ADAPTIVE TWO-TIME-SCALE POSITION CONTROL OF MULTIPLE FLEXIBLE MANIPULATORS

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCES

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Murat Zeren

September 1994

ADAPTIVE TWO-TIME-SCALE POSITION CONTROL OF MULTIPLE FLEXIBLE MANIPULATORS

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND

ELECTRONICS ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCES

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Murat Zeren

September 1994

Murat ZEREN

tarafından [illegible]

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Erol Sezer(Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Ömer Morgül

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Billur Barshan

Approved for the Institute of Engineering and Sciences:

Prof. Dr. Mehmet Baray
Director of Institute of Engineering and Sciences

ABSTRACT

ADAPTIVE TWO-TIME-SCALE POSITION CONTROL OF MULTIPLE FLEXIBLE MANIPULATORS

Murat Zeren
M.S. in Electrical and Electronics Engineering
Supervisor: Prof. Dr. Erol Sezer
September 1994

An adaptive self-tuning control scheme is developed in the independent generalized coordinates for position control of multiple flexible manipulators. First, a suitable model is derived for multiple cooperating manipulators holding a common workpiece. Then flexibility is included in the model where each flexible link is modelled by two rigid sublinks connected by a fictitious joint. The relatively high stiffness of the flexible links is used to decompose the model into two subsystems operating at different rates, and a two-time-scale control is applied to achieve the desired position tracking. Finally adaptive estimation of the manipulator parameters is proposed to achieve a control scheme which is suitable for on-line applications.

Keywords : multiple flexible manipulators, two-time-scale control, adaptive estimation.

ÖZET

ÇOK SAYIDA ESNEK ROBOT KOL İÇİN UYUMLU İKİ ZAMANLI BİR KONUM DENETİMİ

Murat Zeren
Elektrik ve Elektronik Mühendisliği Bölümü Yüksek Lisans
Tez Yöneticisi: Prof. Dr. Erol Sezer
Eylül 1994

Çok sayıda esnek kolun bağımsız genel koordinatlarda modeli elde edilerek, konum denetimi için uyarlamalı bir denetim yöntemi geliştirilmiştir. Önce, ortak bir cismi tutan çok sayıda robot kolu için uygun bir dinamik model çıkarılmış; ardından, her esnek kol yapay bir eklemle birleştirilmiş iki yarı koldan oluşmuş varsayılarak, esneklik oluşturulmuş olan bu modele dahil edilmiştir. Esnek kolların esneme katsayılarının nispeten yüksek oluşundan yararlanılarak, model farklı hızlarda calışan iki yarı sisteme ayrıştırılmış ve istenen konum denetimini sağlamak için iki zamanlı bir denetim uygulanmıştır. Son olarak, oluşturulan metodun gerçek zamanda uygulanabilirliğini sağlamak için robot kol parametrelerinin bir kestirim algoritması ile hesaplanması önerilmiştir.

Anahtar kelimeler : çok sayıda esnek kol, iki zamanli denetim, uyumlu hesaplama.

ACKNOWLEDGMENT

I would like to thank Prof. Dr. M. Erol Sezer for his supervision, guidance, suggestions and encouragement throughout the development of this thesis.

Contents

List of Figures

Chapter 1

Introduction

The present age, with the fast and steady progress in science and technology, has witnessed a large number of new tools developed to make life easy for mankind. In the field of manufacturing, due in large part to recent advances in computer technology, a revolution in automation is evolving. Human workers are being replaced by machines, increasing the productivity and decreasing the production cost.

Increasingly, production has come to mean the introduction of robots into the manufacturing environment. Relative to its role in manufacturing, a robot is defined to be a "reprogrammable multifunctional manipulator designed to move material, parts, tools or specialized devices through variable programmed motions for the performance of a variety of tasks"[1].

After being introduced as a manufacturing tool, robots started to be employed in various other places like handling of hazardous materials in some plants, or as remotely controlled manipulators in deep sea or space explorations. Though we are still quite far from Isaac Asimov's human-like robots, the fact is that robots are gaining more and more importance everyday with the advances in computer technology and the increase in computing power. Robotics has appeared as a new field of modern technology. Moreover, this

[1]Robot Institute of America

field crosses the boundaries of traditional engineering, requiring the knowledge of electrical engineering, mechanical engineering, industrial engineering, computer science and mathematics to fully understand the complexity of robots and their applications.

The expectations from robots and the current trend of technology carried the research on robotics to many different dimensions. This is where the notion of multiple cooperating robot arms originated. In fact it is an everyday scenario which prepares the setting for multiple arms. We, as humans, have two arms and everyday manual work is normally performed by two-handed humans. In fact, manual activities and tasks are normally perceived and designed such that they assume two-handed humans; and if robots are to replace humans in normal manual activities, it seems natural to visualize and design robots with two arms and hands. When two or more arms are used to perform a single task together, as is the case in humans, an increased load carrying, handling and manipulation capacity can be achieved.

The problem of multiple cooperating robot arms have been studied by many investigators. To name a few of the recent work in this area; Tarn, Bejczy and Yun [1] developed a nonlinear control algorithm for multiple robot arms. They have first derived the Lagrange's equation of motion for the closed chain mechanical system by using the Lagrangian of the whole mechanical system. Obtaining a nonlinear, coupled and complicated model, their proposal was to linearize and output decouple the system using a nonlinear feedback and a nonlinear coordinate transformation. Next, assuming that each robot has a force/torque sensor installed at its end-effector, they extended this scheme to force control. Hayati [2] has also developed a position and force control for coordinated multiple robot arms which are holding a single object. In this study, starting with the equation of motion for a single multi-link arm, and using the fact that the sum of forces at the origin of a compliant frame chosen

in the common load is zero, he obtained a formulation to solve for the forward dynamics and interaction forces of the multiple arm system. Then he extended this formulation to a setting in which there exists external constraints imposed by the working environment, and finally he proposed a hybrid control law on generalized forces to yield position tracking and desired interaction forces. In a more recent study, Hsu[3] proposed a coordinated control scheme for multiple manipulator systems. The objective was to perform a parts-matching task. Deriving first the dynamic equation of parts matching system and of manipulators, he combined all into a complete system equation. Then, a control law on generalized coordinates was proposed, which could achieve trajectory tracking as well as achieving the desired internal forces and also the load distribution to each manipulator according to a weighting factor. The implemented system and the experimental results were also reported. Another study by Jean and Fu [5] describes an adaptive control scheme for coordinated multi-manipulator systems. In this study, they have first derived the so called augmented object model for cooperating robots, and then using the fact that this dynamic equation formulation is linear in the dynamic parameters of individual robots and the commonly held object, they converted the dynamic model into a linear regression form. Next, modifying the control scheme previously reported by Hsu [3, 4] to fit their new model, they proposed an adaptive coordinated control. The method is concluded with a discussion of the convergence properties of the method. This control, as well as the one proposed by Hsu, is interesting in the sense that they facilitated the asymptotic tracking of both the object position and internal forces without the need for the measurement of contact forces.

Apart from the many studies investigating the coordinated control of multiple manipulator systems, there also exists some other studies investigating other issues related to multiple manipulator systems. In a study by Li, Tarn and Bejczy [6], they concentrated on the dynamic workspace analysis of multiple cooperating robot arms. In order to fully utilize the benefits obtained

by the use of multiple manipulators, it is necessary to know the maximum force/torque that the robots can jointly apply to the environment within their coupled workspace. This problem was introduced as a maximization problem and a solution was proposed. As multiple manipulator systems are redundant in nature, i.e. they have more actuators then degrees of freedom, there exists some other studies related to optimality on the basis of different cost criteria. A study by Bobrow, McCarthy and Chu [7] presents an algorithm which minimizes the time for two robots holding the same workpiece to move along a given path. Simulation, being an important tool in design and testing of new control algorithms, has also been investigated related to cooperating multiple manipulators. An example is a study carried out by Murphy, Wen and Saridis [8], in which the simulation aspects of cooperating robot manipulators on a mobile platform were investigated.

Putting aside coordinated control of multiple manipulators, another research direction appearing with the current trends in robotics is the modelling and control of lightweight, flexible structures. Lightweight arms are needed to meet the demands of industry for robots that move faster without requiring high-powered bulky actuators. As manipulator arms are made lighter their deformation under stress increases. Thus design of both the mechanical and control systems require accurate models for simulation and manipulation of flexible manipulators under a variety of different operating conditions, without requiring expensive computing machinery.

Based on well-known approaches to analyze the elastic and dynamic behaviour of flexible structures, a variety of different methods are applied in the field of flexible robots. A survey on this topic exists in [9]. Generally these methods can be classified into the following categories:

- kinetostatic methods
- dynamic methods:

finite element methods

vibration mode approach

The considerations leading to kinetostatic methods start with modelling the robot as a system of rigid bodies, and due to inertial forces and loads small elastic deformations are superimposed to the basic rigid body motion. The elastic deformations of a link in this model correspond to those deformations induced by external loads applied at proper positions.

In finite element methods, the flexible parts of the robot are modelled by a finite number of elements, their number depending upon the shape of the part and the properties attached to an element. In order to describe the vibrations of the link structure, a mass or better a mass matrix has to be attached to each finite element. Here this mass matrix can possibly be found by using the kinetic energy of the element described by a homogeneous mass distribution and local displacements computed by the nodal displacements and static shape functions.

As to the vibration mode approach, an analytical description of possible vibration modes of a flexible part can be derived directly from the partial differential equations and the boundary conditions used for the description of time-spatial bending of a flexible beam. The main difficulty with this method lies in choosing the mode shapes and the number of modes that would accurately describe link deflections under time-dependent boundary conditions. In practice, the infinite series that results from solving the Bernoulli-Euler partial differential equations is truncated to an arbitrary number of terms. An experimental study made by Hastings and Book [11] (in which they also used the separability of the modes) showed that the frequencies of vibratory modes in the experiment closely matched the ones in simulation. Here, separability refers to expanding modes as products of two functions, one being a function of the spatial variable only and the other being a function of time only.

To handle flexibility, Nemir, Koivo and Kashyap [12] developed the concept of a pseudolink. A *pseudolink* is an imaginary line connecting the hub of a flexible link to its tip. Assuming that the normalized deflection at the tip is small and that the length of the pseudolink is the same as the length of undeformed link, they wrote the deflection of the pseudolink as a summation of two terms, one being the true joint displacement and the other being the angle of pseudolink relative to rigid link position. This second term was expressed as the weighted sum of some basis functions which were used to describe the link deflection. It was observed that best results were obtained by using the Cantilever eigenfunctions as the basis functions. Another study by King, Gourishankar and Rink [13] made use of this pseudolink concept, and by decomposing deformations into fast and slow components, they developed a two-time-scale decoupled control for the end-point position tracking. In a more recent study by Bodur and Sezer [14], a second order bulk model, consisting of two rigid sublinks joined by a fictitious joint, is used to approximate the dynamics of a flexible link. Elastic potential energy stored in the links as they bend is modeled by the modal stiffness coefficients at the fictitious joints and a small damping coefficient is also included to account for the damping of the vibrations. Using the relatively high stiffness of the fictitious joints, the model is decomposed into two subsystems operating at different rates and a two-time-scale control is proposed. The proposed scheme is in end-point coordinates and has the advantage of being adaptive. An experimental study by Yang, Yang and Kudva [15] considered the load-adaptive control of a single-link flexible manipulator. In this study, by using a discrete-time ARMA model to describe the linearized input/output description of the link, they experimented a least squares identification algorithm and a self-tuning pole placement controller.

In this thesis, the aim is to propose a control scheme for the coordinated

position control of multiple manipulator systems having flexible links. Flexibility is modeled as in [14] and [16], and the governing dynamics of a multi-manipulator system are derived. Next, using the singular perturbations approach, the system is decomposed into slow and fast subsystems, and finally, a two-time-scale adaptive control is proposed. Chapter two explains the derivation of the multiple manipulator dynamics and combines it with flexibility to derive a general model. Chapter three introduces the model decomposition and the proposed control, along with the identification technique. Chapter four is devoted to the simulation results of the proposed control, using two dual planar manipulators each having one flexible link and holding a common load. Chapter five gives the conclusions and proposes directions for further research.

Chapter 2

Modelling and Dynamics

A robot manipulator is in fact a multi-body system. It consists of links which are connected at the joints with the first link being connected to the base and the end-point being in touch with the environment to perform a desired task. Depending on the task to be carried out by the manipulator and the environment in which it functions, the joints of the manipulator can be prismatic, allowing linear displacements, or revolute, allowing angular displacements. The motion is provided by actuators which change the joint displacements so as to change the position of the end-effector. In order to control this system to carry out a specific task, we need to have a dynamic model which relates the torque/force inputs applied at the joints to the resulting displacements at the joints or the position and the orientation of the end-effector.

2.1 Dynamic Model of a Multiple-Joint Manipulator

Since a manipulator is a mechanical device, the laws governing the motion of any mechanism is directly applicable to the manipulators. *Principle of d'Alembert* states that the sum of all (generalized) torques applied at a joint should be zero. *Newton's law* relates net forces to linear acceleration of masses, whereas *Euler's law* relates torques to angular velocity and acceleration of a

rotational moment of inertia. The generalized torques corresponding to the generalized displacements in the mechanism can be evaluated using the Euler-Lagrange relations involving the derivatives, with respect to displacements and time, of the *Lagrangian*, which is the difference of kinetic and potential energies. The equation of motion of a manipulator is obtained using these laws and the constraints of kinematics. Depending on the evaluation of the equations, they are called *Euler-Lagrange* (or Lagrangian), *Newton-Euler* or *Generalized d'Alembert* formulations.

For many applications, the dynamic model can conveniently be obtained from the Lagrangian function using the Euler-Lagrange equations. Below we present a brief summary of the derivations of the Euler-Lagrange equations. Details can be found in [17],[18].

The kinetic energy of the i-th link of an n-link manipulator is given as

$$K_i = \frac{1}{2}\dot{\mathbf{p}}_{0i}^T \mathbf{I}_i \dot{\mathbf{p}}_{0i} \tag{2.1}$$

where $\mathbf{p}_{0i}$ is the position vector of the origin of the i-th coordinate frame attached to the i-th link with respect to the base coordinates and $\mathbf{I}_i$ is the pseudo-inertia matrix of the link given as

$$\mathbf{I}_i = \begin{bmatrix} \frac{1}{2}(-I_{x_i} + I_{y_i} + I_{z_i}) & I_{x_i y_i} & I_{x_i z_i} & \bar{p}_{x_i} m_i \\ I_{x_i y_i} & \frac{1}{2}(I_{x_i} - I_{y_i} + I_{z_i}) & I_{y_i z_i} & \bar{p}_{y_i} m_i \\ I_{x_i z_i} & I_{y_i z_i} & \frac{1}{2}(I_{x_i} + I_{y_i} - I_{z_i}) & \bar{p}_{z_i} m_i \\ \bar{p}_{x_i} m_i & \bar{p}_{y_i} m_i & \bar{p}_{z_i} m_i & m_i \end{bmatrix} \tag{2.2}$$

In equation (2.2), m_i is the total mass of the link, $\underline{\mathbf{p}}_i = [p_{x_i}, p_{y_i}, p_{z_i}, 1]^T$ is the center of gravity of the link expressed in its own coordinate frame, and

$$I_{x_i} = \int (p_{y_i}^2 + p_{z_i}^2) dm_i \quad I_{y_i} = \int (p_{x_i}^2 + p_{z_i}^2) dm_i \quad I_{z_i} = \int (p_{y_i}^2 + p_{z_i}^2) dm_i$$
$$I_{wv} = \int p_w p_v dm_i \quad w \neq v \quad w, v = x_i, y_i, z_i \tag{2.3}$$

are the second moments, where the integration is over the mass of the link. All

quantities in equation (2.2) are independent of the positions and velocities of the joint variables.

The link position vectors in equation (2.1) are related to the joint variables as

$$\mathbf{p}_{0i} = \mathbf{A}_0^i \mathbf{e} \tag{2.4}$$

where $\mathbf{A}_0^i$ is the homogenous transformation matrix that relates the i-th coordinate frame to the base coordinate system, and is a function of the joint variables $\theta_1, \theta_2, \ldots, \theta_i$; and $\mathbf{e}$ is a constant vector that picks the last column of $\mathbf{A}_0^i$. Taking the time derivative of both sides of equation (2.4), substituting in equation (2.1), and summing over all links, the total kinetic energy of the manipulator is obtained as

$$\begin{aligned} K &= \frac{1}{2}\sum_{i=1}^{n}\left[\sum_{j=1}^{i}\mathbf{e}^T\left(\frac{\partial \mathbf{A}_0^i}{\partial \theta_j}\right)^T \dot{\theta}_j\right]\mathbf{I}_i\left[\sum_{k=1}^{i}\left(\frac{\partial \mathbf{A}_0^i}{\partial \theta_k}\,\mathbf{e}\,\dot{\theta}_k\right)\right] \\ &= \frac{1}{2}\sum_{i=1}^{n}\,\mathrm{tr}\left[\sum_{j=1}^{i}\sum_{k=1}^{i}\frac{\partial \mathbf{A}_0^i}{\partial \theta_j}\mathbf{I}_i\left(\frac{\partial \mathbf{A}_0^i}{\partial \theta_k}\right)^T \dot{\theta}_j\dot{\theta}_k\right] \end{aligned} \tag{2.5}$$

where $\mathrm{tr}(\cdot)$ denotes the trace of the indicated matrix.

The total potential energy of the manipulator is simply

$$P = -\sum_{i=1}^{n} m_i \mathbf{g}^T \mathbf{A}_0^i \mathbf{p}_i \tag{2.6}$$

where the vector $\mathbf{g} = [g_{0x}, g_{0y}, g_{0z}, 0]^T$ describes the gravitational acceleration with components in the directions of the coordinates (x_0, y_0, z_0) of the base coordinate system.

With $\mathcal{L} = K - P$ denoting the Lagrange energy function of the manipulator, the equations of motion are obtained by means of Euler-Lagrange's equations:

$$\frac{d}{dt}\left[\frac{\partial \mathcal{L}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}, t)}{\partial \dot{\theta}_i}\right] - \frac{\partial \mathcal{L}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}, t)}{\partial \theta_i} = \tau_i, \qquad i = 1, \ldots, n. \tag{2.7}$$

where $\boldsymbol{\theta} = [\theta_1, \ldots, \theta_n]^T$ is the vector of joint coordinates and τ_i is the joint force/torque acting at the joint in the direction of positive joint displacement.

Now, substituting equations (2.5) and (2.6) in (2.7) we obtain the equation of motion of the i-th link as

$$\sum_{k=1}^{n} m_{ik}\ddot{q}_k + \sum_{k=1}^{n}\sum_{j=1}^{n} c_{ikj}\dot{q}_k\dot{q}_j + g_i = \tau_i \tag{2.8}$$

where

$$\begin{aligned} m_{ik} &= \sum_{l=max(i,k)}^{n} \mathrm{tr}\left[\frac{\partial \mathbf{A}_0^l}{\partial \theta_i}\mathbf{I}_l\left(\frac{\partial \mathbf{A}_0^l}{\partial \theta_k}\right)^T\right] \\ c_{ikj} &= \sum_{l=max(i,k,j)}^{n} \mathrm{tr}\left[\frac{\partial \mathbf{A}_0^l}{\partial \theta_i}\mathbf{I}_l\left(\frac{\partial^2 \mathbf{A}_0^l}{\partial \theta_k \partial \theta_j}\right)^T\right] \\ g_i &= -\sum_{l=1}^{n} m_l \mathbf{g}^T \frac{\partial \mathbf{A}_0^l}{\partial \theta_i}\mathbf{p}_l \end{aligned} \tag{2.9}$$

In this equation, the first term describes the forces due to the acceleration of the inertial masses in the system. The second term represents the Coriolis ($k \neq j$) and centripetal ($k = j$) forces. The last term is to account for the gravitational effect on the motion of joint i.

As the last step, these equations can be combined into a matrix form and can be rewritten as a set of second-order vector differential equation

$$\mathbf{M}(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}} + \mathbf{c}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) + \mathbf{g}(\boldsymbol{\theta}) = \boldsymbol{\tau} \tag{2.10}$$

where the matrix $\mathbf{M}$ is defined to be the mass matrix of robot, $\mathbf{c}$ being the vector of Coriolis and centrifugal terms, $\mathbf{g}$ signifying the effect of gravity and $\boldsymbol{\tau}$ being the vector of joint forces/torques.

2.2 Modelling of Multiple Cooperating Manipulators

There exists many settings in which multiple manipulators work and must work together in a coordinated fashion to accomplish some given task. Often, these manipulators grasp a single object for the purpose of lifting it, turning it or carrying it to some other place; forming a closed chain.

The first concept that has to be introduced in the formulation of a closed chain is the number of degrees of freedom. *Degree of freedom* (D.O.F.) of a system is defined to be the number of independent variables one can pick to describe the entire configuration of the system at hand. In an open chain, D.O.F. is the same as the number of joints, since each joint can move freely without any constraint. However, in a closed-chain structure there exists constraints imposed by the closedness of the system. Thus, D.O.F. is reduced compared to the sum of D.O.F.'s of individual structures forming the closed chain. According to the Kutzbach-Grubler criterion [1], the D.O.F. of a spatial linkage structure with n links each having d_l degrees of freedom, and m joints each having one degree of freedom is given as

$$d = nd_l - m(d_l - 1) \tag{2.11}$$

This formulation reflects the fact that each joint of one D.O.F. causes a loss of $(d_l - 1)$ D.O.F. for a link.

