PARALLEL PROCESSING OF MANIPULATOR DYNAMICS INCORPORATING FRICTIONAL EFFECTS

CHARLES DHANARAJ, B. Tech

PARALLEL PROCESSING OF MANIPULATOR DYNAMICS INCORPORATING FRICTIONAL EFFECTS

By

©CHARLES DHANARAJ, B. TECH.

A thesis submitted to the School of Graduate Studies
in partial fulfillment of the
requirements for the degree of
Master of Engineering

Faculty of Engineering and Applied Sciences
Memorial University of Newfoundland
August 1990

St. John's Newfoundland Canada

ISBN 0-315-61784-5

Abstract

Real-time computation of the inverse dynamics of robotic manipulators is required for ensuring robust control. This thesis presents a modified Newton-Euler algorithm which makes use of symbolic programming for improved computational efficiency. A scheme for modeling the frictional effects at the joints as well as the transmissions for robotic mechanisms is outlined with an illustrative case-study for the PUMA-560 manipulator. The algorithm is parallelized using a 'Task Streamlining Approach' - a systematic mapping scheme using layered task graphs to create the list schedule and a simplified bin-packing heuristic algorithm to schedule the computations on a multiprocessor. The resulting computational load for dynamic torques without friction, is only 12n+9 arithmetic operations, where n is the number of links in the manipulator, indicating a promise for application to precision robot control employing a high sampling rate.

O LORD, OUR LORD!
HOW MAJESTIC IS YOUR NAME IN ALL THE EARTH!
WHEN I CONSIDER YOUR HEAVENS,
THE WORK OF YOUR FINGERS,
THE MOON AND THE STARS,
WHICH YOU HAVE SET IN PLACE,
WHAT IS MAN THAT YOU ARE MINDFUL OF HIM,
THE SON OF MAN THAT YOU CARE FOR HIM?
O LORD, OUR LORD,
HOW MAJESTIC IS YOUR NAME IN ALL THE EARTH!

[King David's Psalms in The Bible]

ACKNOWLEDGEMENTS

It is with sincere appreciation and gratitude that I would like to thank Dr. Anand M. Sharan, for his direction and support during this research work. His encouragement to try new ideas and develop them for practical applications has been the major inspiration for this work.

I would also like to extend my thanks to the School of Graduate Studies and the Faculty of Engineering for supporting me financially through this period. I sincerely thank Dr. J. Malpas, Dean of Graduate Studies and Dr. T. R. Chari, Associate Dean of Engineering for their valuable advice during my graduate work. The 'thesis guide' prepared by the Graduate Studies Committee has been of a practical help in preparing this thesis. A special thanks to Ms. Janet Fairley who was always willing to help, with a smile, despite her busy schedule.

I enjoyed working in the Faculty of Engineering at MUN. The warmth and the friendship of my fellow grad-students and the enthusiasm of the faculty has made the arduous task of this research, quite pleasant and enjoyable. I would like to thank Professors M.J. Hinchey, W.J. Vetter, R.Venkatesan and A.S.J. Swamidas, who have helped me to refine my ideas during the course of this work. I would also like to thank the staff at Center for Computer Aided Engineering for the support they extended to me at various stages of this work. Also, I would like to thank C-CORE for lending me the PARALLON board.

I would like to thank my wife Jayanthi, who was the prime inspiration for me to get into graduate studies and a constant source of encouragement to me.

August 1990 *Charles Dhanaraj*

Contents

List of Figures

List of Tables

List of Symbols

$\{a\}_i$	acceleration of the origin of the ith link co-ordinate frame
$\{a\}_{ci}$	acceleration of the center of gravity of the ith link co-ordinate frame
d	distance between the suppcrt bearings
D_x, D_y, D_z	inertia constants
E_p	efficiency
$\{f\}_i$	reaction force at the ith joint
$\{f\}_i^j$	reaction force at the ith joint referred in jth co-ordinate frame
f	frictional force
F	frictional force
$\{g\}$	gravity vector
$[I]_i$	centroidal inertia tensor of the ith link
i	subscript to denote the link number
k_x, k_y, k_z	kinematic parameters
m	reduction ratio of the hamonic drive
n	the total number of links in the manipulator
$\{n\}_i$	reaction moment at the ith joint
$\{N\}_i$	inertial moment of the ith link
$\{p\}_i$	position vector of the $i+1$th origin
r	radius of the journal
$\{r\}$	position vector of a point
$q_i, \dot{q}_i, \ddot{q}_i$	position,velocity and acceleration of the link movement(angular for revolute joints and

	linear for prismatic joints)
$[R]_i$	Transformation matrix from ith to $i-1$th frame
$\{s\}$	position vector of the center of gravity of the ith link
S_p	speed-up
T_i	ith subtask
x, y, z	subscripts to denote the x, y an z components of a vector
$\{z\}$	unit vector along the local $\vec{z}$ direction (axis of motion)
$\{\omega\}_i$	absolute angular velocity of the ith link
$\{\alpha\}_i$	absolute angular acceleration of the ith link
$[\lambda]_i$	acceleration difference matrix
τ_i	basic dynamic torque of the ith link
$(\tau_f)^J$	frictional torque at the joint
$(\tau_f)^T$	frictional torque in the transmission
τ_{in}	input torque of the harmonic drive
τ_{out}	output torque of the harmonic drive
τ_B	breakaway torque of the harmonic drive
μ	coefficient of kinetic friction
$\|\|$	absolute value
{ }	vector or column matrix
[ ]	matrix

Chapter 1

Introduction and Literature Survey

1.1 Introduction

Modern robotics offers humanity a wide array of economically and socially laudable benefits. Industrial robots are already assuming many hazardous, unpleasant or boring tasks, while simultaneously improving the productivity of factories in the industrialized world. Autonomous robots can potentially handle tasks in hostile or inaccessible environments, such as, underwater, in space, or in nuclear power reactors. In order for robots to satisfactorily fulfill the many potential missions and applications, it is necessary to incorporate many of the recent advances in robot control into *real-time* operation in the robot system.

Robot motion control can be visualized as a hierarchical scheme, where higher levels feed successively lower levels (Fig. 1.1). The 'Task Planning' is at the highest level and provides the lower control levels with the desired robot mo-

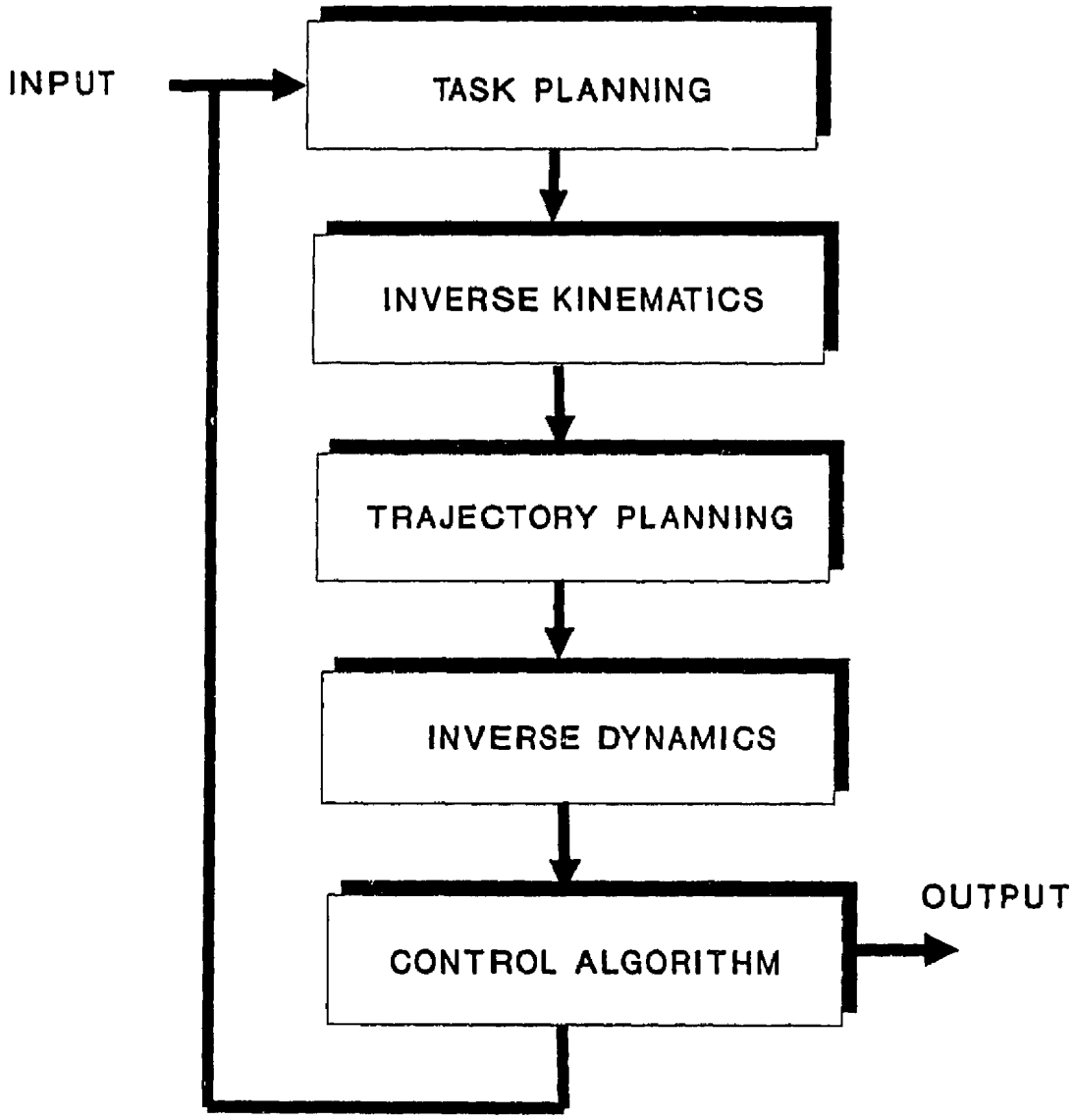


Figure 1.1: Hierarchical Robot Control Scheme

tion, taking cognizance of geometric constraints of the workspace that they have to operate in and the obstacles that may be present within it. The 'Inverse Kinematics' is at the subsequent level of hierarchy, which translates the motion of the end-effector to an equivalent motion of the individual joints. The robot kinematics and dynamics is explained in detail in Chapter 2. The 'Trajectory Planner' goes hand-in-hand with the inverse kinematics module, designing optimal time trajectories for individual motors to achieve the desired motion of the end-effector, ensuring that the resultant motion is satisfying the geometric and the real-world constraints such as the saturation torque of the motor, etc. The 'Inverse Dynamics' computes the torque for the desired motion, using an exact or simplified model of the robot. The 'Control Algorithm' is the final stage of the controller, which compensates the input signal with feed-forward and feed-back correction to ensure robustness.

In a digital control system, which makes use of either minicomputers or microprocessors, time is normalized to the sampling period, Δt; i.e., velocity is expressed as radians per Δt rather than radians per second. To minimize any deterioration of the controller due to sampling, the rate of sampling must be much greater than the natural frequency of the arm (inversely, the sampling period must be much less than the smallest time constant of the arm). Thus to minimize the effect of sampling, usually 20 times the cutoff frequency is chosen (Fu et. al, 1987), i.e.,

$$\Delta t = \frac{1}{20\ \omega_n/2\pi}$$

For industrial manipulators, the natural frequency is in the range of 5 to 10 Hz but could be as much as twice that for smaller arms. This places a limit of 1 to

2 kHz on the bandwidth that can be obtained without considering the effects of flexibility. This brings in the problem of computing the inverse dynamics model within this small sampling period.

Limiting the domain of the research to industrial manipulators, wherein flexibility effects do not play any significant role due to the high rigidity of the manipulators, a rigid body dynamic model incorporating frictional effects would be ideal; however the rigid body dynamic model itself is complicated, when the number of links exceeds three. To give an insight into this problem, the rigid body model of the inverse dynamics of the 6 DOF PUMA-560 manipulator shown in Fig.1.2, using the conventional Lagrangian formulation, requires 66,271 multiplications and 51,548 additions. It would take as high as 800 milliseconds using an 8086 processor. Obviously, this time can be brought down if a higher power processor is used. For example, the PUMA-560 manipulator, which is built around a mini-computer, VAX-700, achieves a sampling period of only 35 ms. This has prompted a search for computationally efficient inverse dynamic models.

1.2 Literature Survey

Many approaches have been taken by researchers to solve the inverse dynamic problem in real time. The Cerebellar Model Articulation Controller (CMAC), developed by Albus (1975; 1981), tries to solve the problem using a look-up table, wherein a wide range of torque values are pre-computed and stored. Such a model needs an extensive memory and may be extremely costly, offsetting other advantages. Yang and Tzeng, (1986) suggested that the design of the robot arms be modified such that it gives a linear model involving a set of

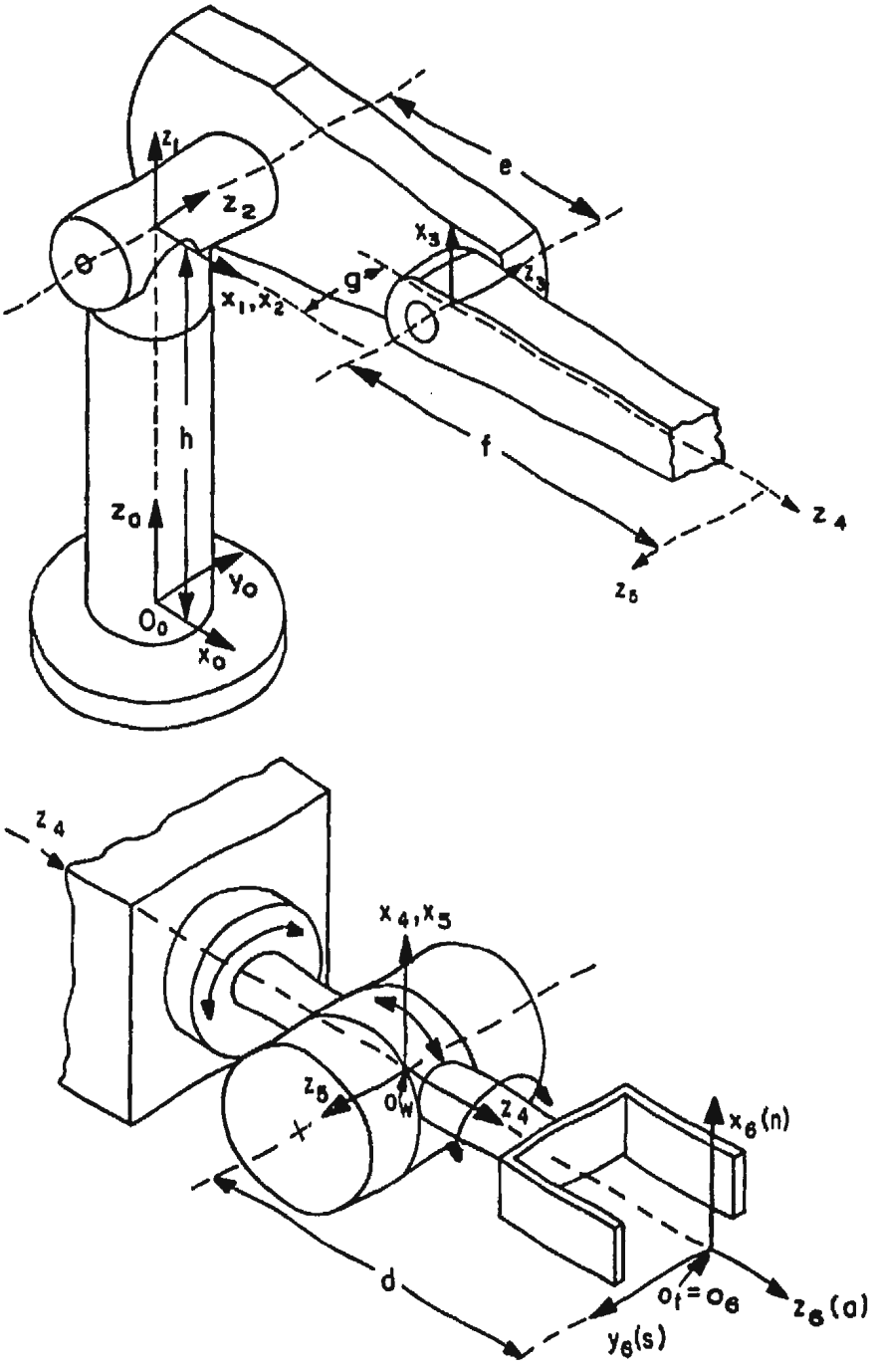


Figure 1.2: PUMA-560 Manipulator

constraint equations for the mass and inertial parameters. This has been only a subject of theoretical interest but the practical feasibility of such a design and manufacture to distribute the inertia as demanded by the constraint equations, has not been explored. Major research has been on a three-tier approach to arrive at efficient equations of motion. A set of equations is termed 'efficient' in the sense that the equations are computationally less demanding. This three-tier approach comprises of

1. Efficient formulation of dynamic equations.

2. Use of symbolic computations to avoid multiplications by zero and one in real-time and for simplifying the algebraic equations.

