions are:

$$x_1(t_f) = 0 \tag{4.23}$$

$$x_2(t_f) = 0 \tag{4.24}$$

The cost functional is:

$$J = \int_{t_0}^{t_f} 1dt \tag{4.25}$$

The Exact Minimum Time Control (EMTC) problem of the one-link vertical robot can be stated as follows:

For the robot governed by the differential Equations (4.16 - 4.17) find an admissible control τ satisfying the constraints ($|\tau| \leq \tau_{max}$) which transfer the robot from initial position the fixed final position in minimum time, i.e., such that the cost functional J defined in Equation(4.25) is minimized.

Define the Hamiltonian as,

$$H(\mathbf{x}, \tau, \lambda) = 1 + \lambda_1 x_2 + \lambda_2[\frac{\tau}{I} - \frac{mglcosx_1}{2I}] \tag{4.26}$$

where λ_1, λ_2 are the costate variables. Using the techniques of variational calculus and applying Pontryagin's maximum principle, the necessary conditions for the EMTC can be written as:

State equation:

$$\dot{\mathbf{x}} = \frac{\partial H}{\partial \lambda} = f(\mathbf{x}, \tau, t) \tag{4.27}$$

Costate equation:

$$\frac{\partial H}{\partial \mathbf{x}} = -\dot{\lambda} \tag{4.28}$$

Stationary Condition:

$$H(\mathbf{x}^*, \lambda^*, \tau^*) \leq H(\mathbf{x}^*, \lambda^*, \tau) \tag{4.29}$$

Where τ^* is the optimal control, $\mathbf{x}^*$ and λ^* are the corresponding optimal state and costate trajectories. Substitute Equation (4.26) into Equation (4.29), we obtain:

$$\lambda_2 \tau^* \leq \lambda_2 \tau \tag{4.30}$$

From above equation, we can see the optimal control τ^* is depend on costate variable λ_2.

$$\tau^*(t) = \begin{cases} \tau_{max} & \text{if } \lambda_2 < 0 \\ -\tau_{max} & \text{if } \lambda_2 > 0 \end{cases}$$

Substitute Equation (4.26) into Equation (4.27) and Equation (4.28), the necessary conditions for EMTC of the one-link vertical robot are:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{\tau}{I} - \frac{mglcosx_1}{2I} \\ \dot{\lambda}_1 &= -\frac{mgl}{2I}\lambda_2 sinx_1 \\ \dot{\lambda}_2 &= -\lambda_1 \end{aligned} \tag{4.31}$$

where the control input τ is depend on λ_2. For the exact minimum time control problem the initial and final position are fixed so the boundary conditions for the above differential equation are:

$$\begin{aligned} x_1(0) &= x_{10} \qquad x_2(0) = 0 \\ x_1(t_f) &= 0 \qquad x_2(t_f) = 0 \end{aligned} \tag{4.32}$$

Since the Hamiltonian function does not explicity depend on time:

$$H(\mathbf{x}^*, \lambda^*, \tau^*) \equiv 0, \qquad \forall t \in [0, t_f] \tag{4.33}$$

From the above optimality conditions, we can see no boundary conditions for the state variable λ_1, λ_2 are defined. This fact leads to the problem of solving the Two Point Boundary Value Problem (TPBVP), which is very difficult to solve. In this thesis, for the purpose of comparing the zero-net-work method with the exact minimum time solution, we will use multiple shooting method to solve the resultant TPBVP. More specifically, the subroutine MUSN written in FORTRAN 77 and available through Netlib repository (http://www.netlib.org/) is used to obtain the solution (see Appendix B.1 for more detail).

MUSN (FDIF,Y0T,G,N,A,B,ER,TI,NTI,NRTI,AMP,ITLIM,Y,Q,U,NU,D,PHI,KP,W,LW, IW,LIW,WG,LWG,IERROR) uses a multiple shooting method to solve nonlinear TPBVP of the form:

$$\mathbf{y}' = f(t, \mathbf{y}), \qquad a < t < b \tag{4.34}$$

$$\mathbf{g}(\mathbf{y}(a), \mathbf{y}(b)) = 0 \tag{4.35}$$

Routine MUSN requires three subroutines. Subroutine FDIF(T,Y,F) evaluate the righthand side of the differential equation f(t,y) for $t = T$ and $y = Y$ and places the result in F(1), ...,F(N). where N is the order of the system; and Subroutine Y0T(T,Y) must evaluate the initial approximation y0(t) of the solution, for any value $t = T$ and place the result in Y(1),

...,Y(N). The third subroutine G(N,YA,YB,FG,DGA,DGB) must evaluate g(y(A),y(B)) for $y(A) = YA$ and $y(B) = YB$ and place the result in FG(1),...,FG(N). Moreover G must evaluate the Jacobians $\partial g(u,v)/\partial u$ for $u = YA$ and $\partial g(u,v)/\partial v$ for $v = YB$ and place the result in the arrays DGA anb DGB respectively. The convergence of the solution depend on the initial guess of the costate vector, so it can only handle problem of relatively less complexity. For the robot of more than three DOF, it becomes less efficient and very difficult to get the solution.

For exact minimum time control problem, the final time t_f is unknown, in order to use MUSN solving the resultant TPBVP, we have to transform it into a fixed time interval problem by a time normalization approach.

Let $T = t/t_f$, then $T \in [0,1]$ for $t \in [0,t_f]$, and $x(t) = x(T), \lambda(t) = \lambda(T)$, so

$$\frac{dx(T)}{dT} = \frac{dx(t)}{dt}\frac{dt}{dT} = t_f\frac{dx(t)}{dt} \tag{4.36}$$

$$\frac{d\lambda(T)}{dT} = t_f\frac{d\lambda(t)}{dt} \tag{4.37}$$

We introduce an additional dynamics equation for the final time to be minimized by letting $t_f = z(T)$, then $dz(T)/dT = 0$. Combining the above equations and invoking the boundary conditions yields the following equivalent TPBVP:

State equation:

$$\frac{dx(T)}{dT} = z(T)f(\mathbf{x},\tau,T) \tag{4.38}$$

Costate equation:

$$\frac{d\lambda(T)}{dT} = z(T)[-\frac{\partial H}{\partial x}] \tag{4.39}$$

Stationary Condition:

$$H(\mathbf{x}^*,\lambda^*,\tau^*) \leq H(\mathbf{x}^*,\lambda^*,\tau) \tag{4.40}$$

Boundary conditions:

$$\mathbf{x}(0) = \mathbf{x}_0 \qquad \mathbf{x}(1) = \mathbf{x}_f \tag{4.41}$$

Additional equation:

$$\frac{dz(T)}{dT} = 0 \tag{4.42}$$

The equivalent TPBVP for the one-link vertical robot is:

$$\begin{aligned}
\frac{dx_1}{dT} &= zx_2 \\
\frac{dx_2}{dT} &= z[\frac{\tau}{I} - \frac{mglcosx_1}{2I}] \\
\frac{d\lambda_1}{dT} &= -z\frac{mgl}{2I}\lambda_2 sinx_1 \\
\frac{d\lambda_2}{dT} &= -z\lambda_1 \\
\frac{dz}{dT} &= 0
\end{aligned} \tag{4.43}$$

The boundary conditions are:

$$\begin{aligned}
x_1(0) = x_{10} \qquad x_2(0) = 0 \\
x_1(1) = 0 \qquad x_2(1) = 0
\end{aligned} \tag{4.44}$$

We then can use subroutine MUSN to solve the above TPBVP and obtain the true minimum time solution.

4.2.3 Zero-Net-Work Solution

Next we will design the zero-net-work controller for above one-link vertical robot manipulator. There is one gravity force and one control torque act on the robot. For the zero-net-work method, we assume the control input is bang-bang and switch once. The torque before the switching point θ_{sw} is $\tau_b = -|\tau_{max}|$, the torque after the switching point is $\tau_a = |\tau_{max}|$, The work done by the torque and gravity before the switching point is

$$\begin{aligned} W_b &= -|\tau_{max}| \int_{\theta_0}^{\theta_{sw}} dt + \int_{\theta_0}^{\theta_{sw}} -mg[l/2cos\theta]d\theta \\ &= -|\tau_{max}|(\theta_{sw} - \theta_0) - mgl/2(sin\theta_{sw} - sin\theta_0) \end{aligned} \tag{4.45}$$

The work done by all the force after the switching point is:

$$\begin{aligned} W_a &= +|\tau_{max}| \int_{\theta_{sw}}^{\theta_f} dt + \int_{\theta_{sw}}^{\theta_f} -mg[l/2cos\theta]d\theta \\ &= +|\tau_{max}|(\theta_f - \theta_{sw}) - mgl/2(sin\theta_f - sin\theta_{sw}) \\ &= -|\tau_{max}|\theta_{sw} + mgl/2sin\theta_{sw} \end{aligned} \tag{4.46}$$

The work energy equation is:

$$\begin{aligned} Work &= W_b + W_a \\ &= -2|\tau_{max}|\theta_{sw} + |\tau_{max}|\theta_0 + 1/2mglsin\theta_0 = 0 \end{aligned} \tag{4.47}$$

So the switching point is:

$$\theta_{sw} = \theta_0/2 + \frac{mglsin\theta_0}{4|\tau_{max}|} \tag{4.48}$$

From above we can see, for the one link vertical manipulator, using the zero-net-work method, the switching point can be easily found. Based on this, we can design the zero-net-work controller for the one-link vertical manipulator as follow: We use the position sensor to get the position of the robot arm at every moment. When $\theta(t) < \theta_{sw}$ the control input is $+\tau_{max}$, and when $\theta(t) \geq \theta_{sw}$ the control input is $-\tau_{max}$. The block diagram of the zero-net-work control system of the one-link vertical robot is the same as for the one-link horizontal manipulator as shown in Figure 4.1.

4.2.4 Computer Simulation

The proposed zero-net-work method will now be simulated for the one-link vertical manipulator, and its performance will be compared with exact minimum time solution. The physical parameters of the manipulator are :

$\tau_{max} = 8N.m$, $m = 5kg$, $l = 0.3m$, $I = 0.15m^2kg$.

We use Simulink for the simulation. Figure 4.8 is the block diagram of the simulation. The system's S-function "one2.m" can be found in Appendix A.2 . The control input is bang-bang and has one switch and at every moment we compare the position output $\theta(t)$ with the switching position θ_{sw}. The control input switches from the maximum value to minimum value when $\theta(t) = \theta_{sw}$. If $\theta(t) \leq \theta_{sw}$, the control input $\tau(t) = -\tau_{max}$; if $\theta(t) > \theta_{sw}$, control input $\tau(t) = +\tau_{max}$. The simulation results for two different motion configurations are presented in the following:

Simulation Case 1:

In this case we will move the manipulator from initial state (1,0) to final state (0,0). Using zero-net-work method from Equation (4.48), we can find the switching position as:

$$\theta_{sw} = 0.8866 \tag{4.49}$$

Once we find the switching point, we can use this value in simulation block diagram shown in Figure 4.8 and use Simulink runs the simulation. Figure 4.9 and Figure 4.10 show the state and control trajectories of the robot. The simulation results show that the control input switches once at $t_{sw} = 0.0531s$, and the final time of the zero-net-work solution is $t_f = 0.6268s$. The exact solution for this case can be obtained by using MUSN to solve the equivalent TPBVP (Equation 4.43). The FORTRAN program for the one-link vertical exact minimum time solution can be found in Appendix A.3. Figure 4.11 is the state trajectories and Figure 4.12 is the control trajectory, Figure 4.13 is the costate trajectories of the exact minimum time solution. For the exact minimum time solution the control input switches once at $t_{sw} = 0.0531s$, the exact minimum time is $t_{tf} = 0.6268s$. The initial values for the costate variable are $\lambda_1 = 0.2446$, $\lambda_2 = 0.0125$. From Figure 4.12 and Figure 4.13 we can see the control input depends on λ_2. When $\lambda_2 > 0$ the control input is $-\tau_{max}$, and when $\lambda_2 > 0$ the control input is τ_{max}. From above we can see for the one-link vertical manipulator in this case, using zero-net-work method

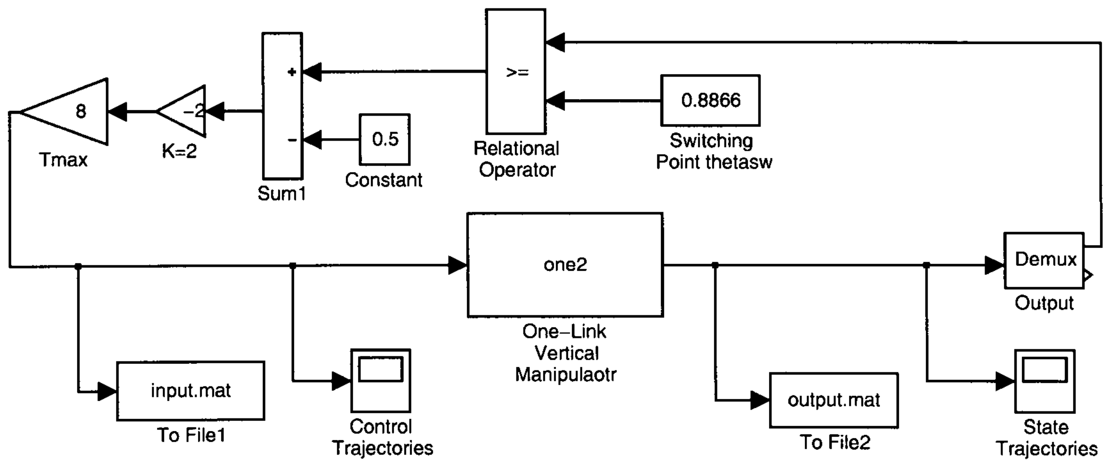

Figure 4.8: Case 1: Simulation block diagram of the one-link vertical manipulator

we obtain the exact minimum time solution.

Simulation Case 2:

In this case we will move the manipulator from initial state $(1.57, 0)$ to final state $(0, 0)$. Using zero-net-work method from Equation (4.48), we can find the switching position as:

$$\theta_{sw} = 1.2444 \tag{4.50}$$

Once we find the switching point, we can use this value in simulation block diagram shown in Figure 4.14 and use Simulink runs the simulation. Figure 4.15 and Figure 4.16 show the state and control trajectories of the manipulator. The simulation results show that the control input switches once at $t_{sw} = 0.1079s$, and the final time of the zero-net-work solution is $t_f = 0.7525s$.

The exact solution for this case can be obtained by using MUSN to solve the equivalent TPBVP (Equation 4.43). Figure 4.17 is the state trajectories and Figure 4.18 is the control trajectory, Figure 4.19 is the costate trajectories of the exact minimum time solution. For the exact minimum time solution the control input switches once at $t_{sw} = 0.1079s$, the exact minimum time is $t_{tf} = 0.7525s$. The initial values for the costate variable are $\lambda_1 = 0.2051$, $\lambda_2 = 0.0187$. From above we can see for the one-link vertical manipulator in case 2, using

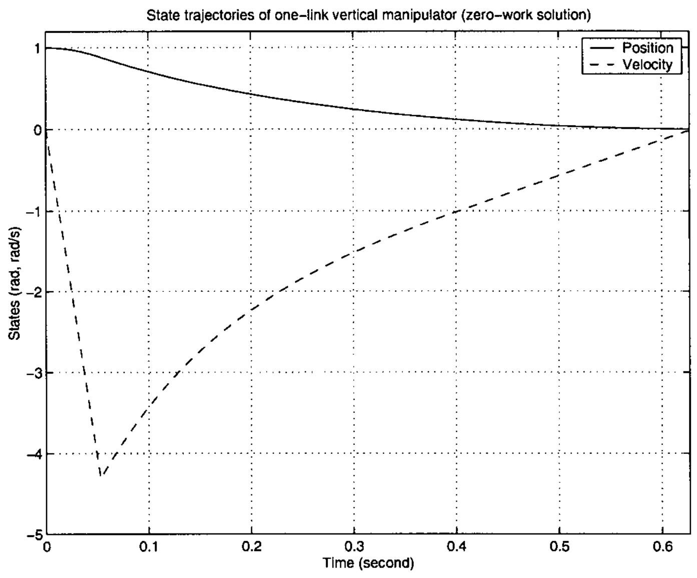

Figure 4.9: Case 1: State trajectories of the one-link vertical manipulator (zero-net-work solution)

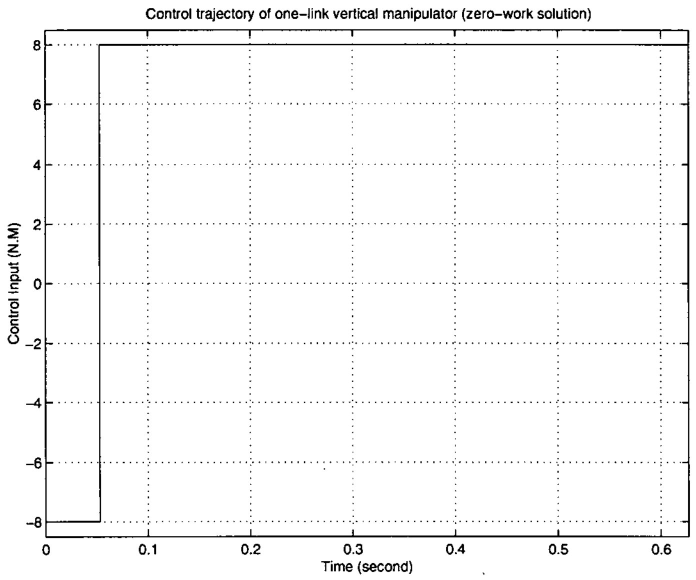

Figure 4.10: Case 1: Control trajectory of the one-link vertical manipulator (zero-net-work solution)

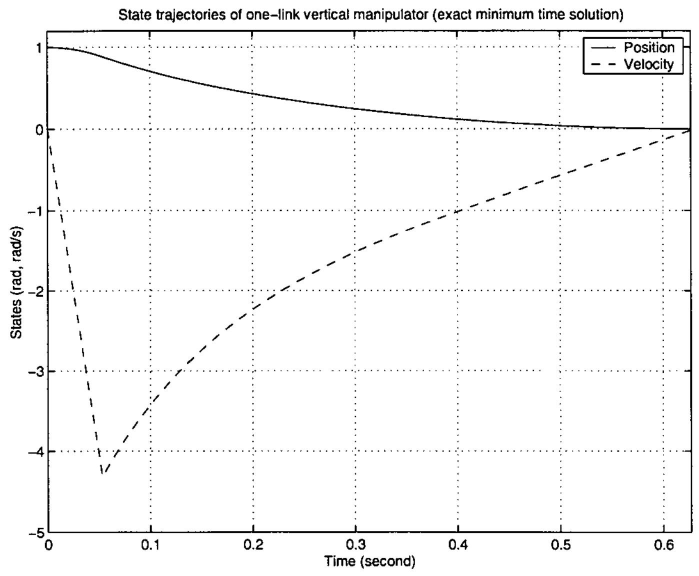

Figure 4.11: Case 1: State trajectories of the one-link vertical manipulator (exact minimum time solution)

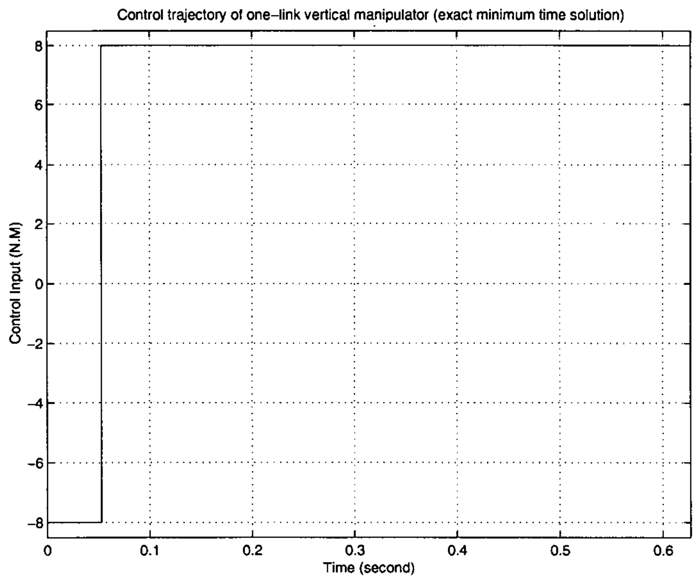

Figure 4.12: Case 1: Control trajectory of one-link vertical manipulator (exact minimum time solution)

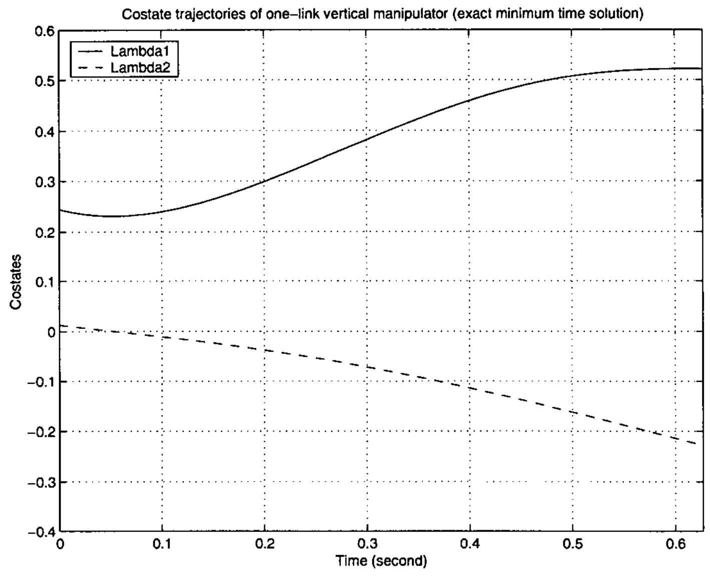

Figure 4.13: Case 1: Costate trajectories of the one-link vertical manipulator (exact minimum time solution)

zero-net-work method we can also obtain the exact minimum time solution.

4.2.5 Summary

From above simulations we can see, for a one-link robot manipulator, the zero-net-work method yields the exact minimum time solution. The zero-net-work method is much easier to implement and much simpler compared to the traditional method which requires the solution of the TPBVP. The traditional method requires the solution of the TPBVP for different initial positions, and also requires a good initial guess for the missing boundary values. For different initial conditions, the zero-net-work method can use the Equation (4.48) to find the switching position very easily, and the control input is based on the switching position.

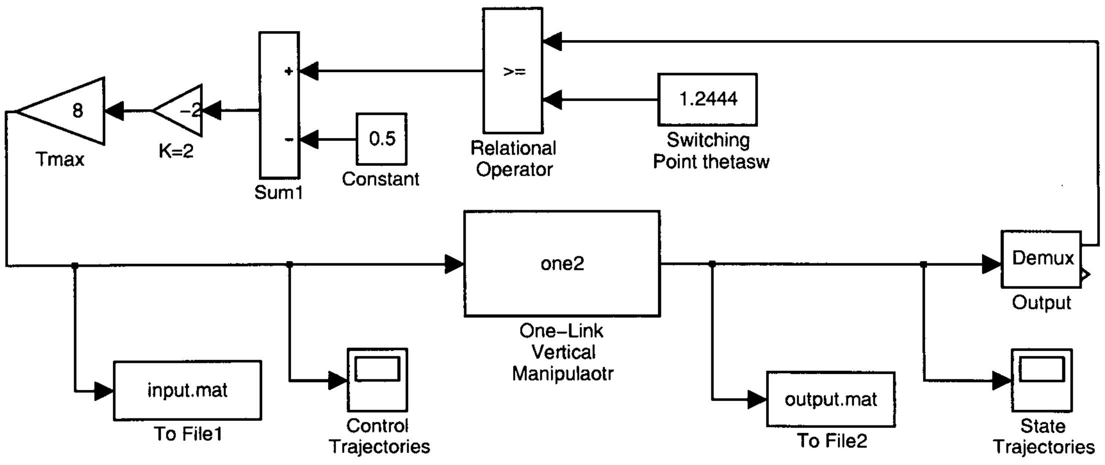

Figure 4.14: Case 2: Simulation block diagram of the one-link vertical manipulator

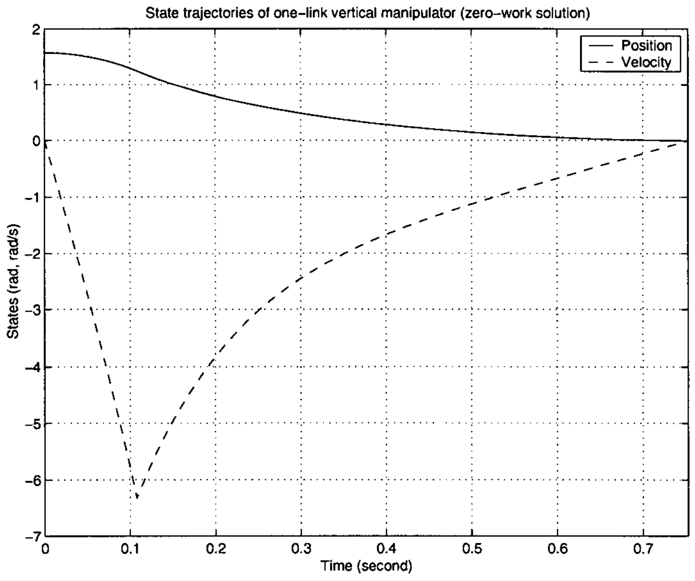

Figure 4.15: Case 2: State trajectories of the one-link vertical manipulator (zero-net-work solution)

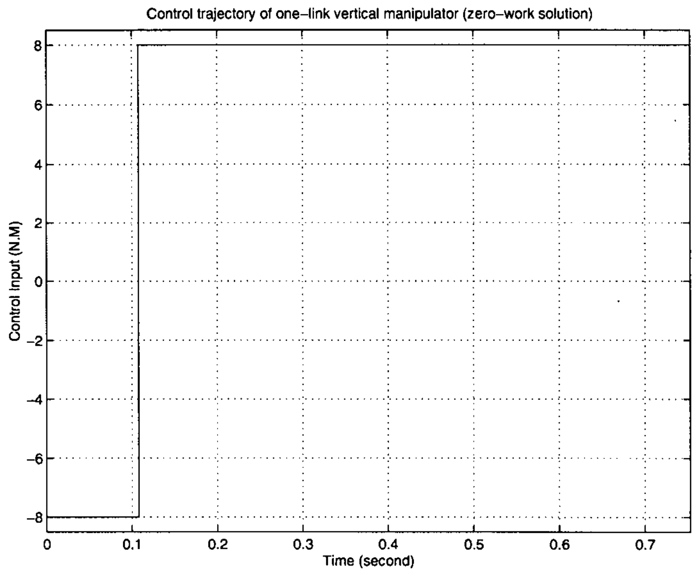

Figure 4.16: Case 2: Control trajectory of the one-link vertical manipulator (zero-net-work solution)

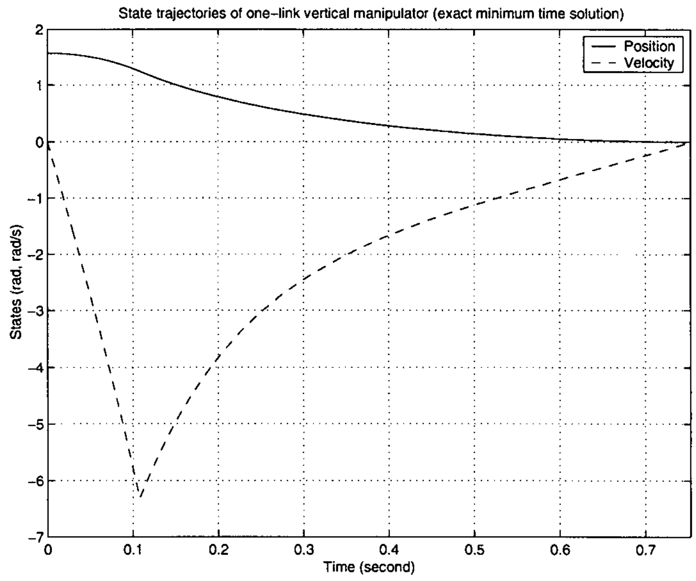

Figure 4.17: Case 2: State trajectories of the one-link vertical manipulator (exact minimum time solution)

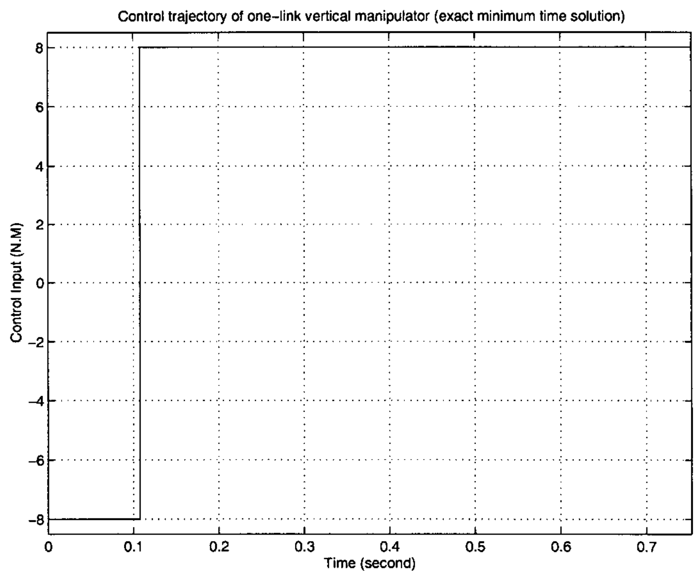

Figure 4.18: Case 2: Control trajectory of one-link vertical manipulator (exact minimum time solution)

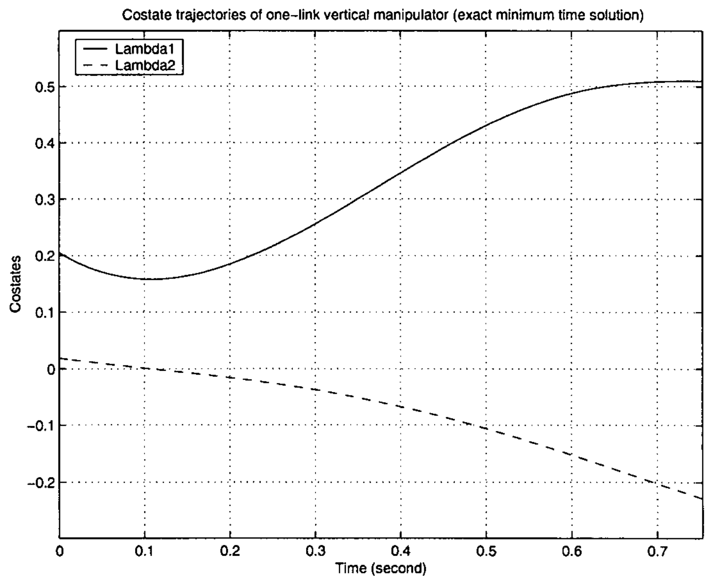

Figure 4.19: Case 2: Costate trajectories of the one-link vertical manipulator (exact minimum time solution)

Chapter 5

Development of the Zero-Net-Work Controller for a Cylindrical Manipulator

5.1 Dynamic Model

Next we will investigate the control of a cylindrical manipulator as shown in Figure 5.1. The first joint is revolute and produces a rotation about the base, while the second and third joints are prismatic. The joints of the robot's hand are neglected here. The first degree of freedom is the rotation θ_1. This coordinate is driven by a limited torque τ_1. The second degree of freedom is the radial translation r. It is driven by a limited force F_2. The third degree of freedom is the vertical translation. This coordinate is decoupled both from the first and second coordinate, so we can treat it separately as a one-link problem. We can use the zero-net-work method shown in Chapter 4 to design the zero-net-work controller for this link. The detail is neglected here, because it is same as the one-link manipulator in Chapter 4.

The control of the cylindrical robot can be treated as two parts: the first part is the one-link robot control problem of the third link; the second part is the planar Revolute and Prismatic (RP) two-link robot control problem. The influence of the third link on the first and second link is neglected and the second and the third link are lumped into a point mass m_2. The configuration for the second part of the cylindrical robot can be simplified as a two-link planar Revolute and Prismatic (RP) robot as shown in Figure 5.2, where I_1 is the moment of the inertial about the center of mass of link 1, m_1 is mass of link 1 and m_2 is mass of link 2. The absolute value of the control torque for link 1 is bounded:

$$|\tau_1| \leq \tau_{1max} \tag{5.1}$$

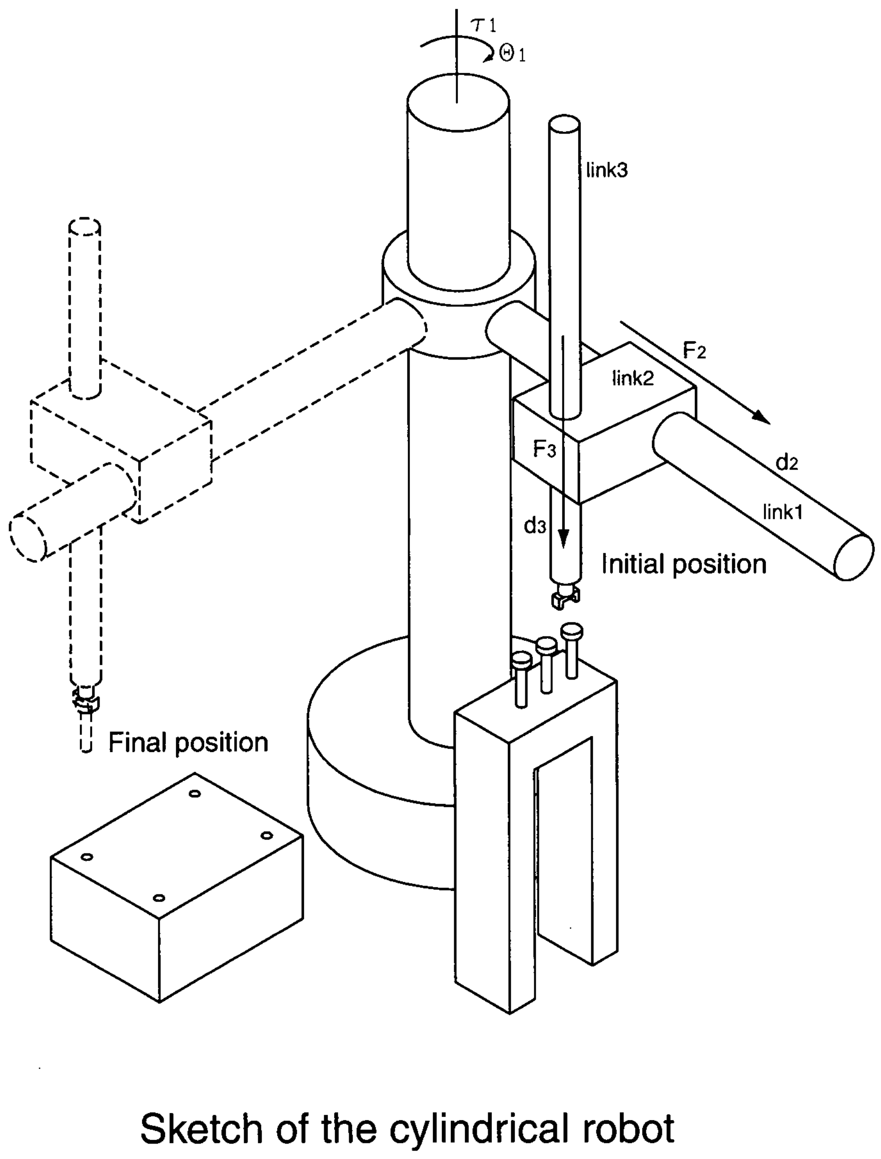

Figure 5.1: Sketch of the cylindrical robot

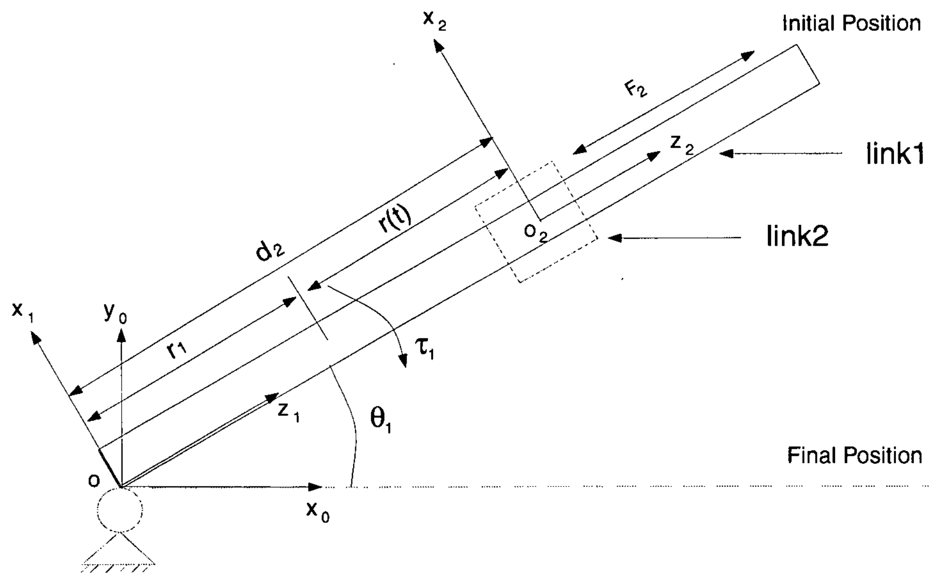

Figure 5.2: Sketch of two-link revolute and prismatic manipulator

Link	a_i	α_i	d_i	θ_i
1	0	$\pi/2$	0	$\pi/2 + \theta_1^*$
2	0	0	d_2^*	0

Table 5.1: D-H parameters of two-link revolute and prismatic manipulator

The control force F_2 for link 2 is also bounded:

$$|F_2| \leq F_{2max} \tag{5.2}$$

We establish the base frame F_0 as shown in Figure 5.2. Once the base frame is established, the F_1 frame and F_2 frame is fixed as shown by the Denavit-Hartenberg (D-H) convention. The D-H parameters are shown in Table 5.1,where $*$ means variable.

The homogeneous transformation from frame F_0 to frame F_1 is:

$$\mathbf{H}_{01} = \begin{bmatrix} -sin\theta_1 & 0 & cos\theta_1 & 0 \\ cos\theta_1 & 0 & sin\theta_1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.3}$$

The homogeneous transformation from F_1 to F_2 is:

$$\mathbf{H}_{12} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.4}$$

So the homogeneous transformation from F_0 to F_2 is:

$$\mathbf{H}_{02} = \mathbf{H}_{01}\mathbf{H}_{12} = \begin{bmatrix} -sin\theta_1 & 0 & cos\theta_1 & d_2cos\theta_1 \\ cos\theta_1 & 0 & sin\theta_1 & d_2sin\theta_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.5}$$

Next we will use the Newton-Euler method shown in Chapter 3 to obtain the dynamic equations of the robot. We begin with the forward recursion to express the various velocities and accelerations in term of θ_1, d_2 and their derivatives. The angular velocity and acceleration of link 1 are:

$$\boldsymbol{\omega}_1 = [0 \ \dot{\theta}_1 \ 0]^T \tag{5.6}$$

$$\boldsymbol{\alpha}_1 = [0 \ \ddot{\theta}_1 \ 0]^T \tag{5.7}$$

The angular velocity and acceleration of link 2 are:

$$\boldsymbol{\omega}_2 = [0 \ \dot{\theta}_1 \ 0]^T \tag{5.8}$$

$$\boldsymbol{\alpha}_2 = [0 \ \ddot{\theta}_1 \ 0]^T \tag{5.9}$$

5.1.1 Forward Recursion

The acceleration of the end of link 1 is:

$$\begin{aligned}\mathbf{a}_{e,1} &= \dot{\boldsymbol{\omega}}_1 \times \mathbf{r}_{1,2} + \boldsymbol{\omega}_1 \times (\boldsymbol{\omega}_1 \times \mathbf{r}_{1,2}) \\ &= [d_2\ddot{\theta}_1 \;\; 0 \;\; -d_2\dot{\theta}_1^2]^T \end{aligned} \tag{5.10}$$

The acceleration of the center of mass of link 1 is:

$$\begin{aligned}\mathbf{a}_{c,1} &= \dot{\boldsymbol{\omega}}_1 \times \mathbf{r}_{1,c1} + \boldsymbol{\omega}_1 \times (\boldsymbol{\omega}_1 \times \mathbf{r}_{1,c1}) \\ &= [\frac{l_1}{2}\ddot{\theta}_1 \;\; 0 \;\; -\frac{l_1}{2}\dot{\theta}_1^2]^T \end{aligned} \tag{5.11}$$

The acceleration of the center of mass of link 2 is:

$$\begin{aligned}\mathbf{a}_{c,2} &= (\mathbf{R}_1^2)^T\mathbf{a}_{e,1} + \dot{\boldsymbol{\omega}}_2 \times \mathbf{r}_{2,c2} + \boldsymbol{\omega}_2 \times (\boldsymbol{\omega}_2 \times \mathbf{r}_{2,c2}) + 2\boldsymbol{\omega}_2 \times \mathbf{z}_1\dot{d}_2 + \mathbf{z}_1\ddot{d}_2 \\ &= [(d_2\ddot{\theta}_1 + 2\dot{\theta}_1\dot{d}_2) \;\; 0 \;\; (-d_2\dot{\theta}_1^2 + \ddot{d}_2)]^T \end{aligned} \tag{5.12}$$

5.1.2 Backward Recursion

Now we carry out the backward recursion to compute the forces and joint torques. The force exerted by link 1 on link 2 is:

$$\begin{aligned}\mathbf{f}_2 &= m_2\mathbf{a}_{c,2} \\ &= m_2\begin{bmatrix} d_2\ddot{\theta}_1 + 2\dot{\theta}_1\dot{d}_2 \\ 0 \\ -d_2\dot{\theta}_1^2 + \ddot{d}_2 \end{bmatrix} \end{aligned} \tag{5.13}$$

The torque exerted by link 1 on link 2 is :

$$\tau_2 = [0 \;\; 0 \;\; 0]^T \tag{5.14}$$

So there are two forces act on link 2, f_{2x} and f_{2z}, Figure 5.3 shows all the forces and torques acting on link 1 and link 2.

$$f_{2x} = (d_2\ddot{\theta}_1 + 2\dot{\theta}_1\dot{d}_2)m_2 \tag{5.15}$$

$$f_{2z} = (-d_2\dot{\theta}_1^2 + \ddot{d}_2)m_2 \tag{5.16}$$

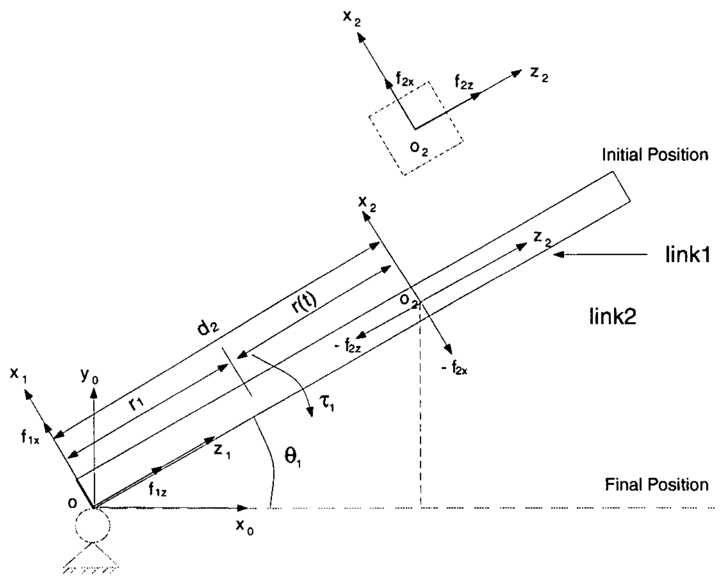

Figure 5.3: Forces and torques act on link 1 and link 2.