Let us consider a closed chain formed by N manipulators holding a single object. Assume that k-th manipulator has m_k joints and links. Then the D.O.F. of the entire system is obtained as

$$\begin{aligned} d &= d_l \left[\sum_{k=1}^{N} m_k + 1 \right] - (d_l - 1) \sum_{k=1}^{N} m_k - r_l N \\ &= \sum_{k=1}^{N} m_k + d_l - r_l N \end{aligned} \tag{2.12}$$

where d_l is the D.O.F. of each link and r_l, depending on the type of contact, is the D.O.F. lost at each joint connecting a manipulator to the workpiece. For a planar configuration $d_l = 3$, and for a spatial configuration $d_l = 6$. This means that, we can find a set of d independent variables, called the *independent generalized coordinates* to describe the configuration of the entire system. In [1], the dynamics of the closed-chain mechanical system was derived by obtaining the Lagrangian of the whole system and applying the Euler-Lagrange equations to get the system equation in generalized coordinates and

generalized forces. Though being quite automatic, the interaction forces never appear in this formulation. Here, a different approach is taken to derive the dynamics of the individual parts in the system, viewing the interaction forces as well.

Let $\boldsymbol{\theta}_k, k = 1, \ldots, N$, be the joint variables defining the position of kth manipulator and similarly $\boldsymbol{\theta}_0$ be an equivalent set defining the position of the commonly held object. The Cartesian position of some point $\mathbf{p}$ in the object can be written as a function of the object coordinates $\boldsymbol{\theta}_0$ alone, $\mathbf{p} = \mathbf{p}(\boldsymbol{\theta}_0)$, or as a function of the end-effector position of any arm and the orientation of the work-piece, i.e.,

$$\mathbf{p}(\boldsymbol{\theta}_0) = \mathbf{p}_k(\boldsymbol{\theta}_k, \boldsymbol{\theta}_0), \qquad k = 1, 2, \ldots, N \tag{2.13}$$

These closure equations represent the geometric constraints on the system. Differentiating the above closure equations with respect to time gives

$$\mathbf{J}_0\dot{\boldsymbol{\theta}}_0 = \mathbf{J}_k\dot{\boldsymbol{\theta}}_k + \mathbf{J}_{0k}\dot{\boldsymbol{\theta}}_0, \qquad k = 1, 2, \ldots, N \tag{2.14}$$

where $\mathbf{J}_0$ is the Jacobian of $\mathbf{p}(\boldsymbol{\theta}_0)$ with respect to $\boldsymbol{\theta}_0$; $\mathbf{J}_k$, the Jacobian of $\mathbf{p}_k$ with respect to $\boldsymbol{\theta}_k$; and $\mathbf{J}_{0k}$, the Jacobian of $\mathbf{p}_k$ with respect to $\boldsymbol{\theta}_0$.

With these in mind, view the robot arms and the object as separate mechanical systems. The equations of motion for these mechanical systems have the form

$$\mathbf{M}_k(\boldsymbol{\theta}_k)\ddot{\boldsymbol{\theta}}_k + \mathbf{h}_k(\boldsymbol{\theta}_k, \dot{\boldsymbol{\theta}}_k) = \boldsymbol{\tau}_k + \mathbf{v}_k, \qquad k = 1, \ldots, N \tag{2.15}$$

for the robots, where $\mathbf{M}_k$'s are the mass matrices and $\mathbf{h}_k$'s collect the Coriolis, centrifugal and gravity terms. Here $\boldsymbol{\tau}_k$ denotes the joint forces/torques in the kth system and $\mathbf{v}_k$ represent the generalized interaction forces acting on the kth system. The equations of motion for the object is given by,

$$\mathbf{M}_0(\boldsymbol{\theta}_0)\ddot{\boldsymbol{\theta}}_0 + \mathbf{h}_0(\boldsymbol{\theta}_0, \dot{\boldsymbol{\theta}}_0) = \mathbf{v}_0 \tag{2.16}$$

through a similar formulation, but this time $\boldsymbol{\tau}_0$ is not included, as the object

is moved only by interaction forces rather than the independent force/torque drives. The coupling among the above equations is obtained through $\mathbf{v}_k$'s.

In order to relate the constraint forces to the dynamics of the system, we use the fact that constraint forces do no work. Defining, $\mathbf{v} = [\mathbf{v}_0^T, \mathbf{v}_1^T, \cdots, \mathbf{v}_N^T]^T$ and $\boldsymbol{\theta} = [\boldsymbol{\theta}_0^T, \boldsymbol{\theta}_1^T, \cdots, \boldsymbol{\theta}_N^T]^T$, we have,

$$\mathbf{v}^T \dot{\boldsymbol{\theta}} = 0 \tag{2.17}$$

for all admissible velocities $\dot{\boldsymbol{\theta}}$. If there were no constraints on the individual systems, as is in the open-chain case, the velocities $\dot{\boldsymbol{\theta}}_k$ of each system could vary arbitrarily and the above equation shows that the forces of interaction, $\mathbf{v}_k$, would be zero.

The joint velocities $\dot{\boldsymbol{\theta}}$ must also satisfy the constraints in equation (2.14), or putting them into matrix form

$$\mathbf{J}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} = 0 \tag{2.18}$$

where,

$$\mathbf{J}(\boldsymbol{\theta}) = \begin{bmatrix} \mathbf{J}_0 - \mathbf{J}_{01} & -\mathbf{J}_1 & & \\ \vdots & & \ddots & \\ \mathbf{J}_0 - \mathbf{J}_{0N} & & & -\mathbf{J}_N \end{bmatrix}$$

Now, equation (2.18) shows that $\dot{\boldsymbol{\theta}}$ is in the null space of $\mathbf{J}(\boldsymbol{\theta})$, and equation (2.17) shows that $\mathbf{v}$ is orthogonal to $\dot{\boldsymbol{\theta}}$. Thus, we conclude that $\mathbf{v}$ is in the range space of $\mathbf{J}^T(\boldsymbol{\theta})$ or that, there exists a vector $\mathbf{f} \neq \mathbf{0}$ such that,

$$\mathbf{v} = \mathbf{J}^T(\boldsymbol{\theta})\mathbf{f}. \tag{2.19}$$

Here $\mathbf{f}$ represents the internal forces built in the closed chain mechanical system. If we partition $\mathbf{f}$ as $\mathbf{f} = [\mathbf{f}_1^T, \cdots, \mathbf{f}_N^T]^T$, with $\mathbf{f}_k, k = 1, \ldots, N$ representing the internal forces acting on the tip of each robot, individual system equations for each robot can be written as

$$\mathbf{M}_k \ddot{\boldsymbol{\theta}}_k + \mathbf{h}_k(\boldsymbol{\theta}_k, \dot{\boldsymbol{\theta}}_k) = \boldsymbol{\tau}_k - \mathbf{J}_k^T \mathbf{f}_k, \qquad k = 1, \ldots, N \tag{2.20}$$

and for the object as

$$\mathbf{M}_0\ddot{\boldsymbol{\theta}}_0 + \mathbf{h}_0(\boldsymbol{\theta}_0, \dot{\boldsymbol{\theta}}_0) = \mathbf{J}_{00}^T\mathbf{f} \tag{2.21}$$

where

$$\mathbf{J}_{00} = [(\mathbf{J}_0 - \mathbf{J}_{01})^T, \cdots, (\mathbf{J}_0 - \mathbf{J}_{0N})^T]^T$$

Now, let's pick a set $\mathbf{q}$ of d independent generalized coordinates describing the configuration of the whole system, where d is the D.O.F. of the system. Then, we know that, since the knowledge of $\mathbf{q}$ determines the configuration of the whole system, they also uniquely determine the joint variables $\boldsymbol{\theta}_k, k = 0, 1, \ldots, N$. So, we have a relation of the form

$$\boldsymbol{\theta}_k = \boldsymbol{\Theta}_k(\mathbf{q}) \tag{2.22}$$

for each subsystem. Here, it is worth mentioning that while choosing the set $\mathbf{q}$, our main concern is the type of control problem we have at hand. For a position tracking problem, it is desirable to put the variables to be controlled into set $\mathbf{q}$.

Next, we convert the dynamic equations of individual subsystems given in their own joint coordinates into generalized coordinates using the derivatives of equation (2.22) as

$$\begin{aligned} \dot{\boldsymbol{\theta}}_k &= \mathbf{J}_{\boldsymbol{\theta}_k}(\mathbf{q})\dot{\mathbf{q}} \\ \ddot{\boldsymbol{\theta}}_k &= \mathbf{J}_{\boldsymbol{\theta}_k}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{J}}_{\boldsymbol{\theta}_k}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}, \qquad k = 1, \ldots, N \end{aligned} \tag{2.23}$$

where $\mathbf{J}_{\boldsymbol{\theta}_k}$ is the Jacobian of $\boldsymbol{\theta}_k$ with respect to $\mathbf{q}$, and $\dot{\mathbf{J}}_{\boldsymbol{\theta}_k}$ is an $n_k \times d$ matrix with the (i,j)-th element given as $\sum_{l=1}^{d} \frac{\partial^2 \Theta_{ki}}{\partial q_j \partial q_l}\dot{q}_l$. Inserting these into equations (2.20) and (2.21) above, we have

$$\tilde{\mathbf{M}}_k(\mathbf{q})\ddot{\mathbf{q}} + \tilde{\mathbf{h}}_k(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{\tau}_k - \mathbf{J}_k^T\mathbf{f}_k, \qquad k = 1, \ldots, N \tag{2.24}$$

$$\tilde{\mathbf{M}}_0(\mathbf{q})\ddot{\mathbf{q}} + \tilde{\mathbf{h}}_0(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{J}_{00}^T\mathbf{f} \tag{2.25}$$

where

$$\begin{aligned}\tilde{\mathbf{M}}_k(\mathbf{q}) &= \mathbf{M}_k\left(\Theta_k(\mathbf{q})\right)\mathbf{J}_{\theta_k}(\mathbf{q}) \\ \tilde{\mathbf{h}}_k(\mathbf{q},\dot{\mathbf{q}}) &= \mathbf{h}_k\left(\Theta_k(\mathbf{q}),\mathbf{J}_{\theta_k}(\mathbf{q})\dot{\mathbf{q}}\right) + \mathbf{M}_k\left(\Theta_k(\mathbf{q})\right)\dot{\mathbf{J}}_{\theta_k}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}, \qquad k = 0,1,\ldots,N.\end{aligned}$$

Now, we have all the dynamics formulated in the generalized coordinates. To proceed, note that the matrix $\mathbf{J}_{00}$ is nonsquare and is in fact of full column rank. So, using the pseudo-inverse of $\mathbf{J}_{00}^T$, we solve for the minimum norm $\mathbf{f}$ from equation (2.25) as

$$\mathbf{f} = (\mathbf{J}_{00}^T)^\dagger[\tilde{\mathbf{M}}_0(\mathbf{q})\ddot{\mathbf{q}} + \tilde{\mathbf{h}}_0(\mathbf{q},\dot{\mathbf{q}})] \tag{2.26}$$

where

$$(\mathbf{J}_{00}^T)^\dagger = \mathbf{J}_{00}(\mathbf{J}_{00}^T\mathbf{J}_{00})^{-1}$$

This $\mathbf{f}$ vector is that particular $\mathbf{f}$ vector which achieves the positioning of the workpiece without building extra internal forces on it, and thus the $\mathbf{f}$ vector which we will use in constructing the computed torque input. With $(\mathbf{J}_{00}^T)_k^\dagger$ denoting the k-th block row of $(\mathbf{J}_{00}^T)^\dagger$, we have

$$\mathbf{f}_k = (\mathbf{J}_{00}^T)_k^\dagger[\tilde{\mathbf{M}}_0(\mathbf{q})\ddot{\mathbf{q}} + \tilde{\mathbf{h}}_0(\mathbf{q},\dot{\mathbf{q}})] \tag{2.27}$$

Finally, substituting this into equation (2.24),

$$\hat{\mathbf{M}}_k(\mathbf{q})\ddot{\mathbf{q}} + \hat{\mathbf{h}}_k(\mathbf{q},\dot{\mathbf{q}}) = \tau_k, \qquad k = 1,\ldots,N \tag{2.28}$$

where

$$\begin{aligned}\hat{\mathbf{M}}_k &= \tilde{\mathbf{M}}_k(\mathbf{q}) + \mathbf{J}_k^T(\mathbf{J}_{00}^T)_k^\dagger\tilde{\mathbf{M}}_0(\mathbf{q}) \\ \hat{\mathbf{h}}_k(\mathbf{q},\dot{\mathbf{q}}) &= \tilde{\mathbf{h}}_k(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{J}_k^T(\mathbf{J}_{00}^T)_k^\dagger\tilde{\mathbf{h}}_0(\mathbf{q},\dot{\mathbf{q}})\end{aligned}$$

Combing these final system equations of individual robots into a larger matrix formulation, we obtain

$$\begin{bmatrix}\hat{\mathbf{M}}_1(\mathbf{q}) \\ \vdots \\ \hat{\mathbf{M}}_N(\mathbf{q})\end{bmatrix}\ddot{\mathbf{q}} + \begin{bmatrix}\hat{\mathbf{h}}_1(\mathbf{q},\dot{\mathbf{q}}) \\ \vdots \\ \hat{\mathbf{h}}_N(\mathbf{q},\dot{\mathbf{q}})\end{bmatrix} = \begin{bmatrix}\tau_1 \\ \vdots \\ \tau_N\end{bmatrix} \tag{2.29}$$

or simply

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{h}(\mathbf{q},\dot{\mathbf{q}}) = \tau \tag{2.30}$$

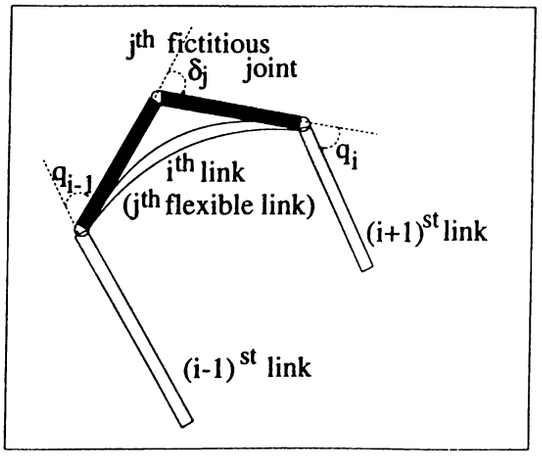


Figure 2.1: *Second-order bulk model of a flexible link*

2.3 Multiple Cooperating Manipulators Having Flexible Links

As introduced earlier, the need in the industry for lightweight manipulators resulted in the production of robots with flexible links. To solve the problem of dynamic modelling of these links various methods have been proposed. In this section, a modelling scheme for multiple manipulators having flexible links is discussed. The dynamics of a flexible link is approximated by a second-order bulk model, which consists of two rigid sublinks joined by a fictitious joint as in [14] and [16]. In this model, the angular displacement of the fictitious joint is a measure of the deflection of the link. Elastic potential energy stored in the links as they bend is modelled by the modal stiffness coefficients at the fictitious joints, and a small damping coefficient is also included to account for the damping of the vibrations.

Let's consider a single manipulator consisting of n links, r of which are flexible. If we model these links by a second-order bulk model as described above, the dynamical model of such a manipulator can easily be obtained by

using the Euler-Lagrange equations as

$$\mathbf{M}(\boldsymbol{\theta},\boldsymbol{\delta})\begin{bmatrix}\ddot{\boldsymbol{\theta}}\\ \ddot{\boldsymbol{\delta}}\end{bmatrix}+\mathbf{c}(\boldsymbol{\theta},\boldsymbol{\delta},\dot{\boldsymbol{\theta}},\dot{\boldsymbol{\delta}})+\mathbf{g}(\boldsymbol{\theta},\boldsymbol{\delta})=\begin{bmatrix}\boldsymbol{\tau}_{\boldsymbol{\theta}}\\ \boldsymbol{\tau}_{\boldsymbol{\delta}}\end{bmatrix} \tag{2.31}$$

where $\boldsymbol{\theta} \in \mathbf{R}^n$ are the true joint displacements and $\boldsymbol{\delta} \in \mathbf{R}^r$ are the fictitious joint displacements. The actuator forces/torques are external inputs to the system and are collected in $\boldsymbol{\tau}_{\boldsymbol{\theta}}$. The torques of fictitious joints, however, are determined by the internal dynamics of the flexible link as

$$\boldsymbol{\tau}_{\boldsymbol{\delta}}(t) = -\mathbf{K}_s\boldsymbol{\delta}(t) - \mathbf{K}_d\dot{\boldsymbol{\delta}}(t) \tag{2.32}$$

where $\mathbf{K}_s = diag\{K_{s1},\cdots,K_{sr}\}$ and $\mathbf{K}_d = diag\{K_{d1},\cdots,K_{dr}\}$ contain the spring constants and the viscous dampings of the fictitious joints, respectively.

This model is quite simple and can easily be combined with the model obtained in equation (2.30). One point that needs to be clarified is, perhaps, the modification in $\mathbf{q}$, the independent generalized coordinates. Set $\mathbf{q}$ can still assume the form it had before the introduction of flexibility, but this time the fictitious joint variables coming from all the flexible links should be appended to the end of this vector. This, in fact, is the same thing that we would have expected if we had added extra true joints with independent actuators. Since an extra joint increases the D.O.F. of the system by one, the number of independent generalized coordinates should also increase by one.

When flexibility is included, equation (2.20) is modified into

$$\mathbf{M}_k\begin{bmatrix}\ddot{\boldsymbol{\theta}}_k\\ \ddot{\boldsymbol{\delta}}_k\end{bmatrix}+\mathbf{h}_k(\boldsymbol{\theta}_k,\boldsymbol{\delta}_k,\dot{\boldsymbol{\theta}}_k,\dot{\boldsymbol{\delta}}_k)=\begin{bmatrix}\boldsymbol{\tau}_{\boldsymbol{\theta}k}\\ \boldsymbol{\tau}_{\boldsymbol{\delta}k}\end{bmatrix}-\mathbf{J}_k^T\mathbf{f}_k,\qquad k=1,\ldots,N \tag{2.33}$$

and the equation (2.23) is modified into

$$\begin{bmatrix}\dot{\boldsymbol{\theta}}_k\\ \dot{\boldsymbol{\delta}}_k\end{bmatrix}=\begin{bmatrix}\mathbf{J}_{\boldsymbol{\theta}k} & \mathbf{J}_{\boldsymbol{\delta}k}\\ 0 & \mathbf{I}_k\end{bmatrix}\begin{bmatrix}\dot{\mathbf{q}}\\ \dot{\boldsymbol{\delta}}\end{bmatrix} \tag{2.34}$$

and

$$\begin{bmatrix} \ddot{\boldsymbol{\theta}}_k \\ \ddot{\boldsymbol{\delta}}_k \end{bmatrix} = \begin{bmatrix} \mathbf{J}_{\boldsymbol{\theta}k} & \mathbf{J}_{\boldsymbol{\delta}k} \\ 0 & \mathbf{I}_k \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}} \\ \ddot{\boldsymbol{\delta}} \end{bmatrix} + \begin{bmatrix} \dot{\mathbf{J}}_{\boldsymbol{\theta}k} & \dot{\mathbf{J}}_{\boldsymbol{\delta}k} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{\mathbf{q}} \\ \dot{\boldsymbol{\delta}} \end{bmatrix}, k = 1, \ldots, N \quad (2.35)$$

where $\boldsymbol{\delta} = [\boldsymbol{\delta}_1^T, \cdots, \boldsymbol{\delta}_N^T]^T$ and the matrix $\mathbf{I}_k$ being a matrix which is of the form $\mathbf{I}_k = [0 \ \mathbf{I} \ 0]$ where the identity matrix $\mathbf{I}$ picks the entries from $\boldsymbol{\delta}$ corresponding to $\boldsymbol{\delta}_k$.

With these modifications, going through the similar steps in construction of the combined system equation and after a suitable reordering, we have

$$\mathbf{M}(\mathbf{q}, \boldsymbol{\delta}) \begin{bmatrix} \ddot{\mathbf{q}} \\ \ddot{\boldsymbol{\delta}} \end{bmatrix} + \mathbf{h}(\mathbf{q}, \boldsymbol{\delta}, \dot{\mathbf{q}}, \dot{\boldsymbol{\delta}}) = \begin{bmatrix} \boldsymbol{\tau}_{\mathbf{q}} \\ \boldsymbol{\tau}_{\boldsymbol{\delta}} \end{bmatrix} \quad (2.36)$$

replacing the equation (2.30). Here, $\boldsymbol{\tau}_{\mathbf{q}} = [\boldsymbol{\tau}_{\mathbf{q}1}^T, \cdots, \boldsymbol{\tau}_{\mathbf{q}N}^T]^T$ and $\boldsymbol{\tau}_{\boldsymbol{\delta}} = [\boldsymbol{\tau}_{\boldsymbol{\delta}1}^T, \cdots, \boldsymbol{\tau}_{\boldsymbol{\delta}N}^T]^T$.