3. Parallel processing of the equations in real-time.

1.2.1 Dynamic Formulations

The literature abounds with formulations for generating complete dynamic robot models. The standard formulation for manipulator dynamics is the Lagrangian formulation, developed by Uicker (1965) for general linkages and later particularized to open loop kinematic chains by Kahn (1969). These Lagrange equations are given in Appendix A. The $O(n^4)$ computational complexity[1] of this formulation rendered it inefficient for real time applications. Stepanenko and Vukobratovic (1976) suggested the Newton-Euler formulation, which is based on the laws governing the dynamics of rigid bodies. In their formulation, they referred all the

[1] $O(n^4)$ complexity means that the number of arithmetic operations required in the algorithm is in the Order of n^4, which indicates that it is proportional to the fourth power of the size of the variable, which in our case, will be the number of links in the manipulator.

link forces and moments as well as the velocities and accelerations to the global coordinate system. Orin et. al (1979) proposed that the forces and moments be referred to the link coordinate system, which brought down the computational requirements. Luh et al. (1980) extended this idea by calculating the velocities and accelerations also in the link co-ordinate system which resulted in an efficient algorithm with a computational complexity of O(n). While the Lagrangian formulation for a typical 6 DOF robot resulted in 66,271 multiplications and 51,548 additions, the Newton-Euler algorithm resulted only in 852 multiplications and 738 additions. This efficiency was mainly attributed to the recursive nature of the Newton-Euler algorithm. This was extended to the conventional Lagrangian formulation by Hollerbach, who proposed two recursive Lagrangian formulations, one using 4x4 transformation matrices and the other using 3x3 rotation transformation matrices, both with computational complexity of O(n). These equations

Method	Type of operation	Computations for n links	Computations for n = 6	Total flops for n = 6
Uicker/Kahn	Mult	$32\frac{1}{2}n^4 + 86\frac{5}{12}n^3 + 171\frac{1}{4}n^2 + 53\frac{1}{3}n - 128$	66,271	
	Addn	$25n^4 + 66\frac{1}{3}n^3 + 129\frac{1}{2}n^2 + 42\frac{1}{3}n - 96$	51,548	117,819
Hollerbach (4x4)	Mult	$830n - 592$	4,388	
	Addn	$675n - 464$	3,586	7,974
Hollerbach (3x3)	Mult	$412n - 277$	2,195	
	Addn	$320n - 201$	1,719	3,914
Newton-Euler	Mult	$150n - 48$	852	
	Addn	$131n - 48$	738	1,590

Table 1.1: Complexity of Dynamic formulations

are also listed in Appendix A. The computational complexity of these formulations is shown in Table 1.1, and as we can clearly see in this table, the NE formulation is much more efficient than the others. The source of this efficiency was brought out by Silver (1982), who showed that with a proper choice for representing the rotational dynamic parameters, the Lagrangian formulation is indeed equivalent to the NE formulation. Other formulations were developed by Kane (1983, 1985) and Balafoutis (1988). Despite the uniqueness in these formulation, they do not offer any significant advantage over the NE scheme. The major draw-back of the NE formulation is that the recursive nature of the equations does not facilitate control analysis unlike the Lagrangian formulation which results in a closed form solution. Also the transformation matrices and the position vectors, for most practical cases are sparse, and the computations involving multiplication with zeros and ones or addition with zeros are unnecessary. These two issues can be effectively addressed by symbolic computations.

1.2.2 Symbolic Computations

Symbolic programming was introduced into robot dynamics for generating the closed form dynamic equations using Lagrange formulation (Vecchio et al. 1980). Luh and Lin (1981) outlined the first systematic method for simplifying robot dynamic models. The simplification procedure mimicked an engineer by comparing similar algebraic expressions and removing negligible terms based upon the relative numerical values of user-specified manipulator parameters. In 1984, Neuman and Murray unveiled the computer program Algebraic Robot Modeler (ARM) for the symbolic generation of complete closed-form dynamic models.

Using Lagrangian as well as the NE formulations, this program can receive inputs on the kinematic and dynamic parameters of the robot and generate the dynamic equations. Though this can result in an error-free and convenient way of arriving at the dynamic equations, it requires a large memory and enormous CPU time and also results in long expressions. For example the ARM output of the complete closed-form dynamic model of a six DOF PUMA-560 Manipulator takes up 28 typewritten pages and takes 1308 seconds of CPU time and 662 pages of memory on a VAX 11/780 (Neuman and Murray, 1985; Neuman and Murray, 1987a; Neuman and Murray, 1987b). Later, they introduced a systematic organization procedure and showed that the efficiency of the NE equations can be improved by such a procedure (Murray and Neuman, 1988). However the elaborate LISP programming restricted the application of the package to only the local researchers due to its lack of portability. Also, the high demands on CPU time as well as memory makes such a program possible only in mini-computers. Some of the other approaches for symbolic implementation are discussed by Vukobratovic et. al (1986), Khalil et. al (1986), Izaguirre and Paul (1986), Leu and Hemati (1986), Burdick (1986), Yin and Yuh (1989), and Toogood (1989).

However, none of the present schemes are able to address all the issues; namely, minimizing the requirements of CPU time and memory size; containing the intermediate expression swelling; automating the process using commercially available symbolic programming packages either on PCs or minicomputers; and minimizing the computational burden for real time applications.

1.2.3 Parallel Processing

Parallel to the developments in symbolic programming, Luh and Lin (1982) proposed a parallel-processing scheme employing inexpensive microprocessors, instead of the conventional mini-computer. In computing the solution for the inverse dynamic problem for the Stanford arm shown in Fig. 1.3, Luh and Lin assigned one microprocessor to each manipulator joint and proposed a variable branch-and-bound search (BBS) algorithm to find a subtask-ordered schedule for the microprocessors which allowed them to compute the joint torques using the NE equations of motion. They also reported a speed-up of 2.64 on a Stanford arm. However, the total processing time for solving the minimum-time scheduling problem could not be easily reduced to a manageable level. Kasahara and Narita (1985) extended the above method and proposed a depth-first/implicit heuristic search method, which combines the BBS method and the critical path method. The scheduling strategy was flexible so that the number of processors could be varied, with an upper bound decided by the critical path. However, the task decomposition was achieved by letting the nine equations of the NE formulation as nine different tasks. This, in itself, is not an efficient process, since the concurrency of the algorithm can be increased by using a higher degree of decomposition.

Lathrop (1985) proposed two parallel algorithms executable on special-purpose processors using the VLSI technology. One is the linear parallel NE algorithm and the other is the logarithmic parallel NE algorithm. They both require potentially massive internal buffering to achieve pipelined computation between forward and backward recursions. They also involve complex communications

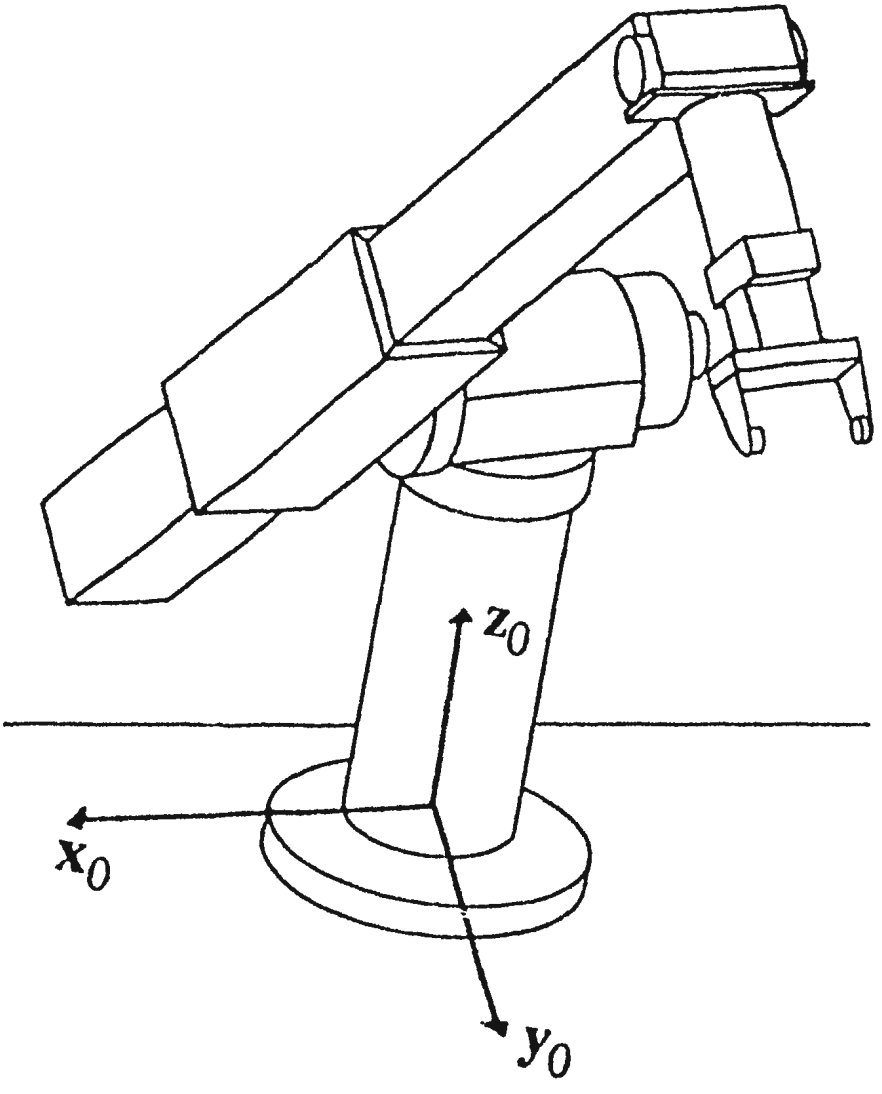


Figure 1.3: Stanford Manipulator

between the computations which degrade performance. Further, Nigam and Lee (1985) proposed an architecture for a multiprocessor-based controller using the NE formulation. Lee and Chang (1986) reformulated the NE equations in a homogeneous linear recurrence form (HLR) and developed an algorithm which could be implemented within a group of general-purpose microprocessors. Chen et. al (1988) applied the A* algorithm and a heuristic search algorithm called dy mical-highest-level-first / most-immediate-successor-first (DHLF/MISF) for scheduling the tasks on a multiprocessor system. However, in all the above works, the ease of hardware or software implementation was not carefully considered. Khosla (1988) did an extensive analysis on the hardware requirements for the NE formulation and the Lagrangian formulation and concluded that the NE formulation was more effective for parallel implementation. Vukobratovic (1988) applied the symbolic equations and subsequently used the BBS algorithm for multiprocessor implementation. Here, the job partitioning was arbitrary and hence the method is not efficient for implementation for robots of arbitrary architecture. Most of the above works consider the NE algorithm without attempting to improve its concurrency and also, do not take into account the fact that the transformation matrices and the inertia matrices are sparse and are the source of a number of multiplications with zeros and ones in real time, which can be avoided by using symbolic equations. Finally, the task partitioning is mostly manual and no systematic procedure for creating the data-base for the scheduling problem is discussed.

1.2.4 Friction Modeling

Significant contributions to the understanding of the frictional effects in robot dynamics were made by Armstrong (1986, 1988). Using an elaborate experimental set-up, he studied the PUMA-560 robot and developed an experimental procedure for modeling friction. While on one hand, his work brought out the significance of the friction terms in industrial robots, there was no analytical approach developed. The friction model depended largely on an expensive and error-prone experimental set-up. Canudas's work (Canudas: 1986, 1989) concentrates on adaptive compensation using non-linear stiction models for friction. Though the adaptive techniques give excellent results, friction being much dependent on the operating conditions, these techniques require extensive computations in real time which may not be the most efficient and cost-effective method. Gogoussis and Donath (1987, 1988) presented a mechanics approach to friction modeling from the basic Coulomb's law. The significant contribution of their work was to establish the independence of the joint reaction forces and moments on the coefficient of friction at the joints. However, no detailed approach was presented nor actual application to existing robots discussed in his work.

1.3 Thesis Objectives

With this background, this thesis tries to extend previous work and develop a systematic means of reducing the computational burden and increasing the speed of real-time computation of the inverse dynamics calculations for the robotic manipulators. The focus is on the rigid models, since most of the industrial manipulators are rigid. They are also high-torque systems, which make use of special

drives such as harmonic drives, which in turn introduce significant amounts of friction. It should be added here that there are other types of errors arising in robot control problems such as due to the backlash in gears and drag forces in underwater arms. However, the intent of this work is to include the frictional effects only. Thus the objectives of this thesis are set out as follows:

1. To improve the Newton-Euler algorithm using symbolic computations for generalized as well as customized robot models for increased computational efficiency.
2. To introduce an analytical model for the friction in robot mechanisms and study the quantitative significance of the frictional torques.
3. To design a parallel algorithm for computation of the inverse dynamics of robotic manipulators using parallel architecture, with emphasis on high speed as well as a systematic procedure for task decomposition and task scheduling.

To begin with, Chapter 2 briefly reviews the kinematics and dynamics of manipulators, and it explains symbolic computation for robot dynamics and reformulates the NE algorithm. Chapter 3 deals with friction modeling in robot dynamics and a case study of a PUMA-560 robot with harmonic drives is done to quantify the frictional torques for practical applications. Chapter 4 explains parallel processing concepts and presents a 'Task Streamlining Approach' for parallel computation of the inverse dynamics and also outlines a systematic mapping scheme for creating a list schedule and a bin-packing heuristic algorithm for scheduling computations on an arbitrary number of processors. Finally, in Chapter 5, the contributions of the thesis and recommendations for future research are outlined.

Chapter 2

Manipulator Dynamics and Symbolic Computations

2.1 Introduction

The control of robotic manipulators requires a complete knowledge of the geometric configuration of the manipulator and the dynamic behavior of the system under the actuator torques/forces. The availability of commercial packages such as REDUCE and MACSYMA has made the formulation of robot dynamic problems less cumbersome and more efficient. This chapter applies symbolic programming to robot dynamic problems for reformulating the conventional Newton-Euler algorithm for increased computational efficiency (Dhanaraj and Sharan, 1990).