The force exerted by base the on link 1 is:

$$\begin{aligned} \mathbf{f}_1 &= m_1\mathbf{a}_{c,1} + \mathbf{R}_1^2\mathbf{f}_2 \\ &= \begin{bmatrix} (m_1l_1/2 + m_2d_2)\ddot{\theta}_1 + 2\dot{\theta}_1\dot{d}_2m_2 \\ 0 \\ -(m_1l_1/2 + m_2d_2)\dot{\theta}_1^2 + m_2\ddot{d}_2 \end{bmatrix} \end{aligned} \tag{5.17}$$

The torque exerted by the base on link 1 is:

$$\begin{aligned} \boldsymbol{\tau}_1 &= \mathbf{R}_1^2\boldsymbol{\tau}^2 - \mathbf{f}_1 \times \mathbf{r}_{1,c1} + (\mathbf{R}_1^2\mathbf{f}_2) \times \mathbf{r}_{2,c1} + I_1\boldsymbol{\alpha}_1 + \boldsymbol{\omega}_1 \times (I_1\boldsymbol{\omega}_1) \\ &= \begin{bmatrix} 0 \\ (I_1 + m_1l_1^2/4 + m_2d_2^2)\ddot{\theta}_1 + 2m_2d_2\dot{\theta}_1\dot{d}_2 \\ 0 \end{bmatrix} \end{aligned} \tag{5.18}$$

From the above we can obtain the dynamic equations of the two-link RP robot as:

$$[I_1 + m_1(l_1/2)^2 + m_2d_2^2]\ddot{\theta}_1 + 2m_2d_2\dot{d}_2\dot{\theta}_1 = \tau_1 \tag{5.19}$$

$$m_2\ddot{d}_2 - m_2d_2\dot{\theta}_1^2 = F_2 \tag{5.20}$$

The moment of inertia of link 1 about the origin of frame F_0 is:

$$I_{01} = I_1 + m_1(l_1/2)^2 \tag{5.21}$$

and

$$d_2(t) = r_1 + r(t) \tag{5.22}$$

Substituting Equation (5.21) and Equation (5.22) into Equation (5.19), (5.20) we obtain:

$$[I_{01} + m_2(r + r_1)^2]\ddot{\theta}_1 + 2m_2(r + r_1)\dot{r}\dot{\theta}_1 = \tau_1 \tag{5.23}$$

$$m_2\ddot{r} - m_2(r + r_1)\dot{\theta}_1^2 = F_2 \tag{5.24}$$

The dynamic equations in state space form are:

$$\dot{x}_1 = x_2 \tag{5.25}$$

$$\dot{x}_2 = \frac{\tau_1 - 2m_2(x_3 + r_1)x_2x_4}{I_{01} + m_2(x_3 + r_1)^2} \tag{5.26}$$

$$\dot{x}_3 = x_4 \tag{5.27}$$

$$\dot{x}_4 = \frac{F_2 + m_2(x_3 + r_1)x_2^2}{m_2} \tag{5.28}$$

where

$$x_1 = \theta_1(t) \qquad x_2 = \dot{\theta}_1(t) \tag{5.29}$$

$$x_3 = r(t) \qquad x_4 = \dot{r}(t) \tag{5.30}$$

5.2 The Exact Minimum Time Solution

The initial conditions of the robot are:

$$x_1(0) = x_{10} \qquad x_2(0) = 0 \tag{5.31}$$

$$x_3(0) = x_{20} \qquad x_4(0) = 0 \tag{5.32}$$

The final conditions are:

$$x_1(t_f) = x_2(t_f) = x_3(t_f) = x_4(t_f) = 0 \tag{5.33}$$

where t_f is the final time.

The absolute values of the control inputs are bounded:

$$|\tau_1| \leq \tau_{1max} \tag{5.34}$$

$$|F_2| \leq F_{2max} \tag{5.35}$$

The cost functional is:

$$J = \int_{t_0}^{t_f} 1dt \tag{5.36}$$

The Exact Minimum Time Control (EMTC) problem of the two-link planar Revolute and Prismatic (RP) can be stated as follows:
For the robot governed by the differential Equations (5.25 - 5.28) find an admissible control τ_1 and F_2 satisfying the constraints in Equation (5.34) and Equation (5.35) which transfer the robot from initial position to the fixed final position in minimum time, i.e., such that the cost functional J defined in Equation(5.36) is minimized.

Referring to Equations (5.25 - 5.28), we define

$$V_1 = \frac{\tau_1 - 2m_2(x_3 + r_1)x_2x_4}{I_{01} + m_2(x_3 + r_1)^2} \tag{5.37}$$

$$V_2 = \frac{F_2 + m_2(x_3 + r_1)x_2^2}{m_2} \tag{5.38}$$

The Hamiltonian is then

$$H(\mathbf{x}, \tau_1, F_2, \lambda) = 1 + \lambda_1 x_2 + \lambda_2 V_1 + \lambda_3 x_4 + \lambda_4 V_2 \tag{5.39}$$

where λ_1, λ_2 , λ_3, λ_4 are the costate variables. The necessary conditions for the EMTC can be written as:

State equation:

$$\dot{\mathbf{x}} = \frac{\partial H}{\partial \lambda} = f(\mathbf{x}, \tau_1, F_2, t) \tag{5.40}$$

Costate equation:

$$\frac{\partial H}{\partial \mathbf{x}} = -\dot{\lambda} \tag{5.41}$$

Stationary Condition:

$$H(\mathbf{x}^*, \lambda^*, \tau_1^*, F_2^*) \leq H(\mathbf{x}^*, \lambda^*, \tau_1, F_2) \tag{5.42}$$

Where τ_1^* and F_2^* are the optimal controls, $\mathbf{x}^*$ and λ^* are the corresponding optimal state and costate trajectories. Substituting Equation (5.39) into Equation (5.42), we can see the optimal controls are dependent on the costate variables λ_2 and λ_4.

$$\tau_1^*(t) = \begin{cases} \tau_{1max} & \text{if } \lambda_2 < 0 \\ -\tau_{1max} & \text{if } \lambda_2 > 0 \end{cases}$$

$$F_2^*(t) = \begin{cases} F_{2max} & \text{if } \lambda_4 < 0 \\ -F_{2max} & \text{if } \lambda_4 > 0 \end{cases}$$

By using the time normalization technique shown in Chapter 4, we can transform EMTC problem into a fixed time interval problem. The equivalent Two Point Boundary Value Problem (TPBVP) for the two link RP robot is:

$$\begin{aligned}
\frac{dx_1}{dT} &= zx_2 && (5.43)\\
\frac{dx_2}{dT} &= zV_1 && (5.44)\\
\frac{dx_3}{dT} &= zx_4 && (5.45)\\
\frac{dx_4}{dT} &= zV_2 && (5.46)\\
\frac{d\lambda_1}{dT} &= 0 && (5.47)\\
\frac{d\lambda_2}{dT} &= -z[\lambda_1 + \lambda_2\frac{\partial V_1}{\partial x_2} + \lambda_4\frac{\partial V_2}{\partial x_2}] && (5.48)\\
\frac{d\lambda_3}{dT} &= -z[\lambda_2\frac{\partial V_1}{\partial x_3} + \lambda_4\frac{\partial V_2}{\partial x_3}] && (5.49)\\
\frac{d\lambda_4}{dT} &= -z[\lambda_3 + \lambda_2\frac{\partial V_1}{\partial x_4}] && (5.50)\\
\frac{dz}{dT} &= 0 && (5.51)
\end{aligned}$$

where $T \in [0,1]$ for $t \in [0, t_f]$, and

$$\begin{aligned} z(T) &= t_f && (5.52) \\ T &= \frac{t}{t_f} && (5.53) \end{aligned}$$

Let

$$\begin{aligned} V_1 &= \frac{U}{V} \\ &= \frac{\tau_1 - 2m_2(x_3+r_1)x_2x_4}{I_{01} + m_2(x_3+r_1)^2} && (5.54) \\ V_2 &= \frac{F_2 + m_2(x_3+r_1)x_2^2}{m_2} && (5.55) \end{aligned}$$

Then

$$\begin{aligned} \frac{\partial V_1}{\partial x_2} &= \frac{-2m_2(x_3+r_1)x_4}{V} && (5.56) \\ \frac{\partial V_1}{\partial x_3} &= \frac{-2m_2x_2x_4V - 2m_2(x_3+r_1)U}{V^2} && (5.57) \\ \frac{\partial V_1}{\partial x_4} &= \frac{-2m_2(x_3+r_1)x_2}{V} && (5.58) \\ \frac{\partial V_2}{\partial x_2} &= 2x_2(x_3+r_1) && (5.59) \\ \frac{\partial V_2}{\partial x_3} &= x_2^2 && (5.60) \end{aligned}$$

The boundary conditions are:

$$\begin{aligned} x_1(0) &= x_{10} & x_2(0) &= 0 \\ x_3(0) &= x_{20} & x_4(0) &= 0 \\ x_1(1) &= 0 & x_2(1) &= 0 \\ x_3(1) &= 0 & x_4(1) &= 0 \qquad (5.61) \end{aligned}$$

From the above equations, we can see the boundary conditions for the costate variables are unknown. In order to get the exact minimum time solution, we have to solve the above TPBVP. We can use subroutine MUSN to solve the TPBVP. The convergence of the solution depend on the initial guess of the unknown parameters. In this case we have to give the initial guesses for 5 unknown parameters, that is λ_1, λ_2, λ_3, λ_4 and the final time z. The convergence of the solution is very sensitive to the unknown initial guesses. For this reason, it is very hard to get the exact minimum time solutions using the traditional calculus of variation method.

5.3 Zero-Net-Work Solution

Next we will design the zero-net-work controller for above two-link Revolute and Prismatic (RP) robot manipulator. The basic algorithm of the zero-net-work method has been given in Chapter 3. For the RP manipulator in this chapter, we assume the final positions and velocities of each link are equal to zero. For zero-net-work method the control torque for link 1 is bang-bang and has one switch and the control force for link 2 is also bang-bang and has one switch. We can use the results from Newton-Euler formulation of the dynamic equations in Section 5.1 to calculate the work done by all the forces/torques for each link. All the work will be calculated in base frame F_0.

For link 1 the work done by all the forces/torques can be calculated as follow: As shown in Figure 5.3, there are one torque τ_1 and four forces $(f_{1x}, f_{1z}, -f_{2x}, -f_{2z})$ act on it. In frame F_0, the displacement of point o is always zero. If we move all the forces to point o, forces $f_{1x}, f_{1z}, -f_{2z}$ pass through point o, so the work done by these forces is always zero.

The displacement of o_2 in x direction in frame F_0 is:

$$\delta x = \delta[(r_1 + r)cos\theta_1] \tag{5.62}$$

The displacement of o_2 in y direction in frame F_0 is:

$$\delta y = \delta[(r_1 + r)sin\theta_1] \tag{5.63}$$

Reaction force:

$$f_{2x} = m_2[(r_1 + r)\ddot{\theta}_1 + 2\dot{\theta}_1\dot{r}_1] \tag{5.64}$$

Work done by all the forces/torques for link 1 is:

$$Work_1 = \tau_1\delta\theta_1 - f_{2x}(r_1 + r)\delta\theta_1 \tag{5.65}$$

For link 2 as shown in Figure 5.3, there are two forces f_{2x}, f_{2z} act on it. $f_{2z} = F_2$ is the control input for link 2. Work done by all the forces for link 2 is:

$$Work_2 = (-f_{2x}sin\theta_1 + F_2cos\theta_1)\delta x + (f_{2x}cos\theta_1 + F_2sin\theta_1)\delta y \tag{5.66}$$

Since we assume the control input for each link is alway bang-bang and switch once. For different physical parameters and motion configurations, we can not always find the solution that both link reach the final position at the same time using maximum or minimum control input only. Following assumption must be made: once the link reaches the final position, the control input will be changed to the torque/force needed to hold the link at the final position. For the two link planar RP manipulator there is only two possibilities:

Case 1: Link 1 reaches the final position first. This means:

$$\theta_1 = 0;\ \dot{\theta}_1 = 0;\ \ddot{\theta}_1 = 0 \tag{5.67}$$

In order to hold link 1 at it's final position, from Equation (5.23) the control input for joint 1 is:

$$\tau_1 = 0 \tag{5.68}$$

This means the movement of link 2 has no effect on link 1, once link 1 reaches the final position. When link 1 reaches the final position, we can turn the control input τ_1 off.

Case 2: Link 2 reaches the final position first. This means:

$$r = 0;\ \dot{r} = 0;\ \ddot{r} = 0 \tag{5.69}$$

In order to hold link 2 at the final position, from Equation (5.24) the control input for joint 2 is:

$$F_2 \quad = \quad -m_2 r_1 \dot{\theta}_1^2 \tag{5.70}$$

Once the link 2 reaches the final position, the control input for link 2 is changed to $-m_2 r_1 \dot{\theta}_1^2$.

We first set the switching positions of link 1 and link 2 to zero, this means the control inputs do not switch until the final point is reached. So each link can reach the final position, but the work done by all the forces and torques for each link is not equal to zero at the final point. We will then increase the switching position for each link, and calculate the work done by all the forces and torques at the final position. When the work done for each link at final position is zero, then we have found the zero-net-work switching point for each link. After we find the zero-net-work switching point for each link, we have specified the zero-net-work controller for the RP robot manipulator. We use the position sensor to obtain the position of the robot arm at every moment. For link 1, when $\theta_1(t) < \theta_{1sw}$ the control input is $+\tau_{1max}$; when $\theta_1(t) \geq \theta_{1sw}$ the control input is $-\tau_{1max}$. For link 2, when $r(t) < r_{sw}$ the control input is $+F_{2max}$; when $r(t) \geq rsw$ the control input is $-F_{2max}$. The FORTRAN program which is used to find the zero-net-work switching positions is shown in Appendix A.4. FORTRAN subroutine DDASSL shown in Appendix B.2 is used to integrate the dynamic equations of the robot system.

5.4 Computer Simulation

The proposed zero-net-work method will now be simulated for the two-link Revolute and Prismatic (RP) manipulator, and its performance will be compared with the exact minimum time solution. We will use the zero-net-work method to two different parameter sets of the RP manipulator. For each parameter set, we will perform simulations for two different motion configurations. Table 5.2 shows the two different parameter set for the simulations.

Set	τ_{1max} N.m	F_{2max} N	r_1 m	l_1 m	m_1 Kg	m_2 Kg	I_0 kgm^2
1	2	2	0.6	0.9	3	1	0.81
2	3	2.5	0.4	1.2	4	1.5	1.92

Table 5.2: Physical parameters of two-link revolute and prismatic manipulator

We use Simulink to perform the simulation. The system's S-function can be found in

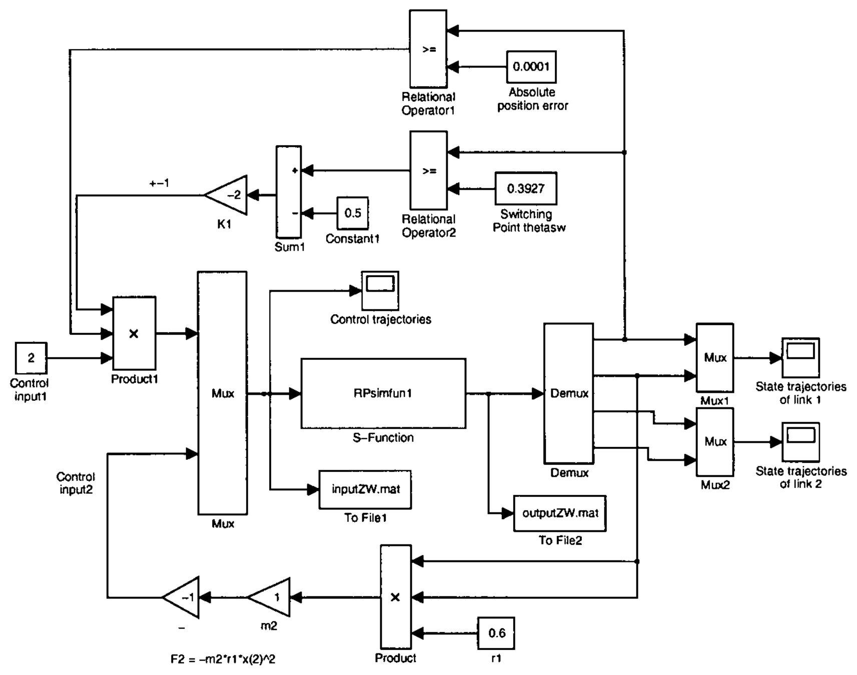

Figure 5.4: Case 1: Simulation block diagram

Appendix A.6. The parameter set 1 shown in Table 5.2 is used for the simulation case 1 and case 2, the parameter set 2 is used for the simulation case 3 and case 4. The simulation results for different parameters and motion configurations are presented in the following:

5.4.1 Simulation Case 1:

Case 1	θ_0	r_0	θ_{sw}	r_{sw}	t_{1sw}	t_{2sw}	t_{1f}	t_{2f}	error
Zero-net-work	$\pi/4$	0	0.3927		0.6778		1.3449		6.74 %
EMTC	$\pi/4$	0			0.6308	0.3510 0.9094		1.2600	

Table 5.3: Case 1: Simulation results of two-link revolute and prismatic manipulator

In this case we will move the manipulator from initial state ($\frac{\pi}{4}$, 0, 0, 0) to final state (0, 0, 0, 0). The FORTRAN program using zero-net-work method to find the switching position for

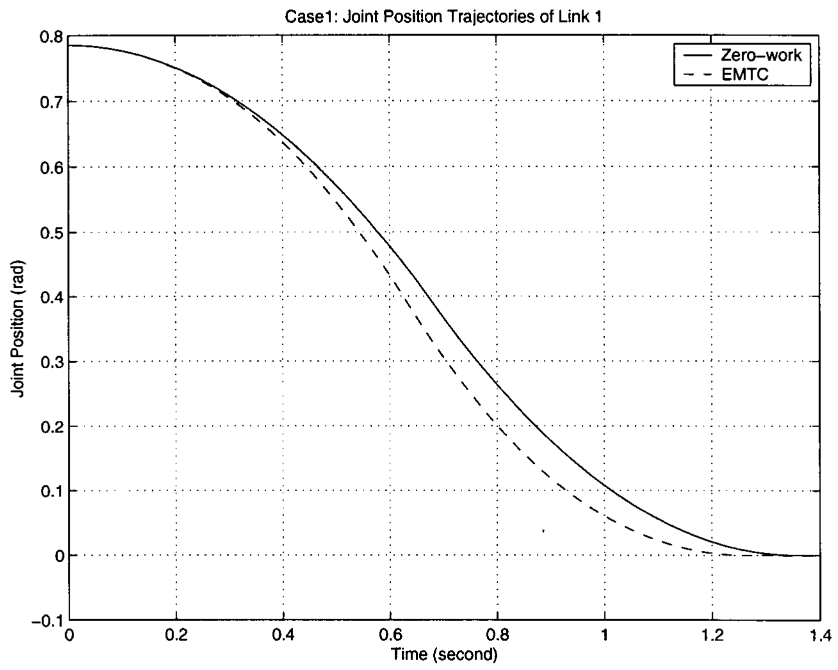

Figure 5.5: Case 1: State trajectories of link 1

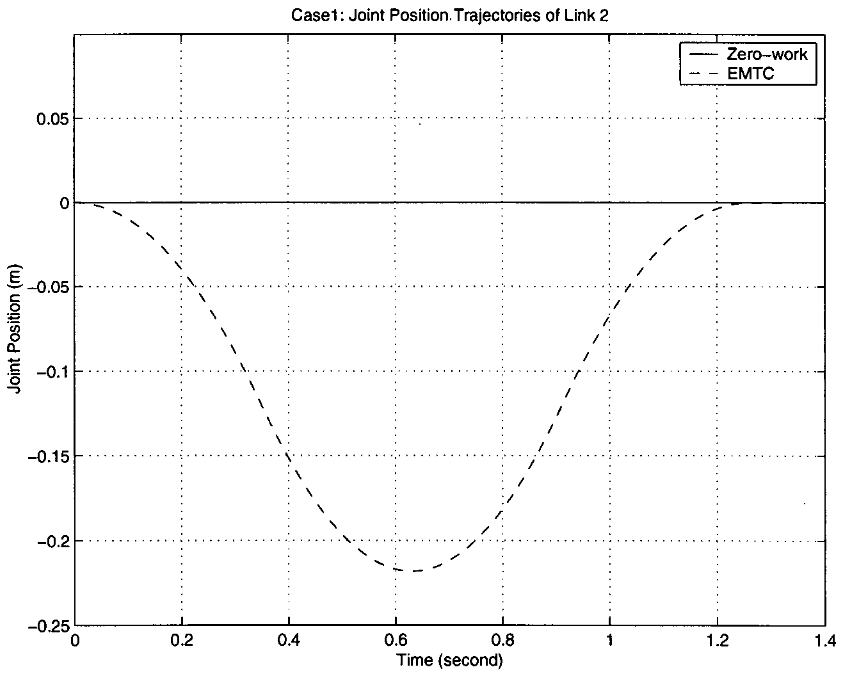

Figure 5.6: Case 1: State trajectories of link 2

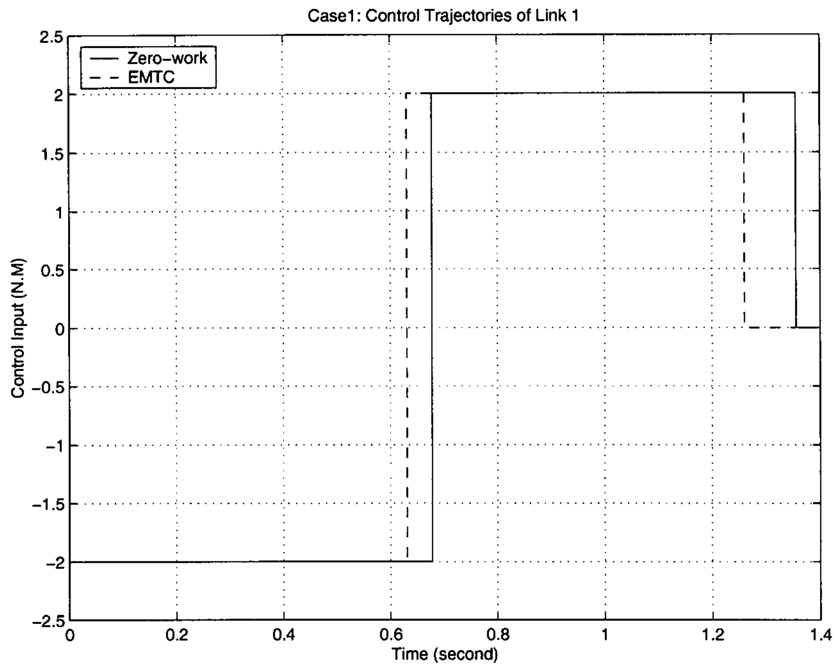

Figure 5.7: Case 1: Control trajectories of link 1

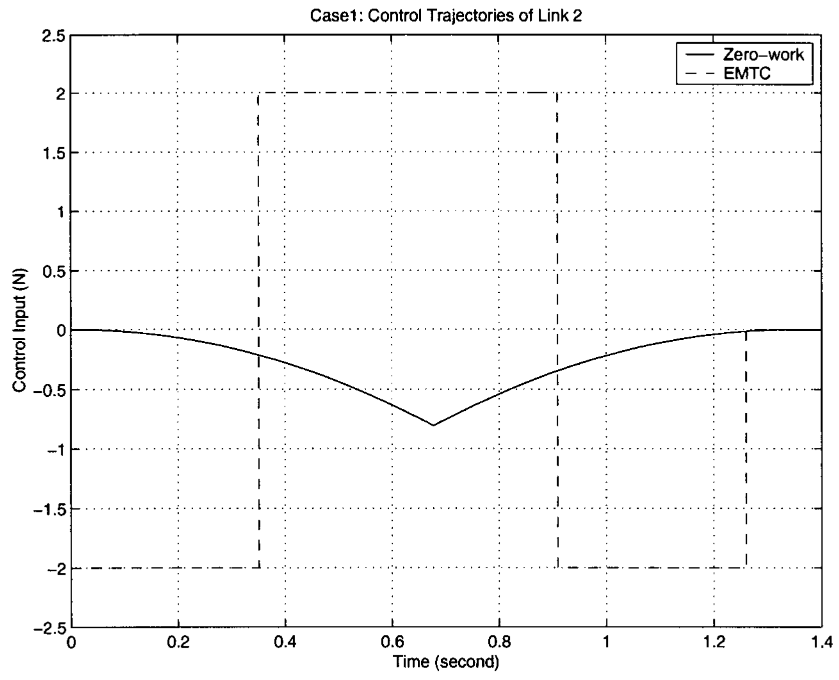

Figure 5.8: Case 1: Control trajectories of link 2

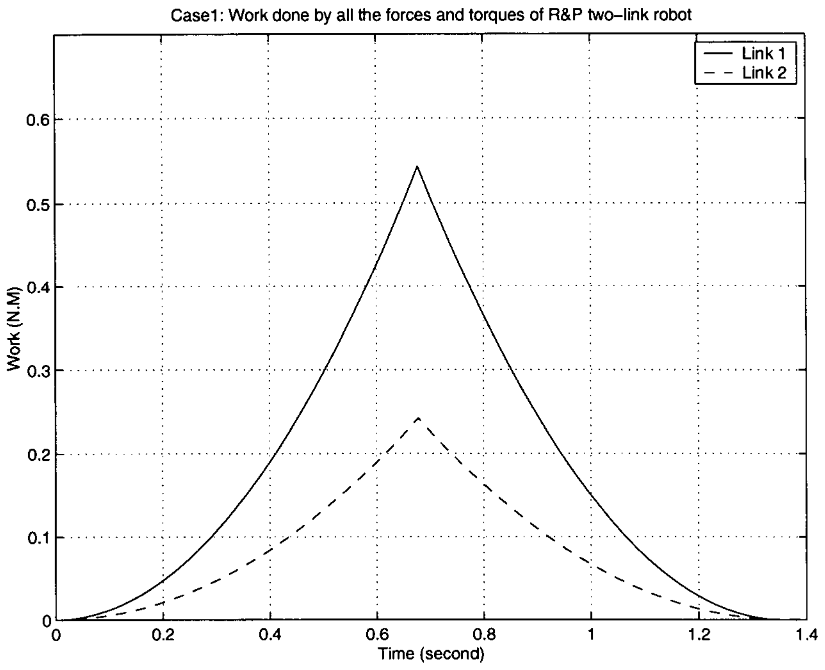

Figure 5.9: Case 1: Work done by forces/torques for each link (zero-net-work solution)

each link is shown in Appendix A.4 . The simulation results are shown in Table 5.3. For link 1, θ_0 is the initial position, θ_{sw} is the switching position, t_{1sw} is the switching time, t_{1f} is the final time. For link 2, r_0 is the initial position, r_{sw} is the switching position, t_{2sw} is the switching time, t_{2f} is the final time.

In this case, the initial velocity and position of link 2 are zero. So there is no minimum to maximum switch for link 2. The control input for link 2 start from the beginning is the force needed to hold link 2 at the final position while link 1 is moving. Control input for link 1 has one switch: $\theta_{1sw} = 0.3927rad$. From offline calculation we find the switching positions, we can then use these values in simulation block diagram shown in Figure 5.4 and use Simulink run the simulation. Exact minimum time solution can be obtained by solving the Two Point Boundary Problem (TPBVP). The FORTRAN program for exact minimum time solution of the two-link revolute and prismatic manipulator can be found in Appendix A.5. The exact minimum time is $t_{fexact} = 1.2600sec$, zero-net-work time is $t_{fZW} = 1.3449sec$. The time difference between the proposed zero-net-work control method and Exact Minimum Time Control (EMTC) is 6.74%. Thus we can say zero-net-work control method leads to a near minimum time solution. Figure 5.5 shows the state trajectories of link 1, Figure 5.6 shows the state trajectories of link 2. Figure 5.7 shows the control trajectories of link 1, Figure 5.8 shows the control trajectories of link 2. The work done by external and reaction forces/torques for zero-net-work method are shown in Figure 5.9.

5.4.2 Simulation Case 2:

Case 2	θ_0	r_0	θ_{sw}	r_{sw}	t_{1sw}	t_{2sw}	t_{1f}	t_{2f}	error
Zero-net-work	0.2	0.3	0.1083	0.1426	0.3709	0.4007	0.7383	0.7810	1.69 %
EMTC	0.2	0.3			0.0170 0.4020	0.3999		0.7680	

Table 5.4: Case 2: Simulation results of two-link revolute and prismatic manipulator

In this case we will move the manipulator from initial state (0.2, 0, 0.3, 0) to final state (0, 0, 0, 0). The simulation results are shown in Table 5.4. Control input for link 1 has one switch: $\theta_{1sw} = 0.1083 rad$. Control input for link 2 has one switch: $r_{sw} = 0.1426m$. From offline calculation we find the switching positions, we can then use these values in simulation block diagram shown in Figure 5.10 and use Simulink run the simulation. The exact minimum time is $t_{fexact} = 0.7680 sec$, zero-net-work time is $t_{fZW} = 0.7810 sec$. The time difference between the proposed zero-net-work control method and EMTC is 1.69%. Figure 5.11 shows the state trajectories of link 1, Figure 5.12 shows the state trajectories of link 2. Figure 5.13 shows the control trajectories of link 1, Figure 5.14 shows the control trajectories of link 2. The work done by external and reaction forces/torques for zero-net-work method are shown in Figure 5.15.

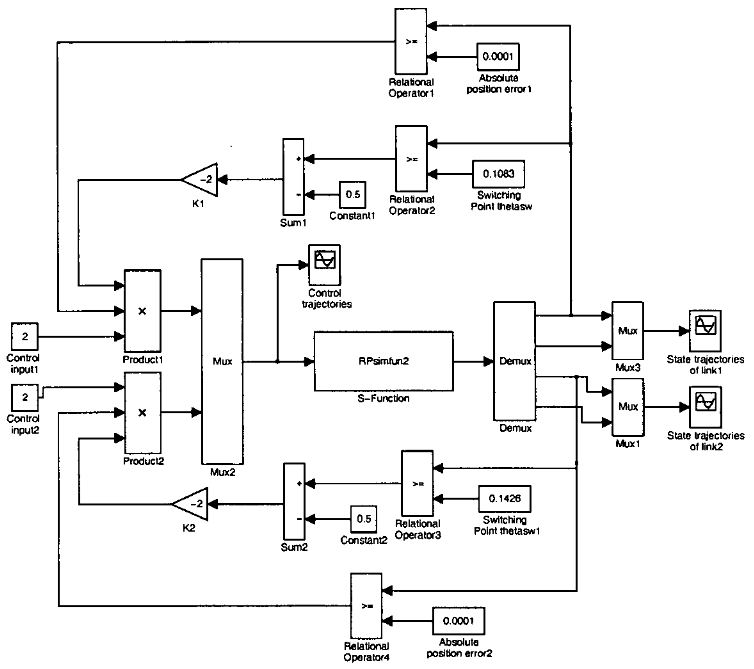

Figure 5.10: Case 2: Simulation block diagram

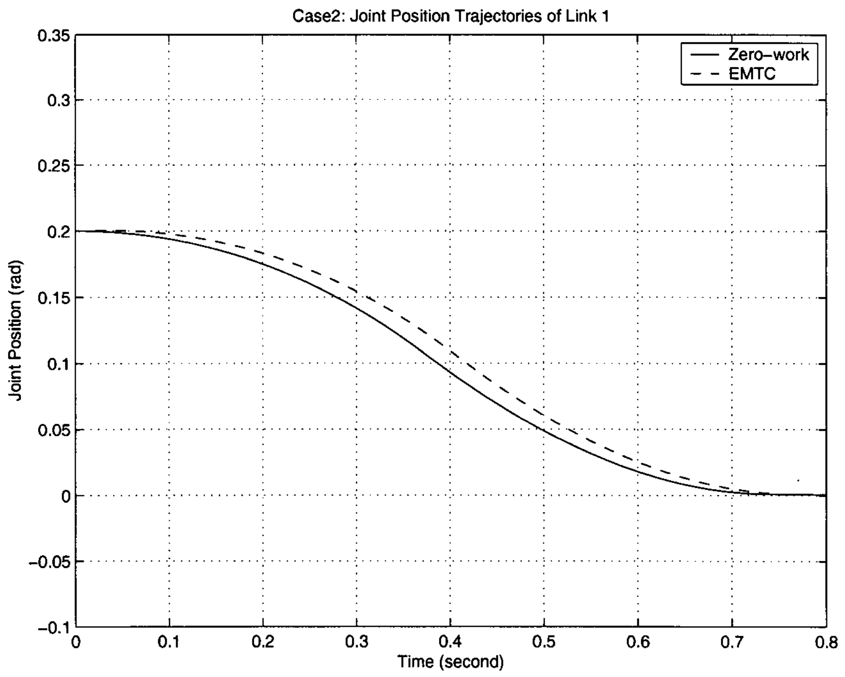

Figure 5.11: Case 2: State trajectories of link 1

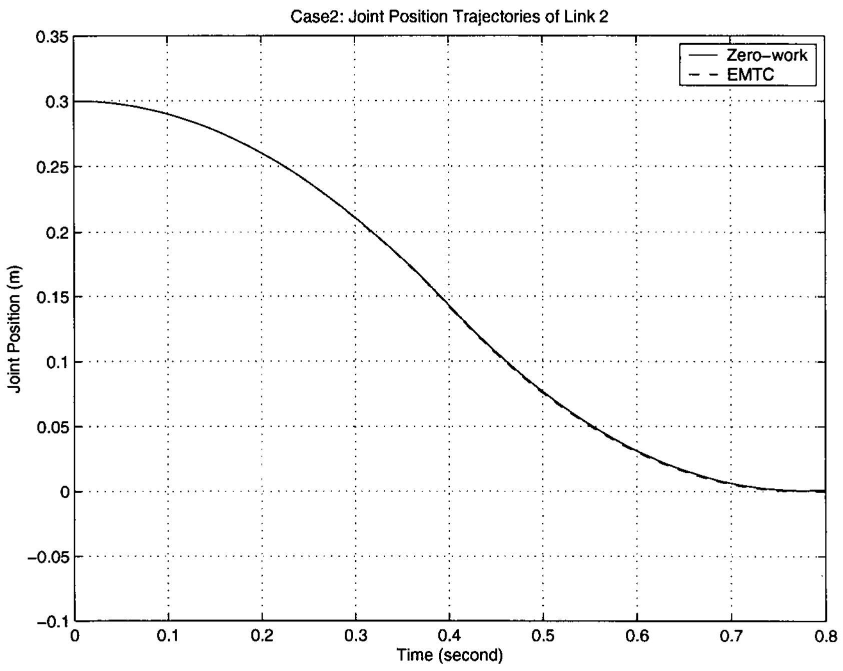

Figure 5.12: Case 2: State trajectories of link 2

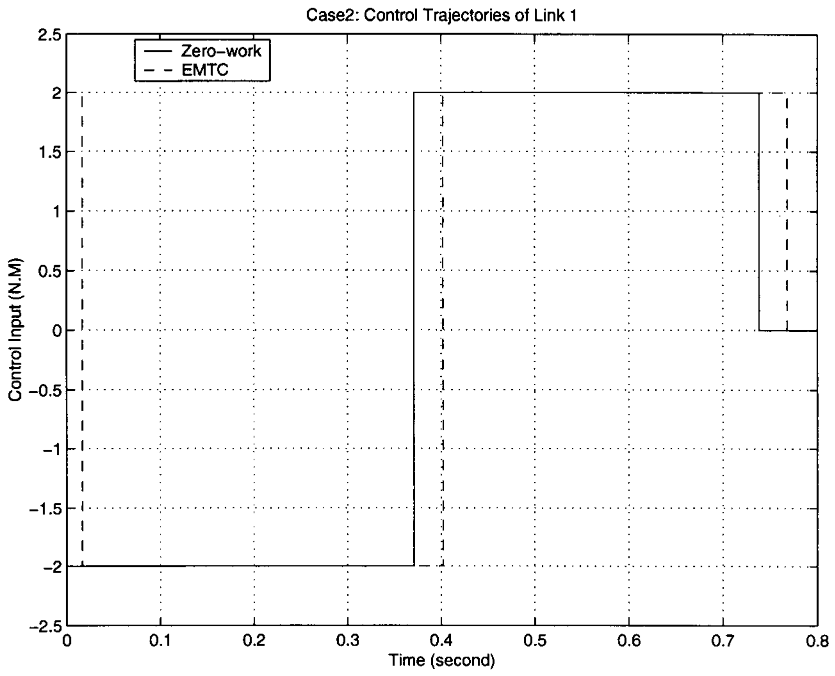

Figure 5.13: Case 2: Control trajectories of link 1

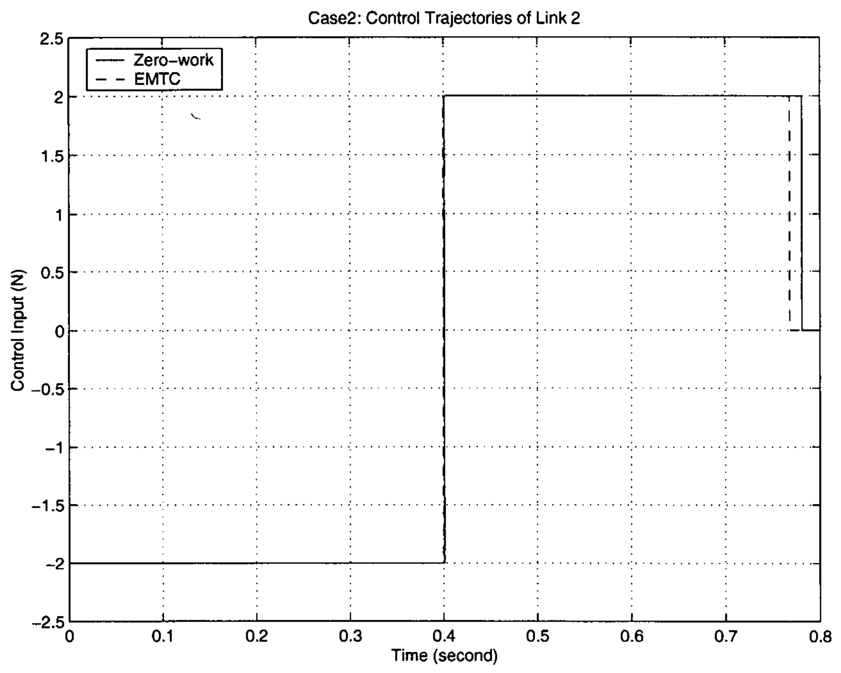

Figure 5.14: Case 2: Control trajectories of link 2

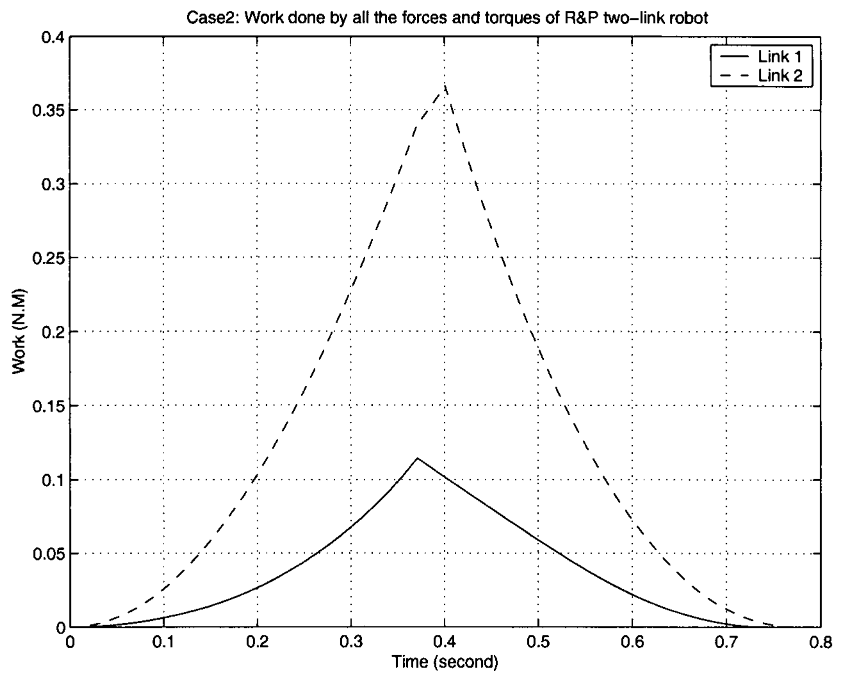

Figure 5.15: Case 2: Work done by forces/torques for each link (zero-net-work solution)