Chapter 3

A Two-Time-Scale Control Method

Accurate positioning of the end-effector is an important issue in robot control systems. Especially, in manipulators having flexible links, this problem becomes more challenging since flexibility induces high frequency vibrations which are difficult to control and should be damped at the end of the trajectory as soon as possible. While deriving the control law, it is assumed that the pseudo-joint angles can be constructed from the strain gage measurements positioned along the link as in [12].

3.1 Model Decomposition

In the position control of multiple coordinated manipulators, if flexibility is included, the desired motion to be performed by the manipulator system is much slower in nature when compared to the oscillations due to flexibility. Though many schemes have been proposed to control the relatively slow desired motion, still special care should be given to prevent these high frequency oscillations. Facilitated by the relatively high stiffness of the flexible links, the method proposed here makes use of the decomposition of the system model into two reduced order models operating at different rates. Once we have this multi-time-scale model, we can implement a suitable control at each time scale.

Let's assume a model in which we have n joint actuators(external inputs), d independent generalized coordinates(D.O.F.) and r flexible links-thus r extra generalized coordinates. The two-time-scale scheme described below applies to the case where $r \leq d$. Otherwise, a multi-time-scale decomposition and a suitable control should be applied by a proper ordering of the diagonal elements of stiffness coefficients matrix, $\mathbf{K}_s$.

Let's start by partitioning the matrices in equation (2.36) resulting in,

$$\begin{bmatrix} \mathbf{M_{qq}} & \mathbf{M_{q\delta}} \\ \mathbf{M_{\delta q}} & \mathbf{M_{\delta\delta}} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}} \\ \ddot{\boldsymbol{\delta}} \end{bmatrix} + \begin{bmatrix} \mathbf{h_q} \\ \mathbf{h_\delta} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\tau_q} \\ \boldsymbol{\tau_\delta} \end{bmatrix} \tag{3.1}$$

where $\mathbf{M_{qq}} \in \mathbf{R}^{n\times d}$, $\mathbf{M_{q\delta}} \in \mathbf{R}^{n\times r}$, $\mathbf{M_{\delta q}} \in \mathbf{R}^{r\times d}$, $\mathbf{M_{\delta\delta}} \in \mathbf{R}^{r\times r}$, $\mathbf{h_q} \in \mathbf{R}^n$ and $\mathbf{h_\delta} \in \mathbf{R}^r$. While deriving the two-time-scale decomposition, we assume that $\mathbf{q}$ is a slow(quasi constant) variable but that $\boldsymbol{\delta}$ consists of both slow and fast parts. Here, we make use of the fact that $\mathbf{K}_s$ is a relatively large quantity and write it as $\mathbf{K}_s = \mu^{-2}\mathbf{K}'_s$ with μ small and $\mathbf{K}'_s$ approximately at the same order as the other parameters in the model.Next, we apply singular perturbations technique to derive the decomposition. However, as $\mathbf{M}$ is non-square it is not invertible and we cannot bring the equations into the standard state space form

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{z}, \mathbf{u}, \mu, t) \\ \mu\dot{\mathbf{z}} &= \mathbf{g}(\mathbf{x}, \mathbf{z}, \mathbf{u}, \mu, t) \end{aligned} \tag{3.2}$$

of a singularly perturbed system([22]). Instead, we first write equation (3.1) as

$$\begin{aligned} &\mathbf{M_{qq}}\ddot{\mathbf{q}} + \mathbf{M_{q\delta}}\ddot{\boldsymbol{\delta}} + \mathbf{h_q} = \boldsymbol{\tau_q} \\ &\mathbf{M_{\delta q}}\ddot{\mathbf{q}} + \mathbf{M_{\delta\delta}}\ddot{\boldsymbol{\delta}} + \mathbf{h_\delta} + \mu^{-2}\mathbf{K}'_s\boldsymbol{\delta} + \mathbf{K}_d\dot{\boldsymbol{\delta}} = 0 \end{aligned} \tag{3.3}$$

and introduce the state variables

$$\mathbf{x}_1 = \mathbf{q}, \quad \mathbf{x}_2 = \dot{\mathbf{q}}, \quad \mathbf{z}_1 = \mu^{-2}\boldsymbol{\delta}, \quad \mathbf{z}_2 = \mu^{-1}\dot{\boldsymbol{\delta}} \tag{3.4}$$

which are decomposed into fast and slow components as

$$\mathbf{x}_i \simeq \bar{\mathbf{x}}_i, \quad \mathbf{z}_i \simeq \bar{\mathbf{z}}_i + \hat{\mathbf{z}}_i, \quad i = 1, 2 \ , \quad \boldsymbol{\tau_q} \simeq \bar{\boldsymbol{\tau}}_\mathbf{q} + \hat{\boldsymbol{\tau}}_\mathbf{q} \tag{3.5}$$

with bar denoting slow components and hat denoting fast components.

With the above state variable assignment, equation (3.3) is changed to

$$\dot{\mathbf{x}}_1 = \mathbf{x}_2, \tag{3.6}$$

$$\mathbf{M}_{\mathbf{qq}}\dot{\mathbf{x}}_2 + \mu\mathbf{M}_{\mathbf{q}\boldsymbol{\delta}}\dot{\mathbf{z}}_2 + \mathbf{h}_{\mathbf{q}} = \boldsymbol{\tau}_{\mathbf{q}}, \tag{3.7}$$

$$\mu\dot{\mathbf{z}}_1 = \mathbf{z}_2, \tag{3.8}$$

$$\mathbf{M}_{\boldsymbol{\delta}\mathbf{q}}\dot{\mathbf{x}}_2 + \mu\mathbf{M}_{\boldsymbol{\delta}\boldsymbol{\delta}}\dot{\mathbf{z}}_2 + \mathbf{h}_{\boldsymbol{\delta}} + \mathbf{K}_s'\mathbf{z}_1 + \mu\mathbf{K}_d\mathbf{z}_2 = 0 \tag{3.9}$$

Now, the slow components of $\mathbf{z}_i, i = 1, 2$ can be solved from the equations (3.8) and (3.9) with $\mu = 0$ as,

$$\begin{aligned} \bar{\mathbf{z}}_1 &= -(\mathbf{K}_s')^{-1}\left(\mathbf{M}_{\boldsymbol{\delta}\mathbf{q}}\dot{\bar{\mathbf{x}}}_2 + \mathbf{h}_{\boldsymbol{\delta}}\right) \\ \bar{\mathbf{z}}_2 &= 0 \end{aligned} \tag{3.10}$$

By using equations (3.6) and (3.7) with $\mu = 0$, the slow components of $\mathbf{x}_i$ are shown to satisfy

$$\begin{aligned} \dot{\bar{\mathbf{x}}}_1 &= \bar{\mathbf{x}}_2 \\ \mathbf{M}_{\mathbf{qq}}\dot{\bar{\mathbf{x}}}_2 + \mathbf{h}_{\mathbf{q}} &= \bar{\boldsymbol{\tau}}_{\mathbf{q}} \end{aligned} \tag{3.11}$$

To obtain the fast dynamics(boundary layer system), we first solve for $\dot{\mathbf{x}}_2$ from equation (3.9) and substitute it into (3.7). Next, we apply the last phase of the singular perturbations method to get the fast dynamics. While solving for $\dot{\mathbf{x}}_2$ from equation (3.9), since we assumed $r \leq d$ at the beginning, $\mathbf{M}_{\boldsymbol{\delta}\mathbf{q}}$ is of full row rank. So, let's use the pseudo-inverse of $\mathbf{M}_{\boldsymbol{\delta}\mathbf{q}}$ to get $\dot{\mathbf{x}}_2$ as

$$\dot{\mathbf{x}}_2 = -\mathbf{M}_{\boldsymbol{\delta}\mathbf{q}}^{\dagger}\left(\mu\mathbf{M}_{\boldsymbol{\delta}\boldsymbol{\delta}}\dot{\mathbf{z}}_2 + \mathbf{h}_{\boldsymbol{\delta}} + \mathbf{K}_s'\mathbf{z}_1 + \mu\mathbf{K}_d\mathbf{z}_2\right) \tag{3.12}$$

where,

$$\mathbf{M}_{\boldsymbol{\delta}\mathbf{q}}^{\dagger} = \mathbf{M}_{\boldsymbol{\delta}\mathbf{q}}^{T}\left(\mathbf{M}_{\boldsymbol{\delta}\mathbf{q}}\mathbf{M}_{\boldsymbol{\delta}\mathbf{q}}^{T}\right)^{-1}$$

Next, we insert this in equation (3.7), switch to the fast time scale by defining a fast time variable

$$\sigma = \frac{t - t_0}{\mu} \tag{3.13}$$

and finally substitute $\mu = 0$. Eliminating the slow quantities, the dynamics of the fast subsystem in the original time variable t is obtained as

$$\mu\left(\mathbf{M}_{\mathbf{q}\delta} - \mathbf{M}_{\mathbf{qq}}\mathbf{M}^{\dagger}_{\delta\mathbf{q}}\mathbf{M}_{\delta\delta}\right)\dot{\hat{z}}_2 - \mathbf{M}_{\mathbf{qq}}\mathbf{M}^{\dagger}_{\delta\mathbf{q}}\mathbf{K}'_s\hat{z}_1 = \hat{\tau}_{\mathbf{q}} \tag{3.14}$$

Going back to the true variables from state variables, we have

$$\mathbf{M}_{\mathbf{qq}}\ddot{\bar{\mathbf{q}}} + \mathbf{h}_{\mathbf{q}} = \bar{\tau}_{\mathbf{q}} \tag{3.15}$$

$$\left(\mathbf{M}_{\mathbf{q}\delta} - \mathbf{M}_{\mathbf{qq}}\mathbf{M}^{\dagger}_{\delta\mathbf{q}}\mathbf{M}_{\delta\delta}\right)\ddot{\hat{\delta}} - \mathbf{M}_{\mathbf{qq}}\mathbf{M}^{\dagger}_{\delta\mathbf{q}}\mathbf{K}_s\hat{\delta} = \hat{\tau}_{\mathbf{q}} \tag{3.16}$$

$$\bar{\delta} = -(\mathbf{K}_s)^{-1}\left(\mathbf{M}_{\delta\mathbf{q}}\ddot{\bar{\mathbf{q}}} + \mathbf{h}_{\delta}\right) \tag{3.17}$$

describing the slow and fast dynamics of the multiple manipulator system.

3.2 Control Scheme

The above decomposition of dynamics into slow and fast components is valid only if the fast subsystem in equation (3.16) is asymptotically stable([22]). If the fast subsystem can be stabilized for any trajectory of interest, the overall system approaches in the limit to the reduced order decomposition. Thus, we first stabilize $\hat{\delta}$ by applying a fast feedback of the form

$$\hat{\tau}_{\mathbf{q}} = \hat{\mathbf{K}}_v\dot{\hat{\delta}} + \hat{\mathbf{K}}_p\hat{\delta} \tag{3.18}$$

which, when combined with the model equation (3.16), results in

$$\left(\mathbf{M}_{\mathbf{q}\delta} - \mathbf{M}_{\mathbf{qq}}\mathbf{M}^{\dagger}_{\delta\mathbf{q}}\mathbf{M}_{\delta\delta}\right)\ddot{\hat{\delta}} - \hat{\mathbf{K}}_v\dot{\hat{\delta}} - \left(\mathbf{M}_{\mathbf{qq}}\mathbf{M}^{\dagger}_{\delta\mathbf{q}}\mathbf{K}_s + \hat{\mathbf{K}}_p\right)\hat{\delta} = 0 \tag{3.19}$$

From equation (3.19) we observe that, if we let

$$\begin{aligned}\hat{\mathbf{K}}_v &= -\left(\mathbf{M}_{\mathbf{q}\delta} - \mathbf{M}_{\mathbf{qq}}\mathbf{M}^{\dagger}_{\delta\mathbf{q}}\mathbf{M}_{\delta\delta}\right)\hat{\mathbf{Z}}_1 \\ \hat{\mathbf{K}}_p &= -\mathbf{M}_{\mathbf{qq}}\mathbf{M}^{\dagger}_{\delta\mathbf{q}}\mathbf{K}_s - \left(\mathbf{M}_{\mathbf{q}\delta} - \mathbf{M}_{\mathbf{qq}}\mathbf{M}^{\dagger}_{\delta\mathbf{q}}\mathbf{M}_{\delta\delta}\right)\hat{\mathbf{Z}}_2\end{aligned} \tag{3.20}$$

equation (3.19) is reduced to

$$\ddot{\hat{\delta}} + \hat{\mathbf{Z}}_1\dot{\hat{\delta}} + \hat{\mathbf{Z}}_2\hat{\delta} = 0 \tag{3.21}$$

Now, $\hat{\mathbf{Z}}_1$ and $\hat{\mathbf{Z}}_2$ can be chosen arbitrarily to stabilize the fast subsystem by placing its $2r$ poles at desired locations. Note that this also allows for the decoupling of fast dynamics, meaning that the vibrational modes associated with the fictitious joints can be controlled independently.

With the fast subsystem stabilized, it now remains to choose the slow components of the slow subsystem defined in equation (3.15). This is the standard tracking problem of rigid manipulators. A computed torque type control can be chosen, in which we feedforward the slow control to achieve the desired trajectory along with a feedback control to compensate for the deviations. So, the slow control takes the form

$$\bar{\boldsymbol{\tau}}_{\mathbf{q}} = \mathbf{M}_{\mathbf{qq}}\ddot{\mathbf{q}}^d + \mathbf{h}_{\mathbf{q}} + \bar{\mathbf{K}}_v\dot{\mathbf{e}}_{\mathbf{q}} + \bar{\mathbf{K}}_p\mathbf{e}_{\mathbf{q}} \tag{3.22}$$

where $\mathbf{q}^d(t)$ describes the desired trajectory in terms of the independent generalized coordinates and $\mathbf{e}_{\mathbf{q}} = \mathbf{q}^d - \bar{\mathbf{q}}$ is the deviation of the actual trajectory from the desired one. Application of this control results in the error dynamics given by,

$$\mathbf{M}_{\mathbf{qq}}\ddot{\mathbf{e}}_{\mathbf{q}} + \bar{\mathbf{K}}_v\dot{\mathbf{e}}_{\mathbf{q}} + \bar{\mathbf{K}}_p\mathbf{e}_{\mathbf{q}} = 0 \tag{3.23}$$

Letting,

$$\begin{aligned} \bar{\mathbf{K}}_v &= \mathbf{M}_{\mathbf{qq}}\bar{\mathbf{Z}}_1 \\ \bar{\mathbf{K}}_p &= \mathbf{M}_{\mathbf{qq}}\bar{\mathbf{Z}}_2 \end{aligned} \tag{3.24}$$

equation (3.23) is reduced to

$$\ddot{\mathbf{e}}_{\mathbf{q}} + \bar{\mathbf{Z}}_1\dot{\mathbf{e}}_{\mathbf{q}} + \bar{\mathbf{Z}}_2\mathbf{e}_{\mathbf{q}} = 0 \tag{3.25}$$

Clearly, $\bar{\mathbf{Z}}_1$ and $\bar{\mathbf{Z}}_2$ can be chosen properly to place the $2d$ poles of the slow subsystem as desired. Again, choosing $\bar{\mathbf{Z}}_1$ and $\bar{\mathbf{Z}}_2$ in a diagonal form decouples the error dynamics of generalized coordinates.

The composite control is the sum of fast and slow controls in equations (3.18) and (3.22). Yet, we still need $\hat{\boldsymbol{\delta}}$ and $\dot{\hat{\boldsymbol{\delta}}}$ for the computation of the fast

control. Since $\hat{\boldsymbol{\delta}}$ is not directly measurable, it somehow has to be computed. Since $\boldsymbol{\delta}$ is measurable and $\bar{\boldsymbol{\delta}}$ can be computed from equation (3.17), $\hat{\boldsymbol{\delta}}$ is computed using $\hat{\boldsymbol{\delta}} \simeq \boldsymbol{\delta} - \bar{\boldsymbol{\delta}}$. Also using $\dot{\bar{\boldsymbol{\delta}}} = 0$, which comes from equation (3.10), $\dot{\hat{\boldsymbol{\delta}}}$ is computed by using $\dot{\hat{\boldsymbol{\delta}}} \simeq \dot{\boldsymbol{\delta}}$. So, the fast control finally takes the form,

$$\hat{\boldsymbol{\tau}}_{\mathbf{q}} = \hat{\mathbf{K}}_v \dot{\boldsymbol{\delta}} + \hat{\mathbf{K}}_p \left(\boldsymbol{\delta} + \mathbf{K}_s^{-1} \left(\mathbf{M}_{\boldsymbol{\delta}\mathbf{q}} \ddot{\mathbf{q}} + \mathbf{h}_{\boldsymbol{\delta}} \right) \right) \tag{3.26}$$

which, like the slow control $\bar{\boldsymbol{\tau}}_{\mathbf{q}}$, involves only available variables.

While implementing the control law in discrete-time, or possibly on a digital computer, most probably time derivatives of the true-joint and pseudo-joint variables will not be available to us but rather the values of the variables at sampling instants. The time derivatives in this case can be computed using backward differencing like $\dot{x}_k = T^{-1}\Delta x_k$ and $\ddot{x}_k = T^{-2}\Delta^2 x_k$ where T is the sampling period, x_k is the value of the variable x at kth sampling instant and Δ is the backward difference operator defined by $\Delta x_k = x_k - x_{k-1}$.

3.3 An Adaptive Self-Tuning Control

Implementation of the control law derived in the previous section requires that the system parameters $\mathbf{M}_{\mathbf{qq}}, \mathbf{M}_{\mathbf{q}\boldsymbol{\delta}}, \mathbf{M}_{\boldsymbol{\delta}\mathbf{q}}, \mathbf{M}_{\boldsymbol{\delta}\boldsymbol{\delta}}, \mathbf{h}_{\mathbf{q}}$ and $\mathbf{h}_{\boldsymbol{\delta}}$ be known. Though they can be computed analytically, these parameters are highly nonlinear, involving lots of sines and cosines, and these extensive computations are not suitable for on-line applications.

An alternative to computing the manipulator parameters is to estimate them from input/output data using an adaptive estimation algorithm and then base the control law on the estimated parameters. The Recursive Least Squares Estimation (RLSE) algorithm ([21]) is a suitable method to use in identification since it has a low computational complexity due to being recursive, and also a fast convergence.

For the development of the identification algorithm, we use the model derived in equation (2.36). Define $\mathit{\Omega} = [\mathbf{q}^T \ \boldsymbol{\delta}^T]^T$ and $\boldsymbol{\tau} = [\boldsymbol{\tau}_\mathbf{q}^T \ \boldsymbol{\tau}_{\boldsymbol{\delta}}^T]^T$. Scaling the equation with α, equation (2.36) can be rewritten as,

$$\alpha\boldsymbol{\tau} = [\mathbf{M} \quad \alpha\mathbf{h}] \begin{bmatrix} \alpha\ddot{\mathit{\Omega}} \\ 1 \end{bmatrix} \tag{3.27}$$

Here, α is incorporated as a scaling factor to increase the numeric precision of the algorithm. Also, as $\ddot{\mathit{\Omega}}$ will not be available to us but rather the sampled values of $\mathit{\Omega}$ only, $\ddot{\mathit{\Omega}}$ can be approximated by backward differencing as $\ddot{\mathit{\Omega}} = T^{-2}\Delta^2\mathit{\Omega}_k$. Here, again T is the sampling period, Δ is the backward difference operator and $\mathit{\Omega}_k$ denoted the sampled values of $\mathit{\Omega}$. At this point we can choose $\alpha = T$ which brings the parameters of above formulation to about same order.

Letting,

$$\mathbf{y}_k = T \cdot \boldsymbol{\tau}_k, \qquad \boldsymbol{\phi}_k = \begin{bmatrix} T^{-1}\Delta^2\mathit{\Omega}_k \\ 1 \end{bmatrix}, \qquad \boldsymbol{\pi}_k = [\mathbf{M}_k \quad T\mathbf{h}_k] \tag{3.28}$$

where the subscript k is to denote the values at the kth sampling instant with $\boldsymbol{\phi}_k$ being the observation vector(regressors) and $\boldsymbol{\pi}_k$ being the parameter matrix, equation (3.27) can be written in the standard form

$$\mathbf{y}_k = \boldsymbol{\pi}_k\boldsymbol{\phi}_k \tag{3.29}$$

Assuming slow variations of parameters, a least-squares estimate of $\boldsymbol{\pi}_k$ which minimizes the weighted sum

$$J_k = \sum_{l=0}^{k} \gamma^{k-l}(\mathbf{y}_l - \boldsymbol{\pi}_l\boldsymbol{\phi}_l)^T(\mathbf{y}_l - \boldsymbol{\pi}_l\boldsymbol{\phi}_l) \tag{3.30}$$

is obtained in [21] as

$$\begin{aligned} \alpha_k &= (1 + \boldsymbol{\phi}_k^T\mathbf{P}_{k-1}\boldsymbol{\phi}_k)^{-1} \\ \boldsymbol{\pi}_k^e &= \boldsymbol{\pi}_{k-1}^e + \alpha_k(\mathbf{y}_k - \boldsymbol{\pi}_{k-1}^e\boldsymbol{\phi}_k)\boldsymbol{\phi}_k^T\mathbf{P}_{k-1} \\ \mathbf{P}_k &= \gamma^{-1}(\mathbf{P}_{k-1} - \alpha_k\mathbf{P}_{k-1}\boldsymbol{\phi}_k\boldsymbol{\phi}_k^T\mathbf{P}_{k-1}) \end{aligned} \tag{3.31}$$

where superscript e denotes the identified parameter matrix. Once $\boldsymbol{\pi}_k$ is calculated, $\mathbf{M}_k$ and $\mathbf{h}_k$ are determined and then are suitably partitioned to compute the next control inputs. The system parameters required for the construction of the fast and slow controls in the $(k+1)$st iteration are replaced by their estimates computed at the kth iteration.