2.2 The Kinematic and Dynamic Equations

2.2.1 Terminology and Definitions

A manipulator arm is a sequence of links connected by joints. Each link is numbered from 0 to n, as depicted in Fig. 2.1, where n denotes the total number of links. A joint between link i-1 and link i is referred to as joint i

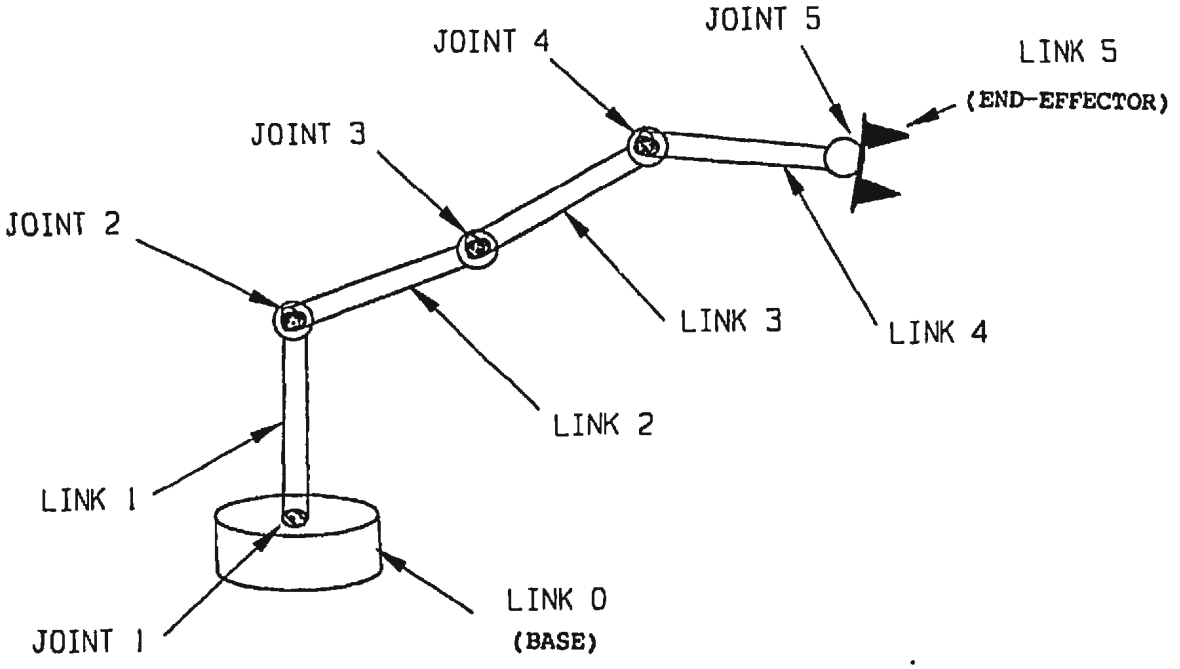


Figure 2.1: Typical Open Chain Serial Link Manipulator

which may be either revolute (relative motion is rotational) or prismatic (relative motion is translational). The 0th link is usually referred to as "base" and the nth link is termed as "end-effector". If the end-effector of a manipulator is unconstrained in free space, the serial linkage has an open loop structure and is referred to as an open kinematic chain. The degrees of freedom (DOF) represent the number of independent joint movements available for the manipulator. In general, the number of DOF of a manipulator is equal to its number of joints. A manipulator arm must have at least 6 DOF in order to locate its end-effector at an arbitrary point with an arbitrary orientation in space and those that have more than 6 DOF are termed as redundant manipulators. The set of positions and orientations in space that can be reached by an end-effector depend on the **configuration of a manipulator** which describes the types of joints and their geometry of connection in the serial linkage. The study of forward kinematics relates the position of the end-effector in the the link coordinates to the global coordinate frame attached to the base and inverse kinematics relates the position in the global frame to the local link coordinates, as shown in Fig. 2.2. In a similar manner, in forward dynamics one computes the joint position, velocity and acceleration and in inverse dynamics one computes the joint torques/forces (torque for a revolute joint and force for a prismatic joint) as shown in Fig. 2.3. For real-time control applications, we are interested in the inverse dynamics and for simulation we will use the forward dynamics.

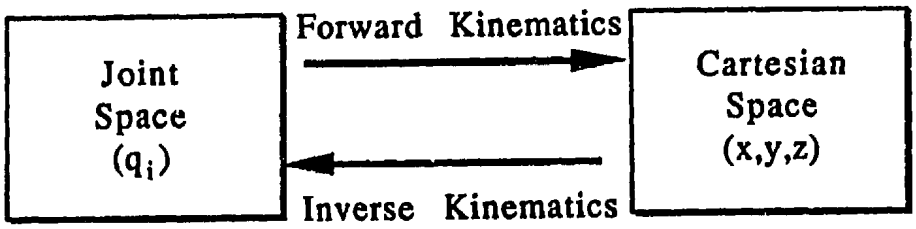


Figure 2.2: Inverse Kinematics and Forward Kinematics

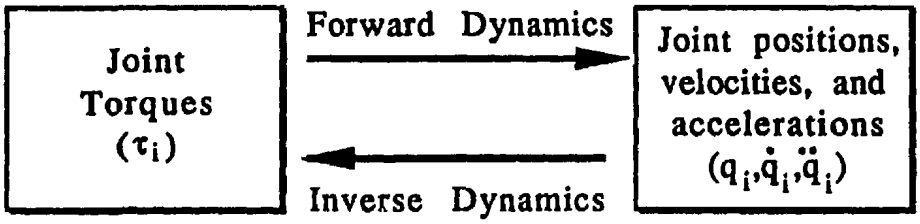


Figure 2.3: Inverse Dynamics and Forward Dynamics

2.2.2 Denavit-Hartenberg Transformation Matrix

Kinematic and dynamic modeling of a multi-body system requires a procedure by which the dynamic configuration of the manipulator can be represented. Moving coordinate frames attached to the links have provided an efficient means for such modeling (Denavit and Hartenberg, 1955). The transformation matrix which will transform a position vector, defined in one frame, to another frame, can be represented by two rotations and two translations, performed in a particular order. A right-handed coordinate frame is assigned to each link i, such that the z_i axis of the coordinate frame attached to the link coincides with the axis of relative motion of the link with respect to the previous link and the x_i axis is normal to the plane of z_i and z_{i+1}. Fig. 2.4 depicts two links i, and i-1 connected by a joint with link frames i (X_i-Y_i-Z_i) and i-1 (X_{i-1}-Y_{i-1}-Z_{i-1}) attached to the two respective links. Note that the axis z_i coincides with the axis of motion of link i and the axis z_{i-1} coincides with the axis of motion of link i-1. The four link parameters are defined as below:

$$\begin{array}{ll} a_{i-1} & \text{distance of translation along } X_{i-1} \text{ from } O_{i-1} \text{ to } O_i \\ d_i & \text{distance of translation along } Z_i \text{ from } O_{i-1} \text{ to } O_i \\ \alpha_{i-1} & \text{angle of rotation about } X_i \text{ to align } Z_{i-1} \text{ with } Z_i \\ \theta_i & \text{angle of rotation about } Z_{i-1} \text{ to align } X_{i-1} \text{ with } X_i \end{array} \qquad (2.1)$$

These are termed as "Denavit-Hartenberg (DH) Parameters" and the DH parameters for the PUMA-560 manipulator shown in Fig. 1.2 and the Stanford manipulator shown in Fig. 1.3 are given in Tables 2.1 and 2.2.

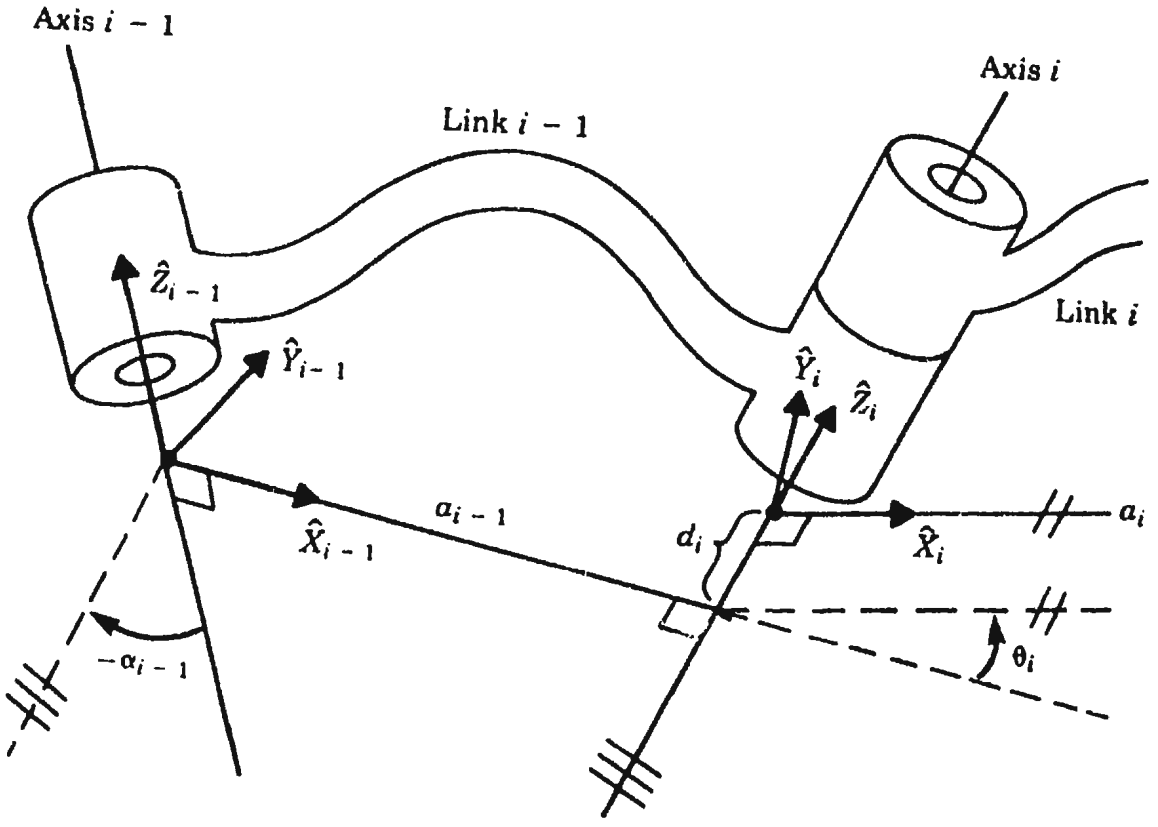


Figure 2.4: Denavit and Hartenberg Parameters

Table 2.1: D-H Parameters of PUMA 560

Link i	α_{i-1} (radians)	θ_{i-1} (radians)	a_{i-1} (meters)	d_i (meters)
1	0	q_1	0	0
2	$-\frac{\pi}{2}$	q_2	0	0.2435
3	0	q_3	0.4318	-0.0934
4	$\frac{\pi}{2}$	q_4	-0.0203	0.4331
5	$-\frac{\pi}{2}$	q_5	0	0
6	$\frac{\pi}{2}$	q_6	0	0

Table 2.2: D-H Parameters of Stanford Manipulator

Link i	α_{i-1} (radians)	θ_{i-1} (radians)	a_{i-1} (meters)	d_i (meters)
1	0	q_1	0	0
2	$-\frac{\pi}{2}$	q_2	0	0.1524
3	0	$\frac{\pi}{2}$	0	0
4	0	q_4	0	q_3
5	$-\frac{\pi}{2}$	q_5	0	0
6	$\frac{\pi}{2}$	q_6	0	0

The transformation matrix, generally known as the DH transformation matrix (Denavit and Hartenburg, 1955), can be expressed as a product of these four transformations, given by

$$[T]_i^{i-1} = Rot(X_i, \alpha_{i-1})\ \ Trans(X_i, a_{i-1})\ \ Rot(Z_i, \theta_i)\ \ Trans(Z_i, d_i) \quad (2.2)$$

where Rot(X,α) implies rotation of α degrees about the X axis and Trans(X,a) means translation along the X axis by 'a' units. These four transformations can also be represented as

$$[T]_i^{i-1} = Screw(X_i, a_{i-1}, \alpha_{i-1})\ \ Screw(Z_i, d_i, \theta_i) \quad (2.3)$$

where Screw(X,a,α) stands for a translation along the X axis by a distance a, and a rotation about the same axis by an angle α. In the expanded form, this can be written as

$$[T]_i^{i-1} = \begin{bmatrix} cos\theta_i & -sin\theta_i & 0 & -a_{i-1} \\ sin\theta_i cos\alpha_{i-1} & cos\theta_i cos\alpha_{i-1} & -sin\alpha_{i-1} & -sin\alpha_{i-1} d_i \\ sin\theta_i sin\alpha_{i-1} & cos\theta_i sin\alpha_{i-1} & cos\alpha_i & cos\alpha_i d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.4)$$

So a vector r^i defined in the frame i can be transformed to the frame i-1 using

$$\{r\}^{i-1} = [T]_i^{i-1}.\{r\}^i \quad (2.5)$$

where $\{r\}^{i-1}$ and $\{r\}^i$ are the position vectors of the same point in i-1th frame and ith frame respectively. Here the position vector $\{r\}$ is defined as a 4x1 column vector to make it compatible with the 4x4 transformation matrix (Craig, 1986), i.e.

$$\{r\} = \begin{Bmatrix} r_x \\ r_y \\ r_z \\ 1 \end{Bmatrix} \quad (2.6)$$

This transformation can be used in the case of robotic manipulators to relate the position vector in the local coordinate system at the end-effector to the inertial coordinate system at the base of the robot. Mathematically, this can be expressed as

$$\{r\}^0 = [T]_0^1[T]_1^2[T]_2^3...[T]_{i-1}^{i-2}[T]_i^{i-1}\{r\}^i = [T]_i^0\{r\}^i \tag{2.7}$$

Referring to Eq. 2.5, we can partition the transformation matrix and re-write the equation using the 3x1 position vector, as