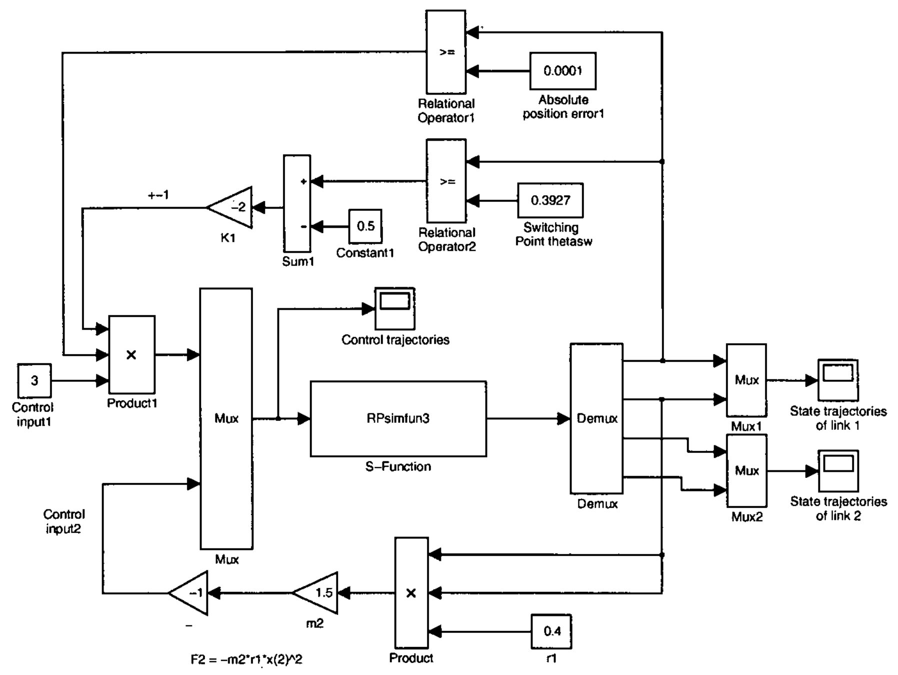

Figure 5.16: Case 3: Simulation block diagram

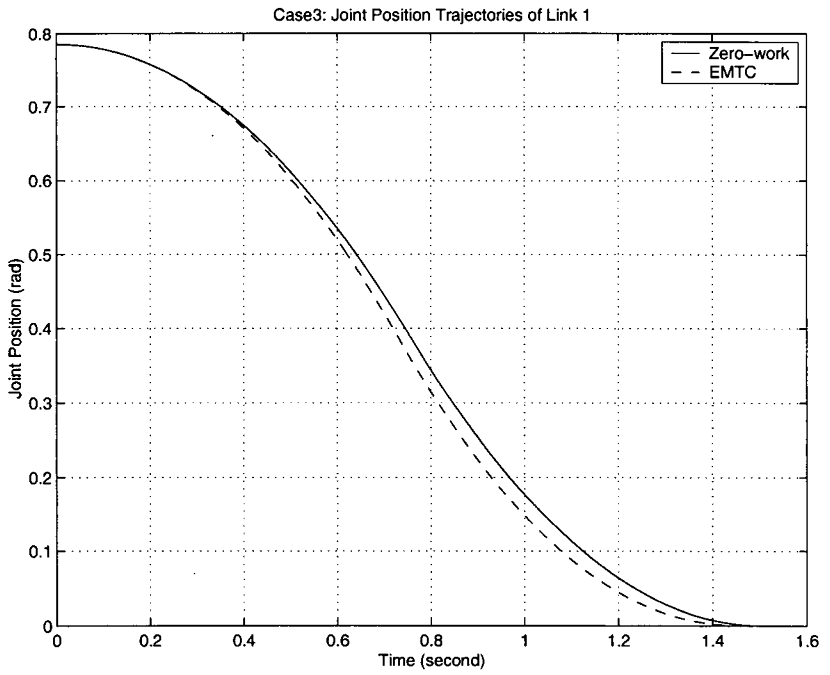

Figure 5.17: Case 3: State trajectories of link 1

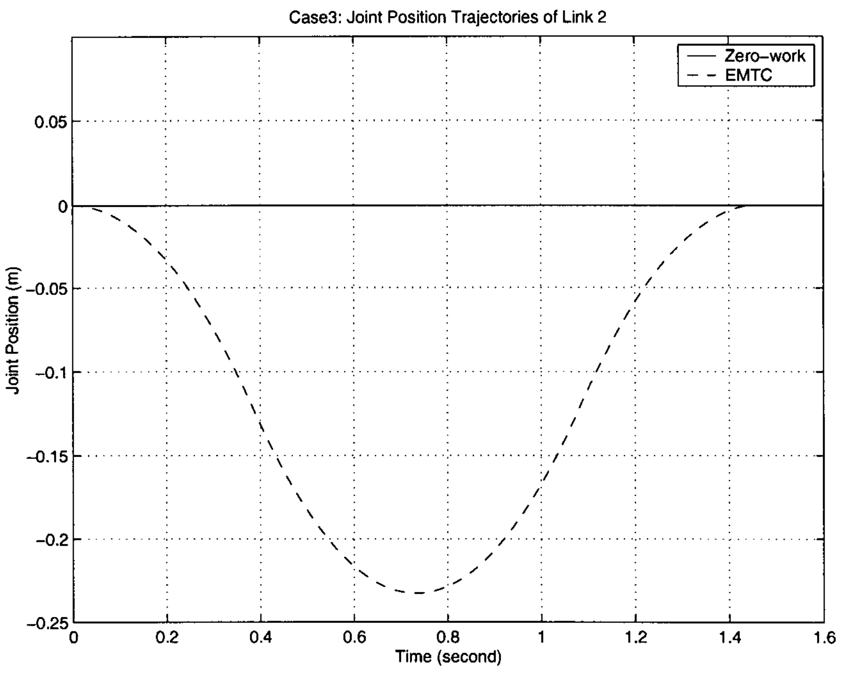

Figure 5.18: Case 3: State trajectories of link 2

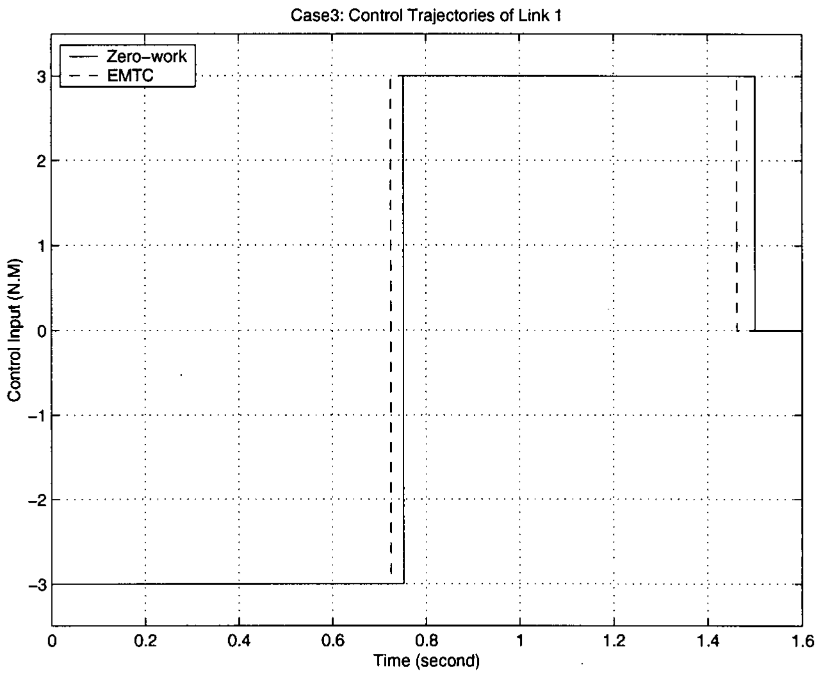

Figure 5.19: Case 3: Control trajectories of link 1

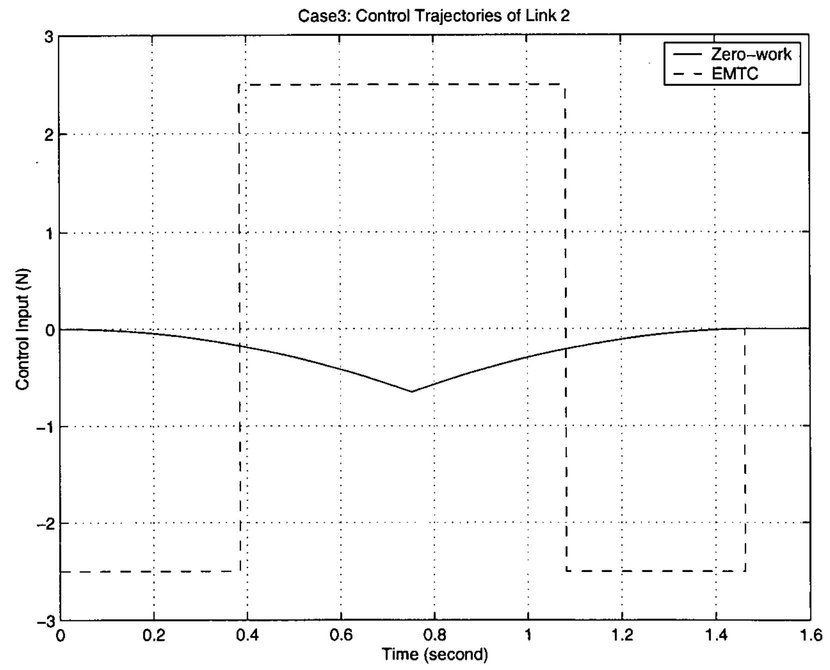

Figure 5.20: Case 3: Control trajectories of link 2

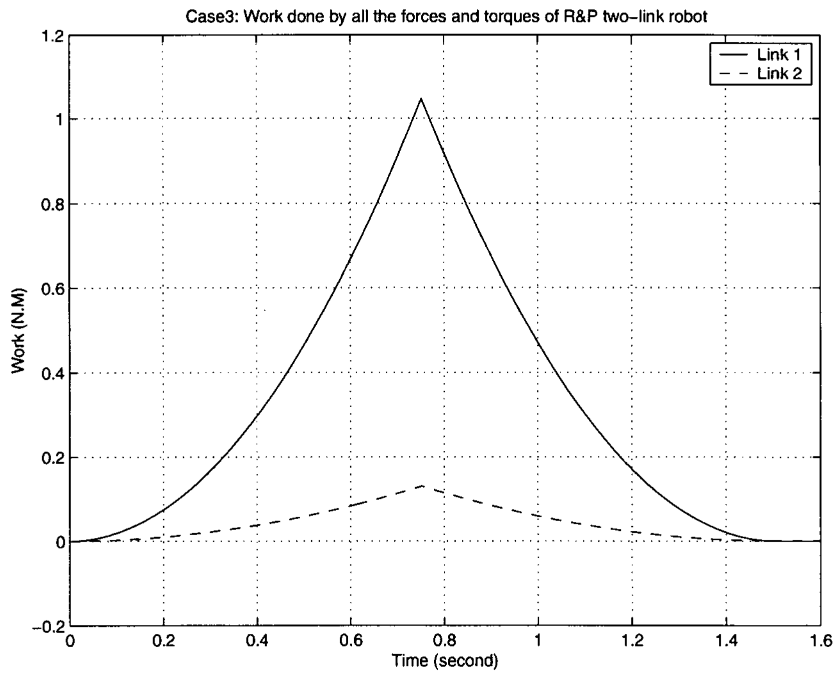

Figure 5.21: Case 3: Work done by forces/torques for each link (zero-net-work solution)

Case 3	θ_0	r_0	θ_{sw}	r_{sw}	t_{1sw}	t_{2sw}	t_{1f}	t_{2f}	error
Zero-net-work	$\pi/4$	0	0.3927		0.7520		1.5005		2.63 %
EMTC	$\pi/4$	0			0.7250	0.3850 1.0816		1.4620	

Table 5.5: Case 3: Simulation results of two-link revolute and prismatic manipulator

5.4.3 Simulation Case 3:

In this case we will move the manipulator from initial state ($\frac{\pi}{4}$, 0, 0, 0) to final state (0, 0, 0, 0), parameter set 2 of the RP manipulator in Table 5.2 will be used for simulation. The simulation results are shown in Table 5.5. For link 1, θ_0 is the initial position, θ_{sw} is the switching position, t_{1sw} is the switching time, t_{1f} is the final time. For link 2, r_0 is the initial position, r_{sw} is the switching position, t_{2sw} is the switching time, t_{2f} is the final time. Control input for link 1 has one switch: $\theta_{1sw} = 0.3927rad$. From offline calculation we find the switching positions, we can then use these values in simulation block diagram shown in Figure 5.16 and use Simulink run the simulation. The exact minimum time is $t_{fexact} = 1.4620sec$, zero-net-work time is $t_{fZW} = 1.5005sec$. The time difference between the proposed zero-net-work control method and EMTC is 2.63%. Figure 5.17 shows the state trajectories of link 1, Figure 5.18 shows the state trajectories of link 2. Figure 5.19 shows the control trajectories of link 1, Figure 5.20 shows the control trajectories of link 2. The work done by external and reaction forces/torques for zero-net-work method are shown in Figure 5.21.

5.4.4 Simulation Case 4:

In this case we will move the manipulator from initial state (0.2, 0, 0.3, 0) to final state (0, 0, 0, 0), parameter set 2 of the RP manipulator in Table 5.2 will be used for simulation. The simulation results are shown in Table 5.6. For link 1, θ_0 is the initial position, θ_{sw} is the switching position, t_{1sw} is the switching time, t_{1f} is the final time. For link 2, r_0 is the initial position, r_{sw} is the switching position, t_{2sw} is the switching time, t_{2f} is the final time. Control

Case 4	θ_0	r_0	θ_{sw}	r_{sw}	t_{1sw}	t_{2sw}	t_{1f}	t_{2f}	error
Zero-net-work	0.2	0.3	0.1050	0.1441	0.3997	0.4363	0.7915	0.8449	2.04 %
EMTC	0.2	0.3			0.0250 0.4465	0.4350		0.8280	

Table 5.6: Case 4: Simulation results of two-link revolute and prismatic manipulator

input for link 1 has one switch: $\theta_{1sw} = 0.1050rad$. Control input for link 2 has one switch: $r_{sw} = 0.1441m$. From offline calculation we find the switching positions, we can then use these values in simulation block diagram shown in Figure 5.22 and use Simulink run the simulation. The exact minimum time is $t_{f_{exact}} = 0.8280sec$, zero-net-work time is $t_{f_{ZW}} = 0.8449sec$. The time difference between the proposed zero-net-work control method and EMTC is 2.04%. Figure 5.23 shows the state trajectories of link 1, Figure 5.24 shows the state trajectories of link 2. Figure 5.25 shows the control trajectories of link 1, Figure 5.26 shows the control trajectories of link 2. The work done by external and reaction forces/torques for zero-net-work method are shown in Figure 5.27.

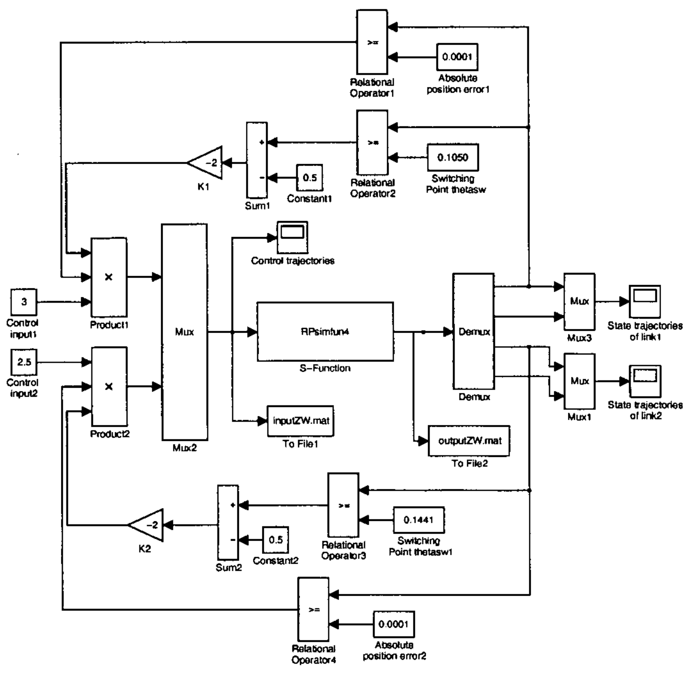

Figure 5.22: Case 4: Simulation block diagram

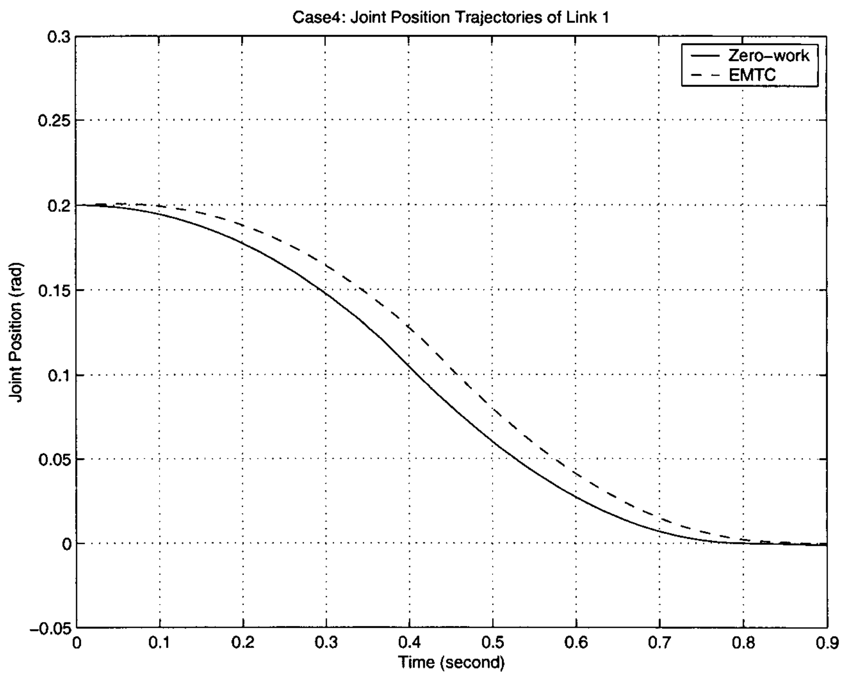

Figure 5.23: Case 4: State trajectories of link 1

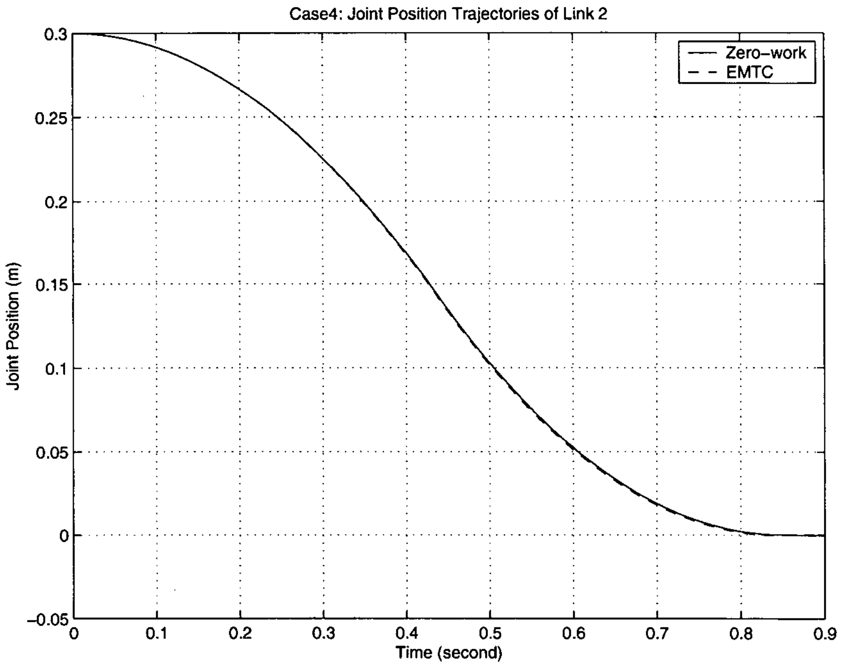

Figure 5.24: Case 4: State trajectories of link 2

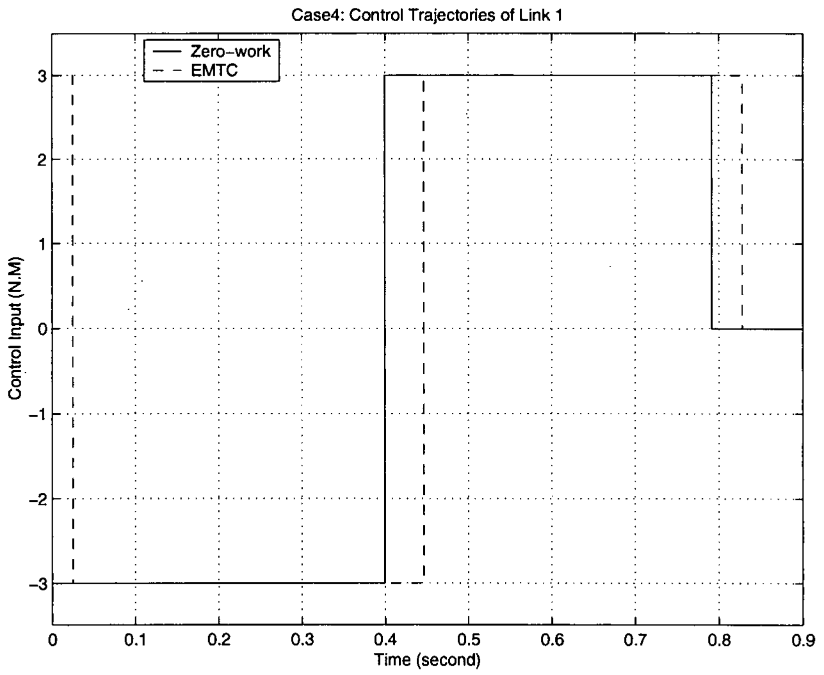

Figure 5.25: Case 4: Control trajectories of link 1

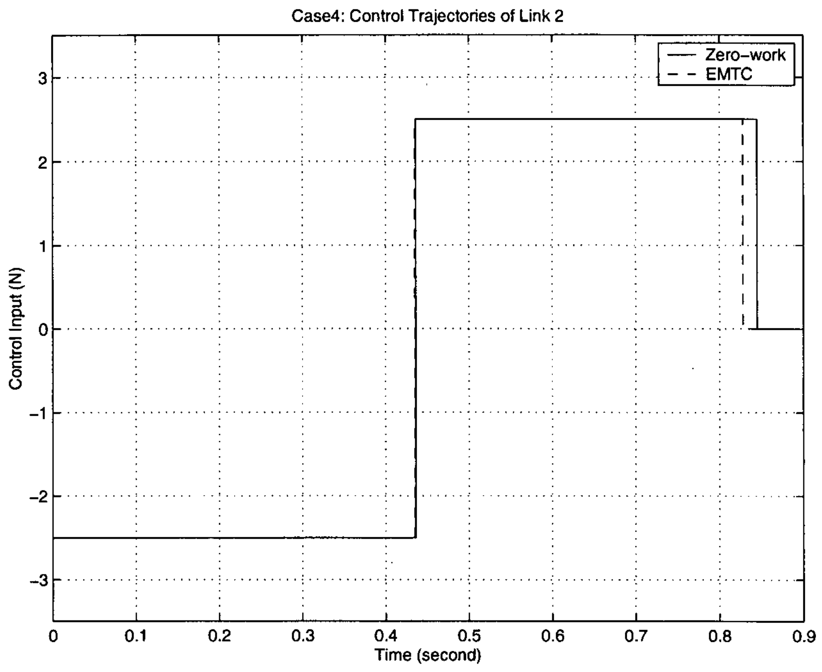

Figure 5.26: Case 4: Control trajectories of link 2

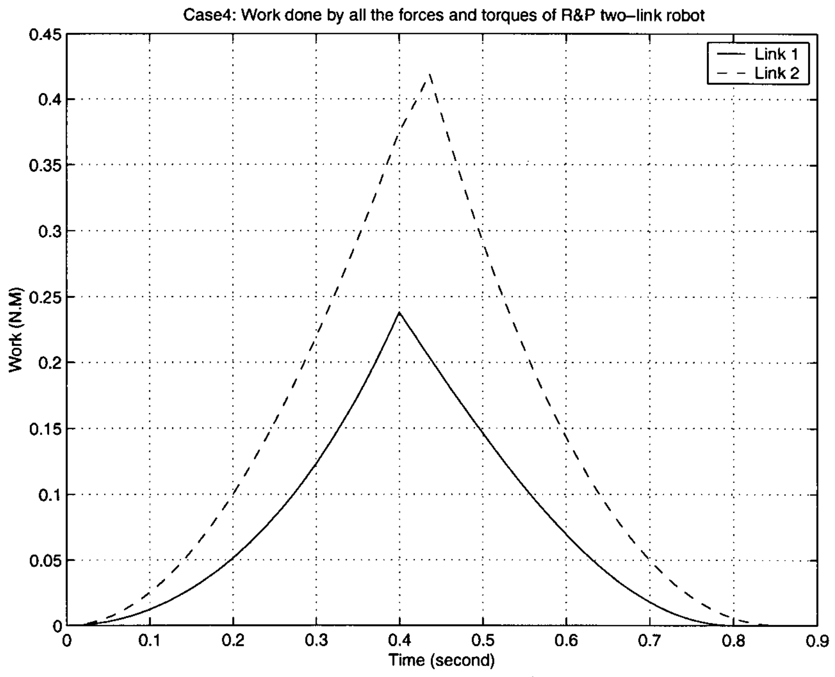

Figure 5.27: Case 4: Work done by forces/torques for each link (zero-net-work solution)

Chapter 6

Development of the Zero-Net-Work Controller for a Revolute Manipulator

6.1 Dynamic Model

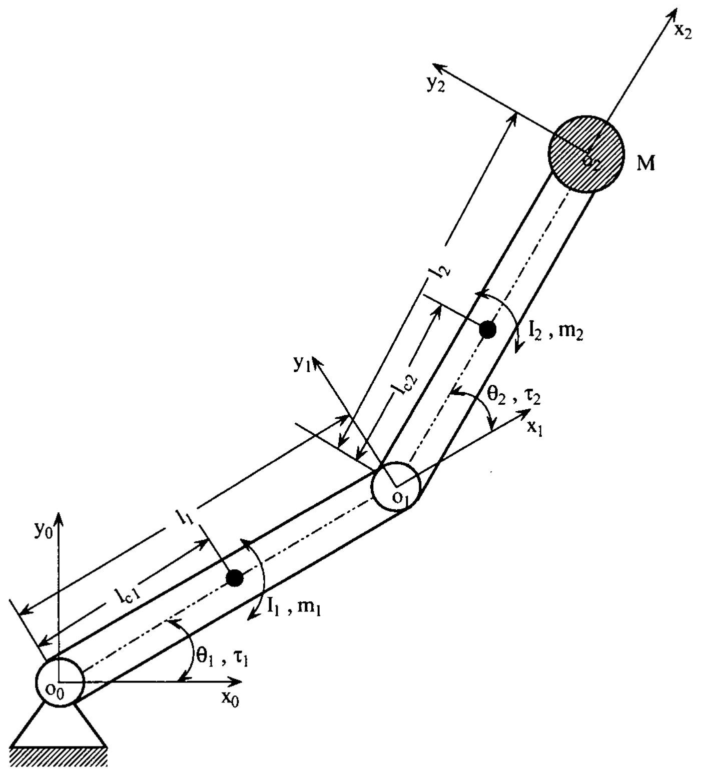

Figure 6.1: Sketch of two-link revolute manipulator

In this chapter we will investigate the control of a Two-Link Revolute Planar Manipulator (TLRPM) as shown in Figure 6.1. The TLRPM consists of two horizontal rigid links, and is controlled by torque inputs τ_1 and τ_2. The first degree of freedom is the angular rotation θ_1 of the inner link (shoulder), and the second degree of freedom is the angular rotation θ_2 of the outer link (elbow). The remaining degrees of freedom used for positioning the end effector are neglected. The absolute value of the control torque for each link is bounded: $|\tau_1| \leq \tau_{1max}$, $|\tau_2| \leq \tau_{2max}$.

Let us give the notations as follow: For $i = 1, 2$, q_i denotes the joint angle, which also serves as a generalized coordinate, m_i is the mass of link i, l_i denotes the length of link i, l_{ci} is the distance from the joint i to the center of mass of link i, I_i is the moment of inertia of link i about the center of mass of link i, and M is the mass of end effector with load. The base frame F_0, frame F_1 and frame F_2 are chosen using D-H convention as shown in Figure 6.1. The D-H parameters are shown in Table 6.1,where $*$ means variable.

Link	a_i	α_i	d_i	θ_i
1	l_1	0	0	q_1^*
2	l_2	0	0	q_2^*

Table 6.1: D-H parameters of two-link revolute manipulator

The homogeneous transformation from frame F_0 to frame F_1 is:

$$\mathbf{H}_{01} = \begin{bmatrix} cosq_1 & -sinq_1 & 0 & l_1cosq_1 \\ sinq_1 & cosq_1 & 0 & l_1sinq_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.1}$$

The homogeneous transformation from F_1 to F_2 is:

$$\mathbf{H}_{12} = \begin{bmatrix} cosq_2 & -sinq_2 & 0 & l_2cosq_2 \\ sinq_2 & cosq_2 & 0 & l_2sinq_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.2}$$

So the homogeneous transformation from F_0 to F_2 is:

$$\mathbf{H}_{02} = \mathbf{H}_{01}\mathbf{H}_{12} = \begin{bmatrix} cos(q_1+q_2) & -sin(q_1+q_2) & 0 & l_1 cosq_1 + l_2 cos(q_1+q_2) \\ sin(q_1+q_2) & cos(q_1+q_2) & 0 & l_1 sinq_1 + l_2 sin(q_1+q_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.3}$$

Next we will use the Newton-Euler method shown in Chapter 3 to obtain the dynamic equations of the TLRPM. We begin with the forward recursion to express the various velocities and accelerations in term of q_1, q_2 and their derivatives. The angular velocity and acceleration of link 1 and link 2 are:

$$\boldsymbol{\omega}_1 = \dot{q}_1\mathbf{k}, \qquad \boldsymbol{\alpha}_1 = \ddot{q}_1\mathbf{k} \tag{6.4}$$

$$\boldsymbol{\omega}_2 = (\dot{q}_1 + \dot{q}_2)\mathbf{k}, \quad \boldsymbol{\alpha}_2 = (\ddot{q}_1 + \ddot{q}_2)\mathbf{k} \tag{6.5}$$

$\{\mathbf{i}, \mathbf{j}, \mathbf{k}\}$ denotes the unit vector along x, y, z axes. The vectors that are independent of the configuration are as follows:

$$\mathbf{r}_{1,c1} = l_{c1}\mathbf{i}, \quad \mathbf{r}_{2,c1} = (l_{c1} - l_1)\mathbf{i}, \quad \mathbf{r}_{1,2} = l_1\mathbf{i}, \tag{6.6}$$

$$\mathbf{r}_{2,c2} = l_{c2}\mathbf{i}, \quad \mathbf{r}_{3,c2} = (l_{c2} - l_2)\mathbf{i}, \quad \mathbf{r}_{2,3} = l_2\mathbf{i}, \tag{6.7}$$

6.1.1 Forward Recursion

Using Equation (3.64) with $i = 1$ and noting that $\mathbf{a}_{e,0} = 0$ gives the acceleration for center of mass of link 1:

$$\begin{aligned} \mathbf{a}_{c,1} &= \dot{\boldsymbol{\omega}}_1 \times \mathbf{r}_{1,c1} + \boldsymbol{\omega}_1 \times (\boldsymbol{\omega}_1 \times \mathbf{r}_{1,c1}) \\ &= \ddot{q}_1\mathbf{k} \times l_{c1}\mathbf{i} + \dot{q}_1\mathbf{k} \times (\dot{q}_1\mathbf{k} \times l_{c1}\mathbf{i}) \\ &= \begin{bmatrix} -l_{c1}\dot{q}_1^2 \\ l_{c1}\ddot{q}_1 \\ 0 \end{bmatrix} \end{aligned} \tag{6.8}$$

The acceleration of the end of link 1 is obtained from above equation by replacing l_{c1} by l_1:

$$\mathbf{a}_{e,1} = \begin{bmatrix} -l_1\dot{q}_1^2 \\ l_1\ddot{q}_1 \\ 0 \end{bmatrix} \tag{6.9}$$

For link 2, using Equation (3.64) with $i = 2$ and substitute $\boldsymbol{\omega}_2$ from Equation 6.5 we have:

$$\begin{aligned} \mathbf{a}_{c,2} &= (\mathbf{R}_1^2)^T \mathbf{a}_{e,1} + \dot{\boldsymbol{\omega}}_2 \times \mathbf{r}_{2,c2} + \boldsymbol{\omega}_2 \times (\boldsymbol{\omega}_2 \times \mathbf{r}_{2,c2}) \\ &= (\mathbf{R}_1^2)^T \mathbf{a}_{e,1} + (\ddot{q}_1 + \ddot{q}_2)\mathbf{k} \times l_{c2}\mathbf{i} + (\dot{q}_1 + \dot{q}_2)\mathbf{k} \times [(\dot{q}_1 + \dot{q}_2)\mathbf{k} \times l_{c2}\mathbf{i}] \end{aligned} \tag{6.10}$$

The first term of the above equation is:

$$\begin{aligned} (\mathbf{R}_1^2)^T \mathbf{a}_{e,1} &= \begin{bmatrix} cosq_2 & sinq_2 & 0 \\ -sinq_2 & cosq_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -l_1\dot{q}_1^2 \\ l_1\ddot{q}_1 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} -l_1\dot{q}_1^2 cosq_2 + l_1\ddot{q}_1 sinq_2 \\ l_1\dot{q}_1^2 sinq_2 + l_1\ddot{q}_1 cosq_2 \\ 0 \end{bmatrix} \end{aligned} \tag{6.11}$$

Substituting Equation (6.11) into Equation (6.10) gives:

$$\mathbf{a}_{c,2} = \begin{bmatrix} -l_1\dot{q}_1^2 cosq_2 + l_1\ddot{q}_1 sinq_2 - l_{c2}(\dot{q}_1 + \dot{q}_2)^2 \\ l_1\dot{q}_1^2 sinq_2 + l_1\ddot{q}_1 cosq_2 + l_{c2}(\ddot{q}_1 + \ddot{q}_2) \\ 0 \end{bmatrix} \tag{6.12}$$

6.1.2 Backward Recursion

Now we carry out the backward recursion to compute the forces and joint torques. Figure 6.2 shows all the forces and torques acting on each link. For the end effector, we assume mass moment of inertia of M about axis z_2 is zero. The force exerted by link 2 on mass M is:

$$\mathbf{f}_3 = M\mathbf{a}_{e,2}$$

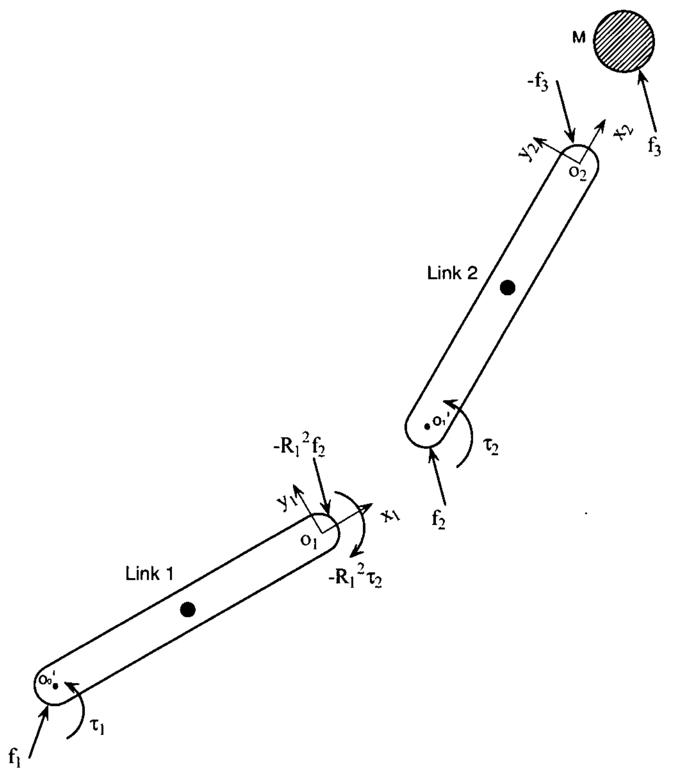

Figure 6.2: Forces and torques act on each link

$$= M \begin{bmatrix} -l_1\dot{q}_1^2 cosq_2 + l_1\ddot{q}_1 sinq_2 - l_2(\dot{q}_1+\dot{q}_2)^2 \\ l_1\dot{q}_1^2 sinq_2 + l_1\ddot{q}_1 cosq_2 + l_2(\ddot{q}_1+\ddot{q}_2) \\ 0 \end{bmatrix} \tag{6.13}$$

The force exerted by link 1 on link 2 is:

$$\begin{aligned} \mathbf{f}_2 &= m_2\mathbf{a}_{c,2} + \mathbf{f}_3 \\ &= m_2\mathbf{a}_{c,2} + M\mathbf{a}_{e,2} \\ &= \begin{bmatrix} (m_2+M)(-l_1\dot{q}_1^2 cosq_2 + l_1\ddot{q}_1 sinq_2) - (m_2 l_{c2} + M l_2)(\dot{q}_1+\dot{q}_2)^2 \\ (m_2+M)(l_1\dot{q}_1^2 sinq_2 + l_1\ddot{q}_1 cosq_2) + (m_2 l_{c2} + M l_2)(\ddot{q}_1+\ddot{q}_2) \\ 0 \end{bmatrix} \end{aligned} \tag{6.14}$$

The torque exerted by link 1 on link 2 is:

$$\boldsymbol{\tau}_2 = I_2\boldsymbol{\alpha}_2 + \boldsymbol{\omega}_2 \times (I_2\boldsymbol{\omega}_2) - \mathbf{f}_2 \times \mathbf{r}_{2,c2} + \mathbf{f}_3 \times \mathbf{r}_{3,c2}$$

$$\begin{aligned}
&= I_2\boldsymbol{\alpha}_2 + \boldsymbol{\omega}_2 \times (I_2\boldsymbol{\omega}_2) - \mathbf{f}_2 \times l_{c2}\mathbf{i} + \mathbf{f}_3 \times (l_{c2} - l_2)\mathbf{i} \\
&= I_2\boldsymbol{\alpha}_2 - m_2\mathbf{a}_{c,2} \times l_{c2}\mathbf{i} - M\mathbf{a}_{e,2} \times l_2\mathbf{i} \\
&= I_2(\ddot{q}_1 + \ddot{q}_2)\mathbf{k} \\
&\quad + m_2 l_{c2}[l_1\dot{q}_1^2 sinq_2 + l_1\ddot{q}_1 cosq_2 + l_{c2}(\ddot{q}_1 + \ddot{q}_2)]\mathbf{k} \\
&\quad + Ml_2[l_1\dot{q}_1^2 sinq_2 + l_1\ddot{q}_1 cosq_2 + l_2(\ddot{q}_1 + \ddot{q}_2)]\mathbf{k}
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{\tau}_2 &= (I_2 + m_2 l_{c2}^2 + Ml_2^2 + m_2 l_{c2} l_1 cosq_2 + Ml_1 l_2 cosq_2)\ddot{q}_1\mathbf{k} \\
&\quad + (I_2 + m_2 l_{c2}^2 + Ml_2^2)\ddot{q}_2\mathbf{k} \\
&\quad + [(m_2 l_{c2} l_1 + Ml_1 l_2) sinq_2]\dot{q}_1^2\mathbf{k}
\end{aligned} \tag{6.15}$$

Backward recursion for link 1, with $i = 1$, the force and torque exerted by link 2 on link 1 are:

$$-\mathbf{R}_1^2\boldsymbol{\tau}_2 = -\tau_2\mathbf{k} \tag{6.16}$$

$$-\mathbf{R}_1^2\mathbf{f}_2 = -\begin{bmatrix} cosq_2 & -sinq_2 & 0 \\ sinq_2 & cosq_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} (m_2 + M)(-l_1\dot{q}_1^2 cosq_2 + l_1\ddot{q}_1 sinq_2) - (m_2 l_{c2} + Ml_2)(\dot{q}_1 + \dot{q}_2)^2 \\ (m_2 + M)(l_1\dot{q}_1^2 sinq_2 + l_1\ddot{q}_1 cosq_2) + (m_2 l_{c2} + Ml_2)(\ddot{q}_1 + \ddot{q}_2) \\ 0 \end{bmatrix}$$

$$= -\begin{bmatrix} -(m_2 + M)l_1\dot{q}_1^2 - (m_2 l_{c2} + Ml_2)[(\dot{q}_1 + \dot{q}_2)^2)cosq_2 + (\ddot{q}_1 + \ddot{q}_2)^2 sinq_2] \\ (m_2 + M)l_1\ddot{q}_1 + (m_2 l_{c2} + Ml_2)[(\ddot{q}_1 + \ddot{q}_2)cosq_2 - (\dot{q}_1 + \dot{q}_2)^2 sinq_2] \\ 0 \end{bmatrix} \tag{6.17}$$

The force equation for link 1 is:

$$\mathbf{f}_1 = m_1\mathbf{a}_{c,1} + \mathbf{R}_1^2\mathbf{f}_2 \tag{6.18}$$

The torque equation for link 1 is:

$$\boldsymbol{\tau}_1 = I_1\boldsymbol{\alpha}_1 + \boldsymbol{\omega}_1 \times (I_1\boldsymbol{\omega}_1) + \mathbf{R}_1^2\boldsymbol{\tau}_2 - \mathbf{f}_1 \times l_{c1}\mathbf{i} - (\mathbf{R}_1^2\mathbf{f}_2) \times (l_1 - l_{c1})\mathbf{i} \tag{6.19}$$

Substitute $\boldsymbol{\tau}_1$, $\mathbf{f}_1$ and $\mathbf{f}_2$ into above equation we have:

$$\begin{aligned}
\boldsymbol{\tau}_1 &= I_1\boldsymbol{\alpha}_1 + \boldsymbol{\tau}_2 - (m_1\mathbf{a}_{c1} + \mathbf{R}_1^2\mathbf{f}_2) \times l_{c1}\mathbf{i} - (\mathbf{R}_1^2\mathbf{f}_2) \times (l_1 - l_{c1})\mathbf{i} \\
&= I_1\ddot{q}\mathbf{k} + \boldsymbol{\tau}_2 - m_1\mathbf{a}_{c1} \times l_{c1}\mathbf{i} - (\mathbf{R}_1^2\mathbf{f}_2) \times l_1\mathbf{i} \\
&= [I_1 + m_1l_{c1}^2 + (m_2 + M)l_1^2 + I_2 + m_2l_{c2}^2 + Ml_2^2 + 2(m_2l_{c2}l_1 + Ml_1l_2)cosq_2]\ddot{q}_1\mathbf{k} \\
&\quad +(I_2 + m_2l_{c2}^2 + Ml_2^2 + (m_2l_{c2}l_1 + Ml_1l_2)cosq_2)\ddot{q}_2\mathbf{k} \\
&\quad -[(m_2l_{c2}l_1 + Ml_1l_2)sinq_2](2\dot{q}_1\dot{q}_2 + \dot{q}_2^2)\mathbf{k}
\end{aligned} \tag{6.20}$$

From Equation (6.15) and Equation (6.20), we obtain the closed form dynamic equations of the system:

$$\tau_1 = (c_1 + c_2 + 2c_3cosq_2)\ddot{q}_1 + (c_2 + c_3cosq_2)\ddot{q}_2 - c_3sinq_2(2\dot{q}_1\dot{q}_2 + \dot{q}_2^2) \tag{6.21}$$

$$\tau_2 = (c_2 + c_3cosq_2)\ddot{q}_1 + c_2\ddot{q}_2 + c_3sinq_2\dot{q}_1^2 \tag{6.22}$$

where

$$\begin{aligned}
c_1 &= I_1 + m_1l_{c1}^2 + (m_2 + M)l_1^2 \\
c_2 &= I_2 + m_2l_{c2}^2 + Ml_2^2 \\
c_3 &= m_2l_1l_{c2} + Ml_1l_2
\end{aligned} \tag{6.23}$$

The dynamic equations in state space form are:

$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= \frac{c_2[u_1 - u_2 + c_3(x_2 + x_4)^2sinx_3] - c_3(u_2 - c_3x_2^2sinx_3)cosx_3}{c_1c_2 - c_3^2(cosx_3)^2} \\
\dot{x}_3 &= x_4 \\
\dot{x}_4 &= \frac{(c_1 + c_3cosx_3)(u_2 - c_3x_2^2sinx_3) - (c_2 + c_3cosx_3)[u_1 - u_2 + c_3(x_2 + x_4)^2sinx_3]}{c_1c_2 - c_3^2(cosx_3)^2}
\end{aligned} \tag{6.24}$$

where

$$x_1 = q_1(t) \qquad x_3 = q_2(t) \tag{6.25}$$

$$x_2 = \dot{q}_1(t) \qquad x_4 = \dot{q}_2(t) \tag{6.26}$$

$$u_1 = \tau_1(t) \qquad u_2 = \tau_2(t) \tag{6.27}$$

6.2 The Exact Minimum Time Solution

The initial conditions of the Two-Link Revolute Planar Manipulator (TLRPM) are:

$$x_1(0) = x_{10} \qquad x_2(0) = 0 \tag{6.28}$$

$$x_3(0) = x_{20} \qquad x_4(0) = 0 \tag{6.29}$$

The final conditions are:

$$x_1(t_f) = x_2(t_f) = x_3(t_f) = x_4(t_f) = 0 \tag{6.30}$$

where t_f is the final time.