In implementation of the RLSE algorithm, the initial value of $\boldsymbol{\pi}_k$ can be given by either calculating it once and for all to initiate the algorithm, or can be suitably chosen based on the a priori information about the system. In choosing the initial value of $\mathbf{P}_k$, the common practice is to choose it as $\lambda \cdot \mathbf{I}$ with $\lambda > 0$ and $\mathbf{I}$ being the identity matrix.

Chapter 4

Simulations

A simulation study has been carried out to test the effectiveness of the proposed adaptive two-time-scale control method. A simulation program has been written in C and run on SUN-Sparc 10 workstations.

4.1 Simulation Setting

The simulation set-up consists of two planar two-link dual manipulators. The manipulators have one rigid link connected to the base and one flexible link each, and are holding a rigid workpiece at both ends as shown in Fig. 4.1.

In the simulation set-up, the links of the manipulator are all $0.5m$ long and have masses $10\ kg$ each. The effective inertias and damping coefficients of the actuators are $6.35\ kg.m^2$ and $2.12\ N.m.s.$ for the first joint and $1.30\ kg.m^2$ and $0.43\ N.m.s.$ for the second joint. The two manipulators are spaced by $1.0\ m$. The commonly held workpiece is $0.2\ m$ long and is $10\ kg$ also.

Each flexible link is modelled by a second-order bulk model approximation, consisting of two sublinks having lengths $0.25\ m$ each. On the assumption that the flexible link is homogeneous, the mass of the flexible link is equally distributed between the two links as $5\ kg$ each. The spring constants and the

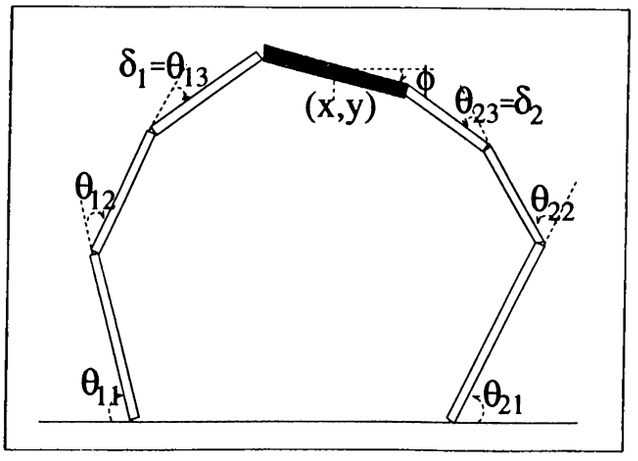


Figure 4.1: *Simulation set-up to test the proposed method*

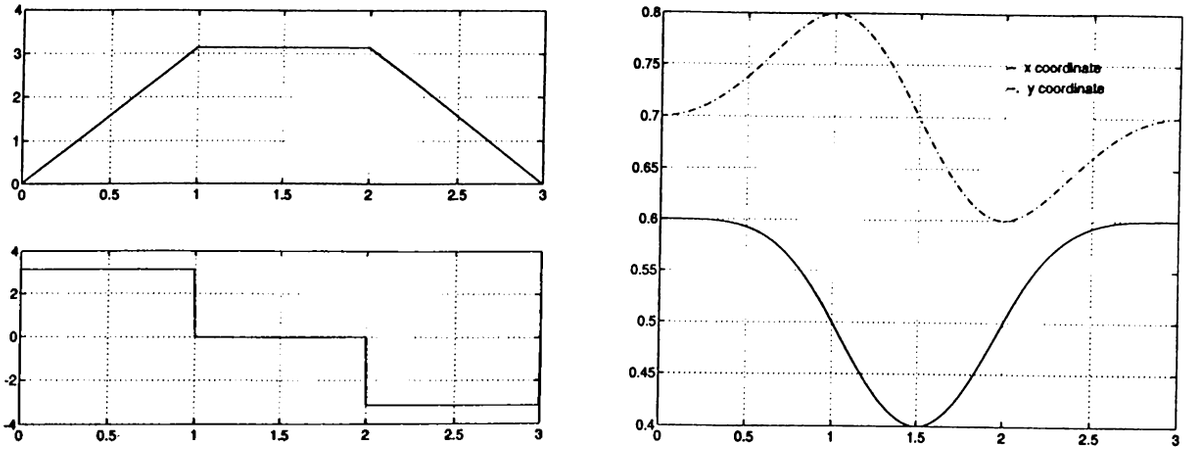
damping coefficients of the fictitious joints are assumed to be $K_s = 400\ N.m$ and $K_d = 10\ N.m.s$, respectively. Note that the assumed value of K_s is large enough to justify a singularly perturbed model.

The closed chain configuration in Fig. 4.1 has 7 links each having a D.O.F $d_l = 3$, and 8 joints. Thus the D.O.F. of the entire configuration is

$$d = 3 \times 7 - 2 \times 8 = 5$$

We choose as generalized coordinates the Cartesian x and y coordinates of the center of mass of the workpiece, its orientation angle ϕ, and the pseudo-joint angles of the flexible links, totaling 5.

The desired trajectory is given in terms of the three variables associated with the workpiece. The trajectory to be followed by the center of mass of the workpiece is a circular path in the plane of the workpiece with radius 0.1 m and centered at $(0.5\ m, 0.7\ m)$. It is to be traced in 3 s starting from the point $(0.6\ m, 0.7\ m)$, with an angular velocity and acceleration as specified in Fig. 4.2.a. Corresponding displacements of the center of mass in Cartesian coordinates are shown in Fig. 4.2.b. While this trajectory is being traced by the center of mass of the workpiece, the orientation angle of the workpiece is desired to trace a sine wave as shown in Fig. 4.3. This trajectory contains large inertial variations along the path, and is chosen to demonstrate the effectiveness



a. Angular velocity$_{(\mathrm{rad/s\ vs\ s})}$ *and acceleration*$_{(\mathrm{rad/s^2 vs\ s})}$ *b. Cartesian position*$_{(\mathrm{m\ vs\ s})}$

Figure 4.2: *Desired center of mass trajectory*

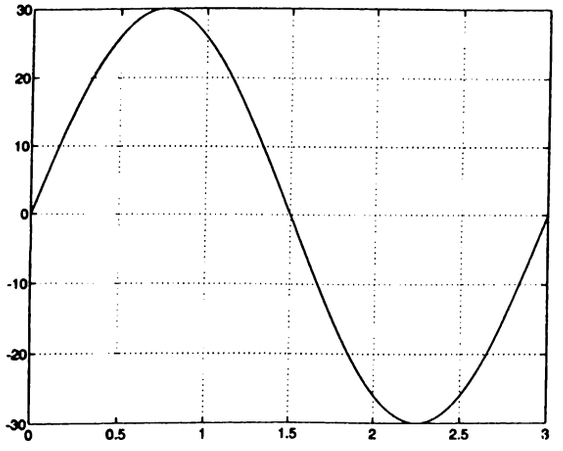
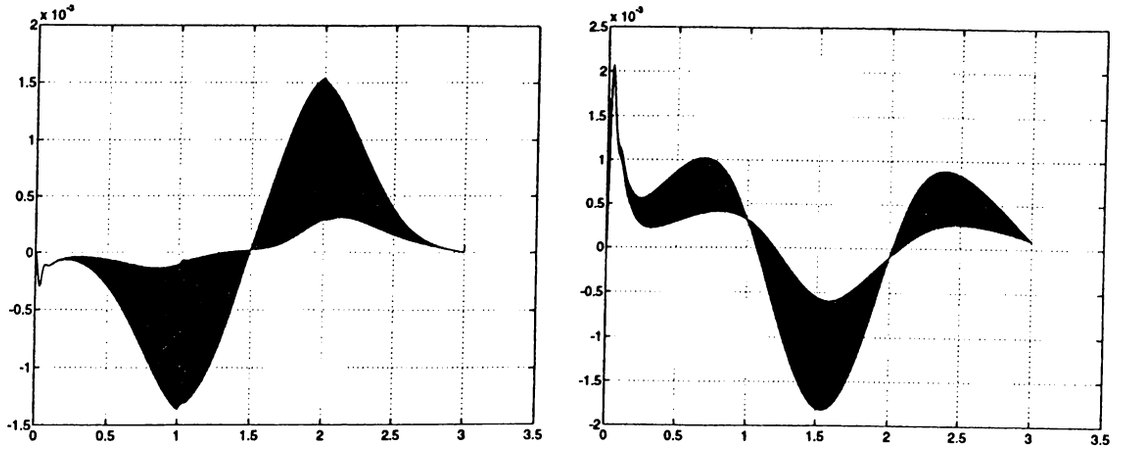


Figure 4.3: *Desired orientation of workpiece in degrees*

of the proposed method.

The simulations are performed by modelling all the links as homogeneous structures with center of masses at their midpoints. A point-mass model is used for the links and their rotational inertias are computed on the homogeneity assumption. In simulations a discrete model of the manipulator corresponding to a sampling period of $T = 0.005\ s$ is used. That is, the manipulator parameters are updated(either by calculation or estimation) once every $0.005\ s$, as well as the value of the control input. Runge-Kutta method with a step size of $h = 0.001\ s$ is used for numerical integration of the manipulator equations during each sampling period.



a. Along x-axis *b. Along y-axis*

Figure 4.4: *Tracking error in the position of center of mass using true parameters (m vs s)*

4.2 Simulation Results

In the simulation, $\bar{\mathbf{K}}_v$ and $\bar{\mathbf{K}}_p$ are selected to place the poles of the error equation (3.25) at $s_{1,2} = -20$ for the slow subsystem. By a proper choice of the feedback gains $\hat{\mathbf{K}}_v$ and $\hat{\mathbf{K}}_p$ fast poles are placed at $s_1 = -50$ and $s_2 = -60$, approximately three times faster than the slow poles, to provide a rapid decay of oscillations.

Simulation results with the manipulator parameters calculated along the desired trajectory are shown in Fig. 4.4 and 4.5. Corresponding fictitious joint displacements are also shown in Fig. 4.6.

The same simulation is repeated with the parameters estimated as described in Section 3.3. This time the same trajectory is traced two times to see that the overshoot observed at the initiation of the control is removed after the identification is completed. The forgetting factor is taken to be 0.8 and the initial value of the matrix $\mathbf{P}_k$ in equation (3.31) is taken to be $\mathbf{P}_0 = 20\,\mathbf{I}$. The initial parameter matrix $\boldsymbol{\pi}_0$ is chosen such that the entries are at the same order as the true values. As the control method requires partial manipulations of the parameter matrices, even by taking inverses, a bad initial estimate may

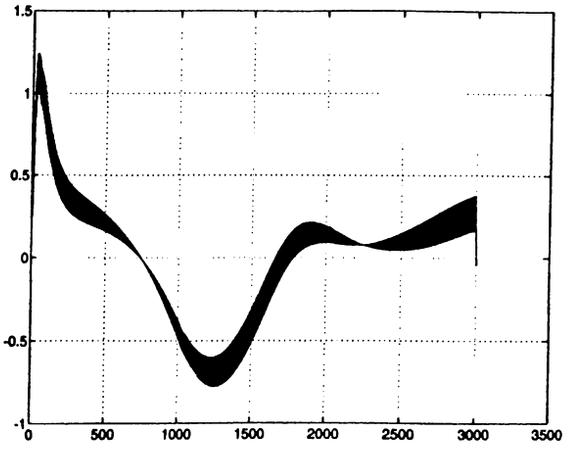
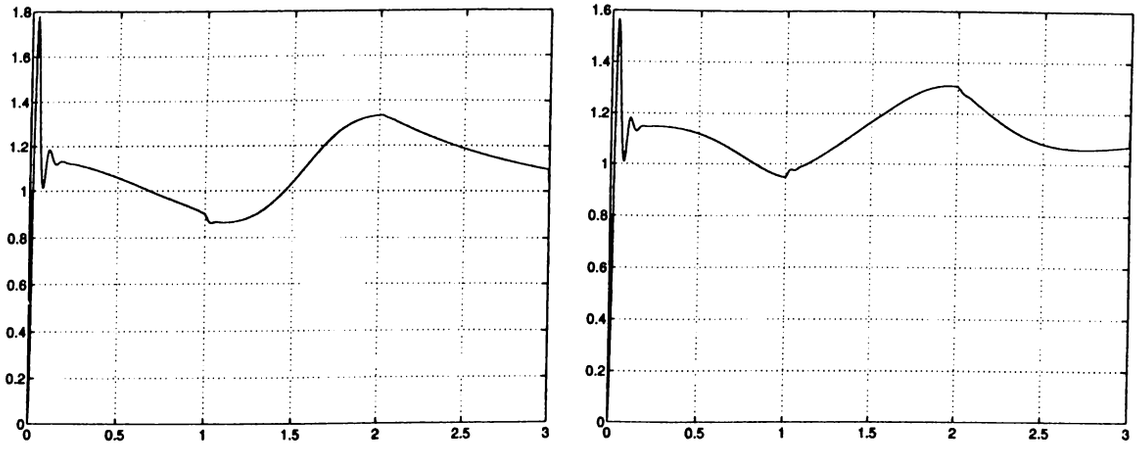


Figure 4.5: *Tracking error in the orientation angle using true parameters (degrees vs s)*

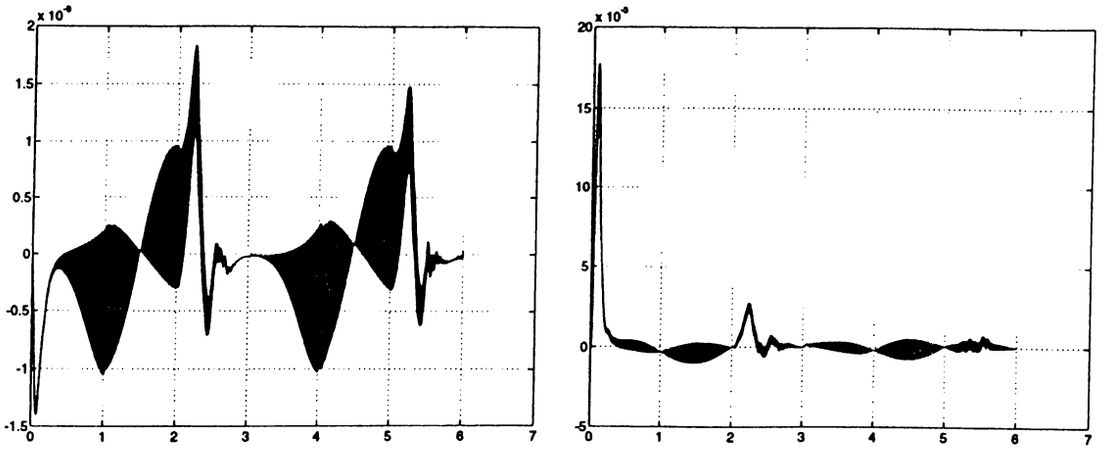


a. For the first joint *b. For the second joint*

Figure 4.6: *Fictitious joint displacements in degrees using true parameters*

cause the system to collapse from the very beginning. However, this is not a major disadvantage as we can almost always assume some previous information on system parameters. It may even be that, though the system parameters are known, we may not want to compute them in each iteration and computing it once and for all at the beginning we may let the identification procedure handle the construction of these parameters in the consecutive iterations. During estimation, $\mathbf{P}_k$ is reset to $\mathbf{P}_0$ whenever the diagonal elements exceed 50.

The results of the simulations are shown in Fig. 4.7 and 4.8. Fictitious joint displacements are plotted in Fig. 4.9.



a. Along x-axis *b. Along y-axis*

Figure 4.7: *Tracking error in the position of center of mass using identification*

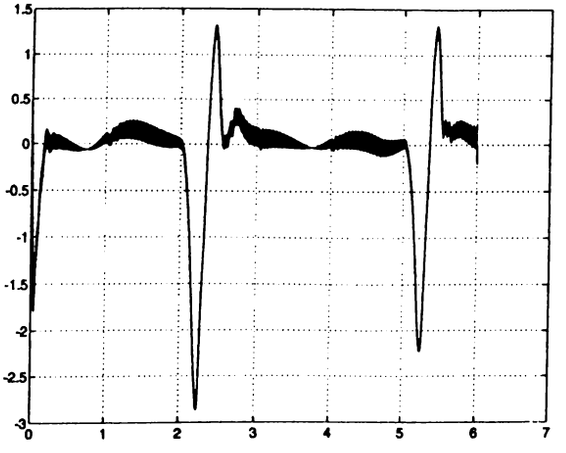
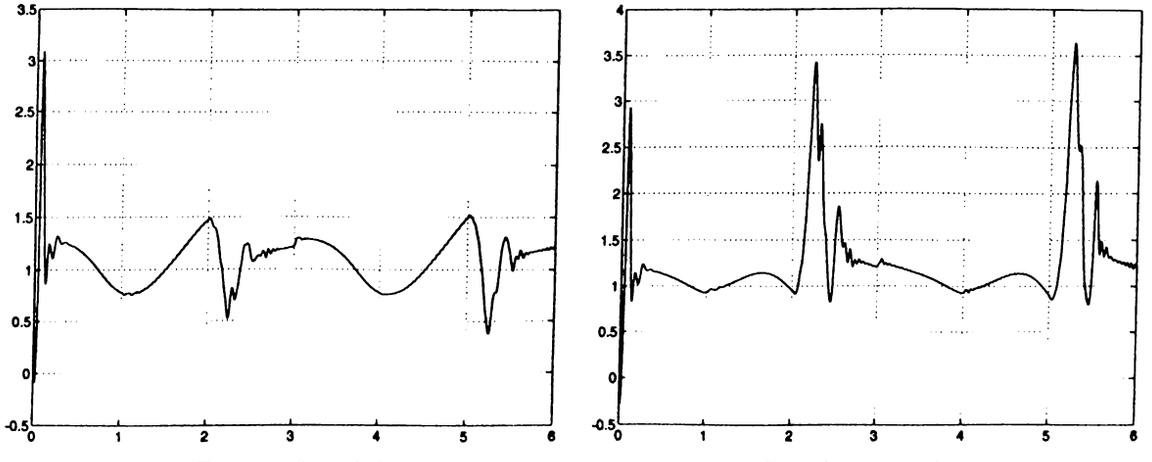


Figure 4.8: *Tracking error in the orientation angle with identification*



a. For the first joint *b. For the second joint*

Figure 4.9: *Fictitious joint displacements with identification*

Chapter 5

Conclusion

To meet the emerging needs of industry, accurate models should be developed to achieve the coordinated control of multiple manipulators, even in the existence of link flexibility. In this thesis, an adaptive two-time-scale position control scheme is proposed for this purpose.

The proposed method has a number of attractive features. First the method is based on modelling the link by a second-order bulk model, of which dynamics can easily be obtained by any method used for the computation of rigid link manipulator dynamics. As well as the easiness of obtaining the dynamics, the model includes a reasonable representation of the flexible modes. Disadvantages of this type of modelling are, perhaps, that it is applicable to links having regular cross sections and also requires the knowledge of the associated stiffness and damping coefficients of the pseudo joints. Yet, these are not major drawbacks since most of the flexible manipulators will have long and slender links with regular cross sections anyway. Also, stiffness and damping coefficients are associated with the material that the link is made of and the geometry of the link, and these can be quite standard parameters for a certain type of link at hand.

Another good feature of this method is its being adaptive. Adaptivity

saves us from the burden of computing the system parameters each time we are going to construct the control, and makes the method suitable for on-line applications. Due to being recursive, the used adaptive scheme has a low computational complexity and is well-suited for this type of an application.

As this method is developed in generalized coordinates, on the simulation model the control converted into an end-point control scheme. This is because the desired trajectory was given in terms of the position and the orientation of the workpiece and that we have chosen these variables as the independent generalized coordinates in simulation. This provides an automatic compensation for the deflection at the flexible links through feedback. Also, the need for the inverse kinematics, which is required to go from end-point coordinates to joint coordinates, for constructing the control is eliminated.

For further research on this topic, one thing that can be investigated is the introduction of a larger order bulk model to model the existence of multiple mode vibrations. The mode in the second-order bulk model being dominant, a third joint with an associated stiffness to model a different mode can be incorporated. On this model, a three- or more-time-scale model decomposition and a suitable control can be investigated.

Another research direction is the incorporation of force control in this model. There exists infinitely many force combinations that can be applied by the tip of the manipulators, and thus infinitely many joint torques, which will achieve the same desired trajectory for the common workpiece. This can be understood by looking at equation (2.21), in which we see that we can add any $\mathbf{f}_I$ to $\mathbf{f}$ which lies in the kernel of $\mathbf{J}_{00}$ and still obtain the same output trajectory. The choice of these forces are important in applications in which we want to exert a reasonable force on the workpiece, large enough to prevent it from slipping but small enough to prevent any damage. This issue has to be combined with position control in a real environment.