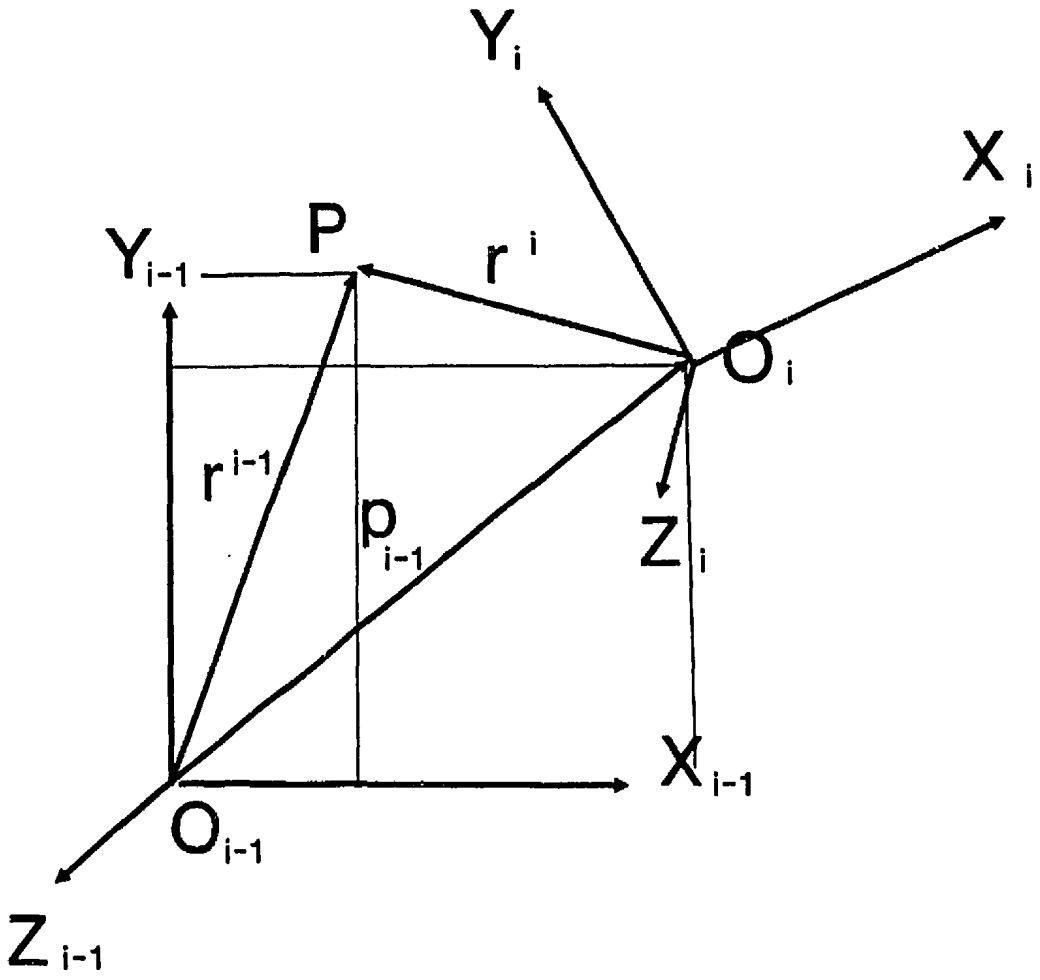
$$\{r\}^{i-1} = \left[\begin{array}{c|c} [R]_i^{i-1} & \{p\}^{i-1} \\ (3\times 3) & (3\times 1) \\ \hline 0 & 1 \\ (1\times 3) & (1\times 1) \end{array}\right] \left\{\begin{array}{c} \{r\}^i \\ (3\text{x}1) \\ \hline 1 \\ (1\text{x}1) \end{array}\right\} \tag{2.8}$$

where $[R]_i^{i-1}$ is defined as the rotation transformation matrix and $\{p\}_{i-1}$ is the position vector of the origin of ith coordinate frame referred in the i-1th frame. So we can write it as[1]

$$\{r\}^{i-1} = [R]_i^{i-1}\{r\}^i + \{p\}_{i-1} \tag{2.9}$$

The above equation can be illustrated by Fig. 2.5, where $\{r\}^{i-1}$ refers to the position vector of point A defined in the reference frame of link i-1, $\{r\}^i$ is the position vector of the same point A defined in the reference frame of link i. Geometrically, the product $[R]_i^{i-1}$ and $\{r^i\}$ yields components of $\{r\}^i$ parallel to the axes in the i-1th frame as shown in the figure. Hence the 3x3, $[R]_i$ matrix, comprised of the first three rows and columns of the Denavit-Hartenberg

[1]The vector $\{p\}_{i-1}$ refers to the position vector of the origin of the ith frame referred in the i-1th frame. Throughout this work this will be associated with the link i-1, and hence the superscript is dropped since it is referred in the i-1th frame.



Rotation of the vector r , projects the vector r in the i-1th frame, as indicated in the figure.

Figure 2.5: Rotation Transformation of Vectors

matrix, can be used effectively to transform the free vectors such as the velocity, acceleration, forces and moments from the frame i to the frame $i-1$ as given below[2]

$$\{\omega\}_i^{i-1} = [R]_i^{i-1}\{\omega\}_i \tag{2.10}$$

$$\{\alpha\}_i^{i-1} = [R]_i^{i-1}\{\alpha\}_i \tag{2.11}$$

$$\{v\}_i^{i-1} = [R]_i^{i-1}\{v\}_i \tag{2.12}$$

$$\{a\}_i^{i-1} = [R]_i^{i-1}\{a\}_i \tag{2.13}$$

$$\{f\}_i^{i-1} = [R]_i^{i-1}\{f\}_i \tag{2.14}$$

$$\{n\}_i^{i-1} = [R]_i^{i-1}\{n\}_i \tag{2.15}$$

where $\{\omega\}_i^{i-1}$ and $\{\alpha\}_i^{i-1}$ refer to the refer to the angular velocity and angular acceleration of the link respectively and $\{\mathrm{v}\}_i^{i-1}$ and $\{\mathrm{a}\}_i^{i-1}$ refer to the linear velocity and linear acceleration of the origin of ith link respectively and $\{\mathrm{f}\}_i^{i-1}$ and $\{\mathrm{n}\}_i^{i-1}$ refer to the reaction forces and moments respectively at the joint i. The superscript i-1 indicates that these vectors are referred to in the frame of the link i-1 and the absence of the superscript indicates that the vector is referred to in its own link frame. For example, $\{\mathrm{n}\}_i$ indicates the joint moment vector of the ith link referred in the ith frame and $\{\mathrm{p}\}_{i-1}$ indicates the vector from the origin of the i-1th frame to the origin of the ith frame, referred in the i-1th frame.

Since these rotation transformation matrices are orthonormal, the transpose of the matrix yields its inverse, i.e.

$$[R]^T = [R]^{-1} \tag{2.16}$$

[2]Note that the vectors do not have a superscript indicating that they are referred in the frame with which the vector is associated. For example, $\{\omega\}_i$ refers to the angular velocity of the link i referred in its own coordinate frame.

Hence we can write

$$[R]_i^{i-1} = [R]_{i-1}^{i}{}^{-1} = [R]_{i-1}^{i}{}^{T} \tag{2.17}$$

Like the 4x4 transformation matrices, the rotation matrices (3 x 3) also can be concatenated, to project vectors from one frame to another, through successive transformation of the intermediate frames.

$$\{v\}_i^0 = [R]_0^1[R]_1^2[R]_2^3...[R]_{i-1}^{i-2}[R]_i^{i-1}\{v\}_i = [R]_i^0\{v\}_i \tag{2.18}$$

Note that the above equation projects a free vector $\{v\}$ such as velocity and acceleration vectors, defined in the ith frame to the global frame and is not applicable for position vectors.

2.2.3 Newton-Euler Recursive Formulation

The dynamics of the robotic manipulator can be modeled as a set of coupled, non-linear differential equations using any one of the various formulations discussed in Chapter 1. It was noted that the Newton-Euler (NE) method is the most efficient in terms of the number of computations. This method is briefly reviewed in this section and symbolic computations are applied to simplify the NE algorithm to make it more efficient in terms of the number of computations.

The NE formulation is based on the laws governing the dynamics of rigid bodies. The manipulator is modeled as a serial chain of rigid links as shown in Fig. 2.6. The force vector acting on a link is related to the acceleration of its center of mass by Newton's second law

$$\{F\} = m\,\{\dot{v}\} \tag{2.19}$$

where {F} is the inertial force and $\{\dot{v}\}$ is the linear acceleration of the center of gravity (CG) of the link. The total moment vector about the CG is related to the angular velocity and angular acceleration of the body by Euler's equation

$$\{N\} = [I]\{\dot{\omega}\} + \{\omega\} \times [I]\{\omega\} \tag{2.20}$$

where $\{\omega\}$ is the angular velocity and $\{\dot{\omega}\}$ is the angular acceleration of the link given in the link coordinate frame. Note that "×" in Eq. (2.20) indicates the cross-product of the angular velocity vector and the angular momentum vector and $[I]$ is the 3x3 inertia tensor about the CG given as

$$[I] = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{bmatrix} \tag{2.21}$$

Fig. 2.6, shows three consecutive links in the kinematic chain of an arbitrary manipulator. Referring to this figure, $\{F\}_i$ and $\{N\}_i$ are the inertial forces and moments acting at the CG of the link i and $\{f\}_i$ and $\{n\}_i$ are the reaction forces and moments acting at the joint i. $\{s_i\}$ is the position vector of the CG of the ith link and the $\{p\}_{i-1}$ is the position vector of O_i, origin of the ith frame, referred in the i-1th frame, as defined in Eq. (2.8).

The set of recursive equations to compute the inverse dynamic torques is given in Table 2.3 and the derivation of these equations is given in Appendix B. As discussed in Chapter 1, the choice of proper coordinate frames is important as it directly affects the computational count. For minimizing the computations, all the kinematic and dynamic parameters of each link are referred to in its local coordinate frame attached to the link. Referring to Table 2.3, q_i, $\dot{q}_i$ and $\ddot{q}_i$ are the position, velocity and accelerations of the link i with respect to the previous link.

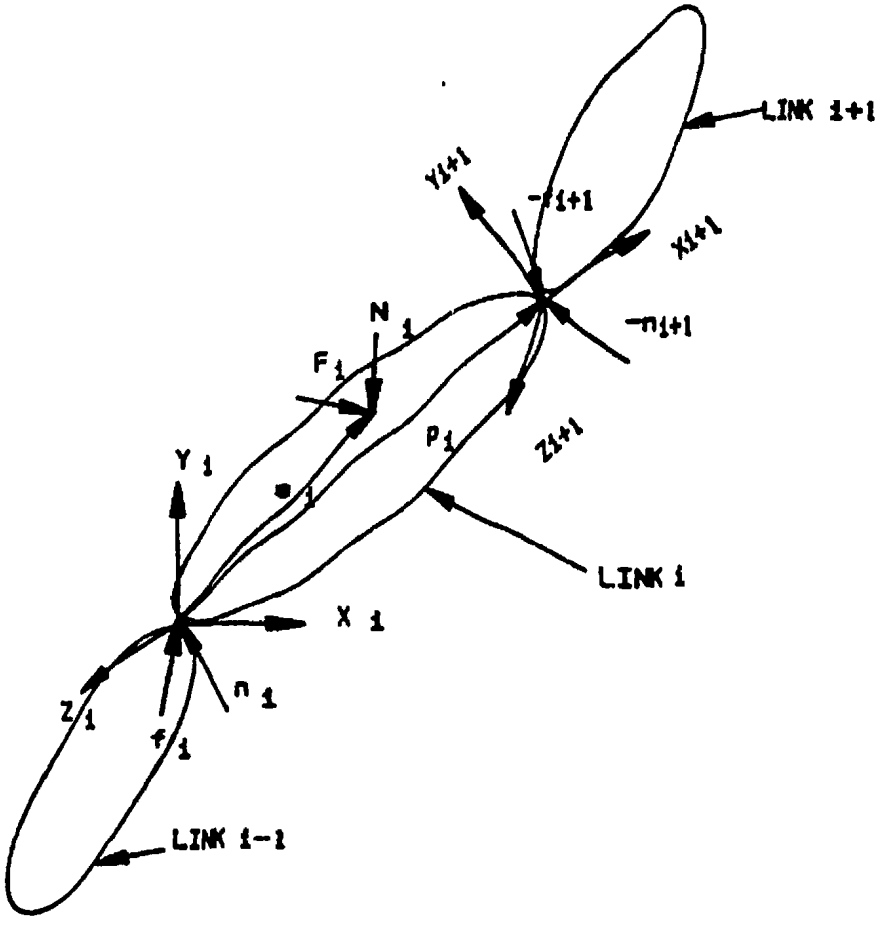


Figure 2.6: Free-body diagram of a link

Table 2.3: Recursive Newton-Euler Algorithm

FORWARD RECURSION:

$$\text{Step 1}: \{\omega\}_i = \begin{cases} [R]_i^T\{\omega\}_{i-1} + \{z\}\dot{q}_i & \text{if joint i rotational} \\ [R]_i^T\{\omega\}_{i-1} & \text{if joint i translational} \end{cases}$$

$$\text{Step 2}: \{\alpha\}_i = \begin{cases} [R]_i^T\{\alpha\}_{i-1} + \{z\}\ddot{q}_i + \boxed{[R]_i^T\{\omega\}_{i-1} \times \{z\}\dot{q}_i} \ \textcircled{1} & \text{if joint i rotational} \\ [R]_i^T\{\alpha\}_{i-1} & \text{if joint i translational} \end{cases}$$

$$\text{Step 3}: \{a\}_i = \begin{cases} [R]_i^T(\{a\}_{i-1} + \boxed{\{\alpha\}_{i-1} \times \{p\}_{i-1} + \{\omega\}_{i-1} \times \{\omega\}_{i-1} \times \{p\}_{i-1}}) \ \textcircled{2} \\ \quad \text{if joint i rotational} \\ [R]_i^T(\{a\}_{i-1} + \boxed{\{\alpha\}_{i-1} \times \{p\}_{i-1} + \{\omega\}_{i-1} \times \{\omega\}_{i-1} \times \{p\}_{i-1}}) \ \textcircled{3} \\ \quad + z_i\ddot{q}_i + \boxed{2[R]_i^T\{\omega\}_{i-1} \times \{z\}\dot{q}_i} \ \textcircled{4} \\ \quad \text{if joint i translational} \end{cases}$$

$$\text{Step 4}: \{a\}_{ci} = \{a\}_i + \boxed{\{\alpha\}_i \times \{s_i\} + \{\omega\}_i \times \{\omega\}_i \times \{s_i\}} \ \textcircled{5}$$

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

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

BACKWARD RECURSION:

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

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

$$\text{Step 9}: \tau_i = \{z\}.\{n\}_i = n_{ii}$$

In case of a rotary joint these will be rotational parameters and for a prismatic joint these will be linear parameters. These are the motion parameters which are directly introduced by the motor movements at the joints. In Table 2.3, Steps 1 to 4 compute the kinematic parameters, namely angular velocity, angular acceleration of the link, and the linear acceleration of the origin of the link coordinate frame and the linear acceleration of the CG of the link. Steps 5 and 6 compute the total forces and moments acting on the body at the CG of the link. Since the velocity and acceleration of the base is known (generally equal to zero), the forward recursion can be done in an iterative manner, starting with the first link and moving successively, link by link, outward to the end-effector (i=n). After completing Step 6 for i=n, the reaction force and moment at the nth joint (f_n, n_n) can be computed using Steps 7 and 8. In Step 9, the z component of the vector $\{n\}_i$ computed in Step 8, is assigned as τ_i (actuator torque) for a revolute joint and for a prismatic joint, the z component of the vector $\{f\}_i$ computed in Step 7 is assigned as τ_i (actuator force). These steps are arranged as a combination of forward recursion for computing the kinematic parameters and backward recursion for computing the torques, as shown in Fig. 2.7.