The absolute values of the control inputs are bounded:

$$|\tau_1| \leq \tau_{1max} \tag{6.31}$$

$$|\tau_2| \leq \tau_{2max} \tag{6.32}$$

The cost function is:

$$J = \int_{t_0}^{t_f} 1dt \tag{6.33}$$

The Exact Minimum Time Control (EMTC) problem of the TLPRM can be stated as follows:

For the robot governed by the differential Equation (6.24) find an admissible control τ_1 and τ_2 satisfying the constraints in Equation (6.31) and Equation (6.32) which transfer the robot from initial position to the fixed final position in minimum time, i.e., such that the cost functional J defined in Equation (6.33) is minimized.

Referring to Equations (6.24), we define

$$\begin{aligned} V_1 &= \frac{c_2[u_1 - u_2 + c_3(x_2 + x_4)^2 sinx_3] - c_3(u_2 - c_3x_2^2 sinx_3)cosx_3}{c_1c_2 - c_3^2(cosx_3)^2} \\ V_2 &= \frac{(c_1 + c_3cosx_3)(u_2 - c_3x_2^2 sinx_3) - (c_2 + c_3cosx_3)[u_1 - u_2 + c_3(x_2 + x_4)^2 sinx_3]}{c_1c_2 - c_3^2(cosx_3)^2} \end{aligned}$$

The Hamiltonian is then

$$H(\mathbf{x}, u_1, u_2, \lambda) = 1 + \lambda_1 x_2 + \lambda_2 V_1 + \lambda_3 x_4 + \lambda_4 V_2 \tag{6.34}$$

where λ_1, λ_2 , λ_3, λ_4 are the costate variables. The necessary conditions for the EMTC can be written as:

State equation:

$$\dot{\mathbf{x}} = \frac{\partial H}{\partial \lambda} = f(\mathbf{x}, u_1, u_2, t) \tag{6.35}$$

Costate equation:

$$\frac{\partial H}{\partial \mathbf{x}} = -\dot{\lambda} \tag{6.36}$$

Stationary Condition:

$$H(x^*, \lambda^*, u_1^*, u_2^*) \leq H(x^*, \lambda^*, u_1, u_2) \tag{6.37}$$

where u_1^* and u_2^* are the optimal controls, $\mathbf{x}^*$ and λ^* are the corresponding optimal state and costate trajectories.

By using the time normalization technique shown in Chapter 4, we can transform EMTC problem into a fixed time interval problem. The equivalent Two Point Boundary Problem (TPBVP) is:

$$\frac{dx_1}{dT} = zx_2 \tag{6.38}$$

$$\frac{dx_2}{dT} = zV_1 \tag{6.39}$$

$$\frac{dx_3}{dT} = zx_4 \tag{6.40}$$

$$\frac{dx_4}{dT} = zV_2 \tag{6.41}$$

$$\frac{d\lambda_1}{dT} = 0 \tag{6.42}$$

$$\frac{d\lambda_2}{dT} = -z[\lambda_1 + \lambda_2\frac{\partial V_1}{\partial x_2} + \lambda_4\frac{\partial V_2}{\partial x_2}] \tag{6.43}$$

$$\frac{d\lambda_3}{dT} = -z[\lambda_2\frac{\partial V_1}{\partial x_3} + \lambda_4\frac{\partial V_2}{\partial x_3}] \tag{6.44}$$

$$\frac{d\lambda_4}{dT} = -z[\lambda_3 + \lambda_2\frac{\partial V_1}{\partial x_4} + \lambda_4\frac{\partial V_2}{\partial x_4}] \tag{6.45}$$

$$\frac{dz}{dT} = 0 \tag{6.46}$$

where $T \in [0,1]$ for $t \in [0, t_f]$, and

$$z(T) = t_f \tag{6.47}$$
$$T = \frac{t}{t_f} \tag{6.48}$$

The boundary conditions are:

$$\begin{array}{rl} x_1(0) = x_{10} & x_2(0) = 0 \\ x_3(0) = x_{20} & x_4(0) = 0 \\ x_1(1) = 0 & x_2(1) = 0 \\ x_3(1) = 0 & x_4(1) = 0 \end{array} \tag{6.49}$$

In order to get the exact minimum time solution, we have to solve the TPBVP. We can use subroutine MUSN to solve above TPBVP. For stability of numerical procedures such as shooting method used by subroutine MUSN, it is required that small changes to the initial values should result in small changes in the solution. Such problem are called well conditioned. Any problem which does not has this property is called ill conditioned. Such problems are numerically unstable, and very sensitive to the initial guess of the unknown initial conditions. Unfortunately the TPBVP for the TLPRM belongs to this class of problem. The convergence of the solution depend on the initial guess of the unknown parameters. In this case we have to give the initial guesses for 5 unknown parameters, that is λ_1, λ_2, λ_3, λ_4 and the final time z. However, it is extremely difficult to have a good guess of the costates intuitively, because the costates do not have any simple interpretation or physical means. It has been shown that for the TLPRM, the initial guess for some of the costates must be within a very narrow range to obtain convergence of the shooting method (Fotouhi, 1996). For example, for a $0.65m$ ($l_1 + l_2 = 0.65m$) long manipulator considered in Geering et al. 1986, for the maneuvers within the range $0.6 < D/(l_1 + l_2) < 2.0$ (D is the geometrical distance between the starting and the final distance between the manipulator tip), the corresponding values of the initial costates were within the limits $-0.31 < \lambda_2(0) < -0.21$ and $-0.088 < \lambda_4(0) < -0.074$. Therefore, if the guessed values of the costates in the shooting method are outside of the narrow limits the target is calculated with large errors causing the method to diverge. For above reasons, it is very hard to get the exact minimum time solutions using the traditional calculus of variation method.

6.3 Zero-Net-Work Solution

Next we will design the zero-net-work controller for the two-link revolute planar manipulator. Figure 6.3 is the block diagram of the overall zero-net-work control scheme. The overall zero-net-work control procedure consists of two step: offline calculation of the bang-bang switching position for each link and online application of the pre-calculated switching positions to control the robot manipulator.

6.3.1 Offline Calculation

We assume the initial and final velocity for each link of the manipulator are zero, also the final position of each link is zero. These are the boundary conditions. When we design the zero-net-work controller, we assume that control input for each link of the manipulator is bang-bang, and switches only once from the minimum to the maximum control constraints. The offline zero-net-work algorithm is to search the bang-bang switching position for each link that satisfies all the boundary conditions, control input limit constraints and work energy constraints. Because we assume the control input is bang-bang when we search the switching position, so the control input will always within the control constraints. This means that the control constraints will always be satisfied. We set the initial switching position for each link equals to the final position, this will guarantee that each link will reach the final position. So the position boundary conditions also are always satisfied. The only constraints we need to search is the work energy constraints. The algorithm will search the switching positions that the work done by all the forces/torques for each link is zero, this will be the switching position used online to control the manipulator.

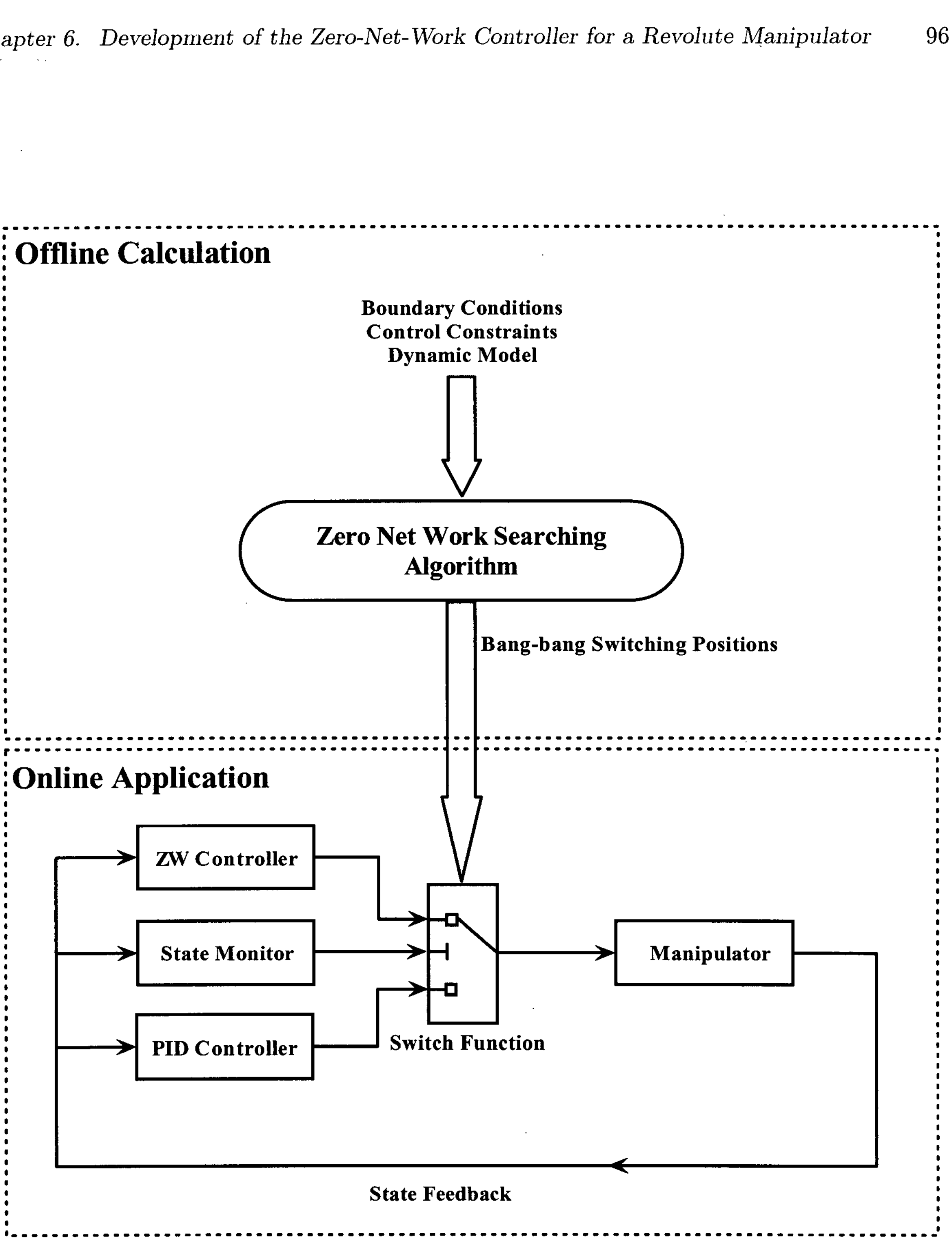

Figure 6.3: Block diagram of overall zero-net-work control scheme

We can use the results from Newton-Euler formulation of the dynamic equations in Section 6.1 to calculate the work done by all the forces/torques for each link. All the work will be calculated in base frame F_0.

For link 1 the work done by all the forces/torques can be calculated as follow: If we move all the forces to point $o_0^{'}$ (see Figure 6.2). The work done by all the forces that passing $o_0^{'}$ is always zero. So we have for link 1 in frame F_0:

$$Work_1 = [(\tau_1 - \tau_2) - (\mathbf{R}_1^2\mathbf{f}_2)_{y_1} l_1]\delta q_1 \tag{6.50}$$

For link 2 the work done by all the forces/torques in frame F_0:

$$Work_2 = (\tau_2 - (\mathbf{f}_3)_{y_2} l_2)(\delta q_1 + \delta q_2) + [(\mathbf{f}_2)_x - (\mathbf{f}_3)_x]\delta x + [(\mathbf{f}_2)_y - (\mathbf{f}_3)_y]\delta y \tag{6.51}$$

where $\delta x = \delta(l_1 cos q_1)$ and $\delta y = \delta(l_1 sin q_1)$

Since the initial control inputs for each link are alway bang-bang and switch once. For different physical parameters and motion configurations, we can not always find the solution that both links reach the final position at the same time. Following assumption must be made: once the link reaches the final position, the control input will be changed to the torque needed to hold the link at the final position. For the two link revolute manipulator there is only two possibilities:

Case 1: Link 1 reaches the final position first. This means:

$$q_1 = 0;\ \dot{q}_1 = 0;\ \ddot{q}_1 = 0 \tag{6.52}$$

In order to hold link 1 at the final position, from Equation (6.21 - 6.22) the control input for joint 1 is:

$$\tau_1 = \frac{(c_2 + c_3 cos q_2)\tau_2}{c_2} - c_3 sin q_2 \dot{q}_2^2 \tag{6.53}$$

Case 2: Link 2 reaches the final position first. This means:

$$q_2 = 0;\ \dot{q}_2 = 0;\ \ddot{q}_2 = 0 \tag{6.54}$$

In order to hold link 2 at the final position, from Equation (6.21 - 6.22) the control input for joint 2 is:

$$\tau_2 = \frac{(c_2 + c_3)\tau_1}{c_1 + c_2 + 2c_3} \tag{6.55}$$

Figure 6.4 shows the flowchart of the overall offline searching method. At the beginning, we set the switching positions to zero. Next we fix the switching position of link 1, and search the switching position of link 2. If no solution is find, then increase the switching position of link 1 by one step. If no solution is find after we search the whole range of link 1 and link 2, this means the searching step is too large. We need to reduce the searching step. At any time, when one of the link reach the final position, we need to change to control input from maximum torque to holding torque based on Equation (6.53) and (6.55). The FORTRAN program which is used to find the zero-net-work switching positions is shown in Appendix A.7.

6.3.2 Online Application

From the offline searching, we find the switching positions. Now we can use them online to control the manipulator. We use Simulink to simulate the online application of the zero-net-work controller. Figure 6.5 shows the simulation block diagram. Block 'ZW Controller' is Simulink S-function that generates the controller inputs. It compares state feedback with the switching positions and then decides the control inputs. The detail program is listed in Appendix A.10. Block 'PD Controller' is also a S-function. It's a PD controller that is used to bring the manipulator to the final position in case of large model error or large disturbance. Block 'Switch Function' is used to make a decision on which controller to use. If the zero-net-work controller can bring the manipulator to the final position accurately, the PD controller will not be turn on. If the position of the manipulator is away from the final position, the PD will be turned on to make sure the manipulator will reach the final position. Detail program of 'PD Controller' is listed in Appendix A.11. Block 'TLRPMsimfun' is a S-function that is used to model the two link planar revolute manipulator, see Appendix A.9 for detail.

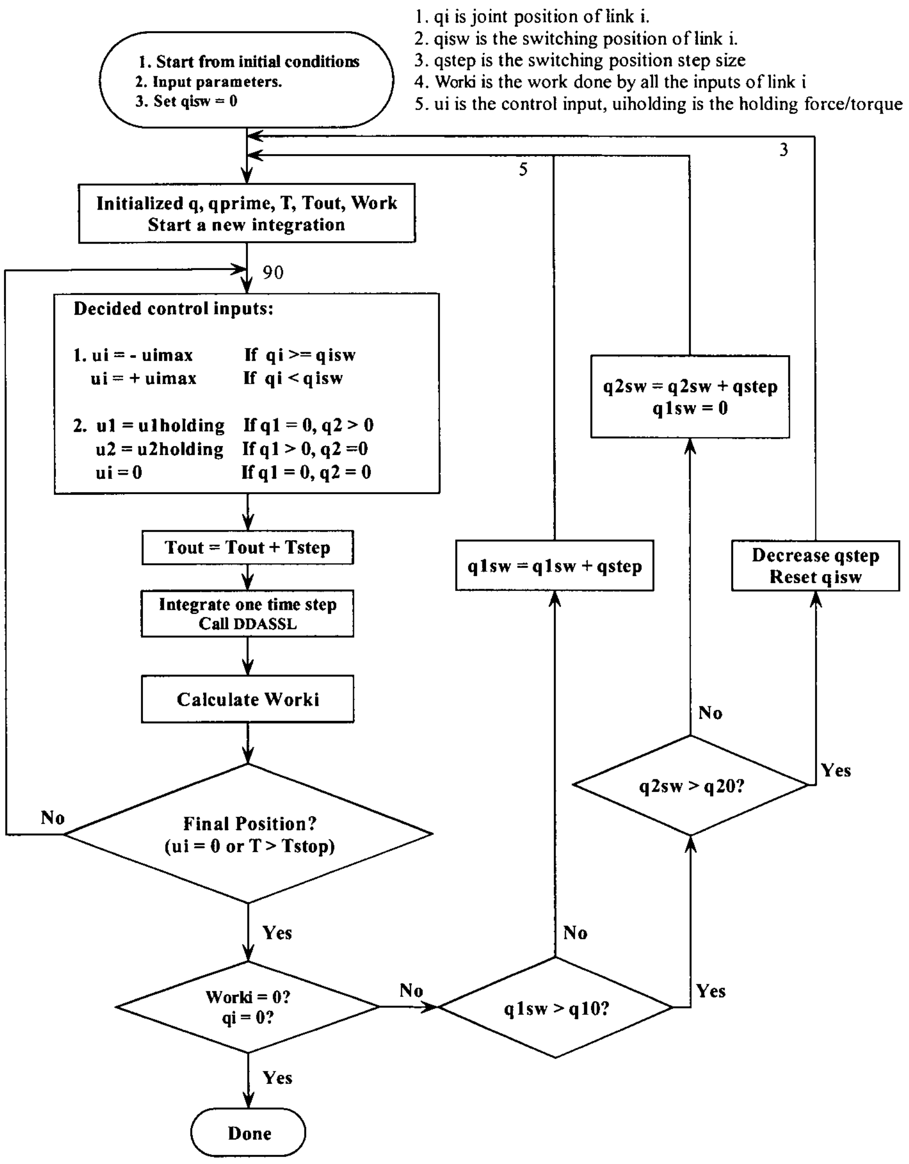
1. qi is joint position of link i.
2. qisw is the switching position of link i.
3. qstep is the switching position step size
4. Worki is the work done by all the inputs of link i
5. ui is the control input, uiholding is the holding force/torque
1. Start from initial conditions
2. Input parameters.
3. Set qisw = 0
Initialized q, qprime, T, Tout, Work
Start a new integration
90
Decided control inputs:
1. ui = - uimax If qi >= qisw
ui = + uimax If qi < qisw
2. u1 = u1holding If q1 = 0, q2 > 0
u2 = u2holding If q1 > 0, q2 = 0
ui = 0 If q1 = 0, q2 = 0
Tout = Tout + Tstep
Integrate one time step
Call DDASSL
Calculate Worki
Final Position?
(ui = 0 or T > Tstop)
No
Yes
Worki = 0?
qi = 0?
Yes
Done
No
q1sw > q10?
No
Yes
q1sw = q1sw + qstep
5
q2sw > q20?
No
Yes
q2sw = q2sw + qstep
q1sw = 0
Decrease qstep
Reset qisw
3

Figure 6.4: Flowchart of offline calculation

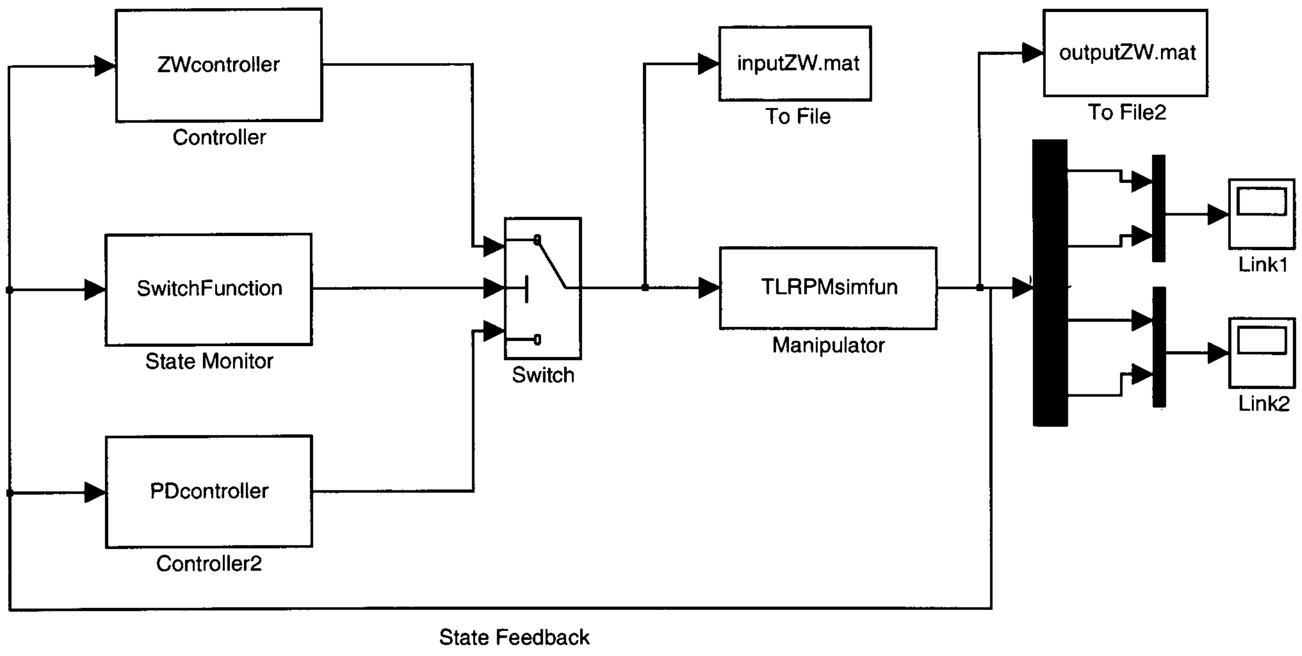

Figure 6.5: Simulation block diagram of two-link revolute manipulator

If there is model error or large disturbance, PD controller will be used to guarantee asymptotic tracking. Because there is no gravitational term in the dynamic equations of the manipulator, we can prove that PD control can achieve zero steady state error. In case there are gravitational terms present in the dynamic equations, there will be a steady state error, PID controller will be used.

To show PD control alone can achieve zero steady state error for the Two-Link Revolute Planar Manipulator (TLRPM). Consider the dynamic equations in matrix form:

$$\mathbf{D}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \mathbf{u} \tag{6.56}$$

PD control scheme in vector form is:

$$\mathbf{u} = K_p\hat{\mathbf{q}} - K_d\dot{\mathbf{q}} \tag{6.57}$$

$\hat{\mathbf{q}} = \mathbf{q}^d - \mathbf{q}$ is the difference between the desired joint position $\mathbf{q}^d$ and the actual joint position $\mathbf{q}$. K_p, K_d are proportional and derivative gains, respectively. Consider Lyapunov function candidate:

$$V = \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{D}(\mathbf{q})\dot{\mathbf{q}} + \frac{1}{2}\hat{\mathbf{q}}^T K_p\hat{\mathbf{q}} \tag{6.58}$$

The derivative of V is:

$$\dot{V} = \dot{\mathbf{q}}^T\mathbf{D}(\mathbf{q})\ddot{\mathbf{q}} + \frac{1}{2}\dot{\mathbf{q}}^T\dot{\mathbf{D}}(\mathbf{q})\dot{\mathbf{q}} - \dot{\mathbf{q}}^T K_p\hat{\mathbf{q}} \tag{6.59}$$

From Equation (6.56), we have:

$$\mathbf{D}(\mathbf{q})\ddot{\mathbf{q}} = \mathbf{u} - \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} \tag{6.60}$$

Substitute above equation into Equation (6.59):

$$\dot{V} = \dot{\mathbf{q}}^T(\mathbf{u} - K_p\hat{\mathbf{q}}) + \frac{1}{2}\dot{\mathbf{q}}^T(\dot{\mathbf{D}}(\mathbf{q}) - 2\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}))\dot{\mathbf{q}} \tag{6.61}$$

Using the fact that $\dot{\mathbf{D}} - \mathbf{2C}$ is skew symmetric and substitute the PD control law for $\mathbf{u}$ into above equation yields:

$$\dot{V} = -\dot{\mathbf{q}}^T K_d\dot{\mathbf{q}} \leq 0 \tag{6.62}$$

LaSalle's Theorem: Suppose a Lyapunov function candidate V is found such that, along solution trajectories $\dot{V} \leq 0$, system is asymptotically stable if the only solution of the system satisfying $\dot{V} = 0$ is the null solution.

From Equation (6.62), if $\dot{V} = 0$, we have $\dot{\mathbf{q}} = 0$ and hence $\ddot{\mathbf{q}} = 0$. From the equation of motion with PD control:

$$\mathbf{D}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} = K_p\hat{\mathbf{q}} - K_d\dot{\mathbf{q}} \tag{6.63}$$

We must then have $K_p\hat{\mathbf{q}} = 0$, which implies that $\hat{\mathbf{q}} = 0$, $\dot{\mathbf{q}} = 0$. Lasalle's Theorem then implies that the system is asymptotically stable.

6.4 Computer Simulation

The proposed zero-net-work method will now be simulated for two-link revolute manipulator, and its performance will be compared with the exact optimal solution. We will use the zero-net-work method to three different parameter sets of two-link revolute manipulator. For each parameter set, we will perform simulations for different motion configurations. Simulink will be used to perform the simulation. The system's S-function can be found in Appendix A.9. The simulation results for different parameters and motion configurations are presented in the following:

6.4.1 Simulation Case 1:

Case 1	q_{10}	q_{20}	q_{1sw}	q_{2sw}	t_{1sw}	t_{2sw}	t_{1f}	t_{2f}	error
Zero-net-work	0.975	0	0.4880		0.5746		1.1440		5.59 %
Exact	0.975	0			0.5417	0.0875 0.5872		1.0834	
PD	0.975	0					3.8865	3.8865	258.73 %

Table 6.2: Case 1: Simulation results of two-link revolute manipulator

Physical parameters for the IBM 7535B 04 robot are used for simulation in this case:

$$\begin{array}{ll} l_1 = 2l_{c1} = 0.4m & l_2 = 2l_{c2} = 0.25m \\ m_1 = 29.58kg & m_2 = 15.00kg \quad M = 6.0kg \\ I_1 = 0.416739kgm^2 & I_2 = 0.205625kgm^2 \\ u_{1max} = 25NM & u_{2max} = 9NM \end{array} \tag{6.64}$$

This example is a rest to rest motion of the manipulator from straight to straight configurations. The initial and the final conditions of the states in $[rad]$ and $[rad/s]$ are:

$$\mathbf{x}(0) = [0.975 \;\; 0.0 \;\; 0.0 \;\; 0.0]^T \tag{6.65}$$

$$\mathbf{x}(t_f) = [0 \;\; 0.0 \;\; 0.0 \;\; 0.0]^T \tag{6.66}$$

The FORTRAN program using zero-net-work method to find the switching positions for each link is shown in Appendix A.7 . The simulation results are shown in Table 6.2. For $i = 1, 2$, q_{i0} is the initial position of link i, q_{isw} is the switching position of link i, t_{isw} is the switching time of link i, and t_{if} is the final time of link i. In this case, the initial velocity and position of link

2 are zero. So there is no minimum to maximum switch for link 2. The control input for link 2 start from the beginning is the torque needed to hold link 2 at the final position while link 1 is moving. Control input for link 1 has one switch: $q_{1sw} = 0.4880rad$. From offline calculation we find the switching positions, we can then use these values in simulation block diagram shown in Figure (6.5) and use Simulink run the simulation. Exact minimum time solution can be obtained by solving the Two Point Boundary Problem (TPBVP). The exact minimum time is $t_{fexact} = 1.0834sec$, zero-net-work time is $t_{fZW} = 1.1440sec$. The time difference between the proposed zero-net-work control method and Exact Minimum Time Control (EMTC) is 5.59%. Thus we can say zero-net-work control method leads to a near minimum time solution. The simulation results of the fastest response using PD control method with input bounds are also shown. In the PD control method, the control gains are tuned by trial and error, so we can get the fastest response while satisfies the control and state constraints. The best PD control gains are: $K_{P1} = 25.6$, $K_{D1} = 27$, $K_{P2} = 300$, $K_{D2} = 100$. K_{P1} and K_{P2} are the proportional gain for link 1 and link 2, respectively. K_{D1} and K_{D2} are the derivative gain for link 1 and link 2, respectively. From the simulation results, we can see the fastest response of PD control ($t_{fPD} = 3.8865sec$) is still far slower than the response of the proposed zero-net-work method ($t_{fZW} = 1.1440sec$). Figure 6.6 shows the state trajectories of link 1, Figure 6.7 shows the state trajectories of link 2. Figure 6.8 shows the control trajectories of link 1, Figure 6.9 shows the control trajectories of link 2. The work done by external and reaction forces/torques for each link are shown in Figure 6.10 and Figure 6.11.

6.4.2 Simulation Case 2:

Physical parameters for the IBM 7535B 04 robot for case 1 is also used in this case. This example is a rest to rest motion of the manipulator from broken to straight configurations. The initial and the final conditions of the states in $[rad]$ and $[rad/s]$ are:

$$\mathbf{x}(0) = [0.76 \;\; 0 \;\; 0.2618 \;\; 0.0]^T \tag{6.67}$$

$$\mathbf{x}(t_f) = [0.0 \;\; 0.0 \;\; 0.0 \;\; 0.0]^T \tag{6.68}$$

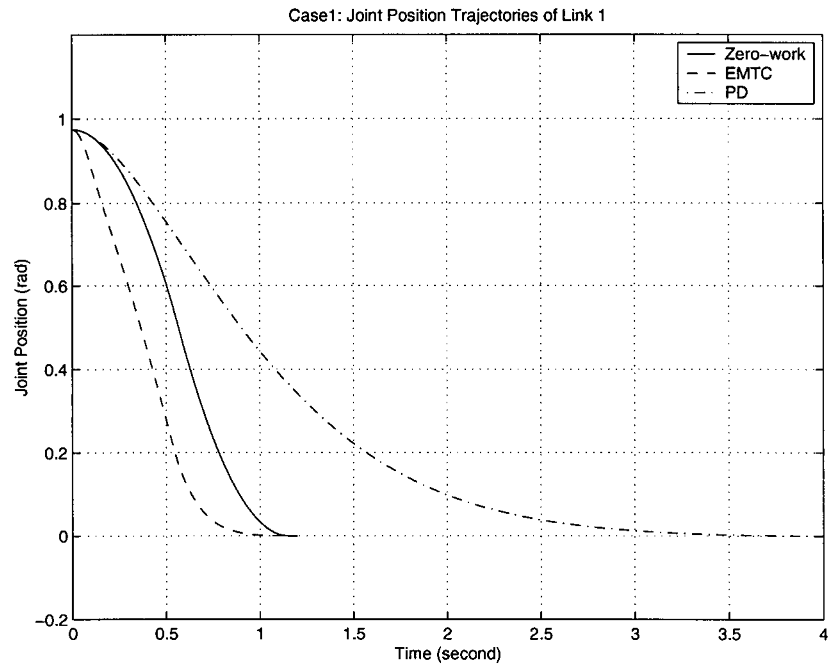

Figure 6.6: Case 1: State trajectories of link 1

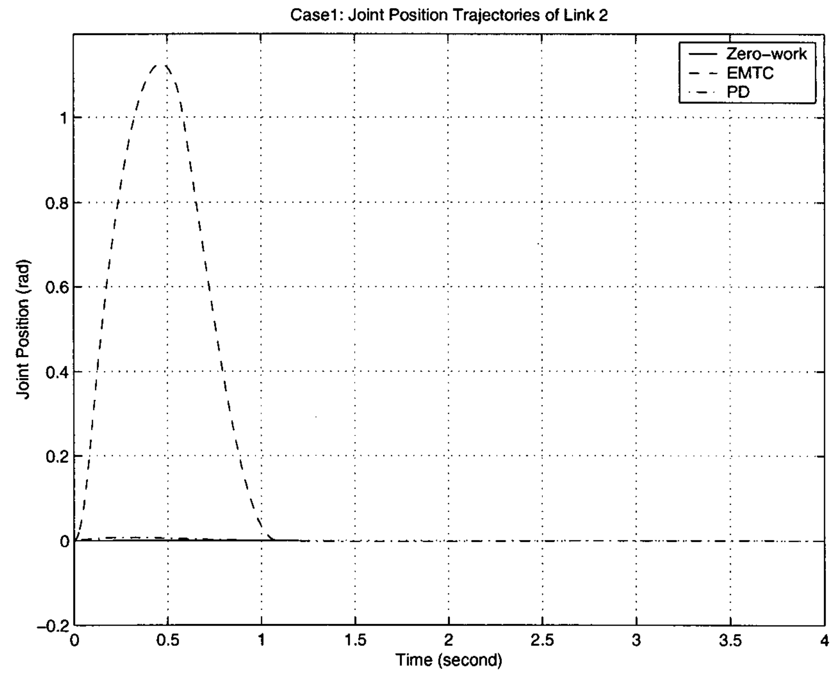

Figure 6.7: Case 1: State trajectories of link 2

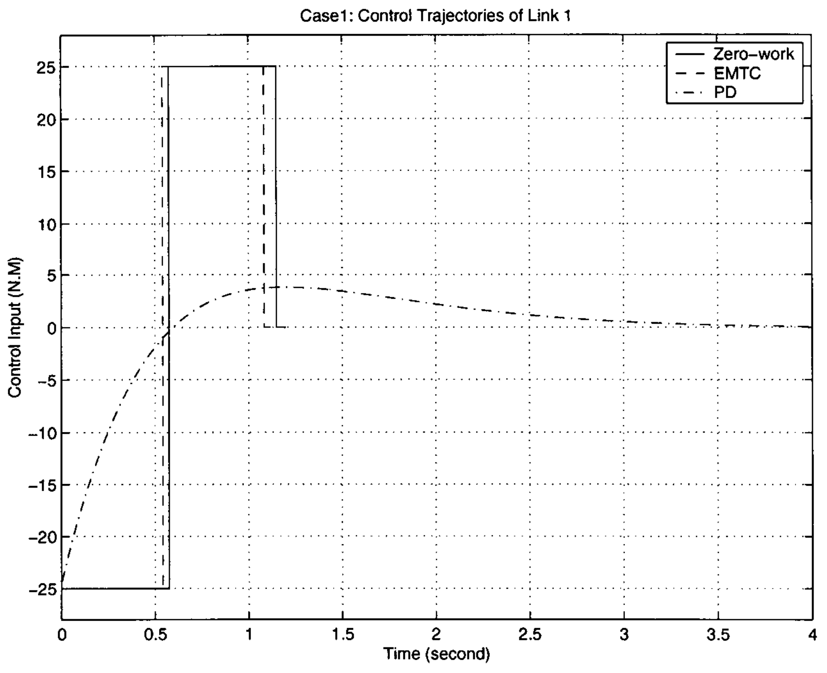

Figure 6.8: Case 1: Control trajectories of link 1

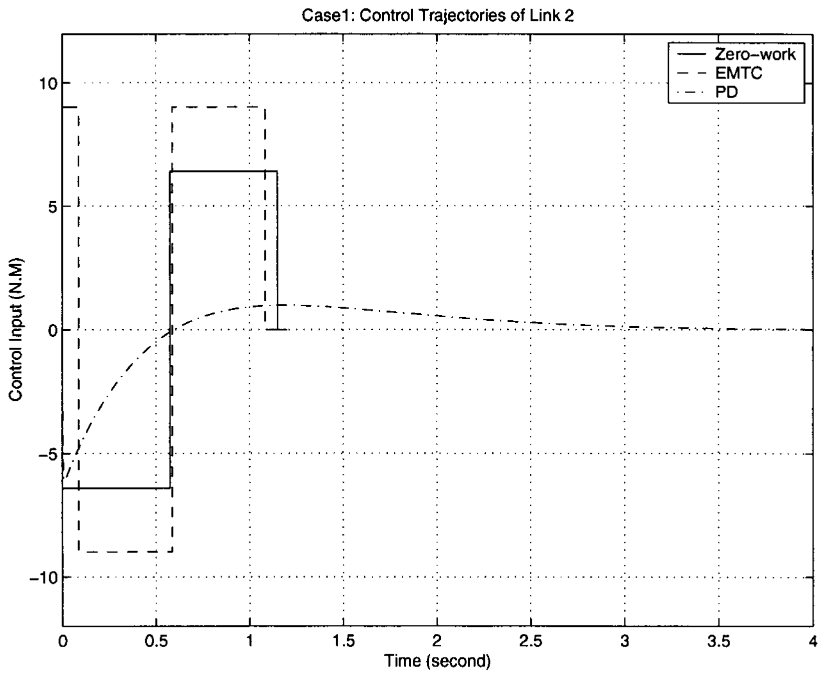

Figure 6.9: Case 1: Control trajectories of link 2

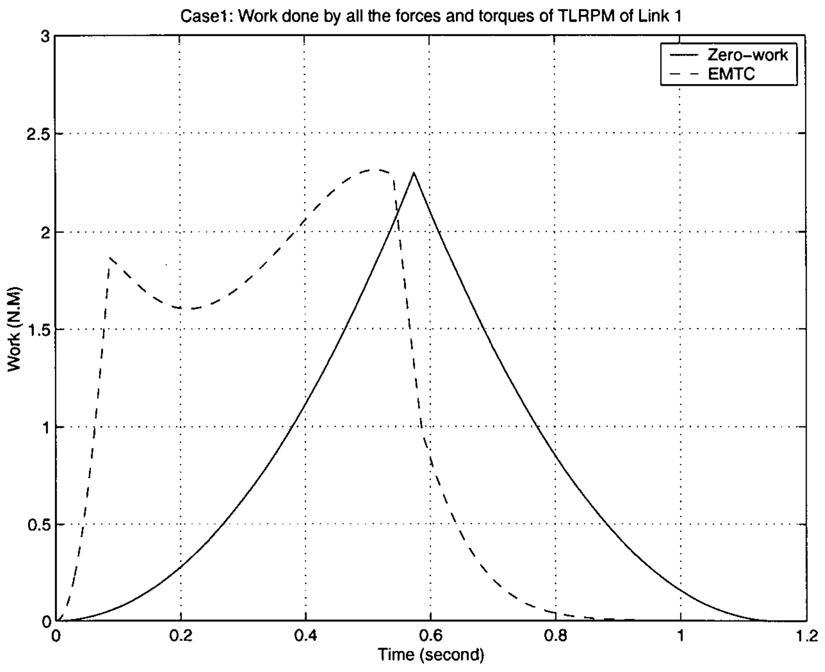

Figure 6.10: Case 1: Work done by forces/torques for link 1

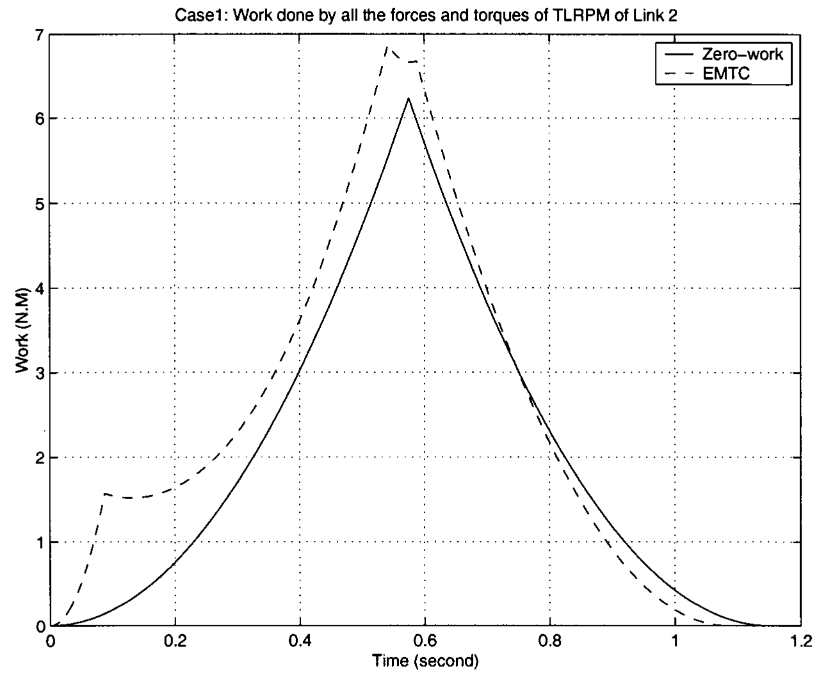

Figure 6.11: Case 1: Work done by forces/torques for link 2

Case 2	q_{10}	q_{20}	q_{1sw}	q_{2sw}	t_{1sw}	t_{2sw}	t_{1f}	t_{2f}	error
Zero-net-work	0.76	0.2618	0.4139	0.0388	0.5302	0.2127	1.0552	0.2485	3.08 %
Exact	0.76	0.2618			0.5119	0.4625 0.9508		1.0237	

Table 6.3: Case 2: Simulation results of two-link revolute manipulator

The simulation results are shown in Table 6.3. In this case, Control input for link 2 has 1 minimum to maximum switch: $q_{2sw} = 0.0388 rad$. Link 2 reaches the final position at $t_{2f} = 0.2485 sec$. The control input for link 2 switches from maximum value to the torque needed to hold link 2 at the final position while link 1 is moving for $t > 0.2485 sec$. Control input for link 1 has one switch: $q_{1sw} = 0.4139 rad$. The exact minimum time is $t_{f exact} = 1.0237 sec$, zero-net-work time is $t_{fZW} = 1.0552 sec$. The time difference between the proposed zero-net-work control method and EMTC is 3.08%. In this case the proposed zero-net-work solution is very close to the exact minimum time. Figure 6.12 shows the state trajectories of link 1, Figure 6.13 shows the state trajectories of link 2. Figure 6.14 shows the control trajectories of link 1, Figure 6.15 shows the control trajectories of link 2. The work done by external and reaction forces/torques for zero-net-work method are shown in Figure 6.16.