References

[1] T.-J. Tarn, A.K. Bejczy and X. Yun, "New Nonlinear Control Algorithms for Multiple Robot Arms", *IEEE Transactions on Aerospace and Electronic Systems*, vol. 24, No. 5, pp. 571-583, September 1988.

[2] Samad A. Hayati, "Position and Force Control of Coordinated Multiple Arms", *IEEE Transactions on Aerospace and Electronic Systems*,vol. 24, No. 5, pp. 584-590, September 1988.

[3] P. Hsu, "Coordinated Control of Multiple Manipulator Systems", *IEEE Transactions on Robotics and Automation*, vol. 9, No. 4, pp. 400-410, August 1993.

[4] P. Hsu, "Control of Mechanical Manipulators", Ph.D. dissertation, Dep. Elec. Eng. Comput. Sci., Univ. Calif., Berkeley, Nov. 1988.

[5] J.-H. Jean and L.-C. Fu, "An Adaptive Control Scheme for Coordinated Multimanipulator Systems", *IEEE Transactions on Robotics and Automation*, vol. 9, No. 2, pp. 226-231, April 1993.

[6] Z. Li, T.-J. Tarn and A.K. Bejczy, "Dynamic Workspace Analysis of Multiple Cooperating Robot Arms", *IEEE Transactions on Robotics and Automation*, vol. 7, No. 5, pp. 589-596, October 1991.

[7] J.E. Bobrow, J.M. McCarthy and V.K. Chu, "Minimum-Time Trajectories for Two Robots Holding the Same Workpiece", *Proceedings of the 29th Conference on Decision and Control*, pp. 3102-3107, December 1990.

[8] S.H. Murphy, J.T.-Y. Wen and G.N. Saridis, "Simulation of Cooperating Robot Manipulators on a Mobile Platform", *IEEE Transactions on Robotics and Automation*, vol. 7, No. 4, pp. 468-477, August 1991.

[9] K. Desoyer, P. Kopacek, P. Lugner and I. Troch, "Flexible Robots - A Survey", *Theory of Robots*, IFAC Proceeding Series, ed. P. Kopacek, I. Troch, K. Desoyer, No. 3, pp. 23-34, Pergamon Press, 1988.

[10] R.H.Jr. Cannon and E. Schmitz, "Precise Control of Flexible Manipulators", *Robotics Research, The First International Sysmposium*, ed. M. Brady, R. Paul, pp. 841-861, The MIT Press, Cambridge, 1984.

[11] G.G. Hastings and W.J. Book, "A Linear Dynamic Model for Flexible Robotic Manipulators", *IEEE Control Systems Magazine*, vol. 7, pp. 61-64, February 1987.

[12] D.C. Nemir, A.J. Koivo and R.L. Kashyap, "Pseudolinks and the Self-Tuning Control of a Nonrigid Link Mechanism", *IEEE Transactions on Systems, Man and Cybernetics*, vol. 18, No. 1, pp. 40-48, January/February 1988.

[13] J.O. King, V.G. Gourishankar and R.E. Rink, "Composite Pseudolink End-Point Control of Flexible Manipulators", *IEEE Transactions on Systems, Man and Cybernetics*, vol. 20, No. 5, pp. 969-977, September/October 1988.

[14] M. Bodur and M.E. Sezer, "Adaptive Control of Flexible Multilink Manipulators", *International Journal on Control*, vol. 58, No. 3, pp. 519-536, 1993.

[15] T.-C. Yang, J.C.S. Yang and P. Kudva, "Load-Adaptive Control of a Single-Link Flexible Manipulator", *IEEE Transactions on Systems, Man and Cybernetics*, vol. 22, No. 1, pp. 85-90, January/February 1988.

[16] M. Bodur, "Dynamic Position Control of Robot Manipulators", Ph.D. thesis, Dept. of Electrical and Electronics Eng., Middle East Technical Univ., Ankara, June 1991.

[17] M.W. Spong and M. Vidyasagar, " Robot Dynamics and Control", John Wiley & Sons Inc., 1989.

[18] A.J. Koivo. "Fundamentals for Control of Robotic Manipulators", John Wiley & Sons Inc., 1989.

[19] D.T. Greenwood, "Principles of Dynamics", Prentice-Hall Inc., 1988.

[20] J.J. Craig, "Adaptive Control of Mechanical Manipulators", Adison-Wesley Publishing Company, 1988.

[21] G.C. Goodwin and K.S. Sin, "Adaptive Filtering Prediction and Control", Prentice-Hall Inc., 1984.

[22] P. Kokotović, H.K. Khalil and J. O'Reilly, "Singular Perturbation Methods in Control: Analysis and Design", Academic Press, 1986.

[23] A. Ben-Israel and T.N.E. Greville, "Generalized Inverses, Theory and Applications", John Wiley & Sons Inc., 1974.

[24] H.R. Schwarz, "Numerical Analysis: A Comprehensive Introduction", John Wiley & Sons Inc., 1989.

Appendix A

Simulation Programs

A simulation program is written in C to test the proposed control method. The flowchart of the program and its listing are given below. A flowchart for the "Simulate" procedure is also given but as the construction of control and identification procedures are direct implementations of the derived formulas their flowcharts are not included.

The included library "Construct.h" includes the "ConsOrig" procedure which constructs the system matrices for a three link planar manipulator formulated in joint coordinates. "JD.h" includes the "ConsJD" procedure which is used to construct the conversion Jacobian $\mathbf{J}_{\boldsymbol{\theta}_k}$ in equation (2.23). Also "JVDD.h" includes the product $\dot{\mathbf{J}}_{\boldsymbol{\theta}_k}\dot{\mathbf{q}}$ given in the same equation. These Jacobians are computed by using the symbolic computation program "xmaple".

A sample input file for the program, which contains the name of the trajectory file, parameters for the simulation setting and control has the following form:

tr	trajectory file
yi	y:singular perturbation desired (else n),
	i:identification desired (else t)
10.0 8.0 2.0	link masses starting with the first being connected to base
0.5 0.25 0.25	link lengths with the first being connected to base
400.0 10.0	spring coefficients and dampings
1.0	manipulator spacing
5.0 0.1	workpiece mass and length
6.35 2.12	actuator inertias
1.30 0.43	actuator dampings
40.0 400.0	diagonal elements of $\bar{\mathbf{Z}}_v$ and $\bar{\mathbf{Z}}_p$
110.0 3000.0	diagonal elements of $\hat{\mathbf{Z}}_v$ and $\hat{\mathbf{Z}}_p$

10.0	1.0	1.0	-1.0	0.0	
1.0	-1.0	-0.1	-0.1	0.0	
1.0	-1.0	-0.01	-0.01	0.0	
-10.0	1.0	-0.1	0.0	-0.1	
-1.0	-1.0	0.1	0.0	-0.1	
-1.0	-1.0	0.01	0.0	-0.01	
0.05	-0.05	-0.05	0.5	-0.05	-0.05

initial parameter matrix

The trajectory file consists of three columns each of which contains the desired trajectories for the x-coordinate of the center of mass of the workpiece(m), y-coordinate of the center of mass of the workpiece(m) and orientation angle of the workpiece(rad), respectively.