2.3 Symbolic Computations

2.3.1 Application of Symbolic Programming

Symbolic mathematical models, which characterize the dynamic behavior of manipulators, are needed for physical insight and engineering analysis and design. Dynamic simulators, parameter identification and real-time control algorithms rely upon efficient numerical models. The dynamic formulation, such as La-

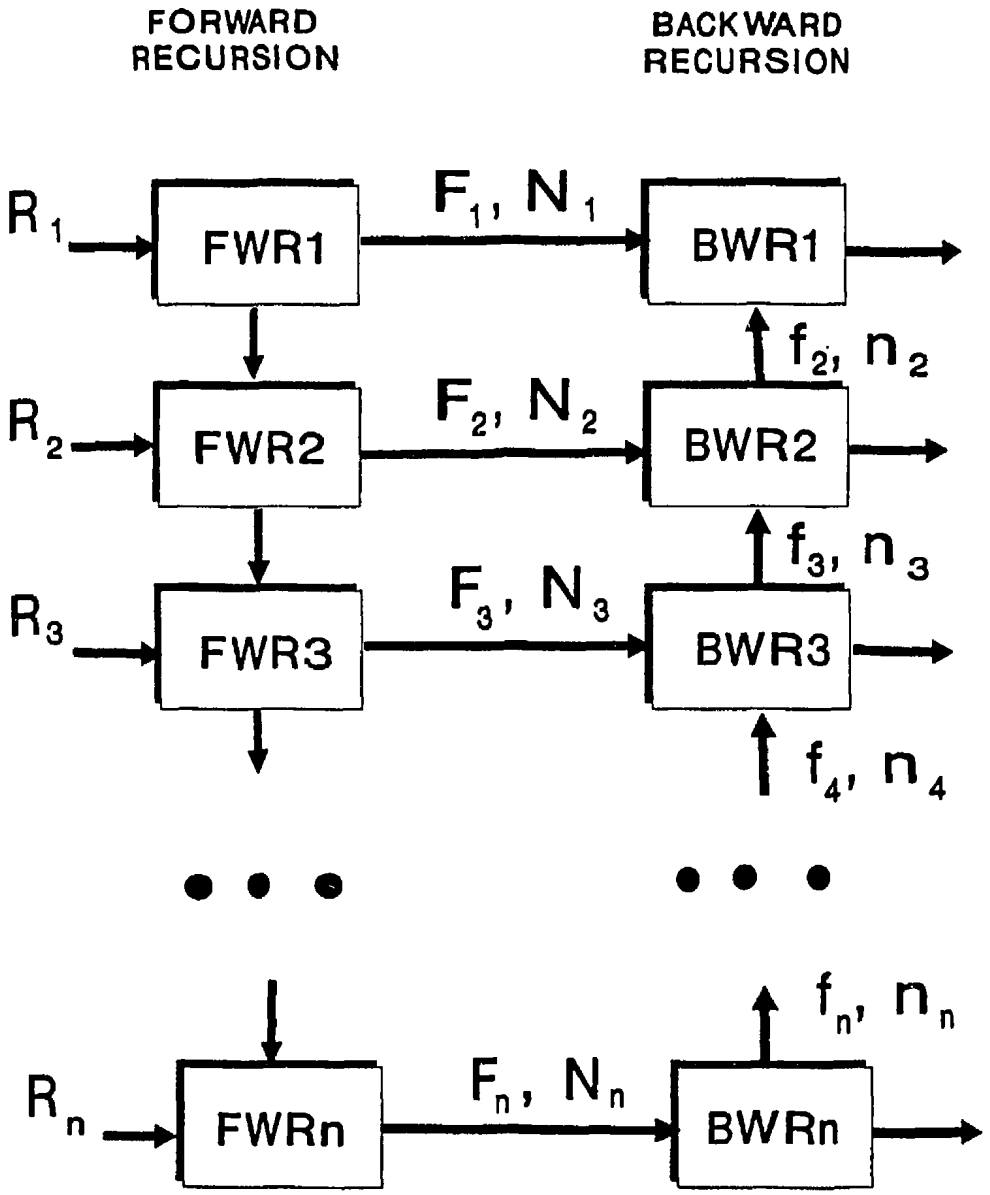


Figure 2.7: Newton-Euler Scheme for Inverse Dynamics of Manipulators

grangian formulation of robotic manipulators is a complex process, involving algebraic manipulation and differentiation, especially when the number of links is greater than 3. Symbolic programs can be used to overcome this difficulty and also to ensure the accuracy of the dynamic model. These symbolic programs manipulate algebraic expressions, in contrast to the conventional application of computers to number crunching. An internal algebraic representation enables the symbolic program to encode uniquely in computer memory the algebraic expressions, and is designed to facilitate the implementation of symbolic mathematical operations. These can be written in languages such as LISP. Also commercially available packages such as REDUCE or MACSYMA can be made use of to develop application packages. Typically a symbolic programming system such as REDUCE can handle tasks such as

(a) expansion and ordering of polynomials and rational functions

(b) substitutions and pattern matching in a wide variety of forms

(c) calculations with symbolic matrices or vectors

(d) analytic differentiation and integration and

(e) factorization of polynomials.

For details of this software package, the reader is referred to Gayna (1988) and the REDUCE User's Manual (1986) or MACSYMA User's Manual (1983).

Symbolic programs can be used in robotics for two types of applications; for developing closed form dynamic equations for engineering design applications and for developing efficient dynamic equations for real time control applications. To develop the closed-form dynamic robot model, the intermediate quantities are generated sequentially (as prescribed by the formulation), injecting the complete

analytical expression for the intermediate quantities whenever they are required in the subsequent calculations. The recursions are thereby expanded, and closed-form symbolic expressions are obtained for the joint torques/forces. The coefficients of the closed-form model are then extracted from each joint torque/force equation of motion.

The second application of symbolic formulations preserves the recursive structure of the NE formulation, thereby leading to a recursive dynamic robot model. The intermediate quantities are again generated sequentially, but each quantity is examined. If a quantity requires no mathematical operation to be evaluated, the value is passed to subsequent calculations. If the quantity does require a mathematical operation to be evaluated then the symbolic quantity's name is passed to subsequent calculations. In the former case, one eliminates unnecessary intermediate quantities, while in the latter, one suppresses the expansion of the recursions, preserving the recursive structure of the formulation. Generating dynamic models through application of the NE recursive formulation requires only basic matrix algebra operations.

For the present work, the NE algorithm is initially reformulated applying symbolic computations to bring down the computational count. This makes it a general algorithm which can be applied to any arbitrary manipulator. Subsequently, this reformulated NE algorithm is applied in a symbolic program to generate customized equations for a particular manipulator. The computational reductions stem from the elimination of additions of zero, multiplications by zeros or ones, and algebraic simplifications, all of which are performed numerically in the general-purpose approach. Also, by recognizing and removing repetitive

calculations within the equations, one can achieve further reduction in the computations.

2.3.2 Reformulation of the NE Algorithm for Reducing the Computations

The NE recursive formulation has been by far the best computationally efficient algorithm. A careful analysis of these equations reveal that some of the terms which have been shown in boxes (these boxes have been numbered in the top right hand corner) can be reformulated using symbolic computations to economize on the on-line computational requirements. Referring to Table 2.3, the terms enclosed in boxes 1 and 4 compute the Coriolis acceleration terms; the terms enclosed in boxes 2, 3 and 5 compute the acceleration difference vector; and the term in Box 6 in Step 4 computes the gyroscopic moment terms. These can be simplified making use of the vector algebra and the symbolic computations.

Coriolis Acceleration (Boxes 1 and 4)

For a revolute joint the Coriolis component of the acceleration appears in the angular acceleration (Box 1) and for a prismatic joint, it appears in the linear acceleration of the origin of the reference frame (Box 4). From Table 2.3 one can see that $[R]_i^T \{\omega\}_{i-1}$ occurs in Step 1 and also in Step 2. The relative velocity of the ith link with respect to the $i-1$th link is always in the z_i direction. Hence we can write

$$[R]_i^T \{\omega\}_{i-1} = \begin{Bmatrix} \omega_{ix} \\ \omega_{iy} \\ \omega_{iz} - \dot{q}_i \end{Bmatrix}$$

Using the matrix representation, we can then reduce the Coriolis term as:

$$[R]_i^T \{\omega\}_{i-1} \times z_i \dot{q}_i = \dot{q}_i . \begin{Bmatrix} \omega_{iy} \\ -\omega_{ix} \\ 0 \end{Bmatrix}$$

Thus the Coriolis term can be computed in just two multiplications, instead of a full matrix multiplication and a cross-product.

Acceleration Difference Matrix (Boxes 2, 3 and 5)

Referring to Table 2.3, Boxes 2,3 and 5 compute the summation of a cross product and a triple cross product, which is the acceleration difference between two points on the same link. Referring to Fig. 2.8, the acceleration of the point

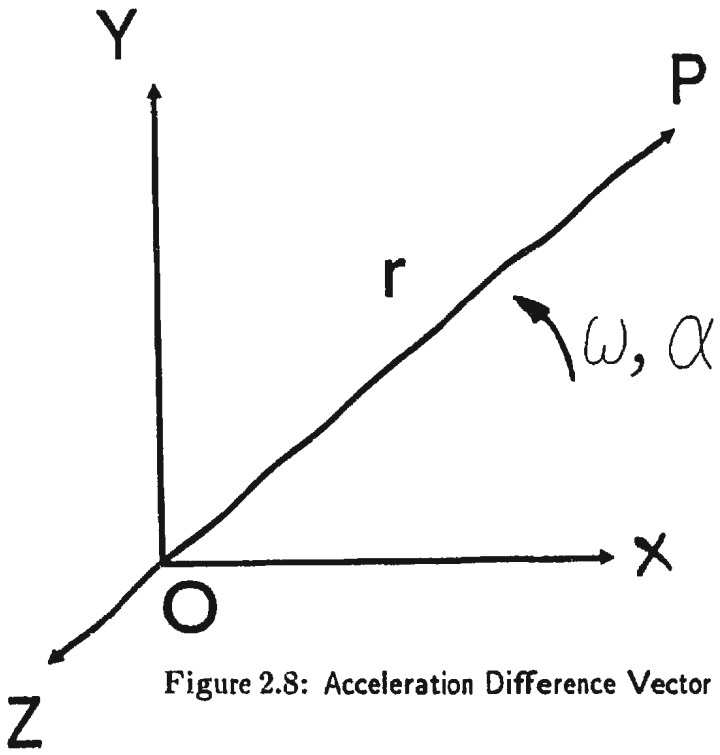


Figure 2.8: Acceleration Difference Vector

P can be written as

$$\begin{aligned} \{a\}_P &= \{a\}_O + \{\alpha\} \times \{r\} + \{\omega\} \times \{\omega\} \times \{r\} \\ &= \{a\}_O + \{a\}_{PO} \end{aligned} \tag{2.22}$$

where $\{a\}_{PO}$ refers to the acceleration difference between O and P. From vector algebra the second term can be converted to a matrix form as

$$\{\alpha\} \times \{r\} = \begin{bmatrix} 0 & -\alpha_z & \alpha_y \\ \alpha_z & 0 & -\alpha_x \\ -\alpha_y & \alpha_x & 0 \end{bmatrix} \begin{Bmatrix} r_x \\ r_y \\ r_z \end{Bmatrix} \tag{2.23}$$

Again using vector algebra the vector cross product can be written as

$$\{\omega\} \times \{\omega\} \times \{r\} = (\{\omega\}.\{r\})\{\omega\} - (\{\omega\}.\{\omega\})\{r\} \tag{2.24}$$

By making use of these two equations we can show that

$$\{a\}_{OP} = \begin{bmatrix} -(\omega_y^2 + \omega_z^2) & \omega_x\omega_y - \alpha_z & \omega_z\omega_x + \alpha_y \\ \omega_x\omega_y + \alpha_z & -(\omega_z^2 + \omega_x^2) & \omega_y\omega_z - \alpha_x \\ \omega_z\omega_x - \alpha_y & \omega_y\omega_z + \alpha_x & -(\omega_x^2 + \omega_y^2) \end{bmatrix} \begin{Bmatrix} r_x \\ r_y \\ r_z \end{Bmatrix} \tag{2.25}$$

where ω_x, ω_y, ω_z are the x, y, and z components of the angular velocity vector and α_x, α_y, α_z are the x, y, and z components of the angular acceleration vector. So we can write,

$$\{a\}_{OP} = [\lambda] * \{r\} \tag{2.26}$$

where λ can be written as

$$[\lambda] = \begin{bmatrix} -(\omega_y^2 + \omega_z^2) & \omega_x\omega_y - \alpha_z & \omega_z\omega_x + \alpha_y \\ \omega_x\omega_y + \alpha_z & -(\omega_z^2 + \omega_x^2) & \omega_y\omega_z - \alpha_x \\ \omega_z\omega_x - \alpha_y & \omega_y\omega_z + \alpha_x & -(\omega_x^2 + \omega_y^2) \end{bmatrix} \tag{2.27}$$

This is shown in Table 2.2 as an intermediate step after Step 2, where the matrix $[\lambda_i]$ is computed which has terms k_x, k_y, k_z which are written as

$$\begin{aligned} k_x &= \omega_y\omega_z \\ k_y &= \omega_z\omega_x \\ k_z &= \omega_x\omega_y \end{aligned} \tag{2.28}$$

These terms are computed at this stage and used again in Step 6. Now the steps 5 and 6 in the Newton - Euler formulation can be rewritten as

$$\{a\}_i = [R]_i^T(\{a\}_{i-1} + [\lambda]_{i-1}.\{p\}_{i-1})$$

$$\{a\}_{ci} = \{a\}_i + [\lambda]_i.\{s\}_i$$

where λ_i is given by Eq. 2.27. It should be noted that $[\lambda]_i$ is dependent only on the angular velocities and angular accelerations of the links, and hence $[\lambda]_0$ is a null matrix. If the ith joint is a prismatic joint then

$$\{\omega\}_i = [R]_i^T.\{\omega\}_{i-1}$$

$$\{\alpha\}_i = [R]_i^T.\{\alpha\}_{i-1}$$

hence $[\lambda]_i$ is obtained by simply transforming $[\lambda]_{i-1}$ as

$$[\lambda_i] = [R]_i[\lambda]_{i-1}[R]_i^T \tag{2.29}$$

If symbolic manipulation is used for evaluating this matrix, the real time computational requirement for $[\lambda]_i$ for the prismatic joint can be brought down to zero. For example, the Stanford manipulator (Fig. 1.3) has a prismatic joint in its design (i=3). The $[\lambda]_3$ is computed for this joint and is shown to be equal to the $[\lambda]_2$ matrix projected in the link coordinate frame, using Eq. 2.29.

$$[\lambda]_3 = [R]_3[\lambda]_2[R]_3^T$$

The rotation transformation matrix for i=3, in this case can be computed using the D-H parameters given in Appendix A and this is given as

$$[R]_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

$\{\omega\}_2$ can be written as

$$\{\omega\}_2 = \begin{Bmatrix} \omega_{2x} \\ \omega_{2y} \\ \omega_{2z} \end{Bmatrix}$$

and $\{\omega\}_3$ is written as

$$\{\omega\}_3 = \begin{Bmatrix} \omega_{3x} \\ \omega_{3y} \\ \omega_{3z} \end{Bmatrix}$$

Using similar notation for $\{\alpha\}_2$ and $\{\alpha\}_3$, we can write the matrices $[\lambda]_2$ and $[\lambda]_3$ as

$$[\lambda]_2 = \begin{bmatrix} -(\omega_{2y}^2 + \omega_{2z}^2) & \omega_{2x}\omega_{2y} - \alpha_{2z} & \omega_{2z}\omega_{2x} + \alpha_{2y} \\ \omega_{2x}\omega_{2y} + \alpha_{2z} & -(\omega_{2z}^2 + \omega_{2x}^2) & \omega_{2y}\omega_{2z} - \alpha_{2x} \\ \omega_{2z}\omega_{2x} - \alpha_{2y} & \omega_{2y}\omega_{2z} + \alpha_{2x} & -(\omega_{2x}^2 + \omega_{2y}^2) \end{bmatrix} \tag{2.30}$$

$$[\lambda]_3 = \begin{bmatrix} -(\omega_{3y}^2 + \omega_{3z}^2) & \omega_{3x}\omega_{3y} - \alpha_{3z} & \omega_{3z}\omega_{3x} + \alpha_{3y} \\ \omega_{3x}\omega_{3y} + \alpha_{3z} & -(\omega_{3z}^2 + \omega_{3x}^2) & \omega_{3y}\omega_{3z} - \alpha_{3x} \\ \omega_{3z}\omega_{3x} - \alpha_{3y} & \omega_{3y}\omega_{3z} + \alpha_{3x} & -(\omega_{3x}^2 + \omega_{3y}^2) \end{bmatrix} \tag{2.31}$$

Using Steps 1 and 2 of Table 2.3, we can write

$$\{\omega\}_3 = [R]_3^T \{\omega\}_2 \tag{2.32}$$

$$\{\alpha\}_3 = [R]_3^T \{\alpha\}_2 \tag{2.33}$$

Symbolically computing these two equations, we can show that

$$\{\omega\}_3 = \begin{Bmatrix} \omega_{2x} \\ \omega_{2z} \\ -\omega_{2y} \end{Bmatrix} \tag{2.34}$$

and

$$\{\alpha\}_3 = \begin{Bmatrix} \alpha_{2x} \\ \alpha_{2z} \\ -\alpha_{2y} \end{Bmatrix} \tag{2.35}$$

Substituting these results in Eq. (2.31), we find

$$[\lambda]_3 = \begin{bmatrix} -(\omega_{2z}^2 + \omega_{2y}^2) & \omega_{2x}\omega_{2z} + \alpha_{2y} & -\omega_{2y}\omega_{2x} + \alpha_{2z} \\ \omega_{2x}\omega_{2z} - \alpha_{2y} & -(\omega_{2y}^2 + \omega_{2x}^2) & -\omega_{2z}\omega_{2y} - \alpha_{2x} \\ -\omega_{2y}\omega_{2x} - \alpha_{2z} & -\omega_{2z}\omega_{2y} + \alpha_{2x} & -(\omega_{2x}^2 + \omega_{2z}^2) \end{bmatrix} \tag{2.36}$$

Noting the terms of $[\lambda]_2$, we can write the above equation as

$$[\lambda]_3 = \begin{bmatrix} \lambda_{211} & -\lambda_{213} & \lambda_{212} \\ -\lambda_{231} & \lambda_{233} & -\lambda_{232} \\ \lambda_{221} & -\lambda_{223} & \lambda_{222} \end{bmatrix} \tag{2.37}$$

where λ_{2ij} refer to the (i,j)th element of the $[\lambda]_2$ matrix.