6.4.3 Simulation Case 3:

Case 3	q_{10}	q_{20}	q_{1sw}	q_{2sw}	t_{1sw}	t_{2sw}	t_{1f}	t_{2f}	error
Zero-net-work	0.376	0	0.1880		0.6340			1.2649	2.58 %
Exact	0.376	0			0.6165	0.0396 0.6645		1.2331	

Table 6.4: Case 3: Simulation results of two-link revolute manipulator

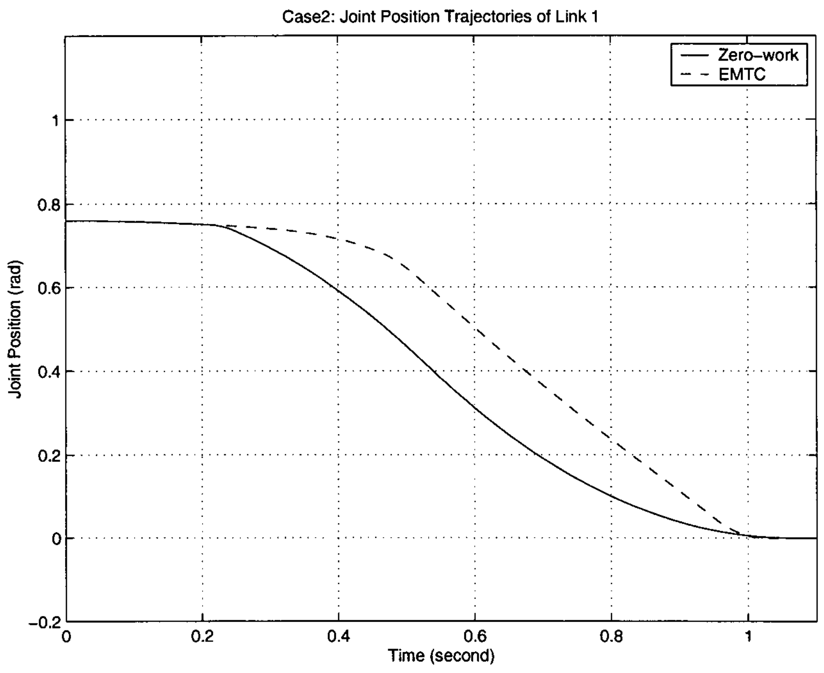

Figure 6.12: Case 2: State trajectories of link 1

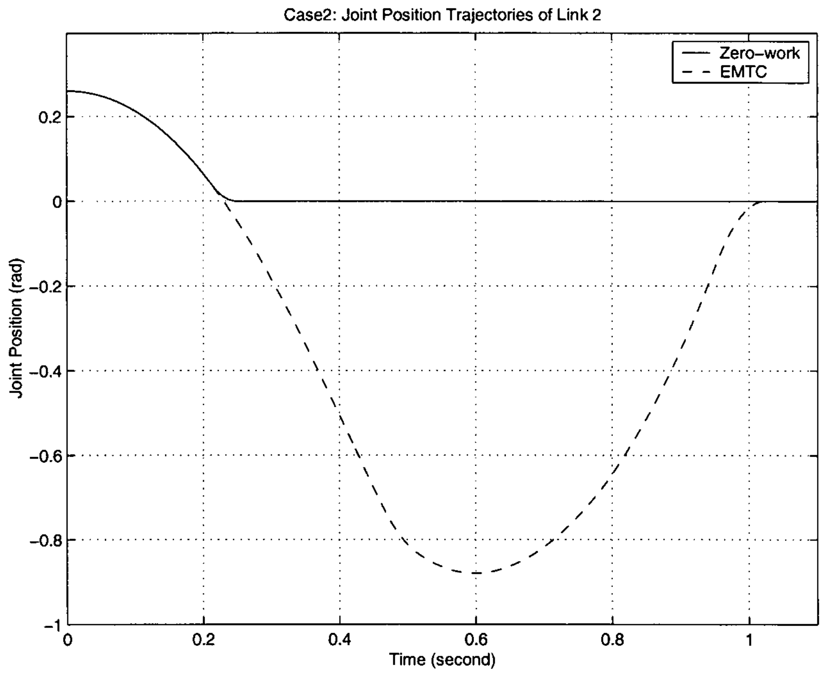

Figure 6.13: Case 2: State trajectories of link 2

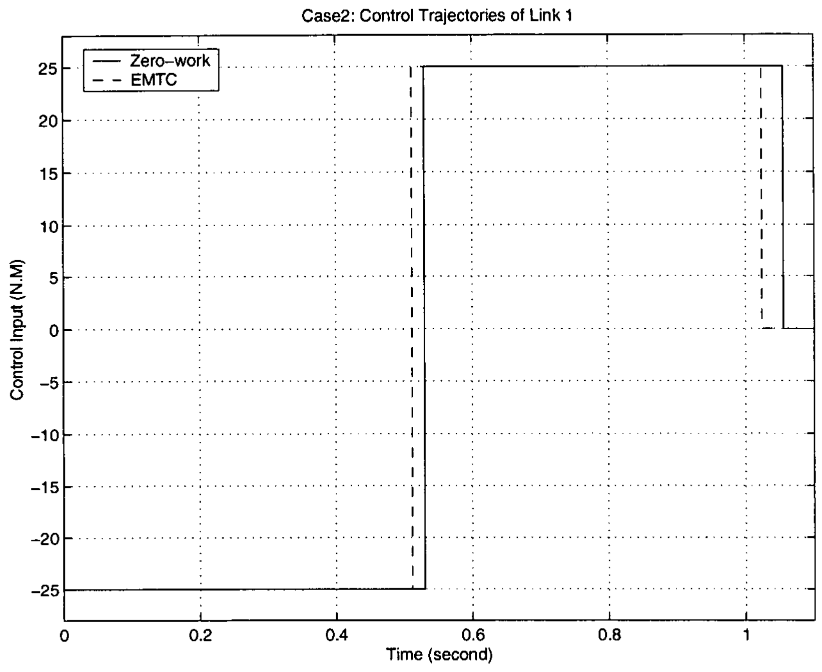

Figure 6.14: Case 2: Control trajectories of link 1

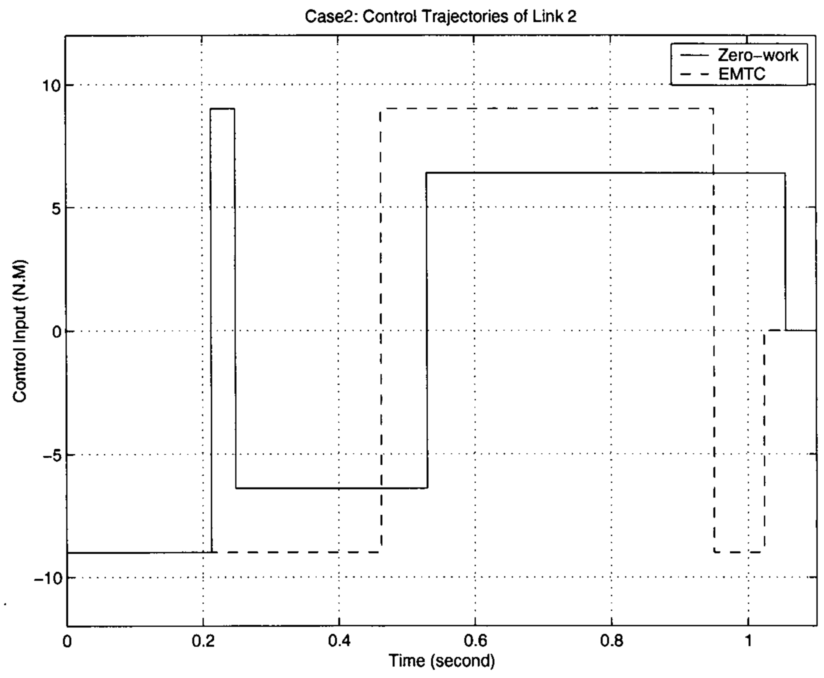

Figure 6.15: Case 2: Control trajectories of link 2

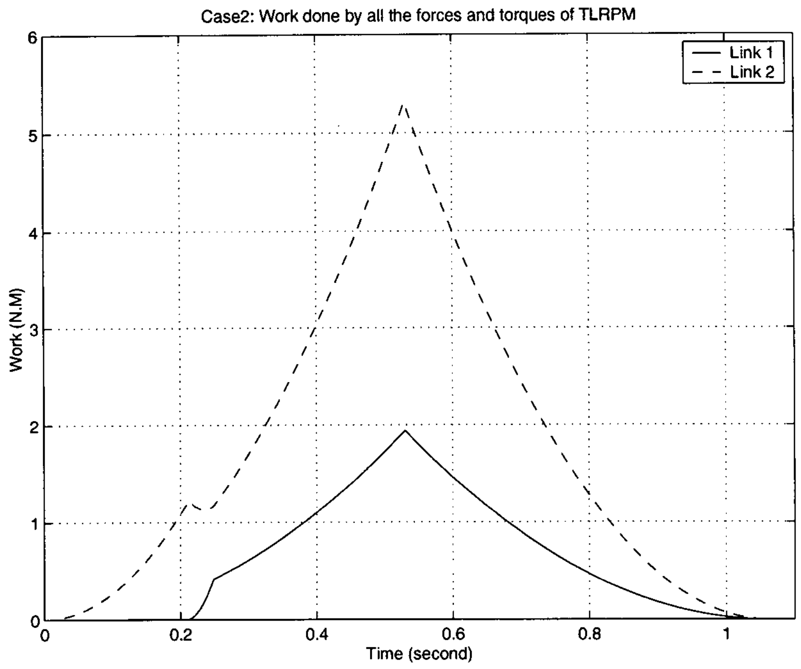

Figure 6.16: Case 2: Work done by forces/torques for each link (zero-net-work solution)

Following physical parameters are used for simulation in this case:

$$\begin{array}{ll} l_1 = 2l_{c1} = 0.4m & l_2 = 2l_{c2} = 0.25m \\ m_1 = 0.245kg & m_2 = 0.15315kg \quad M = 0.5kg \\ I_1 = 3.2688e-3kgm^2 & I_2 = 0.7986e-3kgm^2 \\ u_{1max} = 0.25NM & u_{2max} = 0.10NM \end{array} \tag{6.69}$$

This example is a rest to rest motion of the manipulator from straight to straight configurations. The initial and the final conditions of the states in $[rad]$ and $[rad/s]$ are:

$$\mathbf{x}(0) = [0.376 \ \ 0.0 \ \ 0.0 \ \ 0.0]^T \tag{6.70}$$

$$\mathbf{x}(t_f) = [0.0 \ \ 0.0 \ \ 0.0 \ \ 0.0]^T \tag{6.71}$$

The simulation results are shown in Table 6.4. The exact minimum time is $t_{fexact} = 1.2331sec$, zero-net-work time is $t_{fZW} = 1.2649sec$. The time difference between the proposed zero-net-work control method and EMTC is 2.58%. Figure 6.17 shows the state trajectories

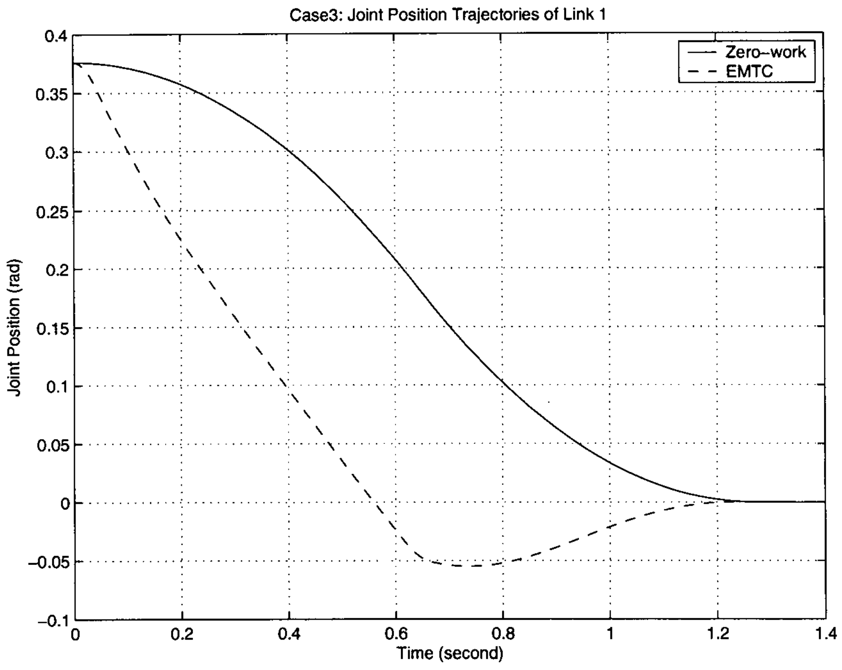

Figure 6.17: Case 3: State trajectories of link 1

of link 1, Figure 6.18 shows the state trajectories of link 2. Figure 6.19 shows the control trajectories of link 1, Figure 6.20 shows the control trajectories of link 2. The work done by external and reaction forces/torques for zero-net-work method are shown in Figure 6.21.

6.4.4 Simulation Case 4:

Case 4	q_{10}	q_{20}	q_{1sw}	q_{2sw}	t_{1sw}	t_{2sw}	t_{1f}	t_{2f}	error
Zero-net-work	0.8	0.2	0.4471	0.0142	0.9572	0.2801	1.7026	0.2998	5.07 %
Exact	0.8	0.2			0.1018 0.9120	0.4322 1.4980		1.6204	

Table 6.5: Case 4: Simulation results of two-link revolute manipulator

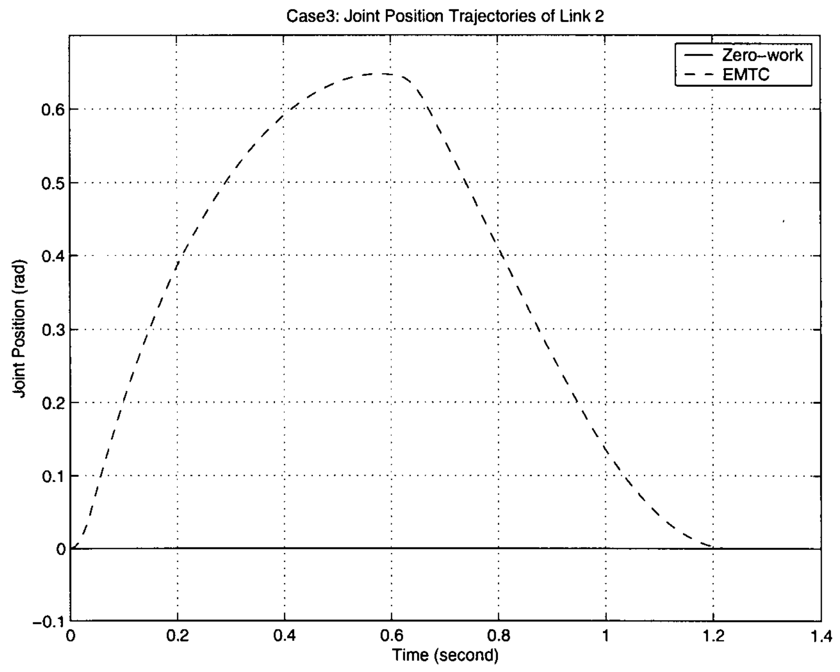

Figure 6.18: Case 3: State trajectories of link 2

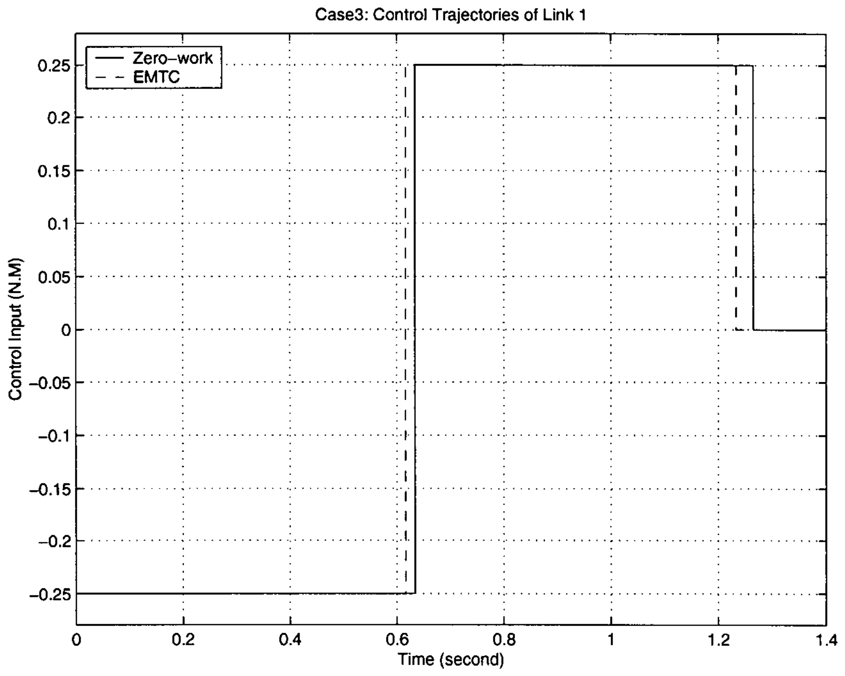

Figure 6.19: Case 3: Control trajectories of link 1

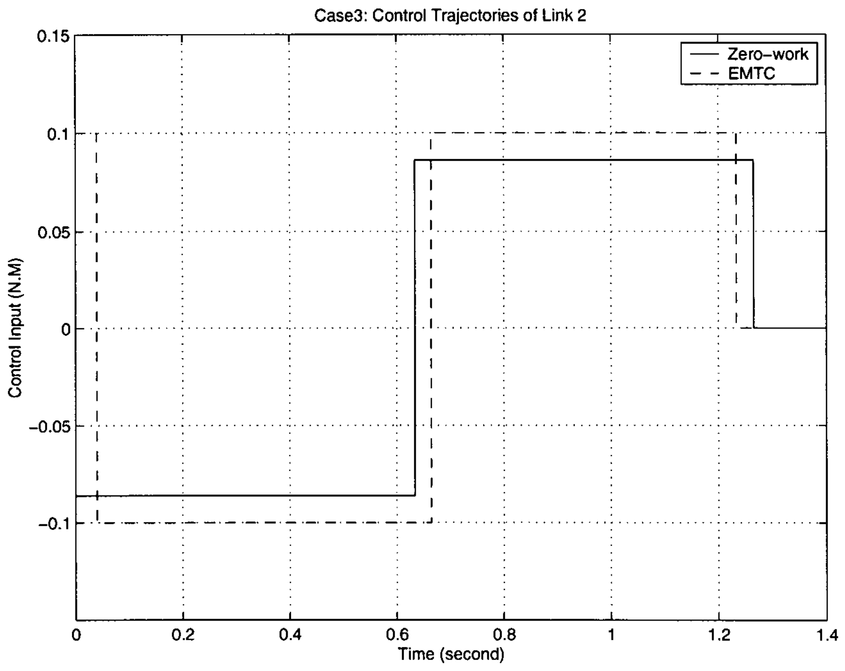

Figure 6.20: Case 3: Control trajectories of link 2

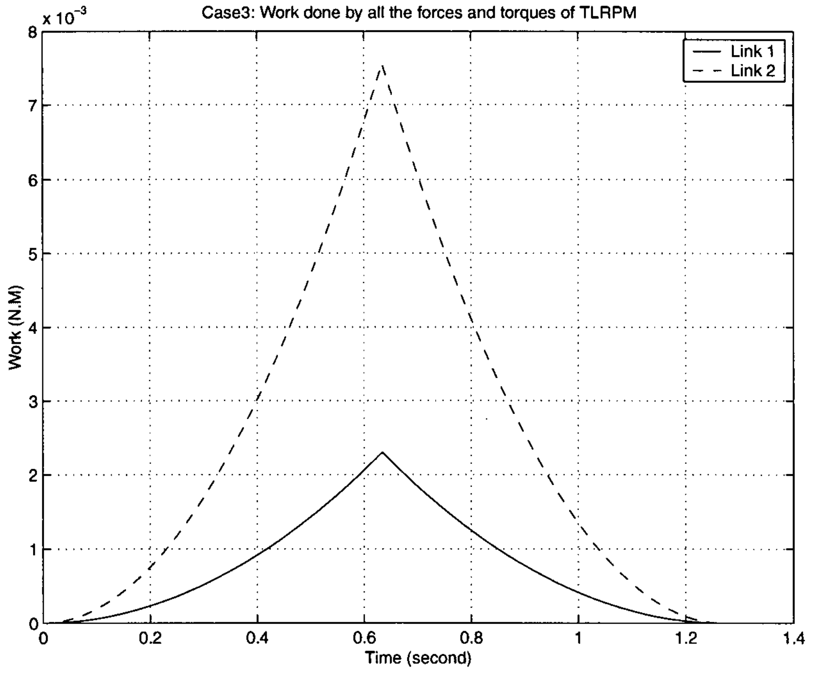

Figure 6.21: Case 3: Work done by forces/torques for each link (zero-net-work solution)

Following physical parameters are used for simulation in this case:

$$\begin{array}{ll} l_1 = 2l_{c1} = 1.0m & l_2 = 2l_{c2} = 0.625m \\ m_1 = 0.61261kg & m_2 = 0.38288kg \quad M = 1.25kg \\ I_1 = 51.0547e - 3kgm^2 & I_2 = 12.4660e - 3kgm^2 \\ u_{1max} = 3.9NM & u_{2max} = 1.56NM \end{array} \tag{6.72}$$

This example is a rest to rest motion of the manipulator from broken to straight configurations. The initial and the final conditions of the states in $[rad]$ and $[rad/s]$ are:

$$\mathbf{x}(0) = [0.8 \;\; 0.0 \;\; 0.2 \;\; 0.0]^T \tag{6.73}$$

$$\mathbf{x}(t_f) = [0.0 \;\; 0.0 \;\; 0.0 \;\; 0.0]^T \tag{6.74}$$

The simulation results are shown in Table 6.5. The exact minimum time is $t_{fexact} = 1.6204sec$, zero-net-work time is $t_{fZW} = 1.7026sec$. The time difference between the proposed zero-net-work control method and EMTC is 5.07%. Figure 6.22 shows the state trajectories of link 1, Figure 6.23 shows the state trajectories of link 2. Figure 6.24 shows the control trajectories of link 1, Figure 6.25 shows the control trajectories of link 2. The work done by external and reaction forces/torques for zero-net-work method are shown in Figure 6.26.

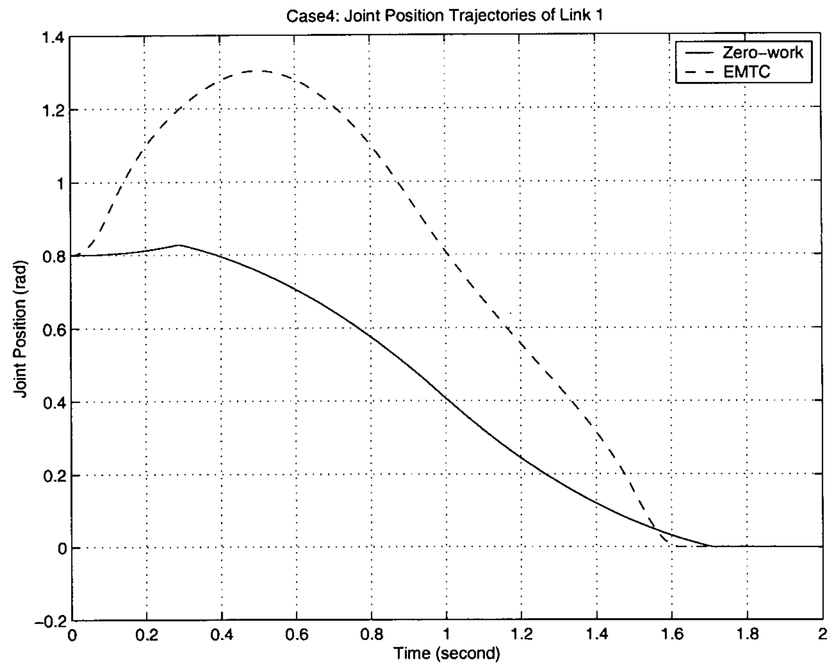

Figure 6.22: Case 4: State trajectories of link 1

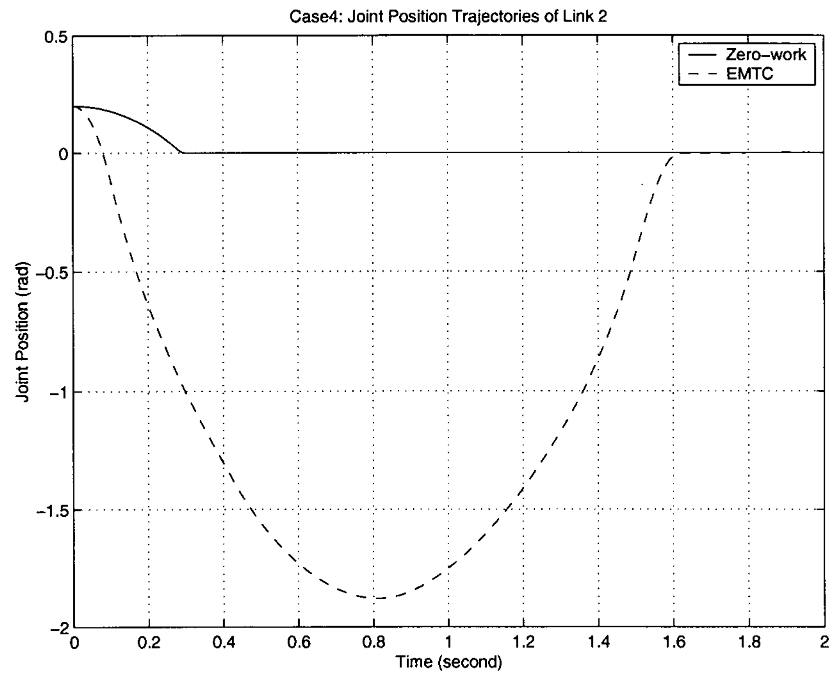

Figure 6.23: Case 4: State trajectories of link 2

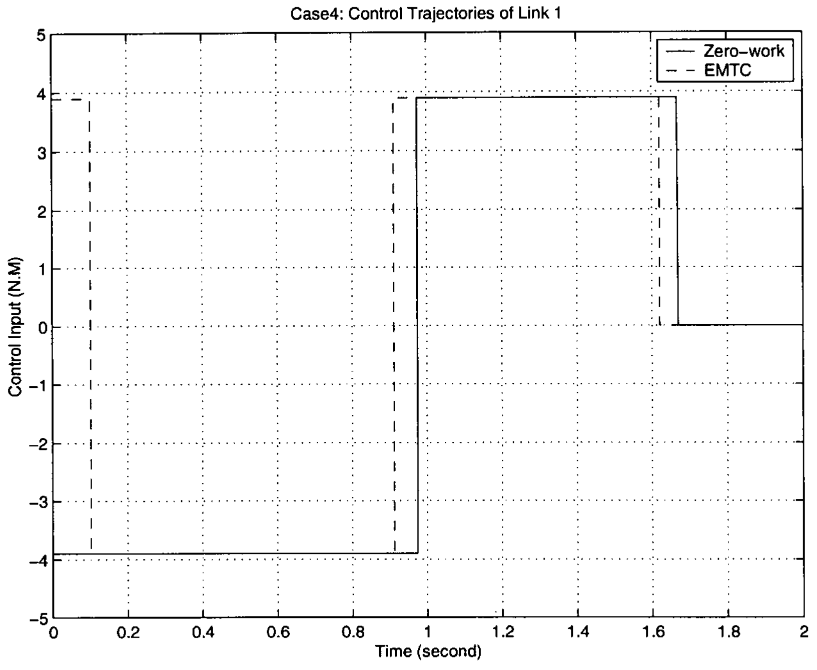

Figure 6.24: Case 4: Control trajectories of link 1

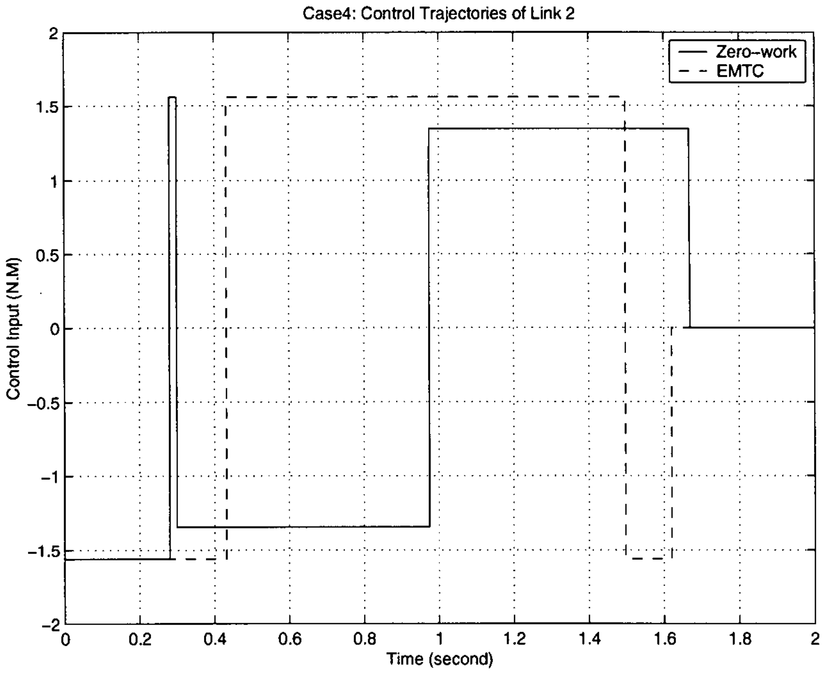

Figure 6.25: Case 4: Control trajectories of link 2

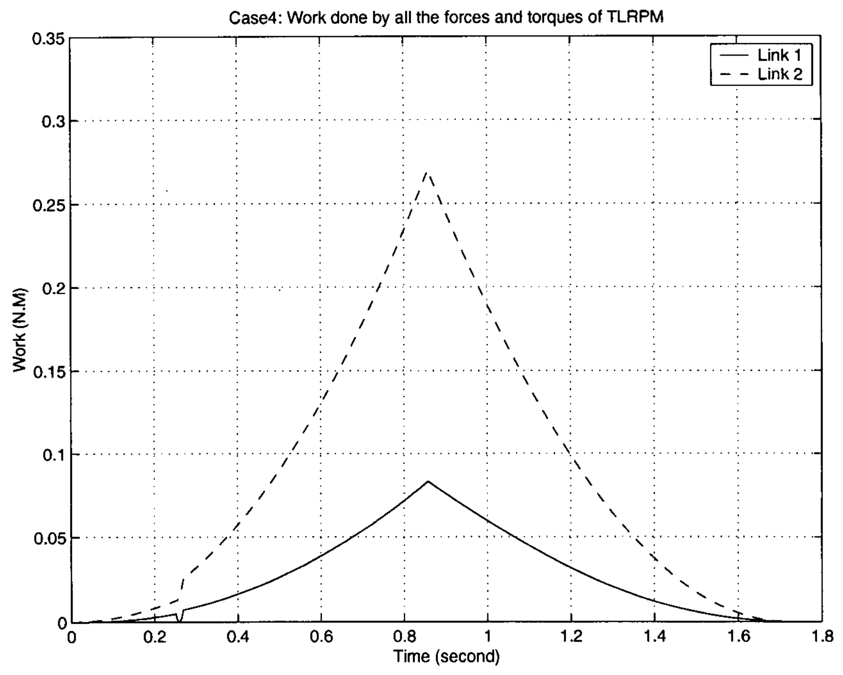

Figure 6.26: Case 4: Work done by forces/torques for each link (zero-net-work solution)

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis a near minimum time zero-net-work control algorithm has been developed to control the point to point motion of a robot manipulator. This proposed zero-net-work method has the basic structure of the exact minimum time control. The zero-net-work method assumes for each link the control input is bang-bang and switches from its minimum to maximum value only once, also the solution satisfies all the constraints (control input, boundary condition and work-energy constraints). Based on this idea, we have designed zero-net-work controllers for one-link manipulators, a cylindrical and a revolute manipulator. From the simulation results, we can see if the exact minimum time solution is bang-bang, and has one switch for each link, using the zero-net-work method we can find the exact minimum time. If the exact minimum time solution has more than one switch for each link, using the zero-net-work method we can find near minimum time solution.

The main advantage of the zero-net-work method is that it is computationally efficient and does not require a good initial guess for the unknown initial boundary conditions. Because the zero-net-work method uses the maximum control input magnitude most of the time, it is much faster than a traditional PID controller. Also, the zero-net-work method does not need to solve the Two Point Boundary Value Problem (TPBVP), it can generate the control input much faster than the exact minimum time method. So we can say the proposed method provides a practical solution to control the motion of a robot manipulator near the minimum time. Currently, the zero-net-work method has two stages: offline calculation and online application. If we further improve the efficient of the algorithm, it can be used online.

7.2 Suggestion for Further Research

The limitations of the zero-net-work method are:

a) Like all model based control algorithm, the zero-net-work method requires a very good model of the plant. It can not deal with large model mismatch or large disturbances. Currently, a PID controller is used as a backup controller to guarantee the manipulator reaches its final position. A better plant model could be of benefit in future work.

b) In this thesis we assume the control constraints for each link is $-u_{imax} \leq u_i \leq u_{imax}$. In practice, most actuators do not allow to switch from minimum to maximum value in very short period of time. More constraints on control input must be defined. We can set the actuator constraints as a saturation limit and a slew rate limit for example.

Further research includes:

a) Improving the efficiency of the zero-net-work method. Currently, the searching start point for each link is at its final position, and using a fixed step size. Computation time can be reduced by using variable searching step size. First we can use a large step size to reduce the searching range, and then use a smaller step size to find the exact switching position.

b) Practical implementation of the algorithm should be investigated.

c) A more practical model of the actuator should be used. A smooth switching period function should be defined and used to search for the solution.

d) A robust controller for large model mismatch or disturbances need to be developed.

Bibliography

[1] Bobrow, J.E. 1982. "Optimal control of robotic manipulators", Ph.D. dissertation, University of California, Los Angeles.

[2] Bobrow, J.E., Dubowsky,S. and Gibson,J.S. 1983 (San Franciso). "On the optimal control of robotic manipulators with actuator constraints", Proc. American Control Conference 2 :pp. 782-787.

[3] Bobrow, J.E., Dubowsky, S.,and Gibson,J.S. 1985. "Time optimal control of robotic manipulators along specified paths", Int. J. Robot. Res. 4(3): pp. 3-17.

[4] Bobrow,J.E. 1988. "Optimal robot path planning using the minimum-time criterion", IEEE J. Robotics and Automation, Vol. RA-4(4), pp. 443-450.

[5] Broyson, A.E., Jr., and Ho, Y-C 1975. Applied optimal control - optimization, estimation, and control, Hemisphere, Washing, D.C. .

[6] Chen,C.C. 1989. "On the structure of the time-optimal controls for robotic manipulators", IEEE Transactions on Automation Control, Vol.34, No.1, January.

[7] Chen,Y.B. 1988. "Minimum-time control of robotic manipulators", Ph.D. Dissertation, Rensselaer Polytechnic Institute, Troy, NY.

[8] Chen,Y.B., and Desrochers, A.A. 1988. "Time-optimal control of Two-Degree of Freedom Robot Arms", Proc. IEEE Conf. Robot. and Autom ., Philadelphia,PA: pp. 1210-1215.

[9] Chen,Y.B. and Desrochers, A.A. 1989. "The Structure of Minimum-Time Control Law for Robotic Manipulators with Constrained ", Proc.IEEE Int. Conf. Robotics and Autom., Scottsdale, AZ, May, pp. 970-975.

[10] Chen,Y.B. and Desrochers, A.A. 1989. "Minimum-time control laws for robotic manipulators", Proc. of IEEE 28th Conference on Decision and Control , pp. 2494-2499.

[11] Chen,Y.B. and Desrochers, A.A. 1990. "A proof of the structure of the minimum-time control law of robotic manipulators using a Hamiltonian formulation", IEEE Trans. Robotics and Automation, Vol. 6, pp. 388-393.

[12] Chen,Y.B. and Huang, J. 1992. "An efficient computation of time-optimal control trajectory for robotic point-to-point motion", Proc. 4th Conf. on Intelligent Robot. Syst. for Space Exploration(Troy,NY), Sept. .

[13] Chen,Y.B. and Desrochers, A.A. 1993. "Minimum-time control laws for robotic manipulators", International Journal of Control, Vol. 57, pp. 1-27.

[14] Chen,Y.B. , Huang, J. and Wen, J.T.Y. 1993. "A continuation method for time-optimal control synthesis for robotic point-to-point motion", Processing of the 32nd Conferance on Decision and Control, San Antonio,Texas. December: pp. 1628-1663.

[15] Dahl, O., 1994. "Path-constrained robot control with limited torques - experimental evaluation" , IEEE Transactions on Robotics and Automation, Vol. 10, No. 5. October 1994, pp. 658-669.

[16] Formal sky, A.M . and Osipov,S.N. 1990. "On the problem of time-optimal manipulator arm turning", IEEE Trans. Automatic Control, Vol. 35, pp. 714-719.

[17] Fotouhi-C., R., 1996. " Time-optimal control of two-link manipulators", PhD thesis, University of Saskatchewan, Saskatoon, Canada.

[18] Fu, K.S., Gonzalez, R.C., Lee, C.S., Lee, 1987. Robotics: control, sensing, vision, intelligence, McGraw-Hill.

[19] Geering , H.P., Guzzella,L., Hepner, S.A.R. and Onder, C.H. 1986. "Time-optimal motions of robots in assembly tasks", IEEE Trans. Automatic Control AC-31(6): pp. 512-518.

[20] Galicki, M. , 1998. "The planning of robotic optimal motions in the presence of obstacles" , The International Journal of Robotics Research, Vol 17, No 3, March 1998, pp248-259.

[21] Hol, C.W.J. , L.G. van Willigenburg, E.J. van Henten and G. van Straten 2001. "A new optimization algorithm for singular and non-singular digital time-optimal control of robots ", Proceedings of the 2001 IEEE International Conference on Robotics and Automation, Seoul, Korea. May 21-26 pp. 1136-1141.

[22] Kahn,M.E. 1970. "The near-minimum time control of open-loop articulated kinematic chains", Ph.D. dissertation, Stanford University.

[23] Kahn, M.E., and Roth,B. 1971. "The near-minimum-time control of open-loop articulated kinematic chains", ASME J. Dyn. Sys., Meas., Contr., Sept., pp. 164-172.

[24] Kim, B.K. and Shin,K.G. 1985. "Suboptimal control of industrial manipulators with weighted minimum-time-fuel criterion", IEEE Trans. Aut. Cont. , Vol. AC-30, pp. 1-10.

[25] Kirk, D.E. 1970. Optimal control theory, an introduction, Prentice-Hall, Englewood Cliffs,N.J. .

[26] Lin, S.-Y. 1992. "A hardware implementable two-level parallel computing algorithm for general minimum-time control", IEEE Trans. Automatic Control, Vol. 37, pp. 589-603.

[27] Luh, J.Y.S., and Walker,W.M. 1977(New Orleans). "Minimum-time along the path for a mechanical arm", Proc. 1977 IEEE Conf. Decision Contr., 2: pp.755-759.

[28] Luh,J.Y.S., and Lin, C.S. 1981. "Optimal path planning for mechanical manipulators", ASME J. Dyn. Sys.,Meas., Contr., Vol.2, pp. 330-335, June 1981.

[29] McCarthy, J.M. and J.E. Bobrow, 1992. "The number of saturated actuators and constraint forces during time-optimal movement of a general robotic system", Proceeding of the 1992 IEEE International Conference on Robotics and Automation, Nice, France - May 1992, pp. 542-546.

[30] Meier,E-B., and Bryson, A.E.Jr. 1990. "Efficient algorithm for time-optimal control of a two-link manipulator" , J.Guidance Control Dynam. 13(5): pp. 859-866.

[31] Niv, M., and Anslander, D.M. 1984. "Optimal control of a robot with obstacles", Proc. American Contr. Conf. (San Diego, CA), June. : pp. 280-287.

[32] Pfeiffer,F. and Johanni, R., 1987. "A concept for manipulator trajectory planning", IEEE J. Robotics and Automation.

[33] Rajan, V.T. . 1985. "Minimum time trajectory planning", Proc. of IEEE Conf. on Robotics and Automation, pp. 759-764.

[34] Sahar, G. and Hollerbach, J.M., 1985. "Planning of minimum-time trajectories for robot arms", Proc. of IEEE Int. Conf. on Robotics and Automation, St.Louis, MO. March. pp. 751-758.

[35] Sahar, G. and Hollerbach, J.M., 1986. "Planning of minimum-time trajectories for robot arms" Int. J. Robot. Res. Vol. 5, No.3 :pp. 90-100.

[36] Sato, O.,et.al, 1983. "Minimum-time control of a manipulator with two-degree of freedom" Bul. of JSME , Vol. 26, No.218, August: pp. 1404-1410.

[37] Sciavicco, L. and Siciliano, B. 1996. Modeling and control of robot manipulators, McGraw Hill, New York.

[38] Shiller, Z. and Dubowsky, S. 1985. "On the optimal control of robotic manipulators with actuator and end-effector constraints" , Proc. IEEE Int. Robotics and Autom. : pp. 614-620.