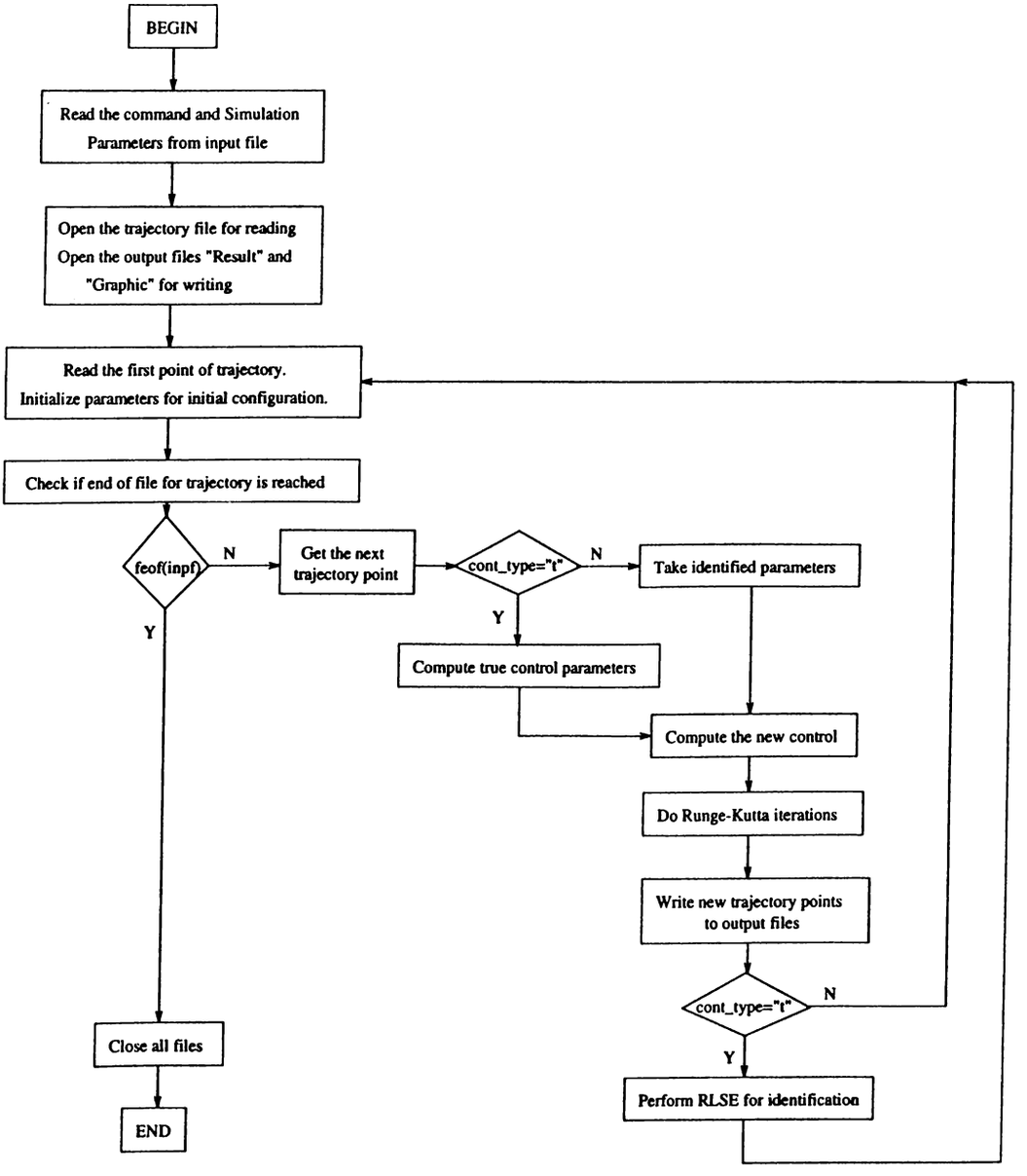


Figure A.1: *Flowchart for simulation program "rbsp.c"*

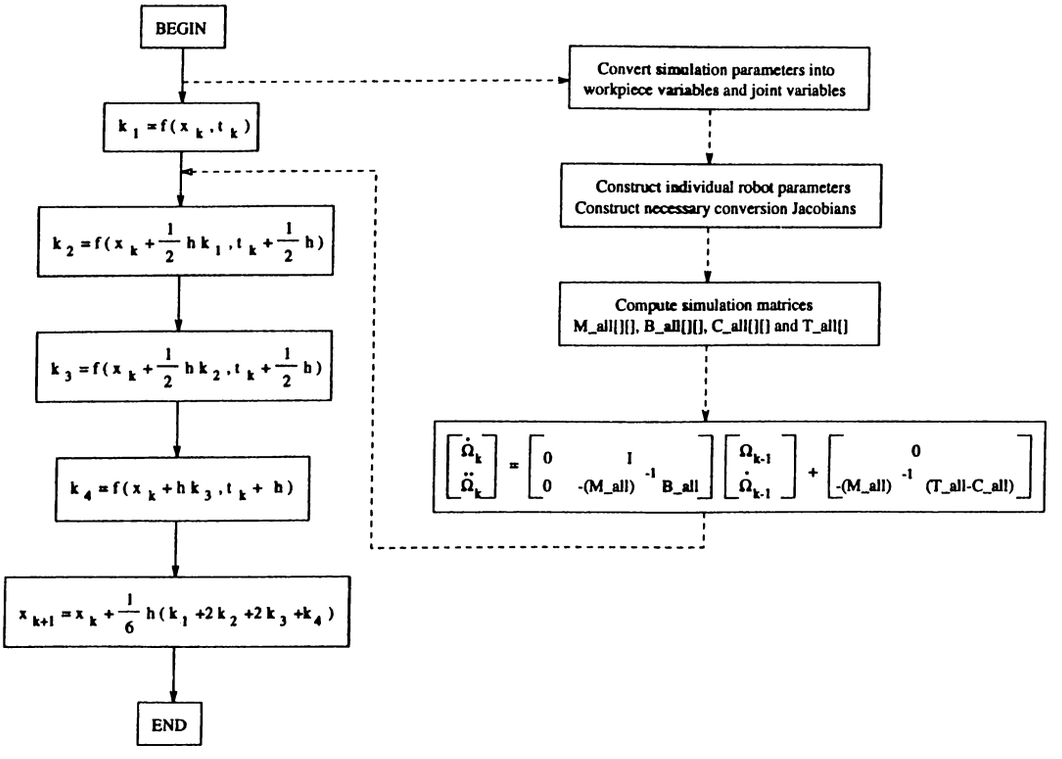


Figure A.2: *Flowchart for Simulation procedure*

Appendix B

Program Codes

```
/*********************************************/
/*            rbsp.c                         */
/*********************************************/
# include <stdio.h>
# include <math.h>
# include <malloc.h>
# define pi 3.141592654
# define g 9.806554
# define MAX 10

double sqr(a)
double a;
{
return (a*a);
}

double sgn (a)
int a;
{
```

```
    return (pow(-1.0,(double)a));
}

# include "JVDD.h"
# include "JD.h"
# include "Construct.h"


/* GLOBAL VARIABLES TO BE USED DURING SIMULATION */
/* ---------------------------------------------- */

double M_all[MAX][MAX],  /* overall simulation inertia matrix        */
    B_all[MAX][MAX], /* overall dampings coefficient matrix          */
    C_all[MAX][MAX], /* overall simulation vector of Coriolis,centrifugal
                        and gravity terms */
    T_all[MAX],      /* overall torque input used for simulation     */
    Mi_mat[MAX][MAX],
    Ci_vec[MAX],
    Ts[MAX],       /* vector of individual torques on robots         */
    mass[3][2],    /* masses of links for the two robots             */
    m_Load,        /* mass of the commonly held load of two robots   */
    teta[3][2],    /* joint angles of the two robots                 */
    teta_dot[3][2],  /* derivatives of joint angles                  */
    x_co[3], y_co[3], phi[3], /* load position and orientation       */
    x_des[3], y_des[3], phi_des[3], /* desired values of x,y,phi     */
    Ks[2], Kd[2], /* stiffness and dampings of fictitious joints     */
    var[10],       /* simulation variables and their derivatives     */
    h = 0.001,     /* Runge-Kutta(simulation) time step              */
    T = 0.005,     /* Control time step                              */
    gama = 0.9,    /* forgetting factor in identification            */
    P_id[MAX][MAX],     /* P matrix to be used during identification */
```

```
    par_id[MAX][MAX];   /* matrix to be identified containing M & C  */

 char sing_pert;

 double leng[3][2],   /* lengths of links for the two robots          */
    D,            /* horizontal distance between the bases of robots */
    l_Load,       /* length of hanging load                          */
    I[3][2],      /* rotational inertias of links                    */
    I_Load,       /* rotational inertia of the load                  */
    Act_damp[2][2], /* dampings of joint actuators                   */
    Act_iner[2][2], /* effective inertias of joint actuators         */
    pseudo[3][2],    /* pseudo joint angles : pseudo[t-2,t-1,t][1,2] */
    pseudo_dot[2],              /* first derivatives                 */
    tip[2][2],                  /* tip coordinates : tip[x,y][1,2]   */
    x_co_dot, y_co_dot, phi_dot, /* first derivatives                */
    M_mat[MAX][MAX][2],  /* inertia matrices of robot 1 & 2          */
    B_mat[MAX][MAX][2],  /* Damping Coefficients of actuators        */
    C_vec[3][2],         /* sum of G_vec and V_vec                   */
    J_mat[2][3][2],  /* matrix transforming the force effects        */
    JVDD[MAX][MAX][2], /* resulting vector from double derivative Jac. */
    JD[MAX][MAX][2],   /* first derivative Jacobian to form teta_dot's */
    Z1b[MAX][MAX],
    Z2b[MAX][MAX],   /* matrices regarding the decoupled slow control */
    Z1h[MAX][MAX],
    Z2h[MAX][MAX];   /* matrices regarding the decoupled fast control */


/* MAIN MATRIX FUNCTIONS TO BE USED DURING SIMULATION */
/* ------------------------------------------------- */

/* M A T R I X    M U L T I P L I C A T I O N */
```

```
 void rmult (res,mat1,mat2,d0,d1,d2)
 double mat1[MAX][MAX], mat2[MAX][MAX], res[MAX][MAX];
 int d0,d1,d2;
 {
 int i,j,k;

 for (i=0; i<d0; i++)
     for (j=0; j<d2; j++) {
         res[i][j] = 0;
         for (k=0; k<d1; k++)
             res[i][j] += mat1[i][k]*mat2[k][j];
     }
 }


 void mult (res,mat1,mat2,d0,d1,d2,k)
 double mat1[MAX][MAX][2], mat2[MAX][MAX][2], res[MAX][MAX];
 int d0,d1,d2,k;
 {
 int i, j;
 double a[MAX][MAX], b[MAX][MAX];

 for (i=0; i<d0; i++)
     for (j=0; j<d1; j++)
         a[i][j] = mat1[i][j][k];
 for (i=0; i<d1; i++)
     for (j=0; j<d2; j++)
         b[i][j] = mat2[i][j][k];
 rmult(res,a,b,d0,d1,d2);
 }
```

```
/* M A T R I X     I N V E R S I O N */

void interchange(p,q,n,mat,mat_inv)
int p,q,n;
double mat[MAX][MAX],mat_inv[MAX][MAX];
{
double t;
int i;

for (i=0; i<n; i++) {
    t=mat[p][i];
    mat[p][i]=mat[q][i];
    mat[q][i]=t;
    t=mat_inv[p][i];
    mat_inv[p][i]=mat_inv[q][i];
    mat_inv[q][i]=t;
}
}


void inv(mat_inv,mat,n)
int n;
double mat[MAX][MAX],mat_inv[MAX][MAX];
{
int i,j,p,q;
double a;

for (i=0; i<n; i++)
   for (j=0; j<n; j++)
```

```
        if (i==j)
            mat_inv[i][j]=1;
        else
            mat_inv[i][j]=0;
for (q=0; q<n; q++) {
    p=q;
    while ((p<n) && (mat[p][q]==0))
        p++;
    if (p==n) {
        printf ("\nSINGULAR");
        break;
    }
    if (p!=q)
        interchange(p,q,n,mat,mat_inv);
    a=mat[q][q];
    for (i=0; i<n; i++) {
        mat[q][i] /= a;
        mat_inv[q][i] /= a;
    }
    for (i=0; i<q; i++) {
        a=mat[i][q];
        for (j=0; j<n; j++) {
            mat[i][j] -= a*mat[q][j];
            mat_inv[i][j] -= a*mat_inv[q][j];
        }
    }
    for (i=q+1; i<n ;i++) {
        a=mat[i][q];
        for (j=0; j<n; j++) {
            mat[i][j] -= a*mat[q][j];
            mat_inv[i][j] -= a*mat_inv[q][j];
```

```
        }
     }
  }  /* for */

  }  /* inv */

  /* M A I N    P R O G R A M    F U N C T I O N S */
  /* ------------------------------------------- */

  void Convert (t_var) /* Converts from t_var(simulation variables) to   */
  double t_var[14];    /* real variables(joint variables,derivatives...) */
  {
  int i, k;
  double l4, l5, py, px, gama, psi;

  x_co[2] = t_var[0];    x_co_dot = t_var[5];
  y_co[2] = t_var[1];    y_co_dot = t_var[6];
  phi[2]  = t_var[2];    phi_dot  = t_var[7];

  for (i=0; i<2; i++) {
      pseudo[2][i] = t_var[3+i];
      pseudo_dot[i] = t_var[8+i];
  }

  for (i=0; i<2; i++) {
      l4 = sqrt(sqr(leng[1][i])+sqr(leng[2][i])+2.*leng[1][i]*leng[2][i
          ]*cos(pseudo[2][i]));

      px = x_co[2] + sgn(i+1)*l_Load*cos(phi[2]);
      tip[0][i] = px;
      py = y_co[2] + sgn(i)*l_Load*sin(phi[2]);
```

```
        tip[1][i] = py;

        if (i==0)
            l5 = sqrt(sqr(py)+sqr(px));
        else l5 = sqrt(sqr(py)+sqr(px-D));
        gama = acos((sqr(leng[1][i])+sqr(l4)-sqr(leng[2][i]))/(2.*leng[1][i]*l4));
        if (pseudo[2][i]<0.0)
            gama = -gama;
        psi = acos((pow(leng[0][i],2.)+pow(l5,2.)-pow(l4,2.))/(2.*leng[0][i]*l5));

        teta[0][i] = pi - psi;
        if (i==0)
            teta[0][i] -= atan(py/px);
        else teta[0][i] -= atan(py/(D-px));

        teta[1][i] = acos((sqr(l5)-sqr(leng[0][i])-sqr(l4))/(2.*leng[0][i]*l4)) -
            gama;

        for (k=0; k<2; k++)
            teta_dot[k][i] = JD[k][0][i]*x_co_dot + JD[k][1][i]*y_co_dot + JD[k]
                [2][i]*phi_dot + JD[k][3][i]*pseudo_dot[0] + JD[k][4][i]*
                pseudo_dot[1];
        teta[2][i] = pseudo[2][i];
        teta_dot[2][i] = pseudo_dot[i];
    }

    } /* Convert */


    void Construct (p)    /* to construct the simulation matrices */
    double p;             /* M_all[5][5], B_all[5][5], C_all[5][1], T_all[5][1] */
```

```
{
double S[2], Mn[MAX][MAX], Bn[MAX][MAX], Cn[MAX][MAX];
double Mn_mat[MAX][MAX][2], Bn_mat[MAX][MAX][2], Cn_vec[MAX][2],
    Mf[MAX][MAX], Bf[MAX][MAX], Cf[MAX][MAX], Te[MAX][MAX],
    Kf[MAX][MAX], Rf[MAX][MAX],
    Ms[MAX][MAX], Bs[MAX][MAX], Cs[MAX][MAX], Ls[MAX][MAX],
    res1[MAX][MAX], res2[MAX][MAX];
int c, i, j, k;

ConsOrig ();          /* Construct simulation matrices in joint Coordinates */

for (i=0; i<2; i++)
    if (pseudo[2][i]==0.0)
        S[i] = 0.0;
    else if (pseudo[2][i]<0.0)
        S[i] = -1.0;
    else
        S[i] = 1.0;

ConsJVDD (phi[2],S);
ConsJD (phi[2],S);                    /* Construct the conversion Jacobians */

for (c=0; c<2; c++) {
    mult (Mn,M_mat,JD,3,3,5,c);
    mult (Bn,B_mat,JD,3,3,5,c);
    for (i=0; i<3; i++)
        for (j=0; j<5; j++) {
            Mn_mat[i][j][c] = Mn[i][j];
            Bn_mat[i][j][c] = Bn[i][j];
        }
   mult(Cn,M_mat,JVDD,3,3,1,c);
```

```
    for (i=0; i<3; i++)
        Cn_vec[i][c] = Cn[i][0] + C_vec[i][c];
}

for (i=0; i<2; i++)
    for (j=0; j<5; j++) {
        Mf[i][j] = m_Load*(double)(i==j);
        Mf[2+i][j] = Mn_mat[2][j][i];
        Bf[i][j] = 0.0;
        Bf[2+i][j] = Bn_mat[2][j][i];
    }
Cf[0][0] = 0.0;            Cf[1][0] = m_Load*g;
Cf[2][0] = Cn_vec[2][0];   Cf[3][0] = Cn_vec[2][1];

for (i=0; i<2; i++) {
    Te[i][0] = 0.0;
    Te[2+i][0] = -Ks[i]*pseudo[2][i] - 2./h/p*Kd[i]*(pseudo[2][i]-pseudo[1][i]);
}

for (i=0; i<2; i++)
    for (j=0; j<4; j++)
        Kf[i][j] = (double)(i==j) + (double)((2+i)==j);
Kf[2][0] = -J_mat[0][2][0];
Kf[2][1] = -J_mat[1][2][0];
Kf[2][2] = 0.0;            Kf[2][3] = 0.0;
Kf[3][0] = 0.0;            Kf[3][1] = 0.0;
Kf[3][2] = -J_mat[0][2][1];
Kf[3][3] = -J_mat[1][2][1];
inv (Rf,Kf,4);

for (i=0; i<2; i++)
```

```
        for (j=0; j<2; j++) {
            for (k=0; k<5; k++) {
                Ms[2*i+j][k] = Mn_mat[j][k][i];
                Bs[2*i+j][k] = Bn_mat[j][k][i];
            }
            Cs[2*i+j][0] = Cn_vec[j][i];
        }
    for (j=0; j<5; j++) {
        Ms[4][j] = I_Load*(double)(2==j);
        Bs[4][j] = 0.0;
    }
    Cs[4][0] = 0.0;

    Ls[0][0] = J_mat[0][0][0];
    Ls[0][1] = J_mat[1][0][0];
    Ls[0][2] = 0.0;          Ls[0][3] = 0.0;
    Ls[1][0] = J_mat[0][1][0];
    Ls[1][1] = J_mat[1][1][0];
    Ls[1][2] = 0.0;          Ls[1][3] = 0.0;
    Ls[2][0] = 0.0;          Ls[2][1] = 0.0;
    Ls[2][2] = J_mat[0][0][1];
    Ls[2][3] = J_mat[1][0][1];
    Ls[3][0] = 0.0;          Ls[3][1] = 0.0;
    Ls[3][2] = J_mat[0][1][1];
    Ls[3][3] = J_mat[1][1][1];
    Ls[4][0] = -l_Load*sin(phi[2]);
    Ls[4][1] = -l_Load*cos(phi[2]);
    Ls[4][2] = l_Load*sin(phi[2]);
    Ls[4][3] = l_Load*cos(phi[2]);

    rmult (res1,Ls,Rf,5,4,4);
```

```
 rmult (res2,res1,Mf,5,4,5);
 for (i=0; i<5; i++)
     for (j=0; j<5; j++)
         M_all[i][j] = Ms[i][j] + res2[i][j];

 rmult (res2,res1,Bf,5,4,5);
 for (i=0; i<5; i++)
     for (j=0; j<5; j++)
         B_all[i][j] = Bs[i][j] + res2[i][j];

 rmult (res2,res1,Cf,5,4,1);
 for (i=0; i<5; i++)
     C_all[i][0] = Cs[i][0] + res2[i][0];

 rmult (res2,res1,Te,5,4,1);
 for (i=0; i<5; i++)
     T_all[i] = Ts[i] + res2[i][0];

} /* Construct   */


double *Compute (t_var,c)      /* returns the k vector for R.K. iterations */
double t_var[10];
double c;
{
int i, j;
double m_inv[MAX][MAX], *k, res[5];

Shift_ps ();

Convert (t_var);
```

```
 Construct (c);                          /* constructs M_all, B_all, C_all    */

 k = (double *)calloc(10,sizeof(double));

 for (i=0; i<5; i++)
     k[i] = t_var[5+i];

 inv (m_inv,M_all,5);
 for (i=0; i<5; i++) {
     res[i] = T_all[i] - C_all[i][0];
     for (j=0; j<5; j++)
         res[i] -= B_all[i][j]*k[j];
     }
 for (i=0; i<5; i++) {
     k[5+i] = 0.0;
     for (j=0; j<5; j++)
         k[5+i] += m_inv[i][j]*res[j];
     }

return(k);
}  /* Compute  */


void Simulate()                 /* Takes current var[10], outputs new var[10] */
{
double *k1, *k2, *k3, *k4, t_var[10];
int i;

k1 = Compute (var,1.0);
```

```
  for (i=0; i<10; i++)
      t_var[i] = var[i] + 1/2.*h*k1[i];
  k2 = Compute (t_var,1.0);

  for (i=0; i<10; i++)
      t_var[i] = var[i] + 1/2.*h*k2[i];
  k3 = Compute (t_var,1.0);

  for (i=0; i<10; i++)
      t_var[i] = var[i] + h*k3[i];
  k4 = Compute (t_var,2.0);

  for (i=0; i<10; i++)
      var[i] += 1/6.*h*(k1[i]+2*k2[i]+2*k3[i]+k4[i]);

  }  /* Simulate */


 void Identified ()
 {
 int i, j;

 for (i=0; i<6; i++)
     for (j=0; j<5; j++)
         Mi_mat[i][j] = par_id[i][j];

 for (i=0; i<6; i++)
     Ci_vec[i] = par_id[i][5]/T;      /* C_all vector identified as Ci_vec[] */

 }  /* Identified */
```

```
 void Identify ()
 {
 double regr[MAX][MAX], regrt[MAX][MAX], res1[MAX][MAX], res2[MAX][MAX],
     res3[MAX][MAX], output[MAX], alpha;
 int i, j, trouble=0;

 for (i=0; i<2; i++)
     output[3*i+2] = -T*Ks[i]*pseudo[2][i] - Kd[i]*(pseudo[2][i]-pseudo[1][i]);
 for (i=0; i<2; i++) {
     output[3*i] = T*Ts[2*i];
     output[3*i+1] = T*Ts[2*i+1];
 }

 regr[0][0] = 1./T*(x_co[2]-2.*x_co[1]+x_co[0]);
 regr[1][0] = 1./T*(y_co[2]-2.*y_co[1]+y_co[0]);
 regr[2][0] = 1./T*(phi[2]-2.*phi[1]+phi[0]);
 regr[3][0] = 1./T*(pseudo[2][0]-2.*pseudo[1][0]+pseudo[0][0]);
 regr[4][0] = 1./T*(pseudo[2][1]-2.*pseudo[1][1]+pseudo[0][1]);
 regr[5][0] = 1.0;
 for (i=0; i<6; i++)
         regrt[0][i] = regr[i][0];

 rmult(res1,regrt,P_id,1,6,6);                              /* 24.6.1994 */
 rmult(res2,res1,regr,1,6,1);
 alpha = 1./(1+res2[0][0]);

 rmult(res1,par_id,regr,6,6,1);
 for (i=0; i<6; i++)
     res2[i][0] = output[i] - res1[i][0];

 rmult(res1,regrt,P_id,1,6,6);
```

```
  rmult(res3,res2,res1,6,1,6);

  for (i=0; i<6; i++)
      for (j=0; j<6; j++)
          par_id[i][j] += alpha*res3[i][j];
  Identified ();

  rmult(res1,P_id,regr,6,6,1);
  rmult(res2,res1,regrt,6,1,6);
  rmult(res1,res2,P_id,6,6,6);
  for (i=0; i<6; i++)
      for (j=0; j<6; j++)
          P_id[i][j] = P_id[i][j]/gama - alpha/gama*res1[i][j];

 for (i=0; i<6; i++)
     if (fabs(P_id[i][i])>50.0)
         trouble = 1;
 if (trouble==1)
     for (i=0; i<6; i++)
         for (j=0; j<6; j++)
             P_id[i][j] = 20.0*(double)(i==j);
 trouble = 0;

 }  /* Identify */


 void fund_Control (matdd,vec)
 double matdd[MAX][MAX], vec[MAX];
 {
 double Mpp[MAX][MAX], Mpe[MAX][MAX], Mep[MAX][MAX],
     Mee[MAX][MAX], cp[MAX][MAX], ce[MAX][MAX],
```

```
     K1h[MAX][MAX], K2h[MAX][MAX],
     ek_1[MAX][MAX], ek_2[MAX][MAX], epsk_1[MAX][MAX],
     d2p_k[MAX][MAX], temp;
 double res1[MAX][MAX], res2[MAX][MAX], res3[MAX][MAX], res4[MAX][MAX];
 int i, j;

 for (j=0; j<5; j++) {
     temp = matdd[2][j];
     matdd[2][j] = matdd[3][j];
     matdd[3][j] = matdd[4][j];
     matdd[4][j] = temp;
 }
 temp = vec[2];   vec[2] = vec[3];   vec[3] = vec[4];   vec[4] = temp;

 d2p_k[0][0] = (x_des[2]-2.*x_des[1]+x_des[0])/sqr(T);
 d2p_k[1][0] = (y_des[2]-2.*y_des[1]+y_des[0])/sqr(T);
 d2p_k[2][0] = (phi_des[2]-2.*phi_des[1]+phi_des[0])/sqr(T);

 for (i=0; i<4; i++)
     for (j=0; j<3; j++)
         Mpp[i][j] = matdd[i][j];
 rmult (res1,Mpp,d2p_k,4,3,1);

 for (i=0; i<4; i++)
     cp[i][0] = vec[i];
 for (i=0; i<2; i++)
     ce[i][0] = vec[4+i];

 for (i=0; i<4; i++)
     Ts[i] = res1[i][0] + cp[i][0];
```

```
rmult (res1,Mpp,Z1b,4,3,3);
rmult (res2,Mpp,Z2b,4,3,3);
ek_1[0][0] = /* ep(dot) */ (x_des[1]-x_des[0])/T - (x_co[2]-x_co[1])/T;
ek_1[1][0] = /* ep(dot) */ (y_des[1]-y_des[0])/T - (y_co[2]-y_co[1])/T;
ek_1[2][0] = /* ep(dot) */ (phi_des[1]-phi_des[0])/T - (phi[2]-phi[1])/T;
ek_2[0][0] = /* ep      */ x_des[1] - x_co[2];
ek_2[1][0] = /* ep      */ y_des[1] - y_co[2];
ek_2[2][0] = /* ep      */ phi_des[1] - phi[2];