Projecting $[\lambda]_2$ in the frame of link 3, we can write

$$[R]_3[\lambda]_2[R]_3^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \lambda_{211} & \lambda_{212} & \lambda_{213} \\ \lambda_{221} & \lambda_{233} & \lambda_{223} \\ \lambda_{231} & \lambda_{232} & \lambda_{233} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \quad (2.38)$$

Multiplying these matrices we get

$$[R]_3[\lambda]_2[R]_3^T = \begin{bmatrix} \lambda_{211} & -\lambda_{213} & \lambda_{212} \\ -\lambda_{231} & \lambda_{233} & -\lambda_{232} \\ \lambda_{221} & -\lambda_{223} & \lambda_{222} \end{bmatrix} \quad (2.39)$$

We note that the RHS of Eqs. (2.37) and (2.39) are both same and hence we can write

$$[\lambda]_3 = [R]_3[\lambda]_2[R]_3^T$$

It should be noted that the $[\lambda]_i$ matrix computed for the ith link is used to compute the linear acceleration of the CG of the ith link as well as that of the origin of the i+1th link.

Gyroscopic Moment (Box 6)

Now we can analyze the term ω x $[I].\omega$, which refers to the gyroscopic moment, M_g, in this section. Using matrix representations, this can be written as

$$\{M\}_g = \{\omega\} \times [I].\{\omega\} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{bmatrix} \begin{Bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{Bmatrix} \quad (2.40)$$

By carrying out the algebraic manipulations, we can write the above equation as a sum of two matrices, one from the diagonal terms and the other from the off-diagonal terms of the inertia tensor given as

$$\{M\}_g = \begin{Bmatrix} \omega_y\omega_z(I_{zz} - I_{yy}) \\ \omega_z\omega_x(I_{xx} - I_{zz}) \\ \omega_x\omega_y(I_{yy} - I_{xx}) \end{Bmatrix} + \begin{bmatrix} \omega_y^2 - \omega_z^2 & \omega_x\omega_y & -\omega_x\omega_z \\ -\omega_x\omega_y & \omega_z^2 - \omega_x^2 & \omega_y\omega_z \\ \omega_x\omega_z & -\omega_y\omega_z & \omega_x^2 - \omega_y^2 \end{bmatrix} \begin{Bmatrix} I_{yz} \\ I_{xz} \\ I_{xy} \end{Bmatrix} \quad (2.41)$$

In most cases, however, the robot design ensures that the principal axes are parallel to the coordinate axes and hence I_{xy}, I_{xz} and I_{yz} are zero. In such cases the above equation reduces to

$$\mathrm{M_g} = \left\{ \begin{array}{l} \omega_y\omega_z(I_{zz} - I_{yy}) \\ \omega_z\omega_x(I_{xx} - I_{zz}) \\ \omega_x\omega_y(I_{yy} - I_{xx}) \end{array} \right\} = \left\{ \begin{array}{l} D_x.k_x \\ D_y.k_y \\ D_z.k_z \end{array} \right\} \tag{2.42}$$

where,

$$\begin{array}{l} D_x = I_{zz} - I_{yy} \\ D_y = I_{xx} - I_{zz} \\ D_z = I_{yy} - I_{xx} \end{array} \tag{2.43}$$

and k_x, k_y, k_z are defined in Eq. (2.28). It should be noted that these have been computed in Step 2 and hence the vector cross-product with a matrix product is replaced by three multiplications. In this way, the overall computations in the dynamic equations can be reduced by a considerable amount. The overall algorithm, incorporating the above modifications and explicitly identifying the intermediate variables, is given in Table 2.4. The reformulated terms are shown in this table, in boxes numbered corresponding to the boxes in Table 2.3. All the modifications appear only in the forward recursion and the backward recursion is not altered. This can be applied to any manipulator in the same way Table 2.1 is applied. The forward recursions are carried out for links 1 to n and subsequently the backward recursions are carried out for links n to 1 and the torques/forces are extracted from the reaction moments/forces at the joint.

Table 2.4: Modified Newton-Euler Algorithm

FORWARD RECURSION:

Initialize : $\{\omega\}_0 = 0$; $\{\alpha\}_0 = 0$; $\{a\}_0 = -\{ g \}$; $[\lambda]_0 = 0$; $\{z\} = \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix}$

1. $\{\omega\}_i = \begin{cases} [R]_i^T\{\omega\}_{i-1} + \{z\}\dot{q}_i & \text{if joint i rotational} \\ [R]_i^T\{\omega\}_{i-1} & \text{if joint i translational} \end{cases}$

2. $\{\alpha\}_i = \begin{cases} [R]_i^T\{\alpha\}_{i-1} + \{z\}\ddot{q}_i + \boxed{\dot{q}_i \begin{Bmatrix} \omega_{iy} \\ -\omega_{ix} \\ 0 \end{Bmatrix}} \ \textcircled{1} & \text{if joint i rotational} \\ [R]_i^T\{\alpha\}_{i-1} & \text{if joint i translational} \end{cases}$

$*[\lambda]_i = \begin{cases} \begin{bmatrix} -(\omega_y^2+\omega_z^2) & k_z-\alpha_z & k_y+\alpha_y \\ k_z+\alpha_z & -(\omega_z^2+\omega_x^2) & k_x-\alpha_x \\ k_y-\alpha_y & k_x+\alpha_x & -(\omega_x^2+\omega_y^2) \end{bmatrix} & k_x = \omega_y\omega_z; k_y = \omega_z\omega_x; k_z = \omega_x\omega_y \\ & \text{if joint i rotational} \\ [R]_i\lambda_{i-1}[R]_i^T & \text{if joint i translational} \end{cases}$

3. $\{a\}_i = \begin{cases} [R]_i^T(\{a\}_{i-1} + \boxed{[\lambda]_{i-1}.\{p\}_{i-1}}\ \textcircled{2}) & \text{if joint i rotational} \\ [R]_i^T(\{a\}_{i-1} + \boxed{[\lambda]_{i-1}.\{p\}_{i-1}}\ \textcircled{3}) + \{z\}\ddot{q}_i + \boxed{2\,\dot{q}_i.\begin{Bmatrix} \omega_{iy} \\ -\omega_{ix} \\ 0 \end{Bmatrix}}\ \textcircled{4} & \text{if joint i translational} \end{cases}$

4. $\{a\}_{ci} = \{a\}_i + \boxed{[\lambda]_i.\{s\}_i}\ \textcircled{5}$

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

6. $\{N\}_i = [I]_i\{\alpha\}_i + \boxed{\begin{Bmatrix} D_x.k_x \\ D_y.k_y \\ D_z.k_z \end{Bmatrix}}\ \textcircled{6}$ $\begin{bmatrix} D_x = I_{zz} - I_{yy} \\ D_y = I_{xx} - I_{zz} \\ D_z = I_{yy} - I_{xx} \end{bmatrix}$ (Inertia Consts)

BACKWARD RECURSION: Initialize : $\{f\}_{n+1} = 0$; $\{n\}_{n+1} = 0$

7. $\{f\}_i = \{F\}_i + \{f\}_{i+1}^i \qquad [\{f\}_{i+1}^i = [R]_{i+1}\{f\}_{i+1}]$

8. $\{n\}_i = [R]_{i+1}\{n\}_{i+1} + \{N\}_i + \{s\}_i \times \{F\}_i + \{p\}_i \times \{f\}_{i+1}^i$

9. $\tau_i = \{z\}\{n\}_i = n_{iz}$

2.4 Symbolic Implementation of the Algorithm

The main objectives of symbolic implementation of the dynamic equations are:

1. To avoid multiplication with zeros and ones and addition with zeros in real time.

2. To simplify the algebraic expressions for minimum computation.

3. To identify and maintain intermediate variables which will minimize the computations, by containing *expression swelling.*

4. To reduce the computational burden of the symbolic modeling software, in terms of the execution time as well as the memory required.

In a recursive form of an equation if symbolic computation is resorted to, in a sequential manner, the final expression tends to be a blown up expression, leading to a much higher arithmetic count than the numerical implementation. This has been termed as *expression swelling*; for computational efficiency this expression swelling has to be contained. For example, let us consider the following problem

$$\begin{aligned} a &= b + c \\ e &= c * a + d \\ g &= ef + ca \end{aligned}$$

In the above set of equations, if the final objective is to compute g, if symbolic computation is resorted to in a sequential manner, the final expression for 'g' in terms of the basic variables b,c,d and f will be

$$g = (c * (b + c) + d) * f + c * (b + c)$$

One can note that the symbolically expanded and simplified expression, has 3 multiplications and 4 additions whereas the numerical implementation would involve 3 multiplications and 3 additions. It is quite obvious that the extra addition is due to the term $(b+c)$ which is being computed twice; had it been computed separately once and substituted later in the final expression, we would have arrived at the same count as the numerical implementation. Hence we see that by expanding expressions by sequentially substituting one expression in another leads to swelling of expressions and the computation is better controlled when intermediate variables are created to avoid this problem.

In this work, each of the parameters in Table 2.4 ($\{\omega\}_i, \{\alpha\}_i, [\lambda]_i$, etc.) are symbolically computed. The expression for one parameter arrived at a step is not substituted in any other step where the same parameter may appear. For example, referring to Table 2.4, the expression for the three components of $\{\omega\}$ is symbolically computed in Step 1, and stored in the numerical program which is written in FORTRAN. In subsequent steps, $\{\omega\}$ is used only as a variable in the symbolic program, without substituting its equivalent expression. In this way, the numerical program computes the numerical value of $\{\omega\}$ initially using the expressions obtained from the symbolic program and subsequently substitutes that value in the other expressions where the parameter $\{\omega\}$ may appear. In addition to the above, all the elements of the various matrices are symbolically computed as new variables and they are subsequently numerically substituted. For example, in computing the matrix $[\lambda]_i$, the expression for the all the elements of the matrix is obtained by using symbolic computations and stored in the numerical program. The subsequent steps in the symbolic program can be carried

out by using the new variables representing these elements. Thus repetitive numerical computation of identical expressions is avoided. This procedure can be conveniently incorporated in any of the commercially available symbolic packages such as MACSYMA or REDUCE (also available for the micro-computers). A FORTRAN or C program can be directly generated from the symbolic program which can be compiled and used in the control software. In this way, one can formulate very efficiently the equations to compute the torques in the inverse dynamics calculations. The dynamic equations of the PUMA-560 robot shown in Fig. 1.2, with and without the wrist and also those of the Stanford robot shown in Fig. 1.3, have been generated using the above procedure. Tables 2.5 to 2.8 give the dynamic parameters of these robot models. The symbolic program in REDUCE for generating the customized equations of PUMA-560 (6 DOF) manipulator is given in Appendix D. The output of such a symbolic program would be a FORTRAN program to compute the inverse dynamic torques/forces and these inverse dynamic equations for some standard manipulators are also given in Appendix D.

Table 2.5: Center of Mass Data for PUMA-560

Link (i)	mass (kg)	$\overline{x}$ (m)	$\overline{y}$ (m)	$\overline{z}$ (m)
2	17.40	0.0680	0.0060	-0.0160
3	4.80	0	-0.0700	0.0140
4	0.82	0	0	-0.0190
5	0.34	0	0	0
6	0.09	0	0	0.032

Table 2.6: Moment of Inertia Parameters for PUMA-560

Link	I_{xx} (kg-m^2)	I_{yy} (kg-m^2)	I_{zz} (kg-m^2)	I_{motor} (kg-m^2)
1	-	-	0.35	1.14
2	0.130	0.524	0.539	4.71
3	0.066	0.0125	0.086	0.83
4	0.0018	0.0018	0.0013	0.20
5	0.0030	0.0030	0.0040	0.179
6	0.0015	0.0015	0.0004	0.193

Table 2.7: Center of Mass Data for Stanford Manipulator

Link (i)	mass (kg)	$\overline{x}$ (m)	$\overline{y}$ (m)	$\overline{z}$ (m)
1	9.29	0	-0.1105	-0.0175
2	5.01	0	0	-0.1054
3	4.25	0	0	0.6447
4	1.08	0	-0.0054	-0.0092
5	0.63	0	-0.0566	0
6	0.51	0	0	0.1554

Table 2.8: Moment of Inertia Parameters for Stanford Manipulator

Link	I_{xx} (kg-m^2)	I_{yy} (kg-m^2)	I_{zz} (kg-m^2)	I_{motor} (kg-m^2)
1	0.276	0.071	0.255	0.953
2	0.108	0.100	0.018	2.193
3	2.510	2.510	-	0.782 (kg)
4	0.002	0.001	0.001	0.106
5	0.003	0.0004	0.003	0.097
6	0.013	0.013	0.0003	0.020

2.5 Computational Efficiency

The comparison of computational count of the algorithm outlined in this work with the conventional NE Algorithm for typical prismatic and revolute joints is given in Table 2.9. The computational count for the inverse dynamic computations when the joint is revolute is 258 floating point operations (flops) using the conventional NE algorithm per joint compared to 168 flops required by the modified NE algorithm. For a prismatic joint the computations reduce from 200 flops to 113 flops. These results show that the modifications as suggested in this chapter can make the NE equations more efficient. Customization of the algorithm for a particular robot further brings down the computational count due to the possible zeroes and ones in the position vectors, namely, $\{p\}_i$, $\{s\}_i$. The comparison of computational count for implementation of the above algorithm for some standard manipulators with some of the earlier published results is given in Table 2.10. The computations for the customized dynamics using the method outlined in this chapter are much less than the published results for most cases. In case of 3 DOF, ARM (Murray and Neuman 1988) seems to be yielding better results but the same procedure results in about 30% more computations for the 6 DOF PUMA robot. Also it should be noted that ARM requires excessively large CPU time for generating these equations whereas the method outlined in this chapter takes only a fraction of a second to compute this model. The comparison of computational counts for PUMA-560 (6 DOF) manipulator is shown graphically in Fig. 2.9.