[39] Shiller, Z., and Dubowsky,S. 1987. "Time optimal paths and acceleration lines of robotic manipulators", 26th IEEE Conf. on Decision and Control. Los Angeles, Dec. .

[40] Shiller, Z., and Dubowsky,S. 1988. "Global time optimal motion of robotic manipulators in the presence of obstacles", Pros. of IEEE Conf. Robotics and Automation , pp. 370-375.

[41] Shiller, Z., and Dubowsky,S. 1991. "On computing the global time optimal motions of robotic manipulators in the prescence of obstacles", IEEE Trans. Robotics and Automation, Vol.7, pp. 759-764.

[42] Shiller, Z., Chang, H. and Wong, V., 1996. "The practical implementation of time-optimal control for robotic manipulators" , Robotics & Computer-Integrate Manufacturing, Vol. 12, No. 1, pp. 29-39.

[43] Shin,K.G. and Mckay,N.D. 1983. "An efficient robot arm control under geometric path constraints", IEEE Conf. on Decision and Control , San Franciso,CA.

[44] Shin,K.G. and Mckay,N.D. 1984(San Diegp). "Open-loop minimum-time control of mechanical manipulators and its application", Proc. American Contr. Conf. : pp. 296-303.

[45] Shin,K.G., and Mckay,N.D. 1985. "Minimum-time control of robotic manipulators with geometric path constraints", IEEE Trans. Automatic Control. Vol. AC-30,No.6: pp. 531-541.

[46] Singh,S. and Leu, M.C. , 1987. "Optimal trajectory generation for robotic manipulators using dynamic programming", ASME Trans. J. Dynamic Sys., Meas. ,and Control, Vol.109, June, pp. 88-86.

[47] Sontag,E.D. and Sussmann 1986. "Time-optimal control of manipulators", Proc. IEEE Int. Conf. on Robotics and Automation, San Franciso,CA,April, pp. 1692-1697.

[48] Slotine, J-J, E., and Yang, H.S., 1989. "Improving the efficiency of minimum-time path-following algorithms", IEEE Trans. Robotics and Automations, Vol.5, No.1..

[49] Spong, M.W. and Vidyasagar, M., 1989. Robot dynamics and control, John Wiley & Sons.

[50] Wen, J. and Desrochers,A.A. 1986. "Sub-time-optimal control statrategies for robotic manipulators", Proc. IEEE Int. Conf. on Robotics and Automation. (San Franciso), April, pp. 402-406.

[51] Weinreb, M.W. and Orin,D.E.. 1982. "Efficient dynamic computer simulation of robot mechanisms", ASME J. Dynam. Sys. Measurement Control 104: pp. 205-211.

[52] Weinreb, A. and Bryson,A.E. . 1985. "Optimal control of systems with hard control bounds", IEEE Trans. Automatic Control Vol.30, pp. 1135-1138.

[53] Weinreb, A. and Bryson,A.E. . 1985. "Minimum-time control of a two-link robot arm" IFAC Control Application of Nonlinear Programming and Optimization, Capri,Italy.

[54] Yang,H.S. and Slotine, J-J.E. 1993. "Efficient algorithms for near-minimum-time control of robot manipulators", M.I.T. Report NSL-931001, Nonlinear System Laboratory.

[55] Yang, H.S., and Slotine,J-J.E. 1994. "Fast algorithms for near-minimum-time control of robot manipulators", Int. J. Robot. Res. 13(6): pp. 521-532.

Appendix A

Simulation Program Listings

A.1 S-function of One-link Horizontal Manipulator

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                     File name one1.m                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Chapter 4
% One link horizontal manipulator
% This is s-function for simulink
function[sys,x0]=robot2(t,x,tau,flag,x01,x02)
I=10;
if abs(flag)==1
a=[x(2);
   tau/I];
sys=a;
elseif abs(flag)==3
sys=x;
elseif abs(flag)==0
sys=[2,0,2,1,0,1];
x0=zeros(2,1);
x0(1)=x01;
x0(2)=x02;
else
sys=[];
end
```

A.2 S-function of One-link Vertical Manipulator

```
% Chapter 4
% Onelink vertical manipulator
% This is s-function for simulink

function[sys,x0]=robot2(t,x,tau,flag,x01,x02)
if abs(flag)==1
l=0.3;
m=5;
I=1/3*m*l^2;
% I = 0.1500
g=9.8;
a=[x(2);
   (tau-m*g*l/2*cos(x(1)))/I];
sys=a;
elseif abs(flag)==3
sys=x;
elseif abs(flag)==0
sys=[2,0,2,1,0,1];
x0=zeros(2,1);
x0(1)=x01;
x0(2)=x02;
else
sys=[];
end
```

A.3 Exact Minimum Time Solution of One-link Vertical Manipulator

```
C%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C%   File name: onelkMT.for   %
C%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C% Chapter 4
C Main program onelkMT.for
C
C***BEGIN PROLOGUE onelkMT.for
C***DATE WRITTEN   030501     (YYMMDD)
C***REVISION DATE  030601     (YYMMDD)
C***AUTHOR  TAO FAN
C***PURPOSE SOLVING TPBVP PROBLEM IN CHAPTER 4.
C***DESCRIPTION
C     This is FORTRAN program used to solve exact minimum time control
C     problem of one-link vertical robot manipulator
C     in chapter 4 of the thesis.
C     Subroutine "MUSN" is used to solve
C     the nonlinear TPBVP.
C***REFERENCES  (NONE)
C***ROUTINES CALLED  MUSN, FDIF, XOT
C   ROUTINES FDIF is used to define the differential equations of the system
C   ROUTINES XOT is used to evaluate the initial approximation
C   XO(t) of the solution.
C   Description of variables used in the program:
C   Taumax is maximum control inputs
C   Tau is control inputs

C***END PROLOGUE onelkMT.for


C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION ER(5),TI(12),X(5,12),Q(5,5,12),U(15,12),D(5,12),
     1          PHIREC(15,12),W(360),WGR(20)
      INTEGER IW(60)
      EXTERNAL FDIF,XOT,G
C
C
C     N is the order of the system.
C
C     NU is one of the dimension of U and PHI.
C        NU must be greater than or equal to N * (N+1) / 2.
C
C     NTI is one of the dimension of  TI, X, S, Q, U en PHI.
C        NTI must be greater than or equal to the total number of
C        necessary output points + 1  (i.e. if the entry value for
C        NRTI > 1, NTI may be equal to the entry value of NRTI + 1)
C
C     LW is the dimension of W. LW >= 7*N + 3*N*NTI + 4*N*N
C
C     LIW is the dimension of IW. LIW >= 3*N + NTI
C
C     ER(3) must contain the machine precision
C
C     LWG is the dimension of WGR. LWG >= (total number of grid points)
C        / 5. The minimum number of grid points between 2 succesive output
C        points is 5, so the minimum value for LWG is the number of actually
C        used output points. Initially a crude estimate for LWG has to be made
C
C     ER(1) must contain the required tolerance for solving the differential
```

```
C        equation.
C     ER(2) must contain the initial tolerance with which a first aproximate
C         solution will be computed. This approximation is then used as an
C         initial approximation for the computation of an solution with an
C         tolerance ER(2)*ER(2) and so on until the required tolerance is
C         reached. As an initial tolerance max(ER(1),min(ER(2),1.d-2)) will
C         be used
C
C     A,B the two boundary points
C
C     NRTI is used to specify the output points. There are 3 ways to
C          specify the output points:
C          1) NRTI = 0, the output points are determined automatically using AMP.
C          2) NRTI = 1, the output points are supplied by the user in the array TI.
C          3) NRTI > 1, the subroutine computes the (NRTI+1) output points TI(k) by
C                           TI(k) = A + (k-1) * (B - A) / NRTI ;
C                     so TI(1) = A and TI(NRTI+1) = B.
C          Depending on the allowed increment between two succesive output points,
C          more output points may be inserted in cases 2 and 3.
C          On exit NRTI contains the total number of output points.

C     AMP must contain the allowed increment between two output
C          points. AMP is used to determine output points and to assure that the
C          increment between two output points is at most AMP*AMP. A small value
C          for AMP may result in a large number of output points.
C          Unless 1 < AMP < .25 * sqrt(ER(1)/ER(3)) the default value
C          .25 * sqrt(ER(1)/ER(3)) is used.

C     ITLIM maximum number of allowed iteration
C     IERROR=1, diagnostics will be printed during computation

      N = 5
      NU = 15
      NTI = 12
      LW = 360
      LIW = 60
      ER(3) = 1.1D-15
      LWG = 20
      ER(1) = 1.D-6
      ER(2) = 1.D-2
      A = 0.D0
      B = 1.D0
      NRTI=10
      AMP=0.01D0
      ITLIM=20
      IERROR=1
      OPEN(FILE='data1MT.m',UNIT=6,status='unknown')
      WRITE(6,30)
   30 FORMAT(2X,'% This is the results of TPBVP')
      CALL MUSN(FDIF,XOT,G,N,A,B,ER,TI,NTI,NRTI,AMP,ITLIM,X,Q,U,NU,D,
     1          PHIREC,KPART,W,LW,IW,LIW,WGR,LWG,IERROR)
      WRITE(6,*) ' MUSN: IERROR =',IERROR
      WRITE(6,200) A,B,ER(1),ER(2),ER(4),ER(5),KPART
  200 FORMAT(' A = ',F8.4,3X,'B = ',F8.4,/,' REQUIRED TOLERANCE = ',1P,
     1 D12.5,3X,'START TOLERANCE  = ',D12.5,/,
     2 ' CONDITION NUMBER = ',D12.5,3X,
     3 'AMPLIFICATION FACTOR = ',D12.5,/,' K-PARTITIONING =',I2,/)
      IF (IERROR.NE.0) GOTO 3000
      WRITE(6,215)
  215 FORMAT(' I ',4X,'T',9X,'X1',12X,'X2',12X,'X3',12X,'X4',12X,'X5',
     1 /)
      DO 2200 K = 1 , NRTI
```

```
        WRITE(6,220) K,TI(K),(X(J,K),J=1,N)
 2200 CONTINUE
  220 FORMAT(' ',I2,1X,F6.4,1P,5(2X,D12.5))
 3000 CONTINUE
      STOP
      END


      SUBROUTINE FDIF(T,Y,F)
C     ----------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION Y(5),F(5)
      DOUBLE PRECISION zm,zl,zI,zg,Taumax,Tau
     * X1,X2,P1,P2,Z

      X1=Y(1)
      X2=Y(2)
      P1=Y(3)
      P2=Y(4)
      Z=Y(5)

C     Physical parameters of the manipulator
      zm=5.0D0
      zl=0.3D0
      zI=0.15D0
      zg=9.8D0
      Taumax=8.0D0

C     Decided the control input
      IF(P2.LT.0.0D0) THEN
        Tau=Taumax
      ELSE
        Tau=-Taumax
      END IF

      F(1) = Z*X2
      F(2) = Z*(Tau-zm*zg*zl*DCOS(X1)/2.0D0)/zI
      F(3) = -Z*zm*zg*zl*P2*DSIN(X1)/(2.0D0*zI)
      F(4) = -Z*P1
      F(5) = 0.0D0
      RETURN
C     END OF FDIF
      END


      SUBROUTINE XOT(T,X)
C     -------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION X(5)
C
C     Case1
      X(1) = 1.D0
      X(2) = 0.0D0
      X(3) = 0.2446D0
      X(4) = 0.0125D0
      X(5) = 0.6268D0

C     Case2
C     X(1) = 1.57D0
C     X(2) = 0.0D0
```

```
C     X(3) = 0.2051D0
C     X(4) = 0.0187D0
C     X(5) = 0.7525D0

      RETURN
C     END OF XOT
      END


      SUBROUTINE G(N,XA,XB,FG,DGA,DGB)
C     -------------------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION XA(N),XB(N),FG(N),DGA(N,N),DGB(N,N)
C
      DO 1100 I = 1 , N
      DO 1100 J = 1 , N
        DGA(I,J) = 0.D0
        DGB(I,J) = 0.D0
 1100 CONTINUE
      DGA(1,1) = 1.D0
      DGA(2,2) = 1.D0
      DGB(1,1) = 1.D0
      DGB(2,2) = 1.D0

C     Case 1
      FG(1) = XA(1) - 1.D0
      FG(2) = XA(2)
      FG(3) = XB(1)
      FG(4) = XB(2)


C     Case 2
C     FG(1) = XA(1) - 1.57D0

      RETURN
C     END OF G
      END
```

A.4 Zero-net-work Solution of Two-link Revolute and Prismatic Manipulator

```
C%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C%   File name: RP2link.for   %
C%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C% Chapter 5
C Main program RP2link.for
C
C***BEGIN PROLOGUE RP2link
C***DATE WRITTEN   030501     (YYMMDD)
C***REVISION DATE  030601     (YYMMDD)
C***AUTHOR  TAO FAN
C***PURPOSE OFFLINE CALCULATION OF ZERO WORK METHOD IN CHAPTER 5.
C
C***DESCRIPTION
C     This is FORTRAN program used to solve near-minimum time control
C     problem of two-link planar R&P arm in chapter 5 of the thesis.
C     The first link is revolute, the second is prismatic.
C     The algorithm is based on the zero work
C     method presented in the thesis.
C     Subroutine "ddassl" is used to solve
C     the ordinary differential equations with initial conditions.
```

```
C***ROUTINES CALLED  DDASSL,DRES
C   ROUTINES DRES is used to define the differential equations of the system
C
C   Description of variables used in the program:
C   r1, l1, m1, m2, J are physical parameters of the manipulator
C   X0(4) initial states,  Xf(4) final states
C   XPRIME0(4) initial values for XPRIME(4)
C   Xsw0(2) initial guess for switch positions
C   Xsw(2) switch positions, Tsw(2) switch times
C   Xsw0step initial position step size
C   Xswstep position step size
C   Tstep time step
C   Nsw(2) switch indicator, it's used to indicate if the link have
C   switched, Nsw=-1 before switch, Nsw=+1 after switch
C   Nfinal(2) final position indicator, initial value=-1,
C   if link reach it's final position,
C   Nfinal=+1, control input changes to holding force/torque.
C   Tau0(2) initial values for control inputs
C   Taumax(2) maximum control inputs
C   Tau(2) control inputs
C   Work(2) work done by all forces/torques for each link
C   RCforce reaction force between two link
C   Xlast(4) store previous state values used to calculate work
C   Tfinal(2) final times for each link
C   Tstop  stop time for simulation
C   Xerror absolute position error,
C   Werror absolute work error
C   Xdeta, Ydeta displancement in X,Y direction


C***END PROLOGUE RP2link


      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      COMMON Tau(2), r1, l1, m1, m2, J
      EXTERNAL DRES
      DIMENSION X(4), XPRIME(4), DELTA(4),INFO(15),RWORK(300),IWORK(60)
      DOUBLE PRECISION  r1, l1, m1, m2, J
      DOUBLE PRECISION  X0(4), Xf(4),XPRIME0(4), Xsw0(2),Xsw(2),Tsw(2),
     *   Xswstep,Xsw0step,Tstep,Nsw(2),Nfinal(2),
     *   Taumax(2),Tau0(2), Work(2), RCforce, Xlast(4), Tfinal(2),
     *   Xerror,Werror, Xdeta, Ydeta, Tstop
      PARAMETER  (PI=3.14159D0)

C     LRW -- Set it to the declared length of the RWORK array.
C            You must have LRW .GE. 40+(MAXORD+4)*NEQ+NEQ**2
      LRW=300

C     LIW -- Set it to the declared length of the IWORK array.
C            You must have LIW .GE. 20+NEQ
      LIW=60

C     Number of differential equations
      NEQ=4

C     Physical parameters for simulation case 1,2
C
      r1=0.6D0
C     length of the first link
      l1=0.9D0
C     mass of the first link
      m1=3.D0
```

```
C     mass moment of inertia of the link1
      J=0.81D0
C     mass of the second link
      m2=1.D0

C     Maximum torque of link1
      Taumax(1)=2.D0
C     Maximum force of link2
      Taumax(2)=2.D0

C     Physical parameters for simulation case 3,4
C
C     r1=0.4D0
C     length of the first link
C     l1=1.2D0
C     mass of the first link
C     m1=4.D0
C     mass moment of inertia of the link1
C     J=1.92D0
C     mass of the second link
C     m2=1.5D0

C     Maximum torque of link1
C     Taumax(1)=3.D0
C     Maximum force of link2
C     Taumax(2)=2.5D0

C     Initial position  for case 1,3
      X0(1)=PI/4.D0
      X0(2)=0.D0
      X0(3)=0.D0
      X0(4)=0.D0
C     Initial position  for case 2,4
C      X0(1)=0.2D0
C      X0(2)=0.D0
C      X0(3)=0.3D0
C      X0(4)=0.D0

C     Final position
      Xf(1)=0.D0
      Xf(2)=0.D0
      Xf(3)=0.D0
      Xf(4)=0.D0
C
      Xerror=1.0D-4
      Werror=3.0D-4
      Tstop=1.5D0

C     Initial torque
      IF (X0(1).LT.Xerror) THEN
      Tau0(1)=0.D0
      ElSE
      Tau0(1)=-Taumax(1)
      END IF

      IF (X0(3).LT.Xerror) THEN
      Tau0(2)=-m2*r1*X0(2)**2
      ElSE
      Tau0(2)=-Taumax(2)
      END IF

C     Initial velocity and accelaration
```

```
      XPRIME0(1)=X0(2)
      XPRIME0(2)=(Tau0(1)-2.D0*m2*(X0(3)+r1)*
     &           X0(2)*X0(4))/(J+m2*(X0(3)+r1)**2)
      XPRIME0(3)=X0(4)
      XPRIME0(4)=(Tau0(2)+m2*(X0(3)+r1)*X0(2)**2)/m2

C     Initial step size
      Xsw0step=0.001D0
      Xswstep=Xsw0step
      Tstep=0.01D0

C     Initial vaules for switch positions
      Xsw0(1)=Xf(1)
      Xsw0(2)=Xf(3)

C     Start of a new integration problem
C     Initialized the switch position
    3 Xsw(1)=Xsw0(1)
      Xsw(2)=Xsw0(2)
    5 CONTINUE
C     Reset the switch indicator
      Nsw(1)=-1.0D0
      Nsw(2)=-1.0D0
      Nfinal(1)=-1.0D0
      Nfinal(2)=-1.0D0
      Tfinal(1)=0.0D0
      Tfinal(2)=0.0D0
      Tsw(1)=0.0D0
      Tsw(2)=0.0D0
C     Dimension of INFO(N)
      DO 7 I=1,15
        INFO(I)=0
    7   CONTINUE
C     Do you want the solution only at
C     TOUT (and not at the next intermediate step) ...
C     Yes - Set INFO(3) = 0
C     No - Set INFO(3) = 1 ****

C     Do you want the code to decide
C     on its own maximum stepsize?
C     Yes - Set INFO(7)=0
C     No - Set INFO(7)=1
C     and define HMAX by setting
C     RWORK(2)=HMAX ****

C     Do you want the code to define
C     its own initial stepsize?
C     Yes - Set INFO(8)=0
C     No - Set INFO(8)=1
C     and define H0 by setting
C     RWORK(3)=H0 ****

        INFO(3)=1
        INFO(7)=1
        INFO(8)=1
C     maximum stepsize
        RWORK(2)=0.0001D0
C     initial stepsize
        RWORK(3)=0.001D0

C     Relative tolerance
      RTOL=1.0D-11
```

```
C     Absolute tolerance
      ATOL=1.0D-8

C     Initialize the parameters
      T=0.D0
      DO 10 I=1,NEQ
      X(I)=X0(I)
   10 XPRIME(I)=XPRIME0(I)


C     Initial value of workdone by external force
C     of each link

      DO 20 I=1,2
   20 Work(I)=0.D0

C     Open file "data1.m' to store data of Link1
      OPEN(FILE='data1.m',UNIT=6,status='unknown')
      WRITE(6,30)
   30 FORMAT(2x,'%% This is the simulation result of Link1')
      WRITE(6,40)
   40 FORMAT(10x,'% Time',9x,'Position',5x,'Velocity',4x,'Force',
     *                                              9x,'Work')
      WRITE(6,50)
   50 FORMAT(2x,'output1=[')

C     Open file "data2.m' to store data of Link2
      OPEN(FILE='data2.m',UNIT=7,status='unknown')
      WRITE(7,60)
   60 FORMAT(2x,'%% This is the simulation result of Link2')
      WRITE(7,40)
      WRITE(7,80)
   80 FORMAT(2x,'output2=[')


C     Decide the control input Tau
   90 CONTINUE
      DO 100 I=1,2
      IF((X(2*I-1) .GE. Xsw(I))) THEN
         Tau(I)=-Taumax(I)
         Tsw(I)=T
         IF (X0(2*I-1).LT.Xerror) THEN
         Tsw(I)=0.0D0
         END IF
      ELSE
      Tau(I)=+Taumax(I)
      Nsw(I)=1.0D0
C     Nsw indicates the link has switched
      END IF
  100 CONTINUE

C     Link1 reaches final position
      IF (((X(1)-Xf(1)).LT.Xerror).OR.(Nfinal(1).GT.0.D0)) THEN
         IF (Nfinal(1).LT.0.D0) THEN
         Tfinal(1)=T
         END IF
       Tau(1)=0.D0
       Nfinal(1)=+1.0D0
      END IF

C     Link2 reaches final position
      IF (((X(3)-Xf(3)).LT.Xerror) .OR.(Nfinal(2).GT.0.D0)) THEN
```

```
            IF (Nfinal(2).LT.0.D0) THEN
            Tfinal(2)=T
            END IF
         Tau(2)=-m2*r1*X(2)**2
         Nfinal(2)=+1.0D0
        END IF

C       Both links reach final position
        IF (((X(1)-Xf(1)).LT.Xerror).AND.((X(3)-Xf(3)).LT.Xerror)) THEN
         Tau(1)=0.D0
         Tau(2)=0.D0
        END IF

C       Output the data
        WRITE(6,110) T,(X(I),I=1,2),Tau(1),Work(1)
  110   FORMAT(7x,6(F10.6,3x))
        WRITE(7,111) T,(X(I),I=3,4),Tau(2),Work(2)
  111   FORMAT(7x,6(F10.6,3x))

C       Reaction force between link1 and link2
        RCforce=m2*((X(3)+r1)*XPRIME(2)+2*X(2)*X(4))
        DO 120 I=1,4
  120   Xlast(I)=X(I)

C       Integrate one time step
        TOUT=T+Tstep
        CALL DDASSL (DRES, NEQ, T, X, XPRIME, TOUT, INFO, RTOL, ATOL,
     *     IDID, RWORK, LRW, IWORK, LIW, RPAR, IPAR,JAC)

C       Calculate work for each link
C       Workd of link1
         Work(1)=Work(1)+(X(1)-Xlast(1))*Tau(1)
     &        -(X(1)-Xlast(1))*RCforce*(r1+Xlast(3))
C
C       Displacement in x direction
        Xdeta=(X(3)+r1)*DCOS(X(1))-(Xlast(3)+r1)*DCOS(Xlast(1))
C       Displacement in y direction
        Ydeta=(X(3)+r1)*DSIN(X(1))-(Xlast(3)+r1)*DSIN(Xlast(1))
C
C       Work of link2
        Work(2)=Work(2)+Tau(2)*(DSIN(Xlast(1))*Ydeta+DCOS(Xlast(1))*Xdeta)
     &      - RCforce*DSIN(Xlast(1))*Xdeta+RCforce*DCOS(Xlast(1))*Ydeta


        IF (((ABS(Tau(1))).LT.1.0D-8).AND.((ABS(Tau(2))).LT.1.0D-8)
     *      .OR.(T.GT.Tstop)) THEN

C       Final point, check for works
            IF ((((ABS(X(1))).LT.Xerror).AND.((ABS(X(3))).LT.Xerror)).AND.
     *      (((ABS(Work(1))).LT.Werror).AND.((ABS(Work(2))).LT.Werror)))
     *       THEN
             GOTO 126
             END IF

            IF (Xsw(1).LT.X0(1)) THEN
C       Increase switch position for link 1 and restart integration
            Xsw(1)=Xsw(1)+Xswstep
            PRINT*,'Updating Xsw1=',Xsw(1)
            CLOSE(6,STATUS='DELETE')
            CLOSE(7,STATUS='DELETE')
            GOTO 5
            END IF
```

```
          IF (Xsw(2).LT.X0(3)) THEN
C      Increase switch position for link 2 and restart integration
          Xsw(1)=Xsw0(1)
          Xsw(2)=Xsw(2)+Xswstep
          PRINT*,'Updating Xsw2=',Xsw(2)
          CLOSE(6,STATUS='DELETE')
          CLOSE(7,STATUS='DELETE')
          GOTO 5
          ELSE
C      Reduce step size by half
          Xswstep=Xswstep/2.0D0
          PRINT*,'Updating Xswstep=',Xswstep
          CLOSE(6,STATUS='DELETE')
          CLOSE(7,STATUS='DELETE')
          GOTO 3
          END IF

      ELSE
C      Continue integration
      GOTO 90
      END IF


 126  WRITE(6,110) T,(X(I),I=1,2),Tau(1),Work(1)
      WRITE(7,111) T,(X(I),I=3,4),Tau(2),Work(2)
      WRITE(6,130)
      WRITE(7,130)
 130  FORMAT(2x,']; ')
      WRITE(6,140)Xsw0(1)
 140  FORMAT(3x,'% The initial guess of Xsw0(1)=',F10.6)
      WRITE(6,150)Xsw(1)
 150  FORMAT(3x,'% The switch position of link1 Xsw(1)=',F10.6)
      WRITE(6,170)Tsw(1)
 170  FORMAT(3x,'% The switch time of link1 Tsw(1)=',F10.6)
      WRITE(6,190)Tfinal(1)
 190  FORMAT(3x,'% The final time of link1 Tfinal(1)=',F10.6)

       WRITE(6,240)Xsw0(2)
 240  FORMAT(3x,'% The initial guess of Xsw0(2)=',F10.6)
      WRITE(6,250)Xsw(2)
 250  FORMAT(3x,'% The switch position of link2 Xsw(2)=',F10.6)
      WRITE(6,270)Tsw(2)
 270  FORMAT(3x,'% The switch time of link2 Tsw(2)=',F10.6)
      WRITE(6,290)Tfinal(2)
 290  FORMAT(3x,'% The final time of link2 Tfinal(2)=',F10.6)

      WRITE(6,300) Xsw0step,Xswstep
 300  FORMAT(1x,'% Xsw0step=',F9.5,3x,'Xswstep=',F10.6)


      END


C     &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
      SUBROUTINE DRES(T,X,XPRIME,DELTA,IRES,RPAR,IPAR)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DIMENSION X(4), XPRIME(4), DELTA(4)
      DOUBLE PRECISION  r1, l1, m1, m2, J
      COMMON Tau(2), r1, l1, m1, m2, J

      DELTA(1)=XPRIME(1)-X(2)
      DELTA(3)=XPRIME(3)-X(4)
      DELTA(2)=(J+m2*(X(3)+r1)**2.0D0)*XPRIME(2)+
     &         2.0D0*m2*(X(3)+r1)*X(2)*X(4)-Tau(1)
      DELTA(4)=m2*XPRIME(4)-m2*(X(3)+r1)*X(2)**2.D0-Tau(2)
      RETURN
      END
```

A.5 Exact Minimum Time Solution of Two-link Revolute and Prismatic Manipulator

```
C%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C%   File name: RP2linkMT.for  %
C%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C% Chapter 5
C Main program RP2linkMT.for
C
C***BEGIN PROLOGUE RP2linkMT.for
C***DATE WRITTEN   030501     (YYMMDD)
C***REVISION DATE  030601     (YYMMDD)
C***AUTHOR  TAO FAN
C***PURPOSE SOLVING TPBVP PROBLEM IN CHAPTER 5.
C***DESCRIPTION
C     This is FORTRAN program used to solve exact minimum time control
C     problem of two-link planar R&P arm in chapter 5 of the thesis.
C     The first link is revolute, the second is prismatic.
C     Subroutine "MUSN" is used to solve
C     the nonlinear TPBVP.
C***REFERENCES  (NONE)
C***ROUTINES CALLED  MUSN, FDIF, XOT
C   ROUTINES FDIF is used to define the differential equations of the system
C   ROUTINES XOT is used to evaluate the initial approximation
C   X0(t) of the solution.
C   Description of variables used in the program:
C   Taumax is maximum control inputs
C   Tau is control inputs

C***END PROLOGUE RP2linkMT.for


C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION ER(5),TI(12),X(5,12),Q(5,5,12),U(15,12),D(5,12),
     1          PHIREC(15,12),W(900),WGR(20)
      INTEGER IW(60)
      EXTERNAL FDIF,XOT,G
C
C
C     N is the order of the system.
C
C     NU is one of the dimension of U and PHI.
C        NU must be greater than or equal to N * (N+1) / 2.
C
C     NTI is one of the dimension of  TI, X, S, Q, U en PHI.
C        NTI must be greater than or equal to the total number of
C        necessary output points + 1  (i.e. if the entry value for
C        NRTI > 1, NTI may be equal to the entry value of NRTI + 1)
C
C     LW is the dimension of W. LW >= 7*N + 3*N*NTI + 4*N*N
C
C     LIW is the dimension of IW. LIW >= 3*N + NTI
```

```
C
C     ER(3) must contain the machine precision
C
C     LWG is the dimension of WGR. LWG >= (total number of grid points)
C        / 5. The minimum number of grid points between 2 succesive output
C        points is 5, so the minimum value for LWG is the number of actually
C        used output points. Initially a crude estimate for LWG has to be made
C
C     ER(1) must contain the required tolerance for solving the differential
C        equation.
C     ER(2) must contain the initial tolerance with which a first aproximate
C         solution will be computed. This approximation is then used as an
C         initial approximation for the computation of an solution with an
C         tolerance ER(2)*ER(2) and so on until the required tolerance is
C         reached. As an initial tolerance max(ER(1),min(ER(2),1.d-2)) will
C         be used
C
C     A,B the two boundary points
C
C     NRTI is used to specify the output points. There are 3 ways to
C          specify the output points:
C          1) NRTI = 0, the output points are determined automatically using AMP.
C          2) NRTI = 1, the output points are supplied by the user in the array TI.
C          3) NRTI > 1, the subroutine computes the (NRTI+1) output points TI(k) by
C                            TI(k) = A + (k-1) * (B - A) / NRTI ;.
C                      so TI(1) = A and TI(NRTI+1) = B.
C          Depending on the allowed increment between two succesive output points,
C          more output points may be inserted in cases 2 and 3.
C          On exit NRTI contains the total number of output points.

C     AMP must contain the allowed increment between two output
C          points. AMP is used to determine output points and to assure that the
C          increment between two output points is at most AMP*AMP. A small value
C          for AMP may result in a large number of output points.
C          Unless 1 < AMP < .25 * sqrt(ER(1)/ER(3)) the default value
C          .25 * sqrt(ER(1)/ER(3)) is used.

C     ITLIM maximum number of allowed iteration
C     IERROR=1, diagnostics will be printed during computation

      N = 9
      NU = 45
      NTI = 12
      LW = 900
      LIW = 60
      ER(3) = 1.1D-15
      LWG = 20
      ER(1) = 1.D-6
      ER(2) = 1.D-2
      A = 0.D0
      B = 1.D0
      NRTI=10
      AMP=0.01D0
      ITLIM=20
      IERROR=1
      OPEN(FILE='data1MT.m',UNIT=6,status='unknown')
      WRITE(6,30)
   30 FORMAT(2X,'% This is the results of TPBVP')
      CALL MUSN(FDIF,XOT,G,N,A,B,ER,TI,NTI,NRTI,AMP,ITLIM,X,Q,U,NU,D,
     1          PHIREC,KPART,W,LW,IW,LIW,WGR,LWG,IERROR)
      WRITE(6,*) ' MUSN: IERROR =',IERROR
      WRITE(6,200) A,B,ER(1),ER(2),ER(4),ER(5),KPART
```

```
  200 FORMAT(' A = ',F8.4,3X,'B = ',F8.4,/,' REQUIRED TOLERANCE = ',1P,
     1 D12.5,3X,'START TOLERANCE  = ',D12.5,/,
     2 ' CONDITION NUMBER = ',D12.5,3X,
     3 'AMPLIFICATION FACTOR = ',D12.5,/,' K-PARTITIONING =',I2,/)
      IF (IERROR.NE.0) GOTO 3000
      WRITE(6,215)
  215 FORMAT(' I ',4X,'T',9X,'X1',12X,'X2',12X,'X3',12X,'X4',12X,'X5',
     1 /)
      DO 2200 K = 1 , NRTI
        WRITE(6,220) K,TI(K),(X(J,K),J=1,N)
 2200 CONTINUE
  220 FORMAT(' ',I2,1X,F6.4,1P,5(2X,D12.5))
 3000 CONTINUE
      STOP
      END


      SUBROUTINE FDIF(T,Y,F)
C     ----------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION Y(9),F(9)
      DOUBLE PRECISION r1,l1,m1,m2,I,Taumax(2),Tau(2),
     * X1,X2,X3,X4,P1,P2,P3,P4,Z,V1,V2,V12,V13,V14,V22,V23

      X1=Y(1)
      X2=Y(2)
      X3=Y(3)
      X4=Y(4)
      P1=Y(5)
      P2=Y(6)
      P3=Y(7)
      P4=Y(8)
      Z=Y(9)

C     Physical parameters for simulation case 1,2
C
      r1=0.6D0
C     length of the first link
      l1=0.9D0
C     mass of the first link
      m1=3.D0
C     mass moment of inertia of the link1
      I=0.81D0
C     mass of the second link
      m2=1.D0

C     Maximum torque of link1
      Taumax(1)=2.D0
C     Maximum force of link2
      Taumax(2)=2.D0

C     Physical parameters for simulation case 3,4
C
C     r1=0.4D0
C     length of the first link
C     l1=1.2D0
C     mass of the first link
C     m1=4.D0
C     mass moment of inertia of the link1
C     I=1.92D0
C     mass of the second link
```

```
C     m2=1.5D0

C     Maximum torque of link1
C     Taumax(1)=3.D0
C     Maximum force of link2
C     Taumax(2)=2.5D0

C     Decided the control input
      IF(P2.LT.0.0D0) THEN
        Tau(1)=Taumax(1)
      ELSE
        Tau(1)=-Taumax(1)
      END IF

      IF(P4.LT.0.0D0) THEN
        Tau(2)=Taumax(2)
      ELSE
        Tau(2)=-Taumax(2)
      END IF

      U = Tau(1)-2.0D0*m2*(X3+r1)*X2*X4
      V = I+m2*(X3+r1)**2
      V1 = U/V
      V12 = (-2.0D0*m2*(X3+r1)*X4)/V
      V13 = (-2.0D0*m2*X2*X4*V-2.0D0*U*m2*(X3+r1))/V**2.0D0
      V14 = -2.0D0*m2*X2*(X3+r1)/V
      V2 = (Tau(2)+m2*(X3+r1)*X2**2.0D0)/m2
      V22 = 2.0D0*X2*(X3+r1)
      V23 = X2**2

      F(1) = Z*X2
      F(2) = Z*V1
      F(3) = Z*X4
      F(4) = Z*V2
      F(5) = 0.0D0
      F(6) = -Z*(P1+P2*V12+P4*V22)
      F(7) = -Z*(P2*V13+P4*V23)
      F(8) = -Z*(P3+P2*V14)
      F(9) = 0.0D0

      RETURN
C     END OF FDIF
      END


      SUBROUTINE XOT(T,X)
C     -------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION X(5)
C
C     Case 1,3
      X(1) = 3.14D0/4.0D0
      X(2) = 0.0D0
      X(3) = 0.0D0
      X(4) = 0.0D0
      X(5) = 1.0D0
      X(6) = 1.0D0
      X(7) = 1.0D0
      X(8) = 1.0D0
      X(9) = 1.0D0
```

```
C     Case 2,4
      X(1) = 0.2D0
      X(2) = 0.0D0
      X(3) = 0.3D0
      X(4) = 0.0D0
      X(5) = 1.0D0
      X(6) = 1.0D0
      X(7) = 1.0D0
      X(8) = 1.0D0
      X(9) = 1.0D0

      RETURN
C     END OF XOT
      END


      SUBROUTINE G(N,XA,XB,FG,DGA,DGB)
C     --------------------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION XA(N),XB(N),FG(N),DGA(N,N),DGB(N,N)
C
      DO 1100 I = 1 , N
      DO 1100 J = 1 , N
        DGA(I,J) = 0.D0
        DGB(I,J) = 0.D0
 1100 CONTINUE
      DGA(1,1) = 1.D0
      DGA(2,2) = 1.D0
      DGA(3,3) = 1.D0
      DGA(4,4) = 1.D0
      DGB(1,1) = 1.D0
      DGB(2,2) = 1.D0
      DGB(3,3) = 1.D0
      DGB(4,4) = 1.D0

C     Case 1,3
      FG(1) = XA(1) - 3.14D0/4.0D0
      FG(2) = XA(2)
      FG(3) = XA(3)
      FG(4) = XA(4)
      FG(5) = XB(1)
      FG(6) = XB(2)
      FG(7) = XB(3)
      FG(8) = XB(4)


C     Case 2,4
C      FG(1) = XA(1) - 0.2D0
C      FG(2) = XA(2)
C      FG(3) = XA(3)- 0.3D0
C      FG(4) = XA(4)
C      FG(5) = XB(1)
C      FG(6) = XB(2)
C      FG(7) = XB(3)
C      FG(8) = XB(4)

      RETURN
C     END OF G
      END
```

A.6 S-function of Two-link Revolute and Prismatic Manipulator

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   File name: RPsimfun1.m    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Chapter 5
% This is the s-function of two link planar revolute and
% prismatic manipulator.
% The first link is revolute, the second is prismatic

function  [sys,x0,str,ts] = robot2(t,x,u,flag,x01,x02,x03,x04)

switch flag
  case 0                                        % Initialization
    sys = [4,       % number of continuous states
           0,       % number of discrete states
           4,       % number of outputs
           2,       % number of inputs
           0,       % reserved must be zero
           0,       % direct feedthrough flag
           1];      % number of sample times
    x0(1)=x01;
    x0(2)=x02;
    x0(3)=x03;
    x0(4)=x04;
    str = [];
    ts  = [0 0];    % sample time: [period, offset]

  case 1                                         % Derivatives

 % Physical parameters for simulation case 1,2
 r1=0.6;
 l1=0.9; % length of the first link
 m1=3;   % mass of the first link
 m2=1;   % mass of the second link
 I=1/3*m1*l1^2; % mass moment of inertia of the link1

% Physical parameters for simulation case 3,4
%r1=0.4;
%l1=1.2; % length of the first link
%m1=4;   % mass of the first link
%m2=1.5;   % mass of the second link
%I=1/3*m1*l1^2; % mass moment of inertia of the link1

 a=[x(2);
 (u(1)-2.*m2.*(x(3)+r1).*x(2).*x(4))/(I+m2.*(x(3)+r1).^2);
 x(4);
 (u(2)+m2.*(x(3)+r1).*x(2).^2)/m2 ;];

sys=a;
  case 2                                        % Discrete state update
    sys = []; % do nothing

  case 3
    sys = x;

  case 9                                        % Terminate
    sys = []; % do nothing

  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end
```