rmult (res3,res1,ek_1,4,3,1);
rmult (res4,res2,ek_2,4,3,1);

for (i=0; i<4; i++)
    Ts[i] += res3[i][0] + res4[i][0];

Ts[4] = 0.0;

if (sing_pert=='y') {
    for (i=0; i<4; i++)
        for (j=0; j<3; j++)
            Mpp[i][j] = matdd[i][j];
    for (i=0; i<4; i++)
        for (j=0; j<2; j++)
            Mpe[i][j] = matdd[i][3+j];
    for (i=0; i<2; i++)
        for (j=0; j<3; j++)
            Mep[i][j] = matdd[4+i][j];
    for (i=0; i<2; i++)
        for (j=0; j<2; j++)
            Mee[i][j] = matdd[4+i][3+j];

    for (i=0; i<3; i++)
```

```
    for (j=0; j<2; j++)
        res1[i][j] = Mep[j][i];
rmult (res2,Mep,res1,2,3,2);
inv (res3,res2,2);
rmult (res2,res1,res3,3,2,2);

rmult (res4,Mpp,res2,4,3,2);

rmult (res3,res4,Mee,4,2,2);
for (i=0; i<4; i++)
    for (j=0; j<2; j++)
        res1[i][j] = Mpe[i][j] - res3[i][j];

rmult (K1h,res1,Z1h,4,2,2);
for (i=0; i<4; i++)
    for (j=0; j<2; j++)
        K1h[i][j] = -K1h[i][j];

for (i=0; i<4; i++)
    for (j=0; j<2; j++)
        res2[i][j] = -res4[i][j]*Ks[j];

rmult (res3,res1,Z2h,4,2,2);
for (i=0; i<4; i++)
    for (j=0; j<2; j++)
        K2h[i][j] = res2[i][j] - res3[i][j];

for (i=0; i<2; i++)
    epsk_1[i][0] = /* eps_cap(dot) */ (pseudo[2][i]-pseudo[1][i])/T;

res1[0][0] = (x_co[2]-2.*x_co[1]+x_co[0])/sqr(T);
```

```
    res1[1][0] = (y_co[2]-2.*y_co[1]+y_co[0])/sqr(T);
    res1[2][0] = (phi[2]-2.*phi[1]+phi[0])/sqr(T);
    rmult (res2,Mep,res1,2,3,1);
    for (i=0; i<2; i++)
        res3[i][0] = pseudo[2][i] + (res2[i][0]+ce[i][0])/Ks[i];

    rmult (res1,K1h,epsk_1,4,2,1);
    rmult (res2,K2h,res3,4,2,1);

}

} /* fund_Control */


void T_Control ()
{
double Mn[MAX][MAX], Bn[MAX][MAX], Cn[MAX][MAX],
    Mn_mat[MAX][MAX][2], Bn_mat[MAX][MAX][2], Cn_vec[MAX][2],
    At[MAX][MAX], A[MAX][MAX], Mo[MAX][MAX], Co[MAX][MAX],
    matdd[MAX][MAX], vec[MAX];
double res1[MAX][MAX], res2[MAX][MAX], res3[MAX][MAX], res4[MAX][MAX];
int i, j, c;

for (c=0; c<2; c++) {
    mult (Mn,M_mat,JD,3,3,5,c);
    mult (Bn,B_mat,JD,3,3,5,c);
    for (i=0; i<3; i++)
        for (j=0; j<5; j++) {
            Mn_mat[i][j][c] = Mn[i][j];
            Bn_mat[i][j][c] = Bn[i][j];
        }
```

```
        mult(Cn,M_mat,JVDD,3,3,1,c);
        for (i=0; i<3; i++)
            Cn_vec[i][c] = Cn[i][0] + C_vec[i][c];
    }

    for (c=0; c<2; c++)
        for (i=0; i<3; i++) {
            for (j=0; j<5; j++)
                matdd[3*c+i][j] = Mn_mat[i][j][c];
            vec[3*c+i] = Cn_vec[i][c];
        }

    for (i=0; i<2; i++)
        for (j=0; j<4; j++)
            At[i][j] = (double)(i==j) + (double)((2+i)==j);
    At[2][0] = l_Load*sin(phi[2]);
    At[2][1] = l_Load*cos(phi[2]);
    At[2][2] = -At[2][0];
    At[2][3] = -At[2][1];
    for (i=0; i<4; i++)
        for (j=0; j<3; j++)
            A[i][j] = At[j][i];
    rmult (res1,At,A,3,4,3);
    inv (res2,res1,3);
    rmult (res1,A,res2,4,3,3);

    for (i=0; i<3; i++) {
        for (j=0; j<5; j++)
            Mo[i][j] = m_Load*(double)(i==j);
        Co[i][0] = 0.0;
    }
```

```
Mo[2][2] = I_Load;
Co[1][0] = m_Load*g;

for (c=0; c<2; c++) {
    for (i=0; i<3; i++)
        for (j=0; j<2; j++)
            res2[i][j] = J_mat[j][i][c];
    for (i=0; i<2; i++)
        for (j=0; j<3; j++)
            res3[i][j] = res1[2*c+i][j];
    rmult (res4,res2,res3,3,2,3);

    rmult (res2,res4,Mo,3,3,5);
    rmult (res3,res4,Co,3,3,1);

    for (i=0; i<3; i++) {
        for (j=0; j<5; j++)
            matdd[3*c+i][j] += res2[i][j];
        vec[3*c+i] += res3[i][0];
    }
}

fund_Control (matdd,vec);

}  /* T_Control */


I_Control ()
{

fund_Control (Mi_mat,Ci_vec);
```

```
 }


 Shift_co ()
 {
 int i;
 for (i=0; i<2; i++) {
     x_co[i] = x_co[i+1];
     y_co[i] = y_co[i+1];
     phi[i] = phi[i+1];

 }
 }  /* Shift_co */


Shift_des ()
{
int i;

for (i=0; i<2; i++) {
    x_des[i] = x_des[i+1];
    y_des[i] = y_des[i+1];
    phi_des[i] = phi_des[i+1];
}
}  /* Shift_des */


Shift_ps ()
{
int i;
```

```
 for (i=0; i<2; i++) {
     pseudo[0][i] = pseudo[1][i];
     pseudo[1][i] = pseudo[2][i];
 }
 }  /* Shift_ps */


 Initialize (iptr)
 FILE *iptr;
 {
 int i,j,k;
 double n1, n2, n3, n4;

 x_co[2] = x_des[2];
 y_co[2] = y_des[2];
 phi[2] = phi_des[2];
 for (i=0; i<2; i++) {
     x_des[i] = x_des[2];
     y_des[i] = y_des[2];
     phi_des[i] = phi_des[2];
     x_co[i] = x_des[2];
     y_co[i] = y_des[2];
     phi[i] = phi_des[2];
     for (j=0; j<3; j++) {
         pseudo[j][i] = 0;
     }
}
x_co_dot = 0.0;
y_co_dot = 0.0;
phi_dot = 0.0;
```

```
pseudo_dot[0] = 0.0;      pseudo_dot[1] = 0.0;

var[0] = x_co[2];     var[5] = x_co_dot;
var[1] = y_co[2];     var[6] = y_co_dot;
var[2] = phi[2];      var[7] = phi_dot;
for (i=0; i<2; i++) {
    var[3+i] = pseudo[2][i];
    var[8+i] = pseudo_dot[i];
}

fscanf (iptr, "%lf %lf %lf", &mass[0][0], &mass[1][0], &mass[2][0]);
fscanf (iptr, "%lf %lf %lf", &leng[0][0], &leng[1][0], &leng[2][0]);
for (i=0; i<3; i++) {
    mass[i][1] = mass[i][0];
    leng[i][1] = leng[i][0];
}
for (i=0; i<3; i++)
    for (j=0; j<2; j++)
        I[i][j] = 1./12.*mass[i][j]*sqr(leng[i][j]);

fscanf (iptr, "%lf %lf", &Ks[0], &Kd[0]);
Ks[1] = Ks[0];          Kd[1] = Kd[0];

fscanf (iptr, "%lf", &D);

fscanf (iptr, "%lf %lf", &m_Load, &l_Load);
I_Load = 1./12.*m_Load*sqr(l_Load);

fscanf (iptr, "%lf %lf %lf %lf", &Act_iner[0][0], &Act_damp[0][0],
    &Act_iner[1][0], &Act_damp[1][0]);
for (i=0; i<2; i++) {
```

```
        Act_damp[i][1] = Act_damp[i][0];
        Act_iner[i][1] = Act_iner[i][0];
    }

    for (i=0; i<3; i++)
        for (j=0; j<5; j++)
            for (k=0; k<2; k++)
                JD[i][j][k] = 0.0;

    fscanf (iptr, "%lf %lf %lf %lf", &n1, &n2, &n3, &n4);
    for (i=0; i<3; i++)
        for (j=0; j<3; j++) {
            Z1b[i][j] = (double)(i==j)*n1;
            Z2b[i][j] = (double)(i==j)*n2;
        }
    for (i=0; i<2; i++)
        for (j=0; j<2; j++) {
            Z1h[i][j] = (double)(i==j)*n3;
            Z2h[i][j] = (double)(i==j)*n4;
        }

    for (i=0; i<6; i++)
        for (j=0; j<6; j++)
            P_id[i][j] = 20.0*(double)(i==j);

    for (i=0; i<6; i++)
        for (j=0; j<5; j++)
            fscanf (iptr, "%lf ", &par_id[i][j]);
    for (i=0; i<6; i++)
        fscanf (iptr, "%lf ", &par_id[i][5]);
```

```
 } /* Initialize */


 main(argc,argv)
 int argc;
 char *argv[];
 {
 FILE *iptr, *inpf, *outf, *grf;
 int i, j, p=0;
 double t1_pseudo[2][2], t2_pseudo[2][2];
 char cont_type, inname[10];

 iptr = fopen(argv[1],"r");

 fscanf (iptr, "%s", inname);
 fscanf (iptr, "%c", &sing_pert);
 fscanf (iptr, "%c", &sing_pert);
 fscanf (iptr, "%c", &cont_type);

 inpf = fopen(inname,"r");
 outf = fopen("result","w");
 grf = fopen("graphic","w");

 fscanf (inpf, "%lf %lf %lf", &x_des[2], &y_des[2], &phi_des[2]);
 fprintf (outf, "%11f %11f %11f %11f %11f %11f\n",
     x_des[2], x_des[2], y_des[2], y_des[2], phi_des[2], phi_des[2]);

 Initialize (iptr);
 for (i=0; i<2; i++)
     for (j=0; j<2; j++) {
         t1_pseudo[i][j] = 0.0;
```

```
        t2_pseudo[i][j] = 0.0;
    }

Identify ();
while (!feof(inpf)) {
    Shift_des ();
    fscanf (inpf, "%lf %lf %lf", &x_des[2], &y_des[2], &phi_des[2]);

    if (cont_type=='t')
        T_Control ();
    else I_Control ();

    Shift_co ();

    for (i=0; i<2; i++)
        for (j=0; j<2; j++) {
            t1_pseudo[j][i] = pseudo[j+1][i];
            pseudo[j+1][i] = t2_pseudo[j][i];
        }

    for (i=0; i<5; i++) {
        Simulate ();

        fprintf (outf, "%11f %11f %11f %11f %11f %11f\n",
            x_des[2], x_co[2], y_des[2], y_co[2], phi_des[2], phi[2]);
    }

    if (p%6==0)
    fprintf (grf, "%9f %9f %9f %9f %9f %9f %9f %9f %9f\n",
        x_co[2], y_co[2], phi[2], teta[0][0], teta[1][0], teta[2][0],
        teta[0][1], teta[1][1], teta[2][1]);
```

```
        for (i=0; i<2; i++)
            for (j=0; j<2; j++)
                t2_pseudo[j][i] = pseudo[j+1][i];

        Convert (var);

        for (i=0; i<2; i++)
            for (j=0; j<2; j++)
                pseudo[j][i] = t1_pseudo[j][i];

        if (cont_type=='i')
            Identify ();

    }  /* while */

    fclose (inpf);
    fclose (outf);
    fclose (grf);
    }
```

```
/*********************************/
/*     Construct.h               */
/*********************************/
# define pi 3.141592654
# define g 9.806554
# define MAX 10

extern double
    leng[3][2],             /* lengths of links for the two robots   */
    I[3][2],          /* rotational inertias of links               */
    Act_damp[2][2],   /* dampings of joint actuators                */
    Act_iner[2][2],   /* effective inertias of joint actuators      */
    D;       /* horizontal distance between two bases of manipulators */

extern double
    M_mat[MAX][MAX][2],  /* inertia matrices of robot 1 & 2         */
    B_mat[MAX][MAX][2],  /* Damping coefficients of actuators       */
    C_vec[3][2],         /* sum of gravitational and Coriolis forces */
    J_mat[2][3][2],      /* matrix transforming the force effects   */
    mass[3][2],    /* masses of links for the two robots            */
    teta[3][2],      /* joint angles of the two robots              */
    teta_dot[3][2];  /* derivatives of joint angles                 */

ConsOrig ()
{
int i;
double m;

for (i=0; i<2; i++) {
```

```
    /* ----  Construct_M_mat  ---- */

 M_mat[0][0][i] = I[0][i] + I[1][i] + I[2][i] + (1./4.*mass[0][i]+mass[1][i]
 +mass[2][i])*sqr(leng[0][i]) + (1./4.*mass[1][i]+mass[2][i])*sqr(leng[1][i])
 + (mass[1][i]+2.*mass[2][i])*leng[0][i]*leng[1][i]*cos(teta[1][i]) + 1./4.*
 mass[2][i]*sqr(leng[2][i]) + mass[2][i]*leng[0][i]*leng[2][i]*cos(teta[1][i]
 +teta[2][i]) + mass[2][i]*leng[1][i]*leng[2][i]*cos(teta[2][i]);

 M_mat[0][1][i] = I[1][i] + I[2][i] + (1./4.*mass[1][i]+mass[2][i])*sqr(leng
 [1][i]) + (1./2.*mass[1][i]+mass[2][i])*leng[0][i]*leng[1][i]*cos(teta[1][i])
 + 1./4.*mass[2][i]*sqr(leng[2][i]) + 1./2.*mass[2][i]*leng[0][i]*leng[2][i]*
 cos(teta[1][i]+teta[2][i]) + mass[2][i]*leng[1][i]*leng[2][i]*cos(teta[2][i]);

 M_mat[0][2][i] = I[2][i] + 1./4.*mass[2][i]*sqr(leng[2][i]) + 1./2.*mass[2]
 [i]*leng[0][i]*leng[2][i]*cos(teta[1][i]+teta[2][i]) + 1./2.*mass[2][i]*leng
 [1][i]*leng[2][i]*cos(teta[2][i]);

 M_mat[1][0][i] = M_mat[0][1][i];

 M_mat[1][1][i] = I[1][i] + I[2][i] + (1./4.*mass[1][i]+mass[2][i])*sqr(leng
 [1][i]) + 1./4.*mass[2][i]*sqr(leng[2][i]) + mass[2][i]*leng[1][i]*leng[2][i]
 *cos(teta[2][i]);

 M_mat[1][2][i] = I[2][i] + 1./4.*mass[2][i]*sqr(leng[2][i]) + 1./2.*mass[2]
 [i]*leng[1][i]*leng[2][i]*cos(teta[2][i]);

 M_mat[2][0][i] = M_mat[0][2][i];

 M_mat[2][1][i] = M_mat[1][2][i];

 M_mat[2][2][i] = I[2][i] + 1./4.*mass[2][i]*sqr(leng[2][i]);
```

```
    /* ----  Construct_B_mat  ---- */

B_mat[0][0][i] = Act_damp[0][i];

B_mat[0][1][i] = 0.0;

B_mat[0][2][i] = 0.0;

B_mat[1][0][i] = 0.0;

B_mat[1][1][i] = Act_damp[1][i];

B_mat[1][2][i] = 0.0;

B_mat[2][0][i] = 0.0;

B_mat[2][1][i] = 0.0;

B_mat[2][2][i] = 0.0;

    /* ----  Construct_C_vec  ---- */

C_vec[0][i] = -(1./2.*mass[1][i]+mass[2][i])*leng[0][i]*leng[1][i]*sin(teta
[1][i])*teta_dot[1][i]*(2.*teta_dot[0][i]+teta_dot[1][i]) -1./2.*mass[2][i]*
leng[0][i]*leng[2][i]*sin(teta[1][i]+teta[2][i])*(teta_dot[1][i]+teta_dot[2]
[i])*(2.*teta_dot[0][i]+teta_dot[1][i]+teta_dot[2][i]) - 1./2.*mass[2][i]*
leng[1][i]*leng[2][i]*sin(teta[2][i])*teta_dot[2][i]*(2.*teta_dot[0][i]+2.*
teta_dot[1][i]+teta_dot[2][i]) + mass[0][i]*g*leng[0][i]/2.*cos(teta[0][i]) +
mass[1][i]*g*(leng[0][i]+cos(teta[0][i])+leng[1][i]/2.*cos(teta[0][i]+teta[1]
[i])) + mass[2][i]*g*(leng[0][i]+cos(teta[0][i])+leng[1][i]*cos(teta[0][i]+
```

```
teta[1][i])+leng[2][i]/2.*cos(teta[0][i]+teta[1][i]+teta[2][i]));

C_vec[1][i] = -1./2.*mass[2][i]*leng[1][i]*leng[2][i]*sin(teta[2][i])*(2.*
teta_dot[0][i]+2.*teta_dot[1][i]+teta_dot[2][i])*teta_dot[2][i] + (1./2.*
mass[1][i]+mass[2][i])*leng[0][i]*leng[1][i]*sin(teta[1][i])*sqr(teta_dot[0]
[i]) + 1./2.*mass[2][i]*leng[0][i]*leng[2][i]*sin(teta[1][i]+teta[2][i])*sqr(
teta_dot[0][i]) + mass[1][i]*g*leng[1][i]/2.*cos(teta[0][i]+teta[1][i]) +
mass[2][i]*g*(leng[1][i]*cos(teta[0][i]+teta[1][i])+leng[2][i]/2.*cos(teta[0]
[i]+teta[1][i]+teta[2][i]));

C_vec[2][i] = 1./2.*mass[2][i]*leng[0][i]*leng[2][i]*sin(teta[1][i]+teta[2]
[i])*sqr(teta_dot[0][i]) + 1./2.*mass[2][i]*leng[1][i]*leng[2][i]*sin(teta
[2][i])*sqr(teta_dot[0][i]+teta_dot[1][i]) + mass[2][i]*g*leng[2][i]/2.*cos(
teta[0][i]+teta[1][i]+teta[2][i]);

    /* ----  Construct_J_mat  ---- */

m = sgn(i);
J_mat[0][0][i] = m*leng[0][i]*sin(teta[0][i]) + m*leng[1][i]*sin(teta[0][i]+
teta[1][i]) + m*leng[2][i]*sin(teta[0][i]+teta[1][i]+teta[2][i]);

J_mat[0][1][i] = m*leng[1][i]*sin(teta[0][i]+teta[1][i]) + m*leng[2][i]*sin(
teta[0][i]+teta[1][i]+teta[2][i]);

J_mat[0][2][i] = m*leng[2][i]*sin(teta[0][i]+teta[1][i]+teta[2][i]);

J_mat[1][0][i] = leng[0][i]*cos(teta[0][i]) + leng[1][i]*cos(teta[0][i]+
teta[1][i]) + leng[2][i]*cos(teta[0][i]+teta[1][i]+teta[2][i]);

J_mat[1][1][i] = leng[1][i]*cos(teta[0][i]+teta[1][i]) + leng[2][i]*cos(
teta[0][i]+teta[1][i]+teta[2][i]);
```

```
J_mat[1][2][i] = leng[2][i]*cos(teta[0][i]+teta[1][i]+teta[2][i]);

}

} /* ConsOrig () */
```

```
/*****************************************/
/*            JD.h                       */
/*****************************************/
extern double
        JD[MAX][MAX][2],   /* first derivative Jacobian to form teta_dot's */
        leng[3][2],                /* link lengths of arms                */
        l_Load,                    /* load length                         */
        pseudo[3][2],   /* pseudo joint angles : pseudo[t-2,t-1,t][1,2]    */
        tip[2][2],      /* tip coordinates : tip[x,y][t-2,t-1,t][1,2]      */
    D;              /* horizontal distance between the bases of robots */


void ConsJD (phi,S)
double phi, S[2];
{
double l14, l24;

l14 = sqrt(pow(leng[1][0],2.)+pow(leng[2][0],2.)+2.*leng[1][0]*leng[2][0]*
cos(pseudo[2][0]));

l24 = sqrt(pow(leng[1][1],2.)+pow(leng[2][1],2.)+2.*leng[1][1]*leng[2][1]*
cos(pseudo[2][1]));

JD[0][0][0] =
((tip[1][0])/pow((tip[0][0]),2.)/(1.+pow((tip[1][0]),2.)/pow((tip[0
][0]),2.))+1./pow((1.-1./4.*pow((pow(leng[0][0],2.)+pow((tip[1][0]
),2.)+pow((tip[0][0]),2.)-pow(l14,2.)),2.)/pow(leng[0][0],2.)/(pow((tip
[1][0]),2.)+pow((tip[0][0]),2.))),(1./2.))*((tip[0][0])/leng[0][0]/pow
((pow((tip[1][0]),2.)+pow((tip[0][0]),2.)),(1./2.))-1./2.*(pow(leng[0
][0],2.)+pow((tip[1][0]),2.)+pow((tip[0][0]),2.)-pow(l14,2.))/leng[0][
```

```
0]/pow((pow((tip[1][0]),2.)+pow((tip[0][0]),2.)),(3./2.))*(tip[0][0])));

JD[0][1][0] =
(-1./(tip[0][0])/(1.+pow((tip[1][0]),2.)/pow((tip[0][0]),2.))+1./
pow((1.-1./4.*pow((pow(leng[0][0],2.)+pow((tip[1][0]),2.)+pow((tip[0
][0]),2.)-pow(l14,2.)),2.)/pow(leng[0][0],2.)/(pow((tip[1][0]),2.)+pow
((tip[0][0]),2.))),(1./2.))*((tip[1][0])/leng[0][0]/pow((pow((tip[1][0
]),2.)+pow((tip[0][0]),2.)),(1./2.))-1./2.*(pow(leng[0][0],2.)+pow((tip
[1][0]),2.)+pow((tip[0][0]),2.)-pow(l14,2.))/leng[0][0]/pow((pow((tip
[1][0]),2.)+pow((tip[0][0]),2.)),(3./2.))*(tip[1][0]));

JD[0][2][0] =
(-(l_Load*cos(phi)/(tip[0][0])-(tip[1][0])/pow((tip[0][0]),2.)*l_Load
*sin(phi))/(1.+pow((tip[1][0]),2.)/pow((tip[0][0]),2.))+1./pow((