Table 2.9: Comparision of Computations

Parameter	Revolute Joint				Prismatic Joint			
	Method I		Method II		Method I		Method II	
	M	A	M	A	M	A	M	A
$\{\omega_i\}$	9	7	9	7	0	0	0	0
$\{\alpha_i\}$	11	9	24	19	0	0	0	0
$\{\lambda_i\}$	6	9	0	0	0	0	0	0
$\{\mathbf{a}_i\}$	18	15	27	21	20	18	33	26
$\{\mathbf{a}_{ci}\}$	9	9	18	15	9	9	18	15
$\{\mathbf{F}_i\}$	3	0	3	0	3	0	3	0
$\{\mathbf{N}_i\}$	6	3	24	18	6	3	24	18
$\{\mathbf{f}_i\}$	0	3	9	9	0	3	9	9
$\{\mathbf{f}^i_{i+1}\}$	9	6	0	0	9	6	0	0
$\{\mathbf{n}_i\}$	21	15	21	24	12	15	21	24
TOTAL	92	76	135	123	59	54	108	92

M= Multiplications; A = Additions
Method I : Present work (Table 2.4)
Method II: Conventional NE algorithm (Table 2.3)

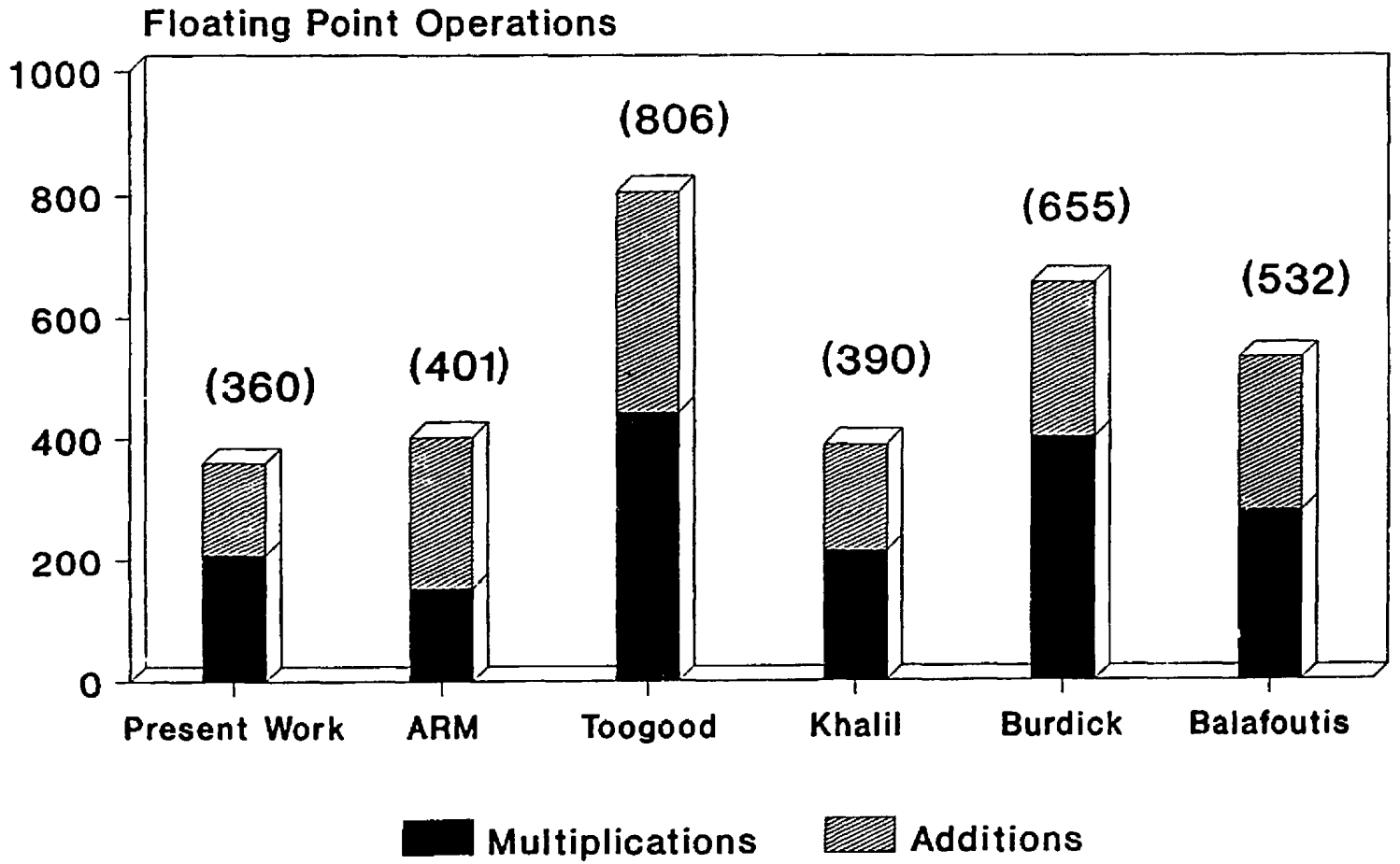
Table 2.10: Implementation of inverse dynamics using symbolic computation

A Comparison of computational counts

ROBOT	Method I		Method II		Method III		Method IV		Method V		Method VI	
	M	A	M	A	M	A	M	A	M	A	M	A
PUMA (3 DOF)	80	55	55	42	114	81	-	-	-	-	-	-
STANFORD (3 DOF)	48	33	40	24	-	-	-	-	-	-	-	-
PUMA (6 DOF)	208	152	152	249	441	365	214	176	401	254	277	255
STANFORD (6 DOF)	183	140	183	147	338	276	187	152	-	-	-	-

M = Multiplications; A = Additions

Method I - Present Work
Method II - Murray and Neuman [1988]
Method III - Toogood, R.W.[1989]
Method IV - Khalil, W. et al. [1986]
Method V - Burdick, J. [1986]
Method VI - Balfoutis, C.A. [1988]



Total flops given in brackets.

Figure 2.9: Comparison of Computations for implementing the inverse dyanmics of PUMA-560 manipulator

The significant reduction in the computations can be attributed to the proper choice of the intermediate variables (ω_i, λ_i, etc.). Due to the efficient symbolic implementation of the algorithm, the CPU time as well as the virtual memory requirements are very low and can be easily carried out on a micro-computer.

2.6 Conclusion

An efficient scheme for dynamic modeling of the robotic manipulators has been developed in this chapter using the λ matrix approach and symbolic programming. Based on the work in this chapter, the following conclusions can be drawn.

1. An efficient scheme for dynamic modeling of the robotic manipulators can be developed from the conventional NE algorithm.
2. Introducing some modifications in the conventional Newton Euler algorithm improves the computational efficiency.
3. This simplified algorithm can be used to derive customized robot dynamic models, using iterative symbolic programming, for real-time control applications.

Chapter 3

Modeling Friction in Inverse Dynamics

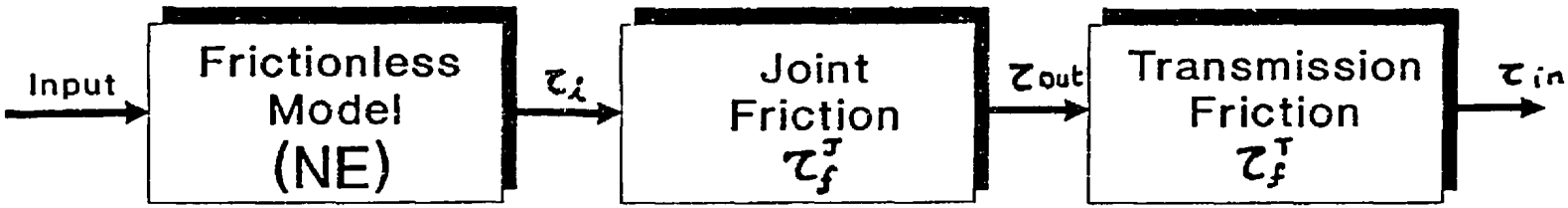
3.1 Introduction

As robotic manipulator systems become increasingly common in industrial applications, accurate manipulator dynamics that govern their operations become essential to ensure control robustness. Much of the published work in computational robot dynamics neglect the frictional effects but in actual task performance they are quite significant. Armstrong's experiments (Armstrong, 1988) reveal that the friction torques can be as high as three times the inertial torques. These effects are significant in robots which operate under high torques, and the errors in the trajectories in such applications can be very large, if friction is not included in the dynamic model. To minimize such errors, not only the model should be accurate but also the computations have to be carried out in real-time. An analytical model for friction in robotic mechanisms is developed in this chapter and a case study of the applicaton of this model to PUMA-560 manipulator is also presented (Dhanaraj and Sharan, 1990).

3.2 Coulomb Friction in Robotic Mechanisms

While viscous friction can be easily modeled as a linear function of relative velocity, Coulomb friction is non-linear and is proportional to the normal forces acting at the contact surface. The laws of Coulomb friction are considered to be valid in bearings (journal and rolling) and also in transmissions (e.g. gearboxes or harmonic drives). Friction at the joints can be expressed as a function of the joint reaction forces and moments at the joints. Friction in transmissions can be conveniently modelled using the input-output graphs of the transmission.

One important point has to be noted here. Given two identical robot manipulators, one operating under frictional conditions and the other considered ideally frictionless, the resultant joint interactions (forces/moments) in corresponding links have to be the same in direction and magnitude for both the manipulators to produce identical motion (Gogoussis, 1988). So, if the kinematic state of the manipulator, (i.e. the position, velocity and acceleration of all the links of the manipulator) is given then the resultant reaction forces/moments at the joints, in the case with friction are equal to the ones in the system without friction. Thus, if the reaction forces for the case of frictionless model are known (which can be computed using the modified NE method), the frictional torques can be computed using the basic Coulomb's law. This can be understood from Fig. 3.1, where τ_i is the dynamic torque required to produce a given set of acceleration, velocity and displacement on a link, when there is no friction. When friction is included, we need to apply two additional torques, one due to the friction at the joints denoted by τ_f^J and another one due to the losses in the transmission denoted by τ_f^T. It should be noted at this point that all these three torques can



τ_i = Basic Dynamic Torque

τ_{out} = Transmission Output Torque

τ_{in} = Transmission Input Torque

Figure 3.1: Frictional Torque in Robotic Mechanisms

take positive or negative values depending upon the direction of the relative motion at the joint and this is explained in the following sections.

3.3 Friction at the Joints

The joint frictional forces arise due to two reasons, one due to the normal reaction forces $\{f\}_i$ at the joints and the second one due to the reaction moments $\{n\}_i$ at the joints. If the reaction moment at a joint is zero, then the frictional force will be of the first kind only and this can be written as a function of the effective normal force F_N at the joint expressed as

$$F_N = \sqrt{f_x^2 + f_y^2} \tag{3.1}$$

where f_x and f_y are the x and y components of the reaction force $\{f\}_i$.

Referring to Fig. 3.2, the friction force f is equal to μ times $\mid F_N \mid$ and the direction of the friction force is opposite to the direction of the relative rotation between the journal and the bearing. In the figure, the relative rotation of the journal is in the clockwise direction and the frictional force acts in such a way to produce a torque in the anticlockwise direction. Hence the frictional moment $(\tau_f)^J$ can be expressed as[1]

$$\begin{aligned} (\tau_f)^J &= fr \\ &= \mu \mid F_N \mid r \end{aligned} \tag{3.2}$$

where r is the journal radius. This would imply that the applied torque has to compensate for this frictional torque in addition to the inertial torque and hence, it will be the sum of these two torques.

The frictional moment arising due to the reaction moments can be understood from Fig. 3.3. Here, a force P is applied at the end effector and its moment

[1] $\mid F_N \mid$ indicates the absolute of F_N.

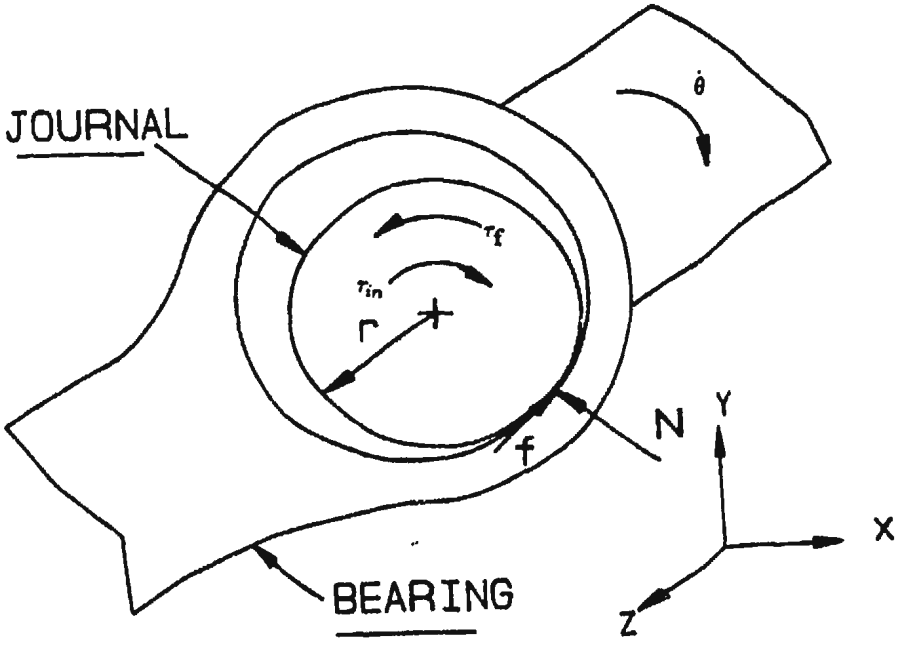


Figure 3.2: Friction in a Journal Bearing

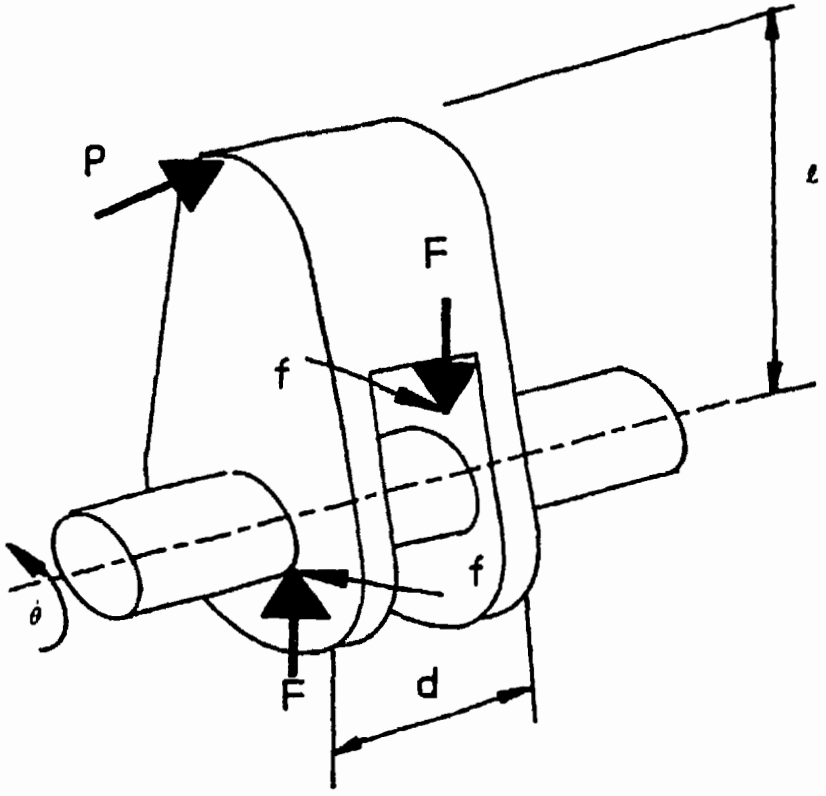


Figure 3.3: Friction due to Reaction Moments

about the x axis of the previous link will be P times l. This must be resisted by a reaction moment at the joint which will be equal to | F | times d, where d is the effective length between the the bearing support points. For a single bearing support, d will be equal to the effective length of the bearing and for a two bearing support, it will be equal to the distance between the support points. There will be frictional forces arising due to this force F which will be given by the expressions similar to Eq. 2. When the force and the moment ($\{f\}_i$ & $\{n\}_i$) act simultaneously at the joints, the frictional torque can be written as [2]

$$\begin{aligned}(\tau_f)^J = \; & sgn(\dot{\theta})\frac{\mu r}{d}[\sqrt{(0.5d\, f_x - n_y)^2 + (0.5d\, f_y - n_x)^2} \\ & +\sqrt{(0.5d\, f_x + n_y)^2 + (0.5d\, f_y + n_x)^2}]\end{aligned} \tag{3.3}$$

where f_x and f_y are the x and y components of the reaction force at the joint and n_x and n_y are the x and y components of the reaction moment at the joint. In case of prismatic joints with a linear bearing, the frictional force is a direct function of the normal forces and can be written as

$$\begin{aligned}(\tau_f)^J = \; & sgn(\dot{\theta})\frac{\mu}{d}[|\, 0.5d\, f_x - n_y \,| + |\, 0.5d\, f_y - n_x \,| \\ & + |\, 0.5d\, f_x + n_y \,| + |\, 0.5d\, f_y + n_x \,|]\end{aligned} \tag{3.4}$$