A.7 Zero-net-work Solution of Two-link Revolute Manipulator

```
C%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C%   File name: TLRPM.for     %
C%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C% Chapter 6
C Main program TLRPM.for
C
C***BEGIN PROLOGUE TLRPM.for
C***DATE WRITTEN   030501     (YYMMDD)
C***REVISION DATE  030601     (YYMMDD)
C***AUTHOR  TAO FAN
C***PURPOSE OFFLINE CALCULATION OF ZERO WORK METHOD IN CHAPTER 6 .
C***DESCRIPTION
C     This is FORTRAN program used to solve near-minimum time control
C     problem of two-link planar revolute arm in chapter 6 of the thesis.
C     The algorithm is based on the zero work
C     method presented in the thesis.
C     Subroutine "ddassl" is used to solve
C     the ordinary differential equations with initial conditions.
C***REFERENCES  (NONE)
C***ROUTINES CALLED  DDASSL,DRES
C   ROUTINES DRES is used to define the differential equations of the system
C
C   Description of variables used in the program:
C   X0(4) initial states,  Xf(4) final states
C   XPRIME0(4) initial values for XPRIME(4)
C   Xsw0(2) initial guess for switch positions
C   Xsw(2) switch positions, Tsw(2) switch times
C   Xsw0step initial position step size
C   Xswstep position step size
C   Tstep time step
C   Nsw(2) switch indicator, it's used to indicate if the link have
C   switched, Nsw=-1 before switch, Nsw=+1 after switch
C   Nfinal(2) final position indicator, initial value=-1,
C   if link reach it's final position,
C   Nfinal=+1, control input changes to holding force/torque.
C   Tau0(2) initial values for control inputs
C   Taumax(2) maximum control inputs
C   Tau(2) control inputs
C   Work(2) work done by all forces/torques for each link
C   RCforce reaction force between two link
C   Xlast(4) store previous state values used to calculate work
C   Tfinal(2) final times for each link
C   Tstop  stop time for simulation
C   Xerror absolute position error,
C   Werror absolute work error
C   Xdeta, Ydeta displancement in X,Y direction


C***END PROLOGUE TLRPM


      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      COMMON Tau(2),l1, lc1, l2,lc2,m1,m2,I1,I2,M,c1,c2,c3
      DOUBLE PRECISION  l1, lc1, l2,lc2,m1,m2,I1,I2,M,c1,c2,c3
      EXTERNAL DRES
      DIMENSION X(4), XPRIME(4), DELTA(4),INFO(15),RWORK(300),IWORK(60)
      DOUBLE PRECISION  X0(4), Xf(4),XPRIME0(4), Xsw0(2),Xsw(2),Tsw(2),
     *   Xswstep,Xsw0step,Tstep,Nsw(2),Nfinal(2),
     *   TaumaX(2),Tau0(2), Work(2), RCforce, Xlast(4), Tfinal(2),
     *   Xerror,Werror, Xdeta, Ydeta, Tstop
      PARAMETER  (PI=3.14159D0)

C     LRW -- Set it to the declared length of the RWORK array.
C            You must have LRW .GE. 40+(MAXORD+4)*NEQ+NEQ**2
      LRW=300

C     LIW -- Set it to the declared length of the IWORK array.
C            You must have LIW .GE. 20+NEQ
      LIW=60

C     Number of differential equations
      NEQ=4


C      Physical parameters for Case 1 and Case 2

C      length of the  link 1
      l1=0.4D0
      lc1=l1/2.0D0
C      length of the  link 2
      l2=0.25D0
      lc2=l2/2.0D0
C      mass of the first link
      m1=29.58D0
C      mass of the second link
      m2=15.00D0
C      mass moment of inertia of the link1
      I1=0.416739D0
C      mass moment of inertia of the link2
      I2=0.205625D0
C      mass of end effector and load
      M=6.0D0
C     maximum torque for link 1
      Taumax(1)=25.0D0
C     maximum torque for link 2
      Taumax(2)=9.0D0


C     Physical parameters for Case 3

C      length of the  link 1
C      l1=0.4D0
C      lc1=l1/2.0D0
C      length of the  link 2
C      l2=0.25D0
C      lc2=l2/2.0D0
C      mass of the first link
C      m1=0.245D0
C      mass of the second link
C      m2=0.15315D0
C      mass moment of inertia of the link1
C      I1=3.2688D-3
C      mass moment of inertia of the link2
C      I2=0.7986D-3
C      mass of end effector and load
C      M=0.5D0
C     maximum torque for link 1
C      Taumax(1)=0.25D0
C     maximum torque for link 2
```

```
C       Taumax(2)=0.1D0


C      Physical parameters for Case 4

C       length of the  link 1
C       l1=1.0D0
C       lc1=l1/2.0D0
C       length of the  link 2
C       l2=0.625D0
C       lc2=l2/2.0D0
C       mass of the first link
C       m1=0.61261D0
C       mass of the second link
C       m2=0.38288D0
C       mass moment of inertia of the link1
C       I1=51.0547D-3
C       mass moment of inertia of the link2
C       I2=12.4660D-3
C       mass of end effector and load
C       M=1.25D0
C      maximum torque for link 1
C       Taumax(1)=3.90D0
C      maximum torque for link 2
C       Taumax(2)=1.56D0

       c1=I1+m1*lc1**2.0D0+(m2+M)*l1**2.0D0
       c2=I2+m2*lc2**2.0D0+M*l2**2.0D0
       c3=m2*l1*lc2+M*l1*l2

C      Initial position  for case 1
       X0(1)=0.975D0
       X0(2)=0.D0
       X0(3)=0.D0
       X0(4)=0.D0

C      Initial position  for case 2
C       X0(1)=0.76D0
C       X0(2)=0.D0
C       X0(3)=0.2618D0
C       X0(4)=0.D0

C      Initial position  for case 3
C       X0(1)=0.376D0
C       X0(2)=0.D0
C       X0(3)=0.D0
C       X0(4)=0.D0

C      Initial position  for case 4
C       X0(1)=0.8D0
C       X0(2)=0.D0
C       X0(3)=0.2D0
C       X0(4)=0.D0


C      Final position
       Xf(1)=0.D0
       Xf(2)=0.D0
       Xf(3)=0.D0
       Xf(4)=0.D0
C
       Xerror=1.0D-4
```

```
      Werror=1.0D-4
      Tstop=2.0D0

C     Initial torque
      Tau0(1)=-Taumax(1)
      Tau0(2)=-Taumax(2)

      IF (X0(1).LT.Xerror) THEN
      Tau0(1)=(c2+c3*DCOS(X0(3)))*Tau0(2)/c2-c3*DSIN(X0(3))*X0(4)**2.0D0
      END IF

      IF (X0(3).LT.Xerror) THEN
      Tau0(2)=(c2+c3)*Tau0(1)/(c1+c2+2*c3)
      END IF

      IF ((X0(1).LT.Xerror).AND.(X0(3).LT.Xerror)) THEN
      Tau0(1)=0.0D0
      Tau0(2)=0.0D0
      END IF

C     Initial velocity and accelaration
      XPRIME0(1)=X0(2)

      XPRIME0(2)=(c2*(Tau0(1)-Tau0(2)+c3*(X0(2)+
     *  X0(4))**2.0D0*DSIN(X0(3)))
     * -c3*(Tau0(2)-c3*(X0(2)**2.0D0)*DSIN(X0(3)))*DCOS(X0(3)))
     * /(c1*c2-c3**2.0D0*(DCOS(X0(3)))**2.0D0)

      XPRIME0(3)=X0(4)

      XPRIME0(4)=((c1+c3*DCOS(X0(3)))*
     * (Tau0(2)-c3*X0(2)**2.0D0*DSIN(X0(3)))
     * -(c2+c3*DCOS(X0(3)))*(Tau0(1)-Tau0(2)
     * +c3*(X0(2)+X0(4))**2.0D0*DSIN(X0(3))))
     * /(c1*c2-c3**2.0D0*(DCOS(X0(3)))**2.0D0)

C     Initial step size
      Xsw0step=0.01D0
      Xswstep=Xsw0step
      Tstep=0.001D0

C     Initial guess for switch positions
      Xsw0(1)=Xf(1)
      Xsw0(2)=Xf(3)

C     Start of a new integration problem
C     Initialized the switch position
    3 Xsw(1)=Xsw0(1)
      Xsw(2)=Xsw0(2)
    5 CONTINUE
C     Reset the switch indicator
      Nsw(1)=-1.0D0
      Nsw(2)=-1.0D0
      Nfinal(1)=-1.0D0
      Nfinal(2)=-1.0D0
      Tfinal(1)=0.0D0
      Tfinal(2)=0.0D0
      Tsw(1)=0.0D0
      Tsw(2)=0.0D0
C     Dimension of INFO(N)
      DO 7 I=1,15
        INFO(I)=0
```

```
   7    CONTINUE
C     Do you want the solution only at
C     TOUT (and not at the next intermediate step) ...
C     Yes - Set INFO(3) = 0
C     No - Set INFO(3) = 1 ****

C     Do you want the code to decide
C     on its own maximum stepsize?
C     Yes - Set INFO(7)=0
C     No - Set INFO(7)=1
C     and define HMAX by setting
C     RWORK(2)=HMAX ****

C     Do you want the code to define
C     its own initial stepsize?
C     Yes - Set INFO(8)=0
C     No - Set INFO(8)=1
C     and define H0 by setting
C     RWORK(3)=H0 ****

        INFO(3)=1
        INFO(7)=1
        INFO(8)=1
C     maximum stepsize
        RWORK(2)=0.0001D0
C     initial stepsize
        RWORK(3)=0.001D0

C     Relative tolerance
      RTOL=1.0D-11
C     Absolute tolerance
      ATOL=1.0D-8

C     Initialize the parameters
      T=0.D0
      DO 10 I=1,NEQ
      X(I)=X0(I)
  10  XPRIME(I)=XPRIME0(I)


C     Initial value of workdone by external force
C     of each link

      DO 20 I=1,2
  20  Work(I)=0.D0

C     Open file "data1.m' to store data of Link1
      OPEN(FILE='data1.m',UNIT=6,status='unknown')
      WRITE(6,30)
  30  FORMAT(2X,'%% This is the simulation result of Link1')
      WRITE(6,40)
  40  FORMAT(10X,'% Time',9X,'Position',5X,'Velocity',4X,'Force',
     *                                               9X,'Work')
      WRITE(6,50)
  50  FORMAT(2X,'output1=[')

C     Open file "data2.m' to store data of Link2
      OPEN(FILE='data2.m',UNIT=7,status='unknown')
      WRITE(7,60)
  60  FORMAT(2X,'%% This is the simulation result of Link2')
      WRITE(7,40)
      WRITE(7,80)
  80  FORMAT(2X,'output2=[')


C     Decide the control input Tau
  90  CONTINUE
      DO 100 I=1,2
      IF((X(2*I-1) .GE. Xsw(I))) THEN
         Tau(I)=-Taumax(I)
         Tsw(I)=T
         IF (X0(2*I-1).LT.Xerror) THEN
         Tsw(I)=0.0D0
         END IF
      ELSE
      Tau(I)=+Taumax(I)
      Nsw(I)=1.0D0
C     Nsw indicates the link has switched
      END IF
 100  CONTINUE

C     Link1 reaches final position
      IF (((X(1)-Xf(1)).LT.Xerror).OR.(Nfinal(1).GT.0.D0)) THEN
         IF (Nfinal(1).LT.0.D0) THEN
         Tfinal(1)=T
         END IF
       Tau(1)=(c2+c3*DCOS(X(3)))*Tau(2)/c2-c3*DSIN(X(3))*X(4)**2.0D0
       Nfinal(1)=+1.0D0
      END IF

C     Link2 reaches final position
      IF (((X(3)-Xf(3)).LT.Xerror) .OR.(Nfinal(2).GT.0.D0)) THEN
         IF (Nfinal(2).LT.0.D0) THEN
         Tfinal(2)=T
         END IF
       Tau(2)=(c2+c3)*Tau(1)/(c1+c2+2.0D0*c3)
       Nfinal(2)=+1.0D0
      END IF

C     Both links reach final position
      IF (((X(1)-Xf(1)).LT.Xerror).AND.((X(3)-Xf(3)).LT.Xerror)) THEN
       Tau(1)=0.D0
       Tau(2)=0.D0
      END IF

C     Output the data
      WRITE(6,110) T,(X(I),I=1,2),Tau(1),Work(1)
 110  FORMAT(7X,6(F10.6,3X))
      WRITE(7,111) T,(X(I),I=3,4),Tau(2),Work(2)
 111  FORMAT(7X,6(F10.6,3X))

      DO 120 I=1,4
 120  Xlast(I)=X(I)

C     Integrate one time step
      TOUT=T+Tstep
      CALL DDASSL (DRES, NEQ, T, X, XPRIME, TOUT, INFO, RTOL, ATOL,
     *   IDID, RWORK, LRW, IWORK, LIW, RPAR, IPAR,JAC)

C     Calculate work for each link

C     Work of link1

       Work(1)=1.0D0/2.0D0*(m1*lc1**2.0D0+I1)*X(2)**2.0D0
C     Work of link2
      Work(2)=1.0D0/2.0D0*(I2+m2*lc2**2.0D0)*(X(2)+X(4))**2.0D0
     * +1.0D0/2.0D0*m2*l1**2.0D0*(X(2))**2.0D0
     * +DCOS(X(3))*X(2)*(X(2)+X(4))*l1*lc2*m2


      IF (((ABS(Tau(1))).LT.1.0D-8).AND.((ABS(Tau(2))).LT.1.0D-8)
     *    .OR.(T.GT.Tstop)) THEN

C     Final point, check for works
          IF ((((ABS(X(1))).LT.Xerror).AND.((ABS(X(3))).LT.Xerror)).AND.
     *    (((ABS(Work(1))).LT.Werror).AND.((ABS(Work(2))).LT.Werror)))
     *     THEN
           GOTO 126
           END IF

          IF (Xsw(1).LT.X0(1)) THEN
C     Increase switch position for link 1 and restart integration
          Xsw(1)=Xsw(1)+Xswstep
          PRINT*,'Updating Xsw1=',Xsw(1)
          CLOSE(6,STATUS='DELETE')
          CLOSE(7,STATUS='DELETE')
          GOTO 5
          END IF

          IF (Xsw(2).LT.X0(3)) THEN
C     Increase switch position for link 2 and restart integration
          Xsw(1)=Xsw0(1)
          Xsw(2)=Xsw(2)+Xswstep
          PRINT*,'Updating Xsw2=',Xsw(2)
          CLOSE(6,STATUS='DELETE')
          CLOSE(7,STATUS='DELETE')
          GOTO 5
          ELSE
C     Reduce step size by half
          Xswstep=Xswstep/2.0D0
          PRINT*,'Updating Xswstep=',Xswstep
          CLOSE(6,STATUS='DELETE')
          CLOSE(7,STATUS='DELETE')
          GOTO 3
          END IF

      ELSE
C     Continue integration
      GOTO 90
      END IF

 126  WRITE(6,110) T,(X(I),I=1,2),Tau(1),Work(1)
      WRITE(7,111) T,(X(I),I=3,4),Tau(2),Work(2)
      WRITE(6,130)
      WRITE(7,130)
 130  FORMAT(2X,']; ')
      WRITE(6,140)Xsw0(1)
 140  FORMAT(3X,'% The initial guess of Xsw0(1)=',F10.6)
      WRITE(6,150)Xsw(1)
 150  FORMAT(3X,'% The switch position of link1 Xsw(1)=',F10.6)
      WRITE(6,170)Tsw(1)
 170  FORMAT(3X,'% The switch time of link1 Tsw(1)=',F10.6)
      WRITE(6,190)Tfinal(1)
 190  FORMAT(3X,'% The final time of link1 Tfinal(1)=',F10.6)
```

```
      WRITE(6,240)Xsw0(2)
 240  FORMAT(3X,'% The initial guess of Xsw0(2)=',F10.6)
      WRITE(6,250)Xsw(2)
 250  FORMAT(3X,'% The switch position of link2 Xsw(2)=',F10.6)
      WRITE(6,270)Tsw(2)
 270  FORMAT(3X,'% The switch time of link2 Tsw(2)=',F10.6)
      WRITE(6,290)Tfinal(2)
 290  FORMAT(3X,'% The final time of link2 Tfinal(2)=',F10.6)

      WRITE(6,300) Xsw0step,Xswstep
 300  FORMAT(1X,'% Xsw0step=',F9.5,3X,'Xswstep=',F10.6)


      END


C     &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
      SUBROUTINE DRES(T,X,XPRIME,DELTA,IRES,RPAR,IPAR)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DIMENSION X(4), XPRIME(4), DELTA(4)
      COMMON Tau(2),l1, lc1, l2,lc2,m1,m2,I1,I2,M,c1,c2,c3
      DOUBLE PRECISION  l1, lc1, l2,lc2,m1,m2,I1,I2,M,c1,c2,c3

      DELTA(1)=XPRIME(1)-X(2)
      DELTA(3)=XPRIME(3)-X(4)
      DELTA(2)=XPRIME(2)-(c2*(Tau(1)-Tau(2)+c3*(X(2)+
     *  X(4))**2.0D0*DSIN(X(3)))
     * -c3*(Tau(2)-c3*(X(2)**2.0D0)*DSIN(X(3)))*DCOS(X(3)))
     * /(c1*c2-c3**2.0D0*(DCOS(X(3)))**2.0D0)

      DELTA(4)=XPRIME(4)-((c1+c3*DCOS(X(3)))*
     * (Tau(2)-c3*X(2)**2.0D0*DSIN(X(3)))
     * -(c2+c3*DCOS(X(3)))*(Tau(1)-Tau(2)
     * +c3*(X(2)+X(4))**2.0D0*DSIN(X(3))))
     * /(c1*c2-c3**2.0D0*(DCOS(X(3)))**2.0D0)
      RETURN
      END
```

A.8 Exact Minimum Time Solution of Two-link Revolute Manipulator

```
C%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C%   File name: TLRPM_MT.for   %
C%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C% Chapter 6
C Main program TLRPM_MT.for
C
C***BEGIN PROLOGUE TLRPM_MT.for
C***DATE WRITTEN   030501     (YYMMDD)
C***REVISION DATE  030601     (YYMMDD)
C***AUTHOR  TAO FAN
C***PURPOSE SOLVING TPBVP PROBLEM IN CHAPTER 6.
C***DESCRIPTION
C     This is FORTRAN program used to solve exact minimum time control
C     problem of two-link planar  revolute arm in chapter 6 of the thesis.
C     Subroutine "MUSN" is used to solve
C     the nonlinear TPBVP.
C***REFERENCES  (NONE)
C***ROUTINES CALLED  MUSN, FDIF, XOT
C   ROUTINES FDIF is used to define the differential equations of the system
C   ROUTINES XOT is used to evaluate the initial approximation
```

```
C   XO(t) of the solution.
C   Description of variables used in the program:
C   Taumax is maximum control inputs
C   Tau is control inputs

C***END PROLOGUE TLRPM_MT.for

C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION ER(5),TI(12),X(5,12),Q(5,5,12),U(15,12),D(5,12),
     1          PHIREC(15,12),W(900),WGR(20)
      INTEGER IW(60)
      EXTERNAL FDIF,XOT,G
C
C
C     N is the order of the system.
C
C     NU is one of the dimension of U and PHI.
C        NU must be greater than or equal to N * (N+1) / 2.
C
C     NTI is one of the dimension of  TI, X, S, Q, U en PHI.
C        NTI must be greater than or equal to the total number of
C        necessary output points + 1  (i.e. if the entry value for
C        NRTI > 1, NTI may be equal to the entry value of NRTI + 1)
C
C     LW is the dimension of W. LW >= 7*N + 3*N*NTI + 4*N*N
C
C     LIW is the dimension of IW. LIW >= 3*N + NTI
C
C     ER(3) must contain the machine precision
C
C     LWG is the dimension of WGR. LWG >= (total number of grid points)
C        / 5. The minimum number of grid points between 2 succesive output
C        points is 5, so the minimum value for LWG is the number of actually
C        used output points. Initially a crude estimate for LWG has to be made
C
C     ER(1) must contain the required tolerance for solving the differential
C        equation.
C     ER(2) must contain the initial tolerance with which a first aproximate
C         solution will be computed. This approximation is then used as an
C         initial approximation for the computation of an solution with an
C         tolerance ER(2)*ER(2) and so on until the required tolerance is
C         reached. As an initial tolerance max(ER(1),min(ER(2),1.d-2)) will
C         be used
C
C     A,B the two boundary points
C
C     NRTI is used to specify the output points. There are 3 ways to
C          specify the output points:
C          1) NRTI = 0, the output points are determined automatically using AMP.
C          2) NRTI = 1, the output points are supplied by the user in the array TI.
C          3) NRTI > 1, the subroutine computes the (NRTI+1) output points TI(k) by
C                           TI(k) = A + (k-1) * (B - A) / NRTI ;
C                     so TI(1) = A and TI(NRTI+1) = B.
C          Depending on the allowed increment between two succesive output points,
C          more output points may be inserted in cases 2 and 3.
C          On exit NRTI contains the total number of output points.

C     AMP must contain the allowed increment between two output
C          points. AMP is used to determine output points and to assure that the
C          increment between two output points is at most AMP*AMP. A small value
C          for AMP may result in a large number of output points.
```

```
C          Unless 1 < AMP < .25 * sqrt(ER(1)/ER(3)) the default value
C          .25 * sqrt(ER(1)/ER(3)) is used.

C     ITLIM maximum number of allowed iteration
C     IERROR=1, diagnostics will be printed during computation

      N = 9
      NU = 45
      NTI = 12
      LW = 900
      LIW = 60
      ER(3) = 1.1D-15
      LWG = 20
      ER(1) = 1.D-6
      ER(2) = 1.D-2
      A = 0.D0
      B = 1.D0
      NRTI=10
      AMP=100.D0
      ITLIM=20
      IERROR=1
      OPEN(FILE='data1MT.m',UNIT=6,status='unknown')
      WRITE(6,30)
   30 FORMAT(2X,'% This is the results of TPBVP')
      CALL MUSN(FDIF,XOT,G,N,A,B,ER,TI,NTI,NRTI,AMP,ITLIM,X,Q,U,NU,D,
     1          PHIREC,KPART,W,LW,IW,LIW,WGR,LWG,IERROR)
      WRITE(6,*) ' MUSN: IERROR =',IERROR
      WRITE(6,200) A,B,ER(1),ER(2),ER(4),ER(5),KPART
  200 FORMAT(' A = ',F8.4,3X,'B = ',F8.4,/,' REQUIRED TOLERANCE = ',1P,
     1 D12.5,3X,'START TOLERANCE  = ',D12.5,/,
     2 ' CONDITION NUMBER = ',D12.5,3X,
     3 'AMPLIFICATION FACTOR = ',D12.5,/,' K-PARTITIONING =',I2,/)
      IF (IERROR.NE.0) GOTO 3000
      WRITE(6,215)
  215 FORMAT(' I ',4X,'T',9X,'X1',12X,'X2',12X,'X3',12X,'X4',12X,'X5',
     1 /)
      DO 2200 K = 1 , NRTI
        WRITE(6,220) K,TI(K),(X(J,K),J=1,N)
 2200 CONTINUE
  220 FORMAT(' ',I2,1X,F6.4,1P,5(2X,D12.5))
 3000 CONTINUE
      STOP
      END


      SUBROUTINE FDIF(T,Y,F)
C     ----------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION Y(9),F(9)
      DOUBLE PRECISION l1,lc1,l2,lc2,I1,I2,M,Taumax(2),Tau(2),
     & c1,c2,c3,c4,a11,a12,a13,a22,a14,a24,S(2),deta
C      f22,f24,f32,f34,f42,f44
C      DOUBLE PRECISION X1,X2,X3,X4,P1,P2,P3,P4,Z

      X1=Y(1)
      X2=Y(2)
      X3=Y(3)
      X4=Y(4)

      P1=Y(5)
      P2=Y(6)
      P3=Y(7)
      P4=Y(8)
      Z=Y(9)

C      Physical parameters for Case 1 and Case 2

C      length of the  link 1
      l1=0.4D0
      lc1=l1/2.0D0
C      length of the  link 2
      l2=0.25D0
      lc2=l2/2.0D0
C      mass of the first link
      m1=29.58D0
C      mass of the second link
      m2=15.00D0
C      mass moment of inertia of the link1
      I1=0.416739D0
C      mass moment of inertia of the link2
      I2=0.205625D0
C      mass of end effector and load
      M=6.0D0
C     maximum torque for link 1
      Taumax(1)=25.0D0
C     maximum torque for link 2
      Taumax(2)=9.0D0


C     Physical parameters for Case 3

C      length of the  link 1
C      l1=0.4D0
C      lc1=l1/2.0D0
C      length of the  link 2
C      l2=0.25D0
C      lc2=l2/2.0D0
C      mass of the first link
C      m1=0.245D0
C      mass of the second link
C      m2=0.15315D0
C      mass moment of inertia of the link1
C      I1=3.2688D-3
C      mass moment of inertia of the link2
C      I2=0.7986D-3
C      mass of end effector and load
C      M=0.5D0
C     maximum torque for link 1
C      Taumax(1)=0.25D0
C     maximum torque for link 2
C      Taumax(2)=0.1D0


C     Physical parameters for Case 4

C      length of the  link 1
C      l1=1.0D0
C      lc1=l1/2.0D0
C      length of the  link 2
C      l2=0.625D0
C      lc2=l2/2.0D0
C      mass of the first link
C      m1=0.61261D0
```

```
C      mass of the second link
C      m2=0.38288D0
C      mass moment of inertia of the link1
C      I1=51.0547D-3
C      mass moment of inertia of the link2
C      I2=12.4660D-3
C      mass of end effector and load
C      M=1.25D0
C     maximum torque for link 1
C      Taumax(1)=3.90D0
C     maximum torque for link 2
C      Taumax(2)=1.56D0

      c1=I1+m1*lc1**2.0D0+(m2+M)*l1**2.0D0
      c2=I2+m2*lc2**2.0D0+M*l2**2.0D0
      c3=m2*l1*lc2+M*l1*l2
      c4=m1*lc1+(m2+M)*l1
      c5=m2*lc2+M*l2

      a11=c1+c2+2.0D0*c3*DCOS(X3)
      a12=c2+c3*DCOS(X3)
      a13=c3*DSIN(X3)
      a22=c2
      a14=c4*DCOS(X1)+c5*DCOS(X1+X3)
      a24=c5*DCOS(X1+X3)

      deta=c1*c2-c3**2.0D0*(DCOS(X3))**2.0D0


C     Decided the control input
      S(1) = (P2*a22-P4*a12)/deta
      S(2) = (-P2*a12+P4*a11)/deta

      IF(S(1).LT.0.0D0) THEN
        Tau(1)=Taumax(1)
      ELSE
        Tau(1)=-Taumax(1)
      END IF

      IF(S(2).LT.0.0D0) THEN
        Tau(2)=Taumax(2)
      ELSE
        Tau(2)=-Taumax(2)
      END IF


      f21=-1.0D0
      f22=-2.0D0*a13*(a12*X2+a22*X4)/deta
      f24=2.0D0*a13*(a11*X2+a12*X4)/deta
      f32=-1.0D0*((a12-a22)*(a22*(2.0D0*X2+X4)*X4+a12*X2**2.0D0)-a13**2.0D0
     *      *X2**2.0D0+a13*Tau(2))/deta + 2.0D0*a13*(a12-a22)*
     *      (a13*(a22*(2.0D0*X2+X4)*X4+a12*X2**2.0D0)
     *      +a22*Tau(1)-a12*Tau(2))/deta**2.0D0

      f34=((a12-a22)*(a12*(2.0D0*X2+X4)*X4+a11*X2**2.0D0)-a13**2.0D0
     &     *((X2+X4)**2.0D0+X2**2.0D0)+a13*(2.0D0*Tau(2)-Tau(1)))/deta
     &     -2.0D0*a13*(a12-a22)*(a13*(a12*(2.0D0*X2+X4)*X4
     &     +a11*X2**2.0D0)+a12*Tau(1)-a11*Tau(2))/deta**2.0D0

      f42=-2.0D0*a13*a22*(X2+X4)/deta

      f44=2.0D0*a13*a12*(X2+X4)/deta
      F(1) = Z*X2

      F(2) = Z*(a13*(a22*(2.0D2*X2+X4)*X4+a12*X2**2.0D0)
     &       +a22*Tau(1)-a12*Tau(2))/deta

      F(3) = Z*X4

      F(4) = Z*(-a13*(a12*(2.0D2*X2+X4)*X4+a11*X2**2.0D0)
     &      -a12*Tau(1)+a11*Tau(2))/deta

      F(5) = 0.0D0
      F(6) = -Z*(-P1+P2*f22+P4*f24)
      F(7) = -Z*(P2*f32+P4*f34)
      F(8) = -Z*(-P3+P2*f42+P4*f44)
      F(9) = 0.0D0

      RETURN
C     END OF FDIF
      END


      SUBROUTINE XOT(T,X)
C     -------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION X(5)
C
C     Case 1
      X(1) = 0.975D0
      X(2) = 0.0D0
      X(3) = 0.0D0
      X(4) = 0.0D0
      X(5) = -0.456692
      X(6) = -0.298622
      X(7) = -0.093548
      X(8) = -0.074258
      X(9) = 1.083378

C     Case 2
C      X(1) = 0.76
C      X(2) = 0.0D0
C      X(3) = 0.2618
C      X(4) = 0.0D0
C      X(5) = -0.534711
C      X(6) = -0.310833
C      X(7) = -0.117210
C      X(8) = -0.07799
C      X(9) = 1.023704
C
C     Case 3
C      X(1) = 0.376
C      X(2) = 0.0D0
C      X(3) = 0.0D0
C      X(4) = 0.0D0
C      X(5) = -0.534711
C      X(6) = -0.310833
C      X(7) = -0.117210
C      X(8) = -0.07799
C      X(9) = 1.233072
C
C     Case 4
```

```
C        X(1) = 0.8
C        X(2) = 0.0D0
C        X(3) = 0.2
C        X(4) = 0.0D0
C        X(5) = -0.534711
C        X(6) = -0.310833
C        X(7) = -0.117210
C        X(8) = -0.07799
C        X(9) = 1.620396

      RETURN
C     END OF XOT
      END


      SUBROUTINE G(N,XA,XB,FG,DGA,DGB)
C     --------------------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION XA(N),XB(N),FG(N),DGA(N,N),DGB(N,N)
C
      DO 1100 I = 1 , N
      DO 1100 J = 1 , N
        DGA(I,J) = 0.D0
        DGB(I,J) = 0.D0
 1100 CONTINUE
      DGA(1,1) = 1.D0
      DGA(2,2) = 1.D0
      DGA(3,3) = 1.D0
      DGA(4,4) = 1.D0
      DGB(1,1) = 1.D0
      DGB(2,2) = 1.D0
      DGB(3,3) = 1.D0
      DGB(4,4) = 1.D0

C     Case 1
      FG(1) = XA(1) - 0.975D0
      FG(2) = XA(2)
      FG(3) = XA(3)
      FG(4) = XA(4)
      FG(5) = XB(1)
      FG(6) = XB(2)
      FG(7) = XB(3)
      FG(8) = XB(4)


C     Case 2
C        FG(1) = XA(1) - 0.76D0
C        FG(2) = XA(2)
C        FG(3) = XA(3)- 0.2618D0
C        FG(4) = XA(4)
C        FG(5) = XB(1)
C        FG(6) = XB(2)
C        FG(7) = XB(3)
C        FG(8) = XB(4)
C     Case 3
C        FG(1) = XA(1) - 0.376D0
C        FG(2) = XA(2)
C        FG(3) = XA(3)
C        FG(4) = XA(4)
C        FG(5) = XB(1)
C        FG(6) = XB(2)
```

```
C      FG(7) = XB(3)
C      FG(8) = XB(4)
C     Case 4
C      FG(1) = XA(1) - 0.8D0
C      FG(2) = XA(2)
C      FG(3) = XA(3)- 0.2D0
C      FG(4) = XA(4)
C      FG(5) = XB(1)
C      FG(6) = XB(2)
C      FG(7) = XB(3)
C      FG(8) = XB(4)

      RETURN
C     END OF G
      END
```

A.9 S-function of Two-link Revolute Manipulator

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   File name: TLRPMsimfun.m %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Chapter 6
% This is the s-function of two-link planar revolute manipulator

function[sys,x0]=robotTLRPM(t,x,u,flag,x01,x02,x03,x04)

if abs(flag)==1

% physical parameters for Case 1 and Case 2
%l1=0.4;   % length of the  link 1
%lc1=l1/2;
%l2=0.25;  % length of the link 2
%lc2=l2/2;
%m1=29.58;   % mass of the first link
%m2=15.00;   % mass of the second link
%I1=0.416739; % mass moment of inertia of the link1
%I2=0.205625; % mass moment of inertia of the link2
%M=6.0;       % mass of end effector and load
%u1max=25;    % maximum torque for link 1
%u2max=9;     % maximum torque for link 2

% physical parameters for Case 3

%l1=0.4;   % length of the  link 1
%lc1=l1/2;
%l2=0.25;  % length of the link 2
%lc2=l2/2;
%m1=0.245;   % mass of the first link
%m2=0.15315;   % mass of the second link
%I1=3.2688e-3; % mass moment of inertia of the link1
%I2=0.7986e-3; % mass moment of inertia of the link2
%M=0.5;       % mass of end effector and load
%u1max=0.25;    % maximum torque for link 1
%u2max=0.1;     % maximum torque for link 2

% physical parameters for Case 4

l1=1.0;   % length of the  link 1
lc1=l1/2;
l2=0.625;  % length of the link 2
```

```
lc2=l2/2;
m1=0.61261;    % mass of the first link
m2=0.38288;    % mass of the second link
I1=51.0547e-3; % mass moment of inertia of the link1
I2=12.4660e-3; % mass moment of inertia of the link2
M=1.25;        % mass of end effector and load
u1max=3.90;     % maximum torque for link 1
u2max=1.56;      % maximum torque for link 2

c1=I1+m1*lc1^2+(m2+M)*l1^2;
c2=I2+m2*lc2^2+M*l2^2;
c3=m2*l1*lc2+M*l1*l2;

a=[x(2);
(c2*(u(1)-u(2)+c3*(x(2)+x(4))^2*sin(x(3)))-c3*(u(2)
-c3*(x(2)^2)*sin(x(3)))*cos(x(3)))/(c1*c2-c3^2*(cos(x(3)))^2);
x(4);
((c1+c3*cos(x(3)))*(u(2)-c3*x(2)^2*sin(x(3)))
-(c2+c3*cos(x(3)))*(u(1)-u(2)+c3*(x(2)+x(4))^2*sin(x(3))))
/(c1*c2-c3^2*(cos(x(3)))^2)];

sys=a;

elseif abs(flag)==3

sys=x;

elseif abs(flag)==0

sys=[4.0,0.0,4.0,2.0,0.0,1.0];
x0(1)=x01;
x0(2)=x02;
x0(3)=x03;
x0(4)=x04;

else
sys=[];
end
```

A.10 S-function of Zero-net-work Controller for Two-link Revolute Manipulator

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File name: ZWcontroller.m %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Chapter 6
% This is the s-function
% used for generating the controller inputs
% of two link planar revolute manipulator.

function[sys,x0,str,ts]=ZWcontroller(t,control,Q,flag,q1sw,q2sw)

% physical parameters for Case 1 and Case 2
l1=0.4;   % length of the  link 1
lc1=l1/2;
l2=0.25;  % length of the link 2
lc2=l2/2;
m1=29.58;   % mass of the first link
m2=15.00;   % mass of the second link
I1=0.416739; % mass moment of inertia of the link1
I2=0.205625; % mass moment of inertia of the link2
```

```
M=6.0;        % mass of end effector and load
u1max=25;     % maximum torque for link 1
u2max=9;      % maximum torque for link 2


% physical parameters for Case 3
%l1=0.4;   % length of the  link 1
%lc1=l1/2;
%l2=0.25;  % length of the link 2
%lc2=l2/2;
%m1=0.245;   % mass of the first link
%m2=0.15315;   % mass of the second link
%I1=3.2688e-3; % mass moment of inertia of the link1
%I2=0.7986e-3; % mass moment of inertia of the link2
%M=0.5;        % mass of end effector and load
%u1max=0.25;     % maximum torque for link 1
%u2max=0.1;      % maximum torque for link 2

% physical parameters for Case 4

%l1=1.0;   % length of the  link 1
%lc1=l1/2;
%l2=0.625;  % length of the link 2
%lc2=l2/2;
%m1=0.61261;   % mass of the first link
%m2=0.38288;   % mass of the second link
%I1=51.0547e-3; % mass moment of inertia of the link1
%I2=12.4660e-3; % mass moment of inertia of the link2
%M=1.25;        % mass of end effector and load
%u1max=3.90;     % maximum torque for link 1
%u2max=1.56;      % maximum torque for link 2

error=0.0001;  % final position error

c1=I1+m1*lc1^2+(m2+M)*l1^2;
c2=I2+m2*lc2^2+M*l2^2;
c3=m2*l1*lc2+M*l1*l2;


switch flag,
  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  % Initialize the states, sample times, and state ordering strings.
  case 0
    [sys,x0,str,ts]=mdlInitializeSizes;

  %%%%%%%%%%%
  % Outputs %
  %%%%%%%%%%%
  % Return the outputs of the S-function block.
  case 3
   sys=mdlOutputs(t,control,Q,u1max,u2max,c1,c2,c3,error,q1sw,q2sw);

  %%%%%%%%%%%%%%%%%%%
  % Unhandled flags %
  %%%%%%%%%%%%%%%%%%%
  % There are no termination tasks (flag=9) to be handled.
  % Also, there are no continuous or discrete states,
  % so flags 1,2, and 4 are not used, so return an emptyu
  % matrix
  case { 1, 2, 4, 9 }
```

```
    sys=[];

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % Unexpected flags (error handling)%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % Return an error message for unhandled flag values.
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% end timestwo

%
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts] = mdlInitializeSizes()

sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 2;
sizes.NumInputs      = 4;
sizes.DirFeedthrough = 1;   % has direct feedthrough
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);
str = [];
x0  = [];
ts  = [-1 0];   % inherited sample time

% end mdlInitializeSizes

%
%=============================================================================
% mdlOutputs
% Return the output vector for the S-function
%=============================================================================
%
function sys = mdlOutputs(t,control,Q,u1max,u2max,c1,c2,c3,error,q1sw,q2sw)

% Q(1) is position of link1
% Q(3) is position of link2
if Q(1)>q1sw
    control(1)= - u1max;
else
    control(1) = u1max;
end

if Q(3)>q2sw
    control(2)= - u2max;
else
    control(2)= u2max;
end

if abs(Q(1)<=error)&abs(Q(3)<=error)
 %turn on PD controller
 % Kp=100;
 % Kd=20;
 % control(1)=-Kp*Q(1)-Kd*Q(2);
 % control(2)=-Kp*Q(3)-Kd*Q(4);
  control(1)=0;
  control(2)=0;
elseif abs(Q(1)<=error)&abs(Q(3)>error)
    control(1)=(c2+c3*cos(Q(3)))*control(2)/c2-c3*sin(Q(3))*Q(4)^2;
elseif abs(Q(1)>error)&abs(Q(3)<=error)
    control(2)=control(1)*(c2+c3)/(c1+c2+c3*2);
end

sys = [control(1);
       control(2)];

% end mdlOutputs
```

A.11 S-function of Proportional and Derivative (PD) Controller for Two-link Revolute Manipulator

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   File name: PDcontroller.m %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Chapter 6
% This is the s-function
% used for generating the PD control inputs
% for two-link planar revolute manipulator.

function[sys,x0,str,ts]=MTcontroller(t,control,Q,flag,kp,kd)

switch flag,
  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  % Initialize the states, sample times, and state ordering strings.
  case 0
    [sys,x0,str,ts]=mdlInitializeSizes;

  %%%%%%%%%%%
  % Outputs %
  %%%%%%%%%%%
  % Return the outputs of the S-function block.
  case 3
   sys=mdlOutputs(t,control,Q,kp,kd);

  %%%%%%%%%%%%%%%%%%%
  % Unhandled flags %
  %%%%%%%%%%%%%%%%%%%
  % There are no termination tasks (flag=9) to be handled.
  % Also, there are no continuous or discrete states,
  % so flags 1,2, and 4 are not used, so return an emptyu
  % matrix
  case { 1, 2, 4, 9 }
    sys=[];

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % Unexpected flags (error handling)%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % Return an error message for unhandled flag values.
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
```

```
end

% end timestwo

%
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts] = mdlInitializeSizes()

sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 2;
sizes.NumInputs      = 4;
sizes.DirFeedthrough = 1;   % has direct feedthrough
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);
str = [];
x0  = [];
ts  = [-1 0];   % inherited sample time

% end mdlInitializeSizes

%
%=============================================================================
% mdlOutputs
% Return the output vector for the S-function
%=============================================================================
%
function sys = mdlOutputs(t,control,Q,kp,kd)

 % turn on PD controller
  control(1)=-kp*Q(1)-kd*Q(2);
  control(2)=-kp*Q(3)-kd*Q(4);

sys = [control(1);
       control(2)];

% end mdlOutputs
```

Appendix B

Subroutines Used in This Thesis

B.1 Subroutine MUSN for Solving Nonlinear TPBVP

```
Subroutine MUSN is written in FORTRAN 77 and is available
through Netlib repository (http://www.netlib.org/).
The Netlib repository contains freely available software,
documents, and databases of interest to the numerical,
scientific computing, and other communities.
The repository is maintained by AT&T Bell Laboratories,
the University of Tennessee and Oak Ridge National Laboratory,
and by colleagues world-wide. The collection is replicated
at several sites around the world, automatically synchronized,
to provide reliable and network efficient service to the
global community.

                                    MUSN

****************
SPECIFICATION  MUSN
****************

      SUBROUTINE MUSN(FDIF,YOT,G,N,A,B,ER,TI,NTI,NRTI,AMP,ITLIM,Y,Q,U,
     1                NU,D,PHI,KP,W,LW,IW,LIW,WG,LWG,IERROR)
C
C     DOUBLE PRECISION A,B,ER(5),TI(NTI),AMP,Y(N,NTI),Q(N,N,NTI),
C    1                 U(NU,NTI),D(N,NTI),PHI(NU,NTI),W(LW),WG(LWG)
C     INTEGER N,NTI,NRTI,ITLIM,NU,KP,LW,IW(LIW),LIW,LWG,IERROR
C     EXTERNAL FDIF,YOT,G

****************
Purpose
****************

MUSN solves the nonlinear two-point BVP

       dy(t)/dt = f(t,y)          A <= t <= B  or B <= t <= A

       g(y(a),y(b)) = 0 ,

where y(t) and f(t,y) are N-vector functions.

****************
Method
****************

MUSN uses a multiple shooting method for computing an approximate solution of
the BVP at specified output points, which are also used as shooting points.
If necessary, more output points (shooting points) are inserted during computa-
tion.
For integration a fixed grid is used. Output points are also grid points and the
minimum number of grid points between two output points is 5.
```

```
****************
Parameters
****************

FDIF    SUBROUTINE, supplied by the user with specification:

        SUBROUTINE FDIF(T,Y,F)
        DOUBLE PRECISION T,Y(N),F(N)

        where N is the order of the system. FDIF must evaluate the righthand-
        side of the differential equation, f(t,y) for t=T and y=Y and places the
        result in F(1),...,F(N).
        FDIF must be declared as EXTERNAL in the (sub)program from which MUSN is
        called.

YOT     SUBROUTINE, supplied by the user with specification

        SUBROUTINE YOT(T,Y)
        DOUBLE PRECISION T,Y(N)

        where N is the order of the system. YOT must evaluate the initial appro-
        ximation y0(t) of the solution, for any value t=T and place the result
        in Y(1),...,Y(N).
        YOT must be declared as EXTERNAL in the (sub)program from which MUSN is
        called.

G       SUBROUTINE, supplied by the user with specification

        SUBROUTINE G(N,YA,YB,FG,DGA,DGB)
        DOUBLE PRECISION YA(N),YB(N),FG(N),DGA(N,N),DGB(N,N)

        where N is the order of the system. G must evaluate g(y(A),y(B)) for
        y(A)=YA and y(B)=YB and place the result in FG(1),...,FG(N). Moreover G
        must evaluate the Jacobians

              dg(u,v)/du  for u = YA   and   dg(u,v)/dv for v = YB

        and place the result in the arrays DGA anb DGB respectively.
        G must be declared as EXTERNAL in the (sub)program from which MUSN is
        called.

N       INTEGER, the order of the system.
        Unchanged on exit.

A,B     DOUBLE PRECISION, the two boundary points.
        Unchanged on exit.

ER      DOUBLE PRECISION array of dimension (5).
        On entry:
        ER(1) must contain the required tolerance for solving the differential
              equation.
        ER(2) must contain the initial tolerance with which a first aproximate
              solution will be computed. This approximation is then used as an
              initial approximation for the computation of an solution with an
              tolerance ER(2)*ER(2) and so on until the required tolerance is
              reached. As an initial tolerance max(ER(1),min(ER(2),1.d-2)) will
              be used.
        ER(3) must contain the machine precision.
        On exit:
        ER(1), ER(2) and ER(3) are unchanged.
        ER(4) contains an estimation of the condition number of the BVP.
        ER(5) contains an estimated error amplification factor.

TI      DOUBLE PRECISION array of dimension (NTI).
        On entry : if NRTI = 1 TI must contain the output points in strict
                   monotone order: A=TI(1) < TI(2) <...< TI(n)=B.
        On exit TI(j), j=1,...,NRTI contains the output points.

NTI     INTEGER, NTI is one of the dimension of  TI, X, S, Q, U en PHI.
        NTI must be greater than or equal to the total number of necessary
        output points + 1  (i.e. if the entry value for NRTI > 1, NTI may be
        equal to the entry value of NRTI + 1).
        Unchanged on exit.