1.-1./4.*pow((pow(leng[0][0],2.)+pow((tip[1][0]),2.)+pow((tip[0][0]),2.
)-pow(l14,2.)),2.)/pow(leng[0][0],2.)/(pow((tip[1][0]),2.)+pow((tip[0
][0]),2.))),(1./2.))*(1./2.*(2.*(tip[1][0])*l_Load*cos(phi)+2.*(tip[0]
[0])*l_Load*sin(phi))/leng[0][0]/pow((pow((tip[1][0]),2.)+pow((tip[0][
0]),2.)),(1./2.))-1./4.*(pow(leng[0][0],2.)+pow((tip[1][0]),2.)+pow((
tip[0][0]),2.)-pow(l14,2.))/leng[0][0]/pow((pow((tip[1][0]),2.)+pow((tip
[0][0]),2.)),(3./2.))*(2.*(tip[1][0])*l_Load*cos(phi)+2.*(tip[0][0])*
l_Load*sin(phi))));

JD[0][3][0] =
1./pow((1.-1./4.*pow((pow(leng[0][0],2.)+pow((tip[1][0]),2.)+pow
((tip[0][0]),2.)-pow(l14,2.)),2.)/pow(leng[0][0],2.)/(pow((tip[1][0]),
2.)+pow((tip[0][0]),2.))),(1./2.))*l14/pow((pow(leng[1][0],2.)+pow(leng
[2][0],2.)+2.*leng[1][0]*leng[2][0]*cos((pseudo[2][0]))),(1./2.))*leng
[1][0]*leng[2][0]*sin((pseudo[2][0]))/leng[0][0]/pow((pow((tip[1][0]),
2.)+pow((tip[0][0]),2.)),(1./2.));
```

```
JD[0][4][0] = 0.0;

JD[1][0][0] =
-1./pow((1.-1./4.*pow((pow((tip[1][0]),2.)+pow((tip[0][0]),2.)-
pow(leng[0][0],2.)-pow(114,2.)),2.)/pow(leng[0][0],2.)/pow(114,2.)),(1.
/2.))*(tip[0][0])/leng[0][0]/114;

JD[1][1][0] =
-1./pow((1.-1./4.*pow((pow((tip[1][0]),2.)+pow((tip[0][0]),2.)-
pow(leng[0][0],2.)-pow(114,2.)),2.)/pow(leng[0][0],2.)/pow(114,2.)),(1.
/2.))*(tip[1][0])/leng[0][0]/114;

JD[1][2][0] =
-1./2./pow((1.-1./4.*pow((pow((tip[1][0]),2.)+pow((tip[0][0]),2.
)-pow(leng[0][0],2.)-pow(114,2.)),2.)/pow(leng[0][0],2.)/pow(114,2.))
,(1./2.))*(2.*(tip[1][0])*l_Load*cos(phi)+2.*(tip[0][0])*l_Load*sin(phi
))/leng[0][0]/114;

JD[1][3][0] =
-1./pow((1.-1./4.*pow((pow((tip[1][0]),2.)+pow((tip[0][0]),2.)
-pow(leng[0][0],2.)-pow(114,2.)),2.)/pow(leng[0][0],2.)/pow(114,2.)),(
1./2.))*(1./pow((pow(leng[1][0],2.)+pow(leng[2][0],2.)+2.*leng[1][0]*
leng[2][0]*cos((pseudo[2][0]))),(1./2.))*leng[1][0]*leng[2][0]*sin((
pseudo[2][0]))/leng[0][0]+1./2.*(pow((tip[1][0]),2.)+pow((tip[0][0]),2.)-
pow(leng[0][0],2.)-pow(114,2.))/leng[0][0]/pow(114,2.)/pow((pow(leng[1
][0],2.)+pow(leng[2][0],2.)+2.*leng[1][0]*leng[2][0]*cos((pseudo[2][0]
))),(1./2.))*leng[1][0]*leng[2][0]*sin((pseudo[2][0])));
if (S[0]!=0.0)
  JD[1][3][0] += S[0]/pow((1.-1./4.
*pow((pow(leng[1][0],2.)+pow(114,2.)-pow(leng[2][0],2.)),2.)/pow(leng
[1][0],2.)/pow(114,2.)),(1./2.))*(-1./pow((pow(leng[1][0],2.)+pow(leng
```

```
[2][0],2.)+2.*leng[1][0]*leng[2][0]*cos((pseudo[2][0]))),(1./2.))*leng
[2][0]*sin((pseudo[2][0]))+1./2.*(pow(leng[1][0],2.)+pow(114,2.)-pow(
leng[2][0],2.))/pow(114,2.)/pow((pow(leng[1][0],2.)+pow(leng[2][0],2.)+
2.*leng[1][0]*leng[2][0]*cos((pseudo[2][0]))),(1./2.))*leng[2][0]*sin(
(pseudo[2][0])));

JD[1][4][0] = 0.0;

JD[2][0][0] = 0.0;

JD[2][1][0] = 0.0;

JD[2][2][0] = 0.0;

JD[2][3][0] = 1.0;

JD[2][4][0] = 0.0;


JD[0][0][1] =
(-(tip[1][1])/pow((D-(tip[0][1])),2.)/(1.+pow((tip[1][1]),2.)/
pow((D-(tip[0][1])),2.))+1./pow((1.-1./4.*pow((pow(leng[0][1],2.)+pow((
tip[1][1]),2.)+pow(((tip[0][1])-D),2.)-pow(124,2.)),2.)/pow(leng[0][1]
,2.)/(pow((tip[1][1]),2.)+pow(((tip[0][1])-D),2.))),(1./2.))*(((tip[0]
[1])-D)/leng[0][1]/pow((pow((tip[1][1]),2.)+pow(((tip[0][1])-D),2.)),(
1./2.))-1./2.*(pow(leng[0][1],2.)+pow((tip[1][1]),2.)+pow(((tip[0][1])
-D),2.)-pow(124,2.))/leng[0][1]/pow((pow((tip[1][1]),2.)+pow(((tip[0][
1])-D),2.)),(3./2.))*((tip[0][1])-D)));

JD[0][1][1] =
(-1./(D-(tip[0][1]))/(1.+pow((tip[1][1]),2.)/pow((D-(tip[0][1])
```

```
),2.))+1./pow((1.-1./4.*pow((pow(leng[0][1],2.)+pow((tip[1][1]),2.)+pow
(((tip[0][1])-D),2.)-pow(124,2.)),2.)/pow(leng[0][1],2.)/(pow((tip[1]
[1]),2.)+pow(((tip[0][1])-D),2.))),(1./2.))*((tip[1][1])/leng[0][1]/pow
((pow((tip[1][1]),2.)+pow(((tip[0][1])-D),2.)),(1./2.))-1./2.*(pow(leng
[0][1],2.)+pow((tip[1][1]),2.)+pow(((tip[0][1])-D),2.)-pow(124,2.))/
leng[0][1]/pow((pow((tip[1][1]),2.)+pow(((tip[0][1])-D),2.)),(3./2.))*
(tip[1][1])));

JD[0][2][1] =
(-(-l_Load*cos(phi)/(D-(tip[0][1]))-(tip[1][1])/pow((D-(tip[0][
1])),2.)*l_Load*sin(phi))/(1.+pow((tip[1][1]),2.)/pow((D-(tip[0][1])),
2.))+1./pow((1.-1./4.*pow((pow(leng[0][1],2.)+pow((tip[1][1]),2.)+pow(
((tip[0][1])-D),2.)-pow(124,2.)),2.)/pow(leng[0][1],2.)/(pow((tip[1][1
]),2.)+pow(((tip[0][1])-D),2.))),(1./2.))*(1./2.*(-2.*(tip[1][1])*l_Load
*cos(phi)-2.*((tip[0][1])-D)*l_Load*sin(phi))/leng[0][1]/pow((pow((tip
[1][1]),2.)+pow(((tip[0][1])-D),2.)),(1./2.))-1./4.*(pow(leng[0][1],
2.)+pow((tip[1][1]),2.)+pow(((tip[0][1])-D),2.)-pow(124,2.))/leng[0][1
]/pow((pow((tip[1][1]),2.)+pow(((tip[0][1])-D),2.)),(3./2.))*(-2.*(tip
[1][1])*l_Load*cos(phi)-2.*((tip[0][1])-D)*l_Load*sin(phi))));

JD[0][3][1] = 0.0;

JD[0][4][1] =
1./pow((1.-1./4.*pow((pow(leng[0][1],2.)+pow((tip[1][1]),2.)+pow
(((tip[0][1])-D),2.)-pow(124,2.)),2.)/pow(leng[0][1],2.)/(pow((tip[1][
1]),2.)+pow(((tip[0][1])-D),2.))),(1./2.))*124/pow((pow(leng[1][1],2.)
+pow(leng[2][1],2.)+2.*leng[1][1]*leng[2][1]*cos((pseudo[2][1]))),(1./
2.))*leng[1][1]*leng[2][1]*sin((pseudo[2][1]))/leng[0][1]/pow((pow((tip
[1][1]),2.)+pow(((tip[0][1])-D),2.)),(1./2.));

JD[1][0][1] =
```

```
 -1./pow((1.-1./4.*pow((pow((tip[1][1]),2.)+pow(((tip[0][1])-D),
 2.)-pow(leng[0][1],2.)-pow(124,2.)),2.)/pow(leng[0][1],2.)/pow(124,2.)
 ),(1./2.))*((tip[0][1])-D)/leng[0][1]/124;

 JD[1][1][1] =
 -1./pow((1.-1./4.*pow((pow((tip[1][1]),2.)+pow(((tip[0][1])-D),
 2.)-pow(leng[0][1],2.)-pow(124,2.)),2.)/pow(leng[0][1],2.)/pow(124,2.)
 ),(1./2.))*(tip[1][1])/leng[0][1]/124;

 JD[1][2][1] =
 -1./2./pow((1.-1./4.*pow((pow((tip[1][1]),2.)+pow(((tip[0][1])-
 D),2.)-pow(leng[0][1],2.)-pow(124,2.)),2.)/pow(leng[0][1],2.)/pow(124,
 2.)),(1./2.))*(-2.*(tip[1][1])*l_Load*cos(phi)-2.*((tip[0][1])-D)*l_Load
 *sin(phi))/leng[0][1]/124;

 JD[1][3][1] = 0.0;

 JD[1][4][1] =
 -1./pow((1.-1./4.*pow((pow((tip[1][1]),2.)+pow(((tip[0][1])-D)
 ,2.)-pow(leng[0][1],2.)-pow(124,2.)),2.)/pow(leng[0][1],2.)/pow(124,2.
 )),(1./2.))*(1./pow((pow(leng[1][1],2.)+pow(leng[2][1],2.)+2.*leng[1][
 1]*leng[2][1]*cos((pseudo[2][1]))),(1./2.))*leng[1][1]*leng[2][1]*sin(
 (pseudo[2][1]))/leng[0][1]+1./2.*(pow((tip[1][1]),2.)+pow(((tip[0][1])
 -D),2.)-pow(leng[0][1],2.)-pow(124,2.))/leng[0][1]/pow(124,2.)/pow((pow
 (leng[1][1],2.)+pow(leng[2][1],2.)+2.*leng[1][1]*leng[2][1]*cos((pseudo
 [2][1]))),(1./2.))*leng[1][1]*leng[2][1]*sin((pseudo[2][1])));
 if (S[1]!=0.0)
   JD[1][4][1] += S[1]/pow(
 (1.-1./4.*pow((pow(leng[1][1],2.)+pow(124,2.)-pow(leng[2][1],2.)),2.)/
 pow(leng[1][1],2.)/pow(124,2.)),(1./2.))*(-1./pow((pow(leng[1][1],2.)+
 pow(leng[2][1],2.)+2.*leng[1][1]*leng[2][1]*cos((pseudo[2][1]))),(1./2.
```

```
))*leng[2][1]*sin((pseudo[2][1]))+1./2.*(pow(leng[1][1],2.)+pow(124,2.
)-pow(leng[2][1],2.))/pow(124,2.)/pow((pow(leng[1][1],2.)+pow(leng[2]
[1],2.) 2.*leng[1][1]*leng[2][1]*cos((pseudo[2][1]))),(1./2.))*leng[2]
[1]*sin((pseudo[2][1])));

JD[2][0][1] = 0.0;

JD[2][1][1] = 0.0;

JD[2][2][1] = 0.0;

JD[2][3][1] = 0.0;

JD[2][4][1] = 1.0;

}
```

```
/************************************/
/*          JVDD.h                  */
/************************************/
extern double
        JVDD[MAX][MAX][2],/* vector from double derivative Jacobian        */
        leng[3][2],               /* link lengths of arms                   */
        l_Load,                   /* load length and connector length       */
        pseudo[3][2],   /* pseudo joint angles : pseudo[t-2,t-1,t][1,2]     */
        tip[2][2],      /* tip coordinates : tip[x,y][t-2,t-1,t][1,2]       */
        D,              /* horizontal distance between the bases of robots */
        x_co_dot, y_co_dot, phi_dot,            /* first derivatives       */
    pseudo_dot[2];


void ConsJVDD (phi,S)
double phi, S[2];
{
double l14, l24;
double t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11;

l14 = sqrt(pow(leng[1][0],2.)+pow(leng[2][0],2.)+2.*leng[1][0]*leng[2][0]*
cos(pseudo[2][0]));

l24 = sqrt(pow(leng[1][1],2.)+pow(leng[2][1],2.)+2.*leng[1][1]*leng[2][1]*
cos(pseudo[2][1]));

t1 = x_co_dot + l_Load*sin(phi)*phi_dot;
t2 = y_co_dot + l_Load*cos(phi)*phi_dot;
t3 = sqr(tip[0][0]);
t4 = sqr(tip[1][0]);
```

```
t5 = sqr(leng[0][0])+t4+t3-sqr(l14);
t6 = tip[0][0]*t1;
t7 = t ip[1][0]*t2;
t8 = pseudo[2][0];
t9 = sqr(leng[1][0])+sqr(leng[2][0])+2.*leng[1][0]*leng[2][0]*cos(t8);
t10 = 2.*t7+2.*t6+2.*l14*leng[1][0]*leng[2][0]*sin(t8)*(pseudo_dot[0])
/sqrt(t9);

JVDD[0][0][0] = -(-l_Load*sin(phi)*sqr(phi_dot)/tip[0][0]-2.*t2*t1/sqr(
tip[0][0])+2.*tip[1][0]*sqr(t1)/pow(tip[0][0],3.)-tip[1][0]*l_Load*cos(
phi)*sqr(phi_dot)/sqr(tip[0][0]))/(1.+sqr(tip[1][0])/sqr(tip[0][0]))+(
t2/tip[0][0]-tip[1][0]*t1/sqr(tip[0][0]))*(2.*tip[1][0]*t2/sqr(tip[0][0])
-2.*sqr(tip[1][0])*t1/pow(tip[0][0],3.))/pow(1.+sqr(tip[1][0])/sqr(tip[0
][0]),2.)-1./2.*(1./2.*t10/leng[0][0]/sqrt(t4+t3)-1./2.*t5*(t7+t6)/leng
[0][0]/pow(t4+t3,3./2.))*(-1./2.*t5*t10/sqr(leng[0][0])/(t4+t3)+1./2.*
sqr(t5)*(t7+t6)/sqr(leng[0][0])/sqr(t4+t3))/pow(1.-1./4.*sqr(t5)/sqr(leng
[0][0])/(t4+t3),3./2.)+(1./2.*(2.*sqr(t2)-2.*tip[1][0]*l_Load*sin(phi)*
sqr(phi_dot)+2.*sqr(t1)+2.*tip[0][0]*l_Load*cos(phi)*sqr(phi_dot)-2.*sqr
(leng[1][0])*sqr(leng[2][0])*sqr(sin(t8))*sqr(pseudo_dot[0])/t9-2.*l14
*(-sqr(leng[1][0])*sqr(leng[2][0])*sqr(sin(t8))*sqr(pseudo_dot[0])/pow
(t9,3./2.)-leng[1][0]*leng[2][0]*cos(t8)*sqr(pseudo_dot[0])/sqrt(t9)))
/(leng[0][0]*sqrt(t4+t3))-t10*(t7+t6)/leng[0][0]/pow(t4+t3,3./2.)+3./2.*
t5*sqr(t7+t6)/leng[0][0]/pow(t4+t3,5./2.)-1./2.*t5*(sqr(t2)-tip[1][0]*
l_Load*sin(phi)*sqr(phi_dot)+sqr(t1)+tip[0][0]*l_Load*cos(phi)*sqr(phi_dot
))/(leng[0][0]*pow(t4+t3,3./2.)))/sqrt(1.-1./4.*sqr(t5)/sqr(leng[0][0])/
(t4+t3));

t1 = pseudo[2][0];
t2 = sqr(leng[2][0]);
t3 = sqr(leng[1][0]);
t4 = t3+t2+2.*leng[1][0]*leng[2][0]*cos(t1);
```

```
 t5 = -sqr(leng[1][0])*sqr(leng[2][0])*sqr(sin(t1))*sqr(pseudo_dot[0])/pow(
 t4,3./2.)-leng[1][0]*leng[2][0]*cos(t1)*sqr(pseudo_dot[0])/sqrt(t4);
 t6 = t  sqr(l14)-t2;
 t7 = sqr(tip[1][0])+sqr(tip[0][0])-sqr(leng[0][0])-sqr(l14);
 t8 = x_co_dot+l_Load*sin(phi)*phi_dot;
 t9 = y_co_dot+l_Load*cos(phi)*phi_dot;
 t10 = 2.*tip[1][0]*t9+2.*tip[0][0]*t8+2.*l14*leng[1][0]*leng[2][0]*sin(t1)*
 pseudo_dot[0]/sqrt(t4);

 JVDD[1][0][0] = 1./2.*(1./2.*t10/leng[0][0]/l14+1./2.*t7*leng[1][0]*leng[2]
 [0]*sin(t1)*pseudo_dot[0]/leng[0][0]/sqr(l14)/sqrt(t4))*(-1./2.*t7*t10/sqr(
 leng[0][0])/sqr(l14)-1./2.*sqr(t7)*leng[1][0]*leng[2][0]*sin(t1)*pseudo_dot
 [0]/sqr(leng[0][0])/pow(l14,3.)/sqrt(t4))/pow(1.-1./4.*sqr(t7)/sqr(leng[0]
 [0])/sqr(l14),3./2.)-(1./2.*(2.*sqr(t9)-2.*tip[1][0]*l_Load*sin(phi)*sqr(
phi_dot)+2.*sqr(t8)+2.*tip[0][0]*l_Load*cos(phi)*sqr(phi_dot)-2.*sqr(leng[1]
 [0])*sqr(leng[2][0])*sqr(sin(t1))*sqr(pseudo_dot[0])/t4-2.*l14*t5)/(leng[0]
 [0]*l14)+t10*leng[1][0]*leng[2][0]*sin(t1)*pseudo_dot[0]/leng[0][0]/sqr(
 l14)/sqrt(t4)+t7*sqr(leng[1][0])*sqr(leng[2][0])*sqr(sin(t1))*sqr(pseudo_dot
 [0])/leng[0][0]/pow(l14,3.)/t4-1./2.*t7*t5/leng[0][0]/sqr(l14))/sqrt(1-1./
4.*sqr(t7)/sqr(leng[0][0])*sqr(l14));
if (S[0]!=0.0)
  JVDD[1][0][0] -= 1./2.*S[0]*(-leng[2][0]*sin(t1)*pseudo_dot[0]/sqrt(t4)+
1./2.*t6*leng[2][0]*sin(t1)*pseudo_dot[0]/sqr(l14)/sqrt(t4))*(t6*leng[2]
[0]*sin(t1)*pseudo_dot[0]/leng[1][0]/l14/sqrt(t4)-1./2.*sqr(t6)*leng[2][0]*
sin(t1)*pseudo_dot[0]/leng[1][0]/pow(l14,3.)/sqrt(t4))/pow(1.-1./4.*sqr(
t6)/sqr(leng[1][0])/sqr(l14),3./2.);
if (S[0]!=0.0)
  JVDD[1][0][0] += S[0]*(t5/leng[1][0]-leng[1][0]*sqr(leng[2][0])*sqr(sin(
t1))*sqr(pseudo_dot[0])/l14/t4+t6*leng[1][0]*sqr(leng[2][0])*sqr(sin(t1))*
sqr(pseudo_dot[0])/pow(l14,3.)/t4-1./2.*t6*t5/leng[1][0]/sqr(l14))/sqrt(1.-
1./4.*sqr(t6)/sqr(leng[1][0])/sqr(l14));
```

```
JVDD[2][0][0] = 0.0;


t1 = tip[0][1]-D;
t2 = x_co_dot - l_Load*sin(phi)*phi_dot;
t3 = y_co_dot - l_Load*cos(phi)*phi_dot;
t4 = sqr(tip[1][1]);
t5 = sqr(leng[0][1])+t4+sqr(t1)-sqr(l24);
t6 = tip[1][1]*t3;
t7 = 2.*t6+2.*t1*t2;
t8 = pseudo[2][1];
t9 = sqr(leng[1][1])+sqr(leng[2][1])+2.*leng[1][1]*leng[2][1]*cos(t8);
t10 = 2.*t6+2.*t1*t2+2.*l24*leng[1][1]*leng[2][1]*sin(t8)*(pseudo_dot[1])
/sqrt(t9);
t11 = D-tip[0][1];

JVDD[0][0][1] = -(l_Load*sin(phi)*sqr(phi_dot)/t11+2.*t3*t2/sqr(t11)
+2.*tip[1][1]*sqr(t2)/pow(t11,3.)-tip[1][1]*l_Load*cos(phi)*sqr(phi_dot)/
sqr(t11))/(1.+sqr(tip[1][1])/sqr(t11))+(t3/t11+tip[1][1]*t2/sqr(t11))*(2.
*tip[1][1]*t3/sqr(t11)+2.*sqr(tip[1][1])*t2/pow(t11,3.))/pow(1.+sqr(tip[1]
[1])/sqr(t11),2.)-1./2.*(1./2.*t10/leng[0][1]/sqrt(t4+sqr(t1))-1./4.*t5*
t7/leng[0][1]/pow(t4+sqr(t1),3./2.))*(-1./2.*t5*t10/sqr(leng[0][1])/(t4+
sqr(t1))+1./4.*sqr(t5)*t7/sqr(leng[0][1])/sqr(t4+sqr(t1)))/pow(1.-1./4.*
sqr(t5)/sqr(leng[0][1])/(t4+sqr(t1)),3./2.)+(1./2.*(2.*sqr(t3)-2.*tip[1]
[1]*l_Load*sin(phi)*sqr(phi_dot)+2.*sqr(t2)-2.*t1*l_Load*cos(phi)*sqr
(phi_dot)-2.*sqr(leng[1][1])*sqr(leng[2][1])*sqr(sin(t8))*sqr(pseudo_dot
[1])/t9-2.*l24*(-sqr(leng[1][1])*sqr(leng[2][1])*sqr(sin(t8))*sqr(pseudo_dot
[1])/pow(t9,3./2.)-leng[1][1]*leng[2][1]*cos(t8)*sqr(pseudo_dot[1])/sqrt(t9)
))/(leng[0][1]*sqrt(t4+sqr(t1)))-1./2.*t10*t7/leng[0][1]/pow(t4+sqr(t1),3./
2.)+3./8.*t5*sqr(t7)/leng[0][1]/pow(t4+sqr(t1),5./2.)-1./4.*t5*(2.*sqr(t3)+
```

```
  2.*tip[1][1]*l_Load*sin(phi)*sqr(phi_dot)+2.*sqr(t2)-2.*t1*l_Load*cos(phi)*
  sqr(phi_dot))/(leng[0][1]*pow(t4+sqr(t1),3./2.)))/sqrt(1.-1./4.*sqr(t5)/
  sqr(leng[0][1])/(t4+sqr(t1)));

  t1 = pseudo[2][1];
  t2 = sqr(leng[2][1]);
  t3 = sqr(leng[1][1]);
  t4 = t3+t2+2.*leng[1][1]*leng[2][1]*cos(t1);
  t5 = -sqr(leng[1][1])*sqr(leng[2][1])*sqr(sin(t1))*sqr(pseudo_dot[1])/pow(
  t4,3./2.)-leng[1][1]*leng[2][1]*cos(t1)*sqr(pseudo_dot[1])/sqrt(t4);
  t6 = t3+sqr(l24)-t2;
  t7 = tip[0][1]-D;
  t8 = sqr(tip[1][1])+sqr(t7)-sqr(leng[0][1])-sqr(l24);
  t9 = x_co_dot-l_Load*sin(phi)*phi_dot;
  t10 = y_co_dot-l_Load*cos(phi)*phi_dot;
  t11 = 2.*tip[1][1]*t10+2.*t7*t9+2.*l24*leng[1][1]*leng[2][1]*sin(t1)*
  pseudo_dot[1]/sqrt(t4);

  JVDD[1][0][1] = 1./2.*(1./2.*t11/leng[0][1]/l24+1./2.*t8*leng[1][1]*leng[2]
  [1]*sin(t1)*pseudo_dot[1]/leng[0][1]/sqr(l24)/sqrt(t4))*(-1./2.*t8*t11/sqr(
  leng[0][1])/sqr(l24)-1./2.*sqr(t8)*leng[1][1]*leng[2][1]*sin(t1)*pseudo_dot
  [1]/sqr(leng[0][1])/pow(l24,3.)/sqrt(t4))/pow(1.-1./4.*sqr(t8)/sqr(leng[0]
  [1])/sqr(l24),3./2.)-(1./2.*(2.*sqr(t10)-2.*tip[1][1]*l_Load*sin(phi)*sqr(
  phi_dot)+2.*sqr(t9)-2.*t7*l_Load*cos(phi)*sqr(phi_dot)-2.*sqr(leng[1]
  [1])*sqr(leng[2][1])*sqr(sin(t1))*sqr(pseudo_dot[1])/t4-2.*l24*t5)/(leng[0]
  [1]*l24)+t11*leng[1][1]*leng[2][1]*sin(t1)*pseudo_dot[1]/leng[0][1]/sqr(
  l24)/sqrt(t4)+t8*sqr(leng[1][1])*sqr(leng[2][1])*sqr(sin(t1))*sqr(pseudo_dot
  [1])/leng[0][1]/pow(l24,3.)/t4-1./2.*t8*t5/leng[0][1]/sqr(l24))/sqrt(1-1./
  4.*sqr(t8)/sqr(leng[0][1])*sqr(l24));
  if (S[1]!=0.0)
    JVDD[1][0][0] -= 1./2.*S[1]*(-leng[2][1]*sin(t1)*pseudo_dot[1]/sqrt(t4)+
```

```
1./2.*t6*leng[2][1]*sin(t1)*pseudo_dot[1]/sqr(124)/sqrt(t4))*(t6*leng[2]
[1]*sin(t1)*pseudo_dot[1]/leng[1][1]/124/sqrt(t4)-1./2.*sqr(t6)*leng[2][1]*
sin(t1 *pseudo_dot[1]/leng[1][1]/pow(124,3.)/sqrt(t4))/pow(1.-1./4.*sqr(
t6)/sqr(leng[1][1])/sqr(124),3./2.);
if (S[1]!=0.0)
  JVDD[1][0][0] += S[1]*(t5/leng[1][1]-leng[1][1]*sqr(leng[2][1])*sqr(sin(
t1))*sqr(pseudo_dot[1])/124/t4+t6*leng[1][1]*sqr(leng[2][1])*sqr(sin(t1))*
sqr(pseudo_dot[1])/pow(124,3.)/t4-1./2.*t6*t5/leng[1][1]/sqr(124))/sqrt(1.-
1./4.*sqr(t6)/sqr(leng[1][1])/sqr(124));

JVDD[2][0][1] = 0.0;

}
```