NRTI    INTEGER.
        On entry NRTI is used to specify the output points. There are 3 ways to
        specify the output points:
        1) NRTI = 0, the output points are determined automatically using AMP.
        2) NRTI = 1, the output points are supplied by the user in the array TI.
        3) NRTI > 1, the subroutine computes the (NRTI+1) output points TI(k) by
                          TI(k) = A + (k-1) * (B - A) / NRTI ;
                     so TI(1) = A and TI(NRTI+1) = B.
        Depending on the allowed increment between two succesive output points,
        more output points may be inserted in cases 2 and 3.
        On exit NRTI contains the total number of output points.

AMP     DOUBLE PRECISION.
        On entry AMP must contain the allowed increment between two output
        points. AMP is used to determine output points and to assure that the
        increment between two output points is at most AMP*AMP. A small value
        for AMP may result in a large number of output points.
        Unless 1 < AMP < .25 * sqrt(ER(1)/ER(3)) the default value
        .25 * sqrt(ER(1)/ER(3)) is used.
        Unchanged on exit.

ITLIM   INTEGER, maximum number of allowed iteration.

Y       DOUBLE PRECISION array of dimension (N,NTI).
        On exit Y(.,i), i=1,...,NRTI contains the solution at the output points
        TI(i), i=1,...,NRTI.

Q       DOUBLE PRECISION array of dimension (N,N,NTI).
        On exit  Q(.,.,i), i=1,...,NRTI contains the orthogonal factors of the
        incremental recursion.

U       DOUBLE PRECISION array of dimension (NU,NTI).
        On exit U(.,i), i=2,...,NRTI contains the upper triangular factors of
        the incremental recursion. The elements are stored column wise, the j-th
        column of U is stored in U(nj+1,.),U(nj+2,.),...,U(nj+j,.), where
        nj = (j-1) * j / 2.

NU      INTEGER, NU is one of the dimension of U and PHI.
        NU must be greater than or equal to N * (N+1) / 2.
        Unchanged on exit.

D       DOUBLE PRECISION array of dimension (N,NTI).
        On exit D(.,i) i=2,...,NRTI contain the inhomogeneous terms of the
        incremental recursion.

PHI     DOUBLE PRECISION array of dimension (NU,NTI).
        On exit PHI(.,i), i=1,...,NRTI contains the fundamental solution of the
        incremental recursion. The fundamental solution is upper triangular and
        stored in the same way as the upper triangular U.
```

```
KP      INTEGER,
        On exit KP contains the dimension of the increasing solution space.

W       DOUBLE PRECISION array of dimension (LW).
        Used as work space.

LW      INTEGER.
        LW is the dimension of W. LW >= 7*N + 3*N*NTI + 4*N*N .

IW      INTEGER array of dimension (LIW).
        Used as work space.

LIW     INTEGER.
        LIW is the dimension of IW. LIW >= 3*N + NTI .

WG      DOUBLE PRECISION array of dimension (LWG).
        WG is used to store the integration grid points.

LWG     INTEGER.
        LWG is the dimension of WG. LWG >= (total number of grid points) / 5.
        The minimum number of grid points between 2 succesive output points is
        5, so the minimum value for LWG is the number of actually used output
        points. Initially a crude estimate for LWG has to be made (see also
        IERROR 219).

IERROR  INTEGER, error indicator.
        On entry : if IERROR=1, diagnostics will be printed during computation.
        On exit : if IERROR = 0 no errors have been detected.


****************
Error indicators
****************

IERROR

0       No errors detected.

101     INPUT error: either ER(1) < 0 or ER(2) < 0 or ER(3) < 0.
        TERMINAL ERROR.

105     INPUT error: either N < 1 or NRTI < 0 or NTI < 3 or NU < N*(N+1)/2 or
        A=B.
        TERMINAL ERROR.

106     INPUT error: either LW < 7*N + 3*N*NTI + 4*N*N  or LIW < 3*N + NTI.
        TERMINAL ERROR.

120     INPUT error: the routine was called with NRTI=1, but the given output
        points in the array TI are not in strict monotone order.
        TERMINAL ERROR.

121     INPUT error: the routine was called with NRTI=1, but the first given
        output point or the last output point is not equal to A or B.
        TERMINAL ERROR.

122     INPUT error: the value of NTI is too small; the number of necessary
        output points is greater than NTI-1.
        TERMINAL ERROR.
```

```
123     INPUT error: the value of LWG is less than the number of output points.
        Increase the dimension of the array WG and the value of LWG.
        TERMINAL ERROR.

216     This indicates that during integration the requested accuracy could not
        be achieved. User must increase error tolerance.
        TERMINAL ERROR.

219     This indicates that the routine needs more space to store the integra-
        tion grid point. An estimate for the required workspace (i.e. the value
        for LWG) is given.
        TERMINAL ERROR.

230     This indicates that Newton failed to converge.
        TERMINAL ERROR.

231     This indicates that the number of iteration has become greater than
        ITLIM.
        TERMINAL ERROR.

240     This indicates that the global error is probably larger than the error
        tolerance due to instabilities in the system. Most likely the system is
        ill-conditioned. Output value is the estimated error amplification
        factor.
        WARNING ERROR.

250     This indicate that one of the upper triangular matrices U is singular.
        TERMINAL ERROR.

260     This indicates that the problem is probably too ill-conditioned with
        respect to the BC.
        TERMINAL ERROR.


****************
Auxiliary routines
****************

Calls are made to the MUS library routines: DBCMAV, DCHINC, DCROUT, DCSAOJ,
DCSHPO, DFQUS, DFUNRC, DGTUR, DINPRO, DINTCH, DJINGX, DKPCH, DLUDEC, DMATVC,
DNEWPO, DPSR, DQEVAL, DQUDEC, DRKF1S, DRKFGG, DRKFGS, DRKFMS, DSBVP, DSOLDE,
DSOLUP, DSORTD, DTAMVC, ERRHAN.

****************
Remarks
****************

MUSN is written by R.M.M. Mattheij and G.W.M. Staarink.
Last update: 02-15-88.

****************
Example of the use of MUSN
****************

Consider the differential equation:

        u' = .5  u*(w-u) / v
        v' = -0.5 (w-u)
        w' = (0.9 - 1000 (w-y) - 0.5 w(w-u)) / x
        x' = 0.5 (w-u)
        y' = -100 (y-w)
```

```
and the boundary conditions:

        u(0) = v(0) = w(0) = 1
        x(0) = -10
        w(1) = y(1)


As an initial guess for the solution we take:

        u(t) = 1 ; v(t) = 1 ; x(t) = -10 ;
        w(t) = -4.5 t*t + 8.91 t + 1 ; y(t) = -4.5 t*t + 9 t + 0.91

The next program computes and prints the solution for t=0, 0.1,0.2,...,1.
This program has been run on a Olivetti M24 PC., operating under MS-DOS V2.11,
using the Olivetti MS-Fortran V3.13 R1.0 compiler and the MS Object Linker V2.01
(large).
```

```
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION ER(5),TI(12),X(5,12),Q(5,5,12),U(15,12),D(5,12),
     1          PHIREC(15,12),W(315),WGR(20)
      INTEGER IW(27)
      EXTERNAL FDIF,YOT,G
C
C     SETTING OF THE INPUT PARAMETERS
C
      N = 5
      NU = 15
      NTI = 12
      LW = 315
      LIW = 27
      ER(3) = 1.1D-15
      LWG = 20
      ER(1) = 1.D-6
      ER(2) = 1.D-2
      A = 0.D0
      B = 1.D0
      NRTI = 10
      AMP = 100
      ITLIM = 20
      CALL MUSN(FDIF,YOT,G,N,A,B,ER,TI,NTI,NRTI,AMP,ITLIM,X,Q,U,NU,D,
     1          PHIREC,KPART,W,LW,IW,LIW,WGR,LWG,IERROR)
      .
      .
      .
      END
      SUBROUTINE FDIF(T,Y,F)
C     ----------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION Y(5),F(5)
C
      Y3MY1 = Y(3) - Y(1)
      Y3MY5 = Y(3) - Y(5)
      F(1) = 0.5D0 * Y(1) * Y3MY1 / Y(2)
      F(2) = - 0.5D0 * Y3MY1
      F(3) = (0.9D0 - 1.D3 * Y3MY5 - 0.5D0 * Y(3) * Y3MY1) / Y(4)
      F(4) = 0.5D0 * Y3MY1
      F(5) =  1.D2 * Y3MY5
      RETURN
C     END OF FDIF
      END
      SUBROUTINE YOT(T,X)
```

```
C     -------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION X(5)
C
      X(1) = 1.D0
      X(2) = 1.D0
      X(4) = -10.D0
      X(3) = - 4.5D0*T*T + 8.91D0 * T + 1.D0
      X(5) = - 4.5D0*T*T + 9.D0 * T + 0.91D0
      RETURN
C     END OF YOT
      END
      SUBROUTINE G(N,XA,XB,FG,DGA,DGB)
C     --------------------------------
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION XA(N),XB(N),FG(N),DGA(N,N),DGB(N,N)
C
      DO 1100 I = 1 , N
      DO 1100 J = 1 , N
        DGA(I,J) = 0.D0
        DGB(I,J) = 0.D0
 1100 CONTINUE
      DGA(1,1) = 1.D0
      DGA(2,2) = 1.D0
      DGA(3,3) = 1.D0
      DGA(4,4) = 1.D0
      DGB(5,3) = 1.D0
      DGB(5,5) = -1.D0
      FG(1) = XA(1) - 1.D0
      FG(2) = XA(2) - 1.D0
      FG(3) = XA(3) - 1.D0
      FG(4) = XA(4) + 10.D0
      FG(5) = XB(3) - XB(5)
      RETURN
C     END OF G
      END
```

B.2 Subroutine DDASS for Solving Nonlinear IVP

```
C     This is FORTRAN subroutine used to solve ordinary differential equations
C     with initial conditions.
C     It is written in FORTRAN 77 and is available through Netlib repository
C     (http://www.netlib.org/).
C     It can direct use in the SUN station. When use PC change the
C     subroutine "d1mach" at the end of the program.

      SUBROUTINE DDASSL (RES, NEQ, T, Y, YPRIME, TOUT, INFO, RTOL, ATOL,
     +   IDID, RWORK, LRW, IWORK, LIW, RPAR, IPAR, JAC)
C***BEGIN PROLOGUE  DDASSL
C***PURPOSE  This code solves a system of differential/algebraic
C            equations of the form G(T,Y,YPRIME) = 0.
C***TYPE      DOUBLE PRECISION (SDASSL-S, DDASSL-D)
C***KEYWORDS  DIFFERENTIAL/ALGEBRAIC, BACKWARD DIFFERENTIATION FORMULAS,
C             IMPLICIT DIFFERENTIAL SYSTEMS
C***DESCRIPTION
C***ROUTINES CALLED  D1MACH, DDAINI, DDANRM, DDASTP, DDATRP, DDAWTS,
C                    XERMSG
```

```
C *Usage:
C
C      EXTERNAL RES, JAC
C      INTEGER NEQ, INFO(N), IDID, LRW, LIW, IWORK(LIW), IPAR
C      DOUBLE PRECISION T, Y(NEQ), YPRIME(NEQ), TOUT, RTOL, ATOL,
C     *   RWORK(LRW), RPAR
C
C      CALL DDASSL (RES, NEQ, T, Y, YPRIME, TOUT, INFO, RTOL, ATOL,
C     *   IDID, RWORK, LRW, IWORK, LIW, RPAR, IPAR, JAC)
C
C
C *Arguments:
C  (In the following, all real arrays should be type DOUBLE PRECISION.)
C
C  RES:EXT     This is a subroutine which you provide to define the
C              differential/algebraic system.
C
C  NEQ:IN      This is the number of equations to be solved.
C
C  T:INOUT     This is the current value of the independent variable.
C
C  Y(*):INOUT  This array contains the solution components at T.
C
C  YPRIME(*):INOUT  This array contains the derivatives of the solution
C              components at T.
C
C  TOUT:IN     This is a point at which a solution is desired.
C
C  INFO(N):IN  The basic task of the code is to solve the system from T
C              to TOUT and return an answer at TOUT.  INFO is an integer
C              array which is used to communicate exactly how you want
C              this task to be carried out.  (See below for details.)
C              N must be greater than or equal to 15.
C
C  RTOL,ATOL:INOUT  These quantities represent relative and absolute
C              error tolerances which you provide to indicate how
C              accurately you wish the solution to be computed.  You
C              may choose them to be both scalars or else both vectors.
C              Caution:  In Fortran 77, a scalar is not the same as an
C                        array of length 1.  Some compilers may object
C                        to using scalars for RTOL,ATOL.
C
C  IDID:OUT    This scalar quantity is an indicator reporting what the
C              code did.  You must monitor this integer variable to
C              decide  what action to take next.
C
C  RWORK:WORK  A real work array of length LRW which provides the
C              code with needed storage space.
C
C  LRW:IN      The length of RWORK.  (See below for required length.)
C
C  IWORK:WORK  An integer work array of length LIW which probides the
C              code with needed storage space.
C
C  LIW:IN      The length of IWORK.  (See below for required length.)
C
C  RPAR,IPAR:IN  These are real and integer parameter arrays which
C              you can use for communication between your calling
C              program and the RES subroutine (and the JAC subroutine)
C
C  JAC:EXT     This is the name of a subroutine which you may choose
C              to provide for defining a matrix of partial derivatives
```

```
C             described below.
C
C  Quantities which may be altered by DDASSL are:
C     T, Y(*), YPRIME(*), INFO(1), RTOL, ATOL,
C     IDID, RWORK(*) AND IWORK(*)
C
C *Description
C
C  Subroutine DDASSL uses the backward differentiation formulas of
C  orders one through five to solve a system of the above form for Y and
C  YPRIME.  Values for Y and YPRIME at the initial time must be given as
C  input.  These values must be consistent, (that is, if T,Y,YPRIME are
C  the given initial values, they must satisfy G(T,Y,YPRIME) = 0.).  The
C  subroutine solves the system from T to TOUT.  It is easy to continue
C  the solution to get results at additional TOUT.  This is the interval
C  mode of operation.  Intermediate results can also be obtained easily
C  by using the intermediate-output capability.
C
C  The following detailed description is divided into subsections:
C    1. Input required for the first call to DDASSL.
C    2. Output after any return from DDASSL.
C    3. What to do to continue the integration.
C    4. Error messages.
C
C
C  -------- INPUT -- WHAT TO DO ON THE FIRST CALL TO DDASSL ------------
C
C  The first call of the code is defined to be the start of each new
C  problem. Read through the descriptions of all the following items,
C  provide sufficient storage space for designated arrays, set
C  appropriate variables for the initialization of the problem, and
C  give information about how you want the problem to be solved.
C
C
C  RES -- Provide a subroutine of the form
C             SUBROUTINE RES(T,Y,YPRIME,DELTA,IRES,RPAR,IPAR)
C         to define the system of differential/algebraic
C         equations which is to be solved. For the given values
C         of T,Y and YPRIME, the subroutine should
C         return the residual of the defferential/algebraic
C         system
C             DELTA = G(T,Y,YPRIME)
C         (DELTA(*) is a vector of length NEQ which is
C         output for RES.)
C
C         Subroutine RES must not alter T,Y or YPRIME.
C         You must declare the name RES in an external
C         statement in your program that calls DDASSL.
C         You must dimension Y,YPRIME and DELTA in RES.
C
C         IRES is an integer flag which is always equal to
C         zero on input. Subroutine RES should alter IRES
C         only if it encounters an illegal value of Y or
C         a stop condition. Set IRES = -1 if an input value
C         is illegal, and DDASSL will try to solve the problem
C         without getting IRES = -1. If IRES = -2, DDASSL
C         will return control to the calling program
C         with IDID = -11.
C
C         RPAR and IPAR are real and integer parameter arrays which
C         you can use for communication between your calling program
C         and subroutine RES. They are not altered by DDASSL. If you
```

```
C        do not need RPAR or IPAR, ignore these parameters by treat-
C        ing them as dummy arguments. If you do choose to use them,
C        dimension them in your calling program and in RES as arrays
C        of appropriate length.
C
C  NEQ -- Set it to the number of differential equations.
C        (NEQ .GE. 1)
C
C  T -- Set it to the initial point of the integration.
C        T must be defined as a variable.
C
C  Y(*) -- Set this vector to the initial values of the NEQ solution
C        components at the initial point. You must dimension Y of
C        length at least NEQ in your calling program.
C
C  YPRIME(*) -- Set this vector to the initial values of the NEQ
C        first derivatives of the solution components at the initial
C        point.  You must dimension YPRIME at least NEQ in your
C        calling program. If you do not know initial values of some
C        of the solution components, see the explanation of INFO(11).
C
C  TOUT -- Set it to the first point at which a solution
C        is desired. You can not take TOUT = T.
C        integration either forward in T (TOUT .GT. T) or
C        backward in T (TOUT .LT. T) is permitted.
C
C        The code advances the solution from T to TOUT using
C        step sizes which are automatically selected so as to
C        achieve the desired accuracy. If you wish, the code will
C        return with the solution and its derivative at
C        intermediate steps (intermediate-output mode) so that
C        you can monitor them, but you still must provide TOUT in
C        accord with the basic aim of the code.
C
C        The first step taken by the code is a critical one
C        because it must reflect how fast the solution changes near
C        the initial point. The code automatically selects an
C        initial step size which is practically always suitable for
C        the problem. By using the fact that the code will not step
C        past TOUT in the first step, you could, if necessary,
C        restrict the length of the initial step size.
C
C        For some problems it may not be permissible to integrate
C        past a point TSTOP because a discontinuity occurs there
C        or the solution or its derivative is not defined beyond
C        TSTOP. When you have declared a TSTOP point (SEE INFO(4)
C        and RWORK(1)), you have told the code not to integrate
C        past TSTOP. In this case any TOUT beyond TSTOP is invalid
C        input.
C
C  INFO(*) -- Use the INFO array to give the code more details about
C        how you want your problem solved.  This array should be
C        dimensioned of length 15, though DDASSL uses only the first
C        eleven entries.  You must respond to all of the following
C        items, which are arranged as questions.  The simplest use
C        of the code corresponds to answering all questions as yes,
C        i.e. setting all entries of INFO to 0.
C
C     INFO(1) - This parameter enables the code to initialize
C          itself. You must set it to indicate the start of every
C          new problem.
C
```

```
C          **** Is this the first call for this problem ...
C                Yes - Set INFO(1) = 0
C                 No - Not applicable here.
C                      See below for continuation calls.  ****
C
C       INFO(2) - How much accuracy you want of your solution
C              is specified by the error tolerances RTOL and ATOL.
C              The simplest use is to take them both to be scalars.
C              To obtain more flexibility, they can both be vectors.
C              The code must be told your choice.
C
C          **** Are both error tolerances RTOL, ATOL scalars ...
C                Yes - Set INFO(2) = 0
C                      and input scalars for both RTOL and ATOL
C                 No - Set INFO(2) = 1
C                      and input arrays for both RTOL and ATOL ****
C
C       INFO(3) - The code integrates from T in the direction
C              of TOUT by steps. If you wish, it will return the
C              computed solution and derivative at the next
C              intermediate step (the intermediate-output mode) or
C              TOUT, whichever comes first. This is a good way to
C              proceed if you want to see the behavior of the solution.
C              If you must have solutions at a great many specific
C              TOUT points, this code will compute them efficiently.
C
C          **** Do you want the solution only at
C                TOUT (and not at the next intermediate step) ...
C                 Yes - Set INFO(3) = 0
C                  No - Set INFO(3) = 1 ****
C
C       INFO(4) - To handle solutions at a great many specific
C              values TOUT efficiently, this code may integrate past
C              TOUT and interpolate to obtain the result at TOUT.
C              Sometimes it is not possible to integrate beyond some
C              point TSTOP because the equation changes there or it is
C              not defined past TSTOP. Then you must tell the code
C              not to go past.
C
C           **** Can the integration be carried out without any
C                restrictions on the independent variable T ...
C                 Yes - Set INFO(4)=0
C                  No - Set INFO(4)=1
C                       and define the stopping point TSTOP by
C                       setting RWORK(1)=TSTOP ****
C
C       INFO(5) - To solve differential/algebraic problems it is
C              necessary to use a matrix of partial derivatives of the
C              system of differential equations. If you do not
C              provide a subroutine to evaluate it analytically (see
C              description of the item JAC in the call list), it will
C              be approximated by numerical differencing in this code.
C              although it is less trouble for you to have the code
C              compute partial derivatives by numerical differencing,
C              the solution will be more reliable if you provide the
C              derivatives via JAC. Sometimes numerical differencing
C              is cheaper than evaluating derivatives in JAC and
C              sometimes it is not - this depends on your problem.
C
C           **** Do you want the code to evaluate the partial
C                derivatives automatically by numerical differences ...
C                   Yes - Set INFO(5)=0
```

```
C                 No - Set INFO(5)=1
C               and provide subroutine JAC for evaluating the
C               matrix of partial derivatives ****
C
C       INFO(6) - DDASSL will perform much better if the matrix of
C              partial derivatives, DG/DY + CJ*DG/DYPRIME,
C              (here CJ is a scalar determined by DDASSL)
C              is banded and the code is told this. In this
C              case, the storage needed will be greatly reduced,
C              numerical differencing will be performed much cheaper,
C              and a number of important algorithms will execute much
C              faster. The differential equation is said to have
C              half-bandwidths ML (lower) and MU (upper) if equation i
C              involves only unknowns Y(J) with
C                             I-ML .LE. J .LE. I+MU
C              for all I=1,2,...,NEQ. Thus, ML and MU are the widths
C              of the lower and upper parts of the band, respectively,
C              with the main diagonal being excluded. If you do not
C              indicate that the equation has a banded matrix of partial
C              derivatives, the code works with a full matrix of NEQ**2
C              elements (stored in the conventional way). Computations
C              with banded matrices cost less time and storage than with
C              full matrices if 2*ML+MU .LT. NEQ. If you tell the
C              code that the matrix of partial derivatives has a banded
C              structure and you want to provide subroutine JAC to
C              compute the partial derivatives, then you must be careful
C              to store the elements of the matrix in the special form
C              indicated in the description of JAC.
C
C          **** Do you want to solve the problem using a full
C               (dense) matrix (and not a special banded
C               structure) ...
C                Yes - Set INFO(6)=0
C                 No - Set INFO(6)=1
C                       and provide the lower (ML) and upper (MU)
C                       bandwidths by setting
C                       IWORK(1)=ML
C                       IWORK(2)=MU ****
C
C
C        INFO(7) -- You can specify a maximum (absolute value of)
C              stepsize, so that the code
C              will avoid passing over very
C              large regions.
C
C          ****  Do you want the code to decide
C                on its own maximum stepsize?
C                Yes - Set INFO(7)=0
C                 No - Set INFO(7)=1
C                      and define HMAX by setting
C                      RWORK(2)=HMAX ****
C
C        INFO(8) -- Differential/algebraic problems
C              may occaisionally suffer from
C              severe scaling difficulties on the
C              first step. If you know a great deal
C              about the scaling of your problem, you can
C              help to alleviate this problem by
C              specifying an initial stepsize H0.
C
C          ****  Do you want the code to define
C                its own initial stepsize?
```

```
C                Yes - Set INFO(8)=0
C                 No - Set INFO(8)=1
C                      and define HO by setting
C                      RWORK(3)=HO ****
C
C        INFO(9) -- If storage is a severe problem,
C              you can save some locations by
C              restricting the maximum order MAXORD.
C              the default value is 5. for each
C              order decrease below 5, the code
C              requires NEQ fewer locations, however
C              it is likely to be slower. In any
C              case, you must have 1 .LE. MAXORD .LE. 5
C          ****  Do you want the maximum order to
C                default to 5?
C                Yes - Set INFO(9)=0
C                 No - Set INFO(9)=1
C                      and define MAXORD by setting
C                      IWORK(3)=MAXORD ****
C
C        INFO(10) --If you know that the solutions to your equations
C               will always be nonnegative, it may help to set this
C               parameter. However, it is probably best to
C               try the code without using this option first,
C               and only to use this option if that doesn't
C               work very well.
C           ****  Do you want the code to solve the problem without
C                 invoking any special nonnegativity constraints?
C                  Yes - Set INFO(10)=0
C                   No - Set INFO(10)=1
C
C        INFO(11) --DDASSL normally requires the initial T,
C               Y, and YPRIME to be consistent. That is,
C               you must have G(T,Y,YPRIME) = 0 at the initial
C               time. If you do not know the initial
C               derivative precisely, you can let DDASSL try
C               to compute it.
C          ****   Are the initialHE INITIAL T, Y, YPRIME consistent?
C                 Yes - Set INFO(11) = 0
C                  No - Set INFO(11) = 1,
C                       and set YPRIME to an initial approximation
C                       to YPRIME.  (If you have no idea what
C                       YPRIME should be, set it to zero. Note
C                       that the initial Y should be such
C                       that there must exist a YPRIME so that
C                       G(T,Y,YPRIME) = 0.)
C
C  RTOL, ATOL -- You must assign relative (RTOL) and absolute (ATOL
C         error tolerances to tell the code how accurately you
C         want the solution to be computed.  They must be defined
C         as variables because the code may change them.  You
C         have two choices --
C               Both RTOL and ATOL are scalars. (INFO(2)=0)
C               Both RTOL and ATOL are vectors. (INFO(2)=1)
C         in either case all components must be non-negative.
C
C         The tolerances are used by the code in a local error
C         test at each step which requires roughly that
C               ABS(LOCAL ERROR) .LE. RTOL*ABS(Y)+ATOL
C         for each vector component.
C         (More specifically, a root-mean-square norm is used to
C         measure the size of vectors, and the error test uses the
C         magnitude of the solution at the beginning of the step.)
C
C         The true (global) error is the difference between the
C         true solution of the initial value problem and the
C         computed approximation.  Practically all present day
C         codes, including this one, control the local error at
C         each step and do not even attempt to control the global
C         error directly.
C         Usually, but not always, the true accuracy of the
C         computed Y is comparable to the error tolerances. This
C         code will usually, but not always, deliver a more
C         accurate solution if you reduce the tolerances and
C         integrate again.  By comparing two such solutions you
C         can get a fairly reliable idea of the true error in the
C         solution at the bigger tolerances.
C
C         Setting ATOL=0. results in a pure relative error test on
C         that component.  Setting RTOL=0. results in a pure
C         absolute error test on that component.  A mixed test
C         with non-zero RTOL and ATOL corresponds roughly to a
C         relative error test when the solution component is much
C         bigger than ATOL and to an absolute error test when the
C         solution component is smaller than the threshhold ATOL.
C
C         The code will not attempt to compute a solution at an
C         accuracy unreasonable for the machine being used.  It will
C         advise you if you ask for too much accuracy and inform
C         you as to the maximum accuracy it believes possible.
C
C  RWORK(*) --  Dimension this real work array of length LRW in your
C         calling program.
C
C  LRW -- Set it to the declared length of the RWORK array.
C               You must have
C                    LRW .GE. 40+(MAXORD+4)*NEQ+NEQ**2
C               for the full (dense) JACOBIAN case (when INFO(6)=0), or
C                    LRW .GE. 40+(MAXORD+4)*NEQ+(2*ML+MU+1)*NEQ
C               for the banded user-defined JACOBIAN case
C               (when INFO(5)=1 and INFO(6)=1), or
C                     LRW .GE. 40+(MAXORD+4)*NEQ+(2*ML+MU+1)*NEQ
C                           +2*(NEQ/(ML+MU+1)+1)
C               for the banded finite-difference-generated JACOBIAN case
C               (when INFO(5)=0 and INFO(6)=1)
C
C  IWORK(*) --  Dimension this integer work array of length LIW in
C         your calling program.
C
C  LIW -- Set it to the declared length of the IWORK array.
C               You must have LIW .GE. 20+NEQ
C
C  RPAR, IPAR -- These are parameter arrays, of real and integer
C         type, respectively.  You can use them for communication
C         between your program that calls DDASSL and the
C         RES subroutine (and the JAC subroutine).  They are not
C         altered by DDASSL.  If you do not need RPAR or IPAR,
C         ignore these parameters by treating them as dummy
C         arguments.  If you do choose to use them, dimension
C         them in your calling program and in RES (and in JAC)
C         as arrays of appropriate length.
C
C  JAC -- If you have set INFO(5)=0, you can ignore this parameter
C         by treating it as a dummy argument.  Otherwise, you must
```

```
C         provide a subroutine of the form
C             SUBROUTINE JAC(T,Y,YPRIME,PD,CJ,RPAR,IPAR)
C         to define the matrix of partial derivatives
C             PD=DG/DY+CJ*DG/DYPRIME
C         CJ is a scalar which is input to JAC.
C         For the given values of T,Y,YPRIME, the
C         subroutine must evaluate the non-zero partial
C         derivatives for each equation and each solution
C         component, and store these values in the
C         matrix PD.  The elements of PD are set to zero
C         before each call to JAC so only non-zero elements
C         need to be defined.
C
C         Subroutine JAC must not alter T,Y,(*),YPRIME(*), or CJ.
C         You must declare the name JAC in an EXTERNAL statement in
C         your program that calls DDASSL.  You must dimension Y,
C         YPRIME and PD in JAC.
C
C         The way you must store the elements into the PD matrix
C         depends on the structure of the matrix which you
C         indicated by INFO(6).
C               *** INFO(6)=0 -- Full (dense) matrix ***
C                   Give PD a first dimension of NEQ.
C                   When you evaluate the (non-zero) partial derivative
C                   of equation I with respect to variable J, you must
C                   store it in PD according to
C                   PD(I,J) = "DG(I)/DY(J)+CJ*DG(I)/DYPRIME(J)"
C               *** INFO(6)=1 -- Banded JACOBIAN with ML lower and MU
C                   upper diagonal bands (refer to INFO(6) description
C                   of ML and MU) ***
C                   Give PD a first dimension of 2*ML+MU+1.
C                   when you evaluate the (non-zero) partial derivative
C                   of equation I with respect to variable J, you must
C                   store it in PD according to
C                   IROW = I - J + ML + MU + 1
C                   PD(IROW,J) = "DG(I)/DY(J)+CJ*DG(I)/DYPRIME(J)"
C
C         RPAR and IPAR are real and integer parameter arrays
C         which you can use for communication between your calling
C         program and your JACOBIAN subroutine JAC. They are not
C         altered by DDASSL. If you do not need RPAR or IPAR,
C         ignore these parameters by treating them as dummy
C         arguments. If you do choose to use them, dimension
C         them in your calling program and in JAC as arrays of
C         appropriate length.
C
C
C  OPTIONALLY REPLACEABLE NORM ROUTINE:
C
C     DDASSL uses a weighted norm DDANRM to measure the size
C     of vectors such as the estimated error in each step.
C     A FUNCTION subprogram
C       DOUBLE PRECISION FUNCTION DDANRM(NEQ,V,WT,RPAR,IPAR)
C       DIMENSION V(NEQ),WT(NEQ)
C     is used to define this norm. Here, V is the vector
C     whose norm is to be computed, and WT is a vector of
C     weights.  A DDANRM routine has been included with DDASSL
C     which computes the weighted root-mean-square norm
C     given by
C       DDANRM=SQRT((1/NEQ)*SUM(V(I)/WT(I))**2)
C     this norm is suitable for most problems. In some
C     special cases, it may be more convenient and/or
```

```
C     efficient to define your own norm by writing a function
C     subprogram to be called instead of DDANRM. This should,
C     however, be attempted only after careful thought and
C     consideration.
C
C
C  -------- OUTPUT -- AFTER ANY RETURN FROM DDASSL ---------------------
C
C  The principal aim of the code is to return a computed solution at
C  TOUT, although it is also possible to obtain intermediate results
C  along the way. To find out whether the code achieved its goal
C  or if the integration process was interrupted before the task was
C  completed, you must check the IDID parameter.
C
C
C  T -- The solution was successfully advanced to the
C               output value of T.
C
C  Y(*) -- Contains the computed solution approximation at T.
C
C  YPRIME(*) -- Contains the computed derivative
C               approximation at T.
C
C  IDID -- Reports what the code did.
C
C                     *** Task completed ***
C                Reported by positive values of IDID
C
C           IDID = 1 -- A step was successfully taken in the
C                   intermediate-output mode. The code has not
C                   yet reached TOUT.
C
C           IDID = 2 -- The integration to TSTOP was successfully
C                   completed (T=TSTOP) by stepping exactly to TSTOP.
C
C           IDID = 3 -- The integration to TOUT was successfully
C                   completed (T=TOUT) by stepping past TOUT.
C                   Y(*) is obtained by interpolation.
C                   YPRIME(*) is obtained by interpolation.
C
C                    *** Task interrupted ***
C                Reported by negative values of IDID
C
C           IDID = -1 -- A large amount of work has been expended.
C                   (About 500 steps)
C
C           IDID = -2 -- The error tolerances are too stringent.
C
C           IDID = -3 -- The local error test cannot be satisfied
C                   because you specified a zero component in ATOL
C                   and the corresponding computed solution
C                   component is zero. Thus, a pure relative error
C                   test is impossible for this component.
C
C           IDID = -6 -- DDASSL had repeated error test
C                   failures on the last attempted step.
C
C           IDID = -7 -- The corrector could not converge.
C
C           IDID = -8 -- The matrix of partial derivatives
C                   is singular.
C
```

```
C         IDID = -9 -- The corrector could not converge.
C                 there were repeated error test failures
C                 in this step.
C
C         IDID =-10 -- The corrector could not converge
C                 because IRES was equal to minus one.
C
C         IDID =-11 -- IRES equal to -2 was encountered
C                 and control is being returned to the
C                 calling program.
C
C         IDID =-12 -- DDASSL failed to compute the initial
C                 YPRIME.
C
C
C
C         IDID = -13,..,-32 -- Not applicable for this code
C
C                  *** Task terminated ***
C              Reported by the value of IDID=-33
C
C         IDID = -33 -- The code has encountered trouble from which
C                 it cannot recover. A message is printed
C                 explaining the trouble and control is returned
C                 to the calling program. For example, this occurs
C                 when invalid input is detected.
C
C  RTOL, ATOL -- These quantities remain unchanged except when
C             IDID = -2. In this case, the error tolerances have been
C             increased by the code to values which are estimated to
C             be appropriate for continuing the integration. However,
C             the reported solution at T was obtained using the input
C             values of RTOL and ATOL.
C
C  RWORK, IWORK -- Contain information which is usually of no
C             interest to the user but necessary for subsequent calls.
C             However, you may find use for
C
C             RWORK(3)--Which contains the step size H to be
C                     attempted on the next step.
C
C             RWORK(4)--Which contains the current value of the
C                     independent variable, i.e., the farthest point
C                     integration has reached. This will be different
C                     from T only when interpolation has been
C                     performed (IDID=3).
C
C             RWORK(7)--Which contains the stepsize used
C                     on the last successful step.
C
C             IWORK(7)--Which contains the order of the method to
C                     be attempted on the next step.
C
C             IWORK(8)--Which contains the order of the method used
C                     on the last step.
C
C             IWORK(11)--Which contains the number of steps taken so
C                      far.
C
C             IWORK(12)--Which contains the number of calls to RES
C                      so far.
C
```

```
C               IWORK(13)--Which contains the number of evaluations of
C                        the matrix of partial derivatives needed so
C                        far.
C
C               IWORK(14)--Which contains the total number
C                        of error test failures so far.
C
C               IWORK(15)--Which contains the total number
C                        of convergence test failures so far.
C                        (includes singular iteration matrix
C                        failures.)
C
C
C  -------- INPUT -- WHAT TO DO TO CONTINUE THE INTEGRATION ------------
C                    (CALLS AFTER THE FIRST)
C
C  This code is organized so that subsequent calls to continue the
C  integration involve little (if any) additional effort on your
C  part. You must monitor the IDID parameter in order to determine
C  what to do next.
C
C  Recalling that the principal task of the code is to integrate
C  from T to TOUT (the interval mode), usually all you will need
C  to do is specify a new TOUT upon reaching the current TOUT.
C
C  Do not alter any quantity not specifically permitted below,
C  in particular do not alter NEQ,T,Y(*),YPRIME(*),RWORK(*),IWORK(*)
C  or the differential equation in subroutine RES. Any such
C  alteration constitutes a new problem and must be treated as such,
C  i.e., you must start afresh.
C
C  You cannot change from vector to scalar error control or vice
C  versa (INFO(2)), but you can change the size of the entries of
C  RTOL, ATOL. Increasing a tolerance makes the equation easier
C  to integrate. Decreasing a tolerance will make the equation
C  harder to integrate and should generally be avoided.
C
C  You can switch from the intermediate-output mode to the
C  interval mode (INFO(3)) or vice versa at any time.
C
C  If it has been necessary to prevent the integration from going
C  past a point TSTOP (INFO(4), RWORK(1)), keep in mind that the
C  code will not integrate to any TOUT beyond the currently
C  specified TSTOP. Once TSTOP has been reached you must change
C  the value of TSTOP or set INFO(4)=0. You may change INFO(4)
C  or TSTOP at any time but you must supply the value of TSTOP in
C  RWORK(1) whenever you set INFO(4)=1.
C
C  Do not change INFO(5), INFO(6), IWORK(1), or IWORK(2)
C  unless you are going to restart the code.
C
C                 *** Following a completed task ***
C  If
C     IDID = 1, call the code again to continue the integration
C                   another step in the direction of TOUT.
C
C     IDID = 2 or 3, define a new TOUT and call the code again.
C                   TOUT must be different from T. You cannot change
C                   the direction of integration without restarting.
C
C                 *** Following an interrupted task ***
C               To show the code that you realize the task was
```

```
C               interrupted and that you want to continue, you
C               must take appropriate action and set INFO(1) = 1
C  If
C    IDID = -1, The code has taken about 500 steps.
C                  If you want to continue, set INFO(1) = 1 and
C                  call the code again. An additional 500 steps
C                  will be allowed.
C
C    IDID = -2, The error tolerances RTOL, ATOL have been
C                  increased to values the code estimates appropriate
C                  for continuing. You may want to change them
C                  yourself. If you are sure you want to continue
C                  with relaxed error tolerances, set INFO(1)=1 and
C                  call the code again.
C
C    IDID = -3, A solution component is zero and you set the
C                  corresponding component of ATOL to zero. If you
C                  are sure you want to continue, you must first
C                  alter the error criterion to use positive values
C                  for those components of ATOL corresponding to zero
C                  solution components, then set INFO(1)=1 and call
C                  the code again.
C
C    IDID = -4,-5  --- Cannot occur with this code.
C
C    IDID = -6, Repeated error test failures occurred on the
C                  last attempted step in DDASSL. A singularity in the
C                  solution may be present. If you are absolutely
C                  certain you want to continue, you should restart
C                  the integration. (Provide initial values of Y and
C                  YPRIME which are consistent)
C
C    IDID = -7, Repeated convergence test failures occurred
C                  on the last attempted step in DDASSL. An inaccurate
C                  or ill-conditioned JACOBIAN may be the problem. If
C                  you are absolutely certain you want to continue, you
C                  should restart the integration.
C
C    IDID = -8, The matrix of partial derivatives is singular.
C                  Some of your equations may be redundant.
C                  DDASSL cannot solve the problem as stated.
C                  It is possible that the redundant equations
C                  could be removed, and then DDASSL could
C                  solve the problem. It is also possible
C                  that a solution to your problem either
C                  does not exist or is not unique.
C
C    IDID = -9, DDASSL had multiple convergence test
C                  failures, preceeded by multiple error
C                  test failures, on the last attempted step.
C                  It is possible that your problem
C                  is ill-posed, and cannot be solved
C                  using this code. Or, there may be a
C                  discontinuity or a singularity in the
C                  solution. If you are absolutely certain
C                  you want to continue, you should restart
C                  the integration.
C
C    IDID =-10, DDASSL had multiple convergence test failures
C                  because IRES was equal to minus one.
C                  If you are absolutely certain you want
C                  to continue, you should restart the
```

```
C               integration.
C
C   IDID =-11, IRES=-2 was encountered, and control is being
C               returned to the calling program.
C
C   IDID =-12, DDASSL failed to compute the initial YPRIME.
C               This could happen because the initial
C               approximation to YPRIME was not very good, or
C               if a YPRIME consistent with the initial Y
C               does not exist. The problem could also be caused
C               by an inaccurate or singular iteration matrix.
C
C   IDID = -13,..,-32  --- Cannot occur with this code.
C
C
C              *** Following a terminated task ***
C
C  If IDID= -33, you cannot continue the solution of this problem.
C               An attempt to do so will result in your
C               run being terminated.
C
C
C  -------- ERROR MESSAGES ---------------------------------------------
C
C     The SLATEC error print routine XERMSG is called in the event of
C   unsuccessful completion of a task.  Most of these are treated as
C   "recoverable errors", which means that (unless the user has directed
C   otherwise) control will be returned to the calling program for
C   possible action after the message has been printed.
C
C   In the event of a negative value of IDID other than -33, an appro-
C   priate message is printed and the "error number" printed by XERMSG
C   is the value of IDID.  There are quite a number of illegal input
C   errors that can lead to a returned value IDID=-33.  The conditions
C   and their printed "error numbers" are as follows:
C
C   Error number       Condition
C
C        1       Some element of INFO vector is not zero or one.
C        2       NEQ .le. 0
C        3       MAXORD not in range.
C        4       LRW is less than the required length for RWORK.
C        5       LIW is less than the required length for IWORK.
C        6       Some element of RTOL is .lt. 0
C        7       Some element of ATOL is .lt. 0
C        8       All elements of RTOL and ATOL are zero.
C        9       INFO(4)=1 and TSTOP is behind TOUT.
C       10       HMAX .lt. 0.0
C       11       TOUT is behind T.
C       12       INFO(8)=1 and H0=0.0
C       13       Some element of WT is .le. 0.0
C       14       TOUT is too close to T to start integration.
C       15       INFO(4)=1 and TSTOP is behind T.
C       16       --( Not used in this version )--
C       17       ML illegal.  Either .lt. 0 or .gt. NEQ
C       18       MU illegal.  Either .lt. 0 or .gt. NEQ
C       19       TOUT = T.
C
C   If DDASSL is called again without any action taken to remove the
C   cause of an unsuccessful return, XERMSG will be called with a fatal
C   error flag, which will cause unconditional termination of the
C   program.  There are two such fatal errors:
```

```
C
C   Error number -998:  The last step was terminated with a negative
C       value of IDID other than -33, and no appropriate action was
C       taken.
C
C   Error number -999:  The previous call was terminated because of
C       illegal input (IDID=-33) and there is illegal input in the
C       present call, as well.  (Suspect infinite loop.)
C
C ---------------------------------------------------------------------
C***END PROLOGUE  DDASSL
```