When a thrust bearing is used, the frictional forces will be a function of the axial force, and hence in such cases, the frictional torque can be written as

$$(\tau_f)^J = sgn(\dot{\theta})\mu r \,|\, f_z \,| \tag{3.5}$$

The applied torque should compensate the frictional torques and hence the total torque will be equal to the applied torque from the frictionless model plus the moment due to the friction force. In Eqs. 3.3 to 3.5, $(\tau_f)^J$ is the frictional

[2] $\mathrm{sgn}(\dot{\theta})$ indicates the sign function.
$\mathrm{sgn}(\dot{\theta}) = +1$ if $\dot{\theta} > 0$
$\mathrm{sgn}(\dot{\theta}) =$ -1 if $\dot{\theta} > 0$

Table 3.1: Friction at the Joints

Type of Bearing	Frictional Torque Equation
1. JOURNAL BEARING	$(\tau_f)^J = sgn(\dot{\theta})\frac{\mu r}{d}\left[\sqrt{(0.5d\, f_x - n_y)^2 + (0.5d\, f_y - n_x)^2} + \sqrt{(0.5d\, f_x + n_y)^2 + (0.5d\, f_y + n_x)^2}\right]$
2. LINEAR BEARING	$(\tau_f)^J = sgn(\dot{\theta})\frac{\mu}{d}[\mid 0.5d\, f_x - n_y \mid + \mid 0.5d\, f_y - n_x \mid + \mid 0.5d\, f_x + n_y \mid + \mid 0.5d\, f_y + n_x \mid]$
3. THRUST BEARING	$(\tau_f)^J = sgn(\dot{\theta})\mu r \mid f_z \mid$

torque required to compensate friction at the joint. The torque required at the joint, or at the output end of the transmission, τ_{out}, can be computed by summing up the dynamic torque and the frictional torque, when the applied torque is in the direction of motion, given as

$$\tau_{out} = \tau_i + \tau_f^J \tag{3.6}$$

When the direction of the dynamic torque is opposite to that of the motion (braking motion), the frictional torque will be aiding the applied torque and hence τ_{out} will be given as

$$\tau_{out} = \tau_i - \tau_f^J \tag{3.7}$$

Table 3.1 summarizes the various equations for computing the frictional torque at the joints.

3.4 Friction in Transmissions

The major source of friction is in the transmission systems, which may comprise of gear drives, belt drives, etc. For a detailed analysis, a commonly used harmonic drive system can chosen. Harmonic drives have been extensively used in industrial robots owing to their high efficiency, low weight and compactness (Dudley, 1956; Chironis, 1967). For trajectory control the robot drives require a wide range of torques, and a precision control is possible only if the friction in the transmissions is also considered in the dynamic model. The efficiency curves of the transmission system (these are generally available from the manufacturers) such as harmonic drives can be used to model the frictional losses in the transmission. The efficiency curve of a typical harmonic drive, shown in Fig. 3.4 is a non-linear curve with high frictional losses at low torque operations leading to very low efficiencies at such regions (Dudley, 1956). In robotic mechanisms, such regions cannot be avoided in trajectory control.

Defining τ_{in} as the torque generated by the motor at the input shaft of the harmonic drive and τ_{out} as the torque available at the output shaft, the frictional torque in the transmission, $\tau_f)^T$ can be written as

$$\tau_f^T = m \times \tau_{in} - \tau_{out} \tag{3.8}$$

where m is the torque amplification ratio, which is given by

$$m = \frac{Input\ \ Speed}{Output\ \ Speed} \tag{3.9}$$

It should be noted that the efficiency for the transmission system can be written

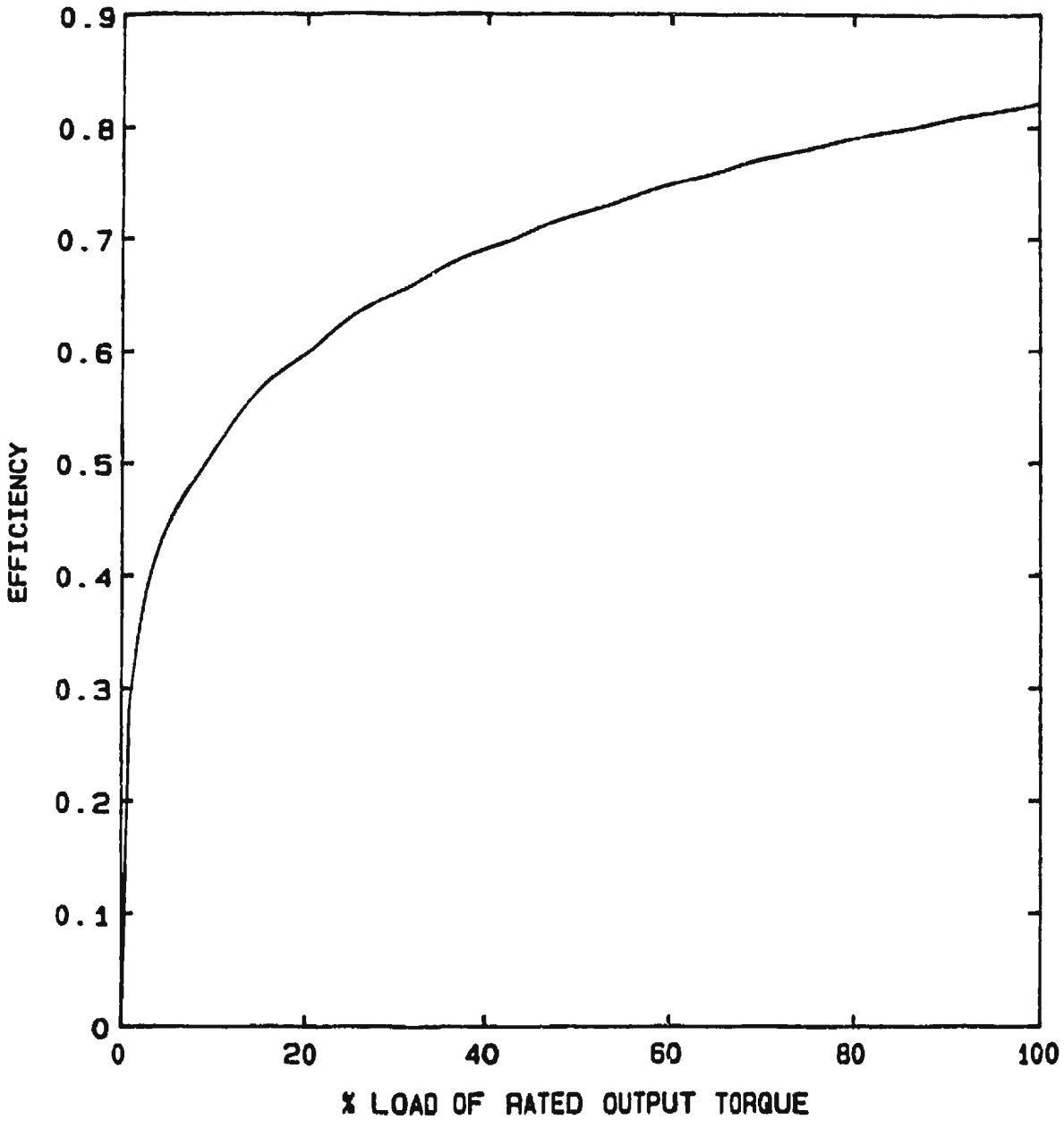


Figure 3.4: Efficiency of a Harmonic Drive

as

$$\begin{aligned}\eta &= \frac{Power\ \ Output}{Power\ \ Input} \\ &= \frac{Output\ \ Torque \times Output\ \ Speed}{Input\ \ Torque \times Input\ \ Speed} \\ &= \frac{\tau_{out}}{\tau_{in}}\frac{1}{m}\end{aligned} \tag{3.10}$$

Due to the high inertia of the transmission systems and the high static friction, the no-load torque or the break-away torque of the drives (τ_B) are normally very high and these can be incorporated in the input-output model as

$$\tau_{in} = \frac{\tau_{out}}{\eta} + \tau_B \tag{3.11}$$

In addition to the factors discussed above, one has to note that when the output torque and the velocity of the shaft are in opposite directions, the frictional torque will be in the same direction as the applied torque and hence the input torque will be less than the output torque. The complete set of equations for the input-output relationships are given in Table 3.2.

Note that in these expressions η is a function of the ratio of the output torque to the rated torque. Fig. 3.5 demonstrates the input-output relationship described by these equations. One should note that 'curve I' corresponds to the positive velocity and 'curve II' corresponds to the negative velocity. Note that at point A ($\tau_{out} > 0$), the required input torque is greater than τ_{out} for positive velocity and is less for negative velocity. In the same manner at point B ($\tau_{out} < 0$), the required input torque is less than τ_{out} for positive velocity and is greater for negative velocity.

The computational count for including the frictional model is summarized in Table 3.3. The efficiency data for the transmission system can be generated using

Table 3.2: Friction at the Transmission

Case 1 :	$\tau_{out} > 0$ $\dot{\theta} > 0$	$\tau_{in} = \frac{\tau_{out}}{\eta} + \tau_B$
Case 2 :	$\tau_{out} < 0$ $\dot{\theta} < 0$	$\tau_{in} = \frac{\tau_{out}}{\eta} - \tau_B$
Case 3 :	$\tau_{out} > 0$ $\dot{\theta} < 0$	$\tau_{in} = \tau_{out}\eta - \tau_B$
Case 4 :	$\tau_{out} < 0$ $\dot{\theta} > 0$	$\tau_{in} = \tau_{out}\eta + \tau_B$

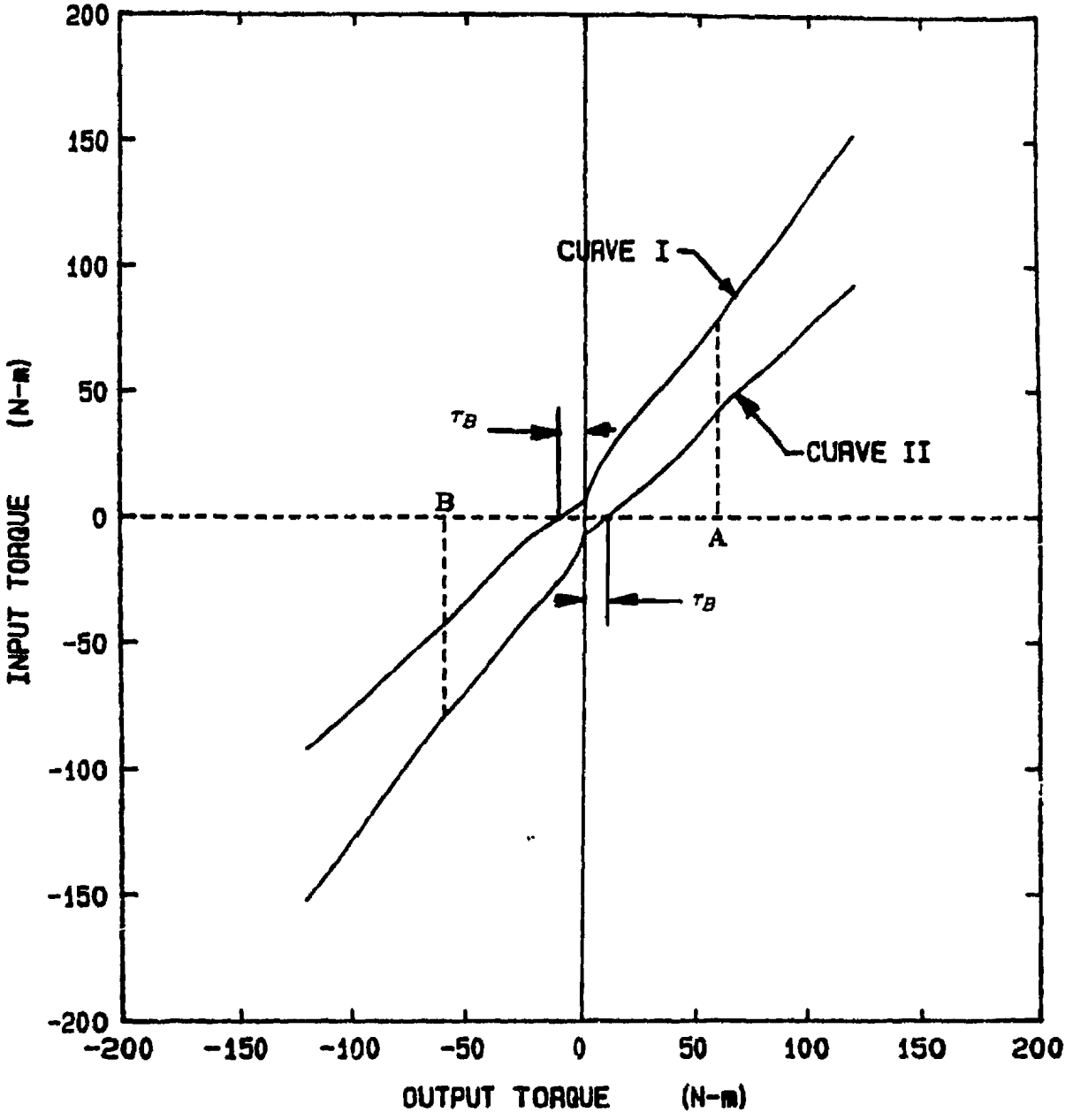


Figure 3.5: Input-Output Curve for a Harmonic Drive

Table 3.3: Computational Count for the Friction Model

Dynamic Model Type		Revolute Joint		Prismatic Joint	
		M	A	M	A
Frictionless Model (Table 2)		92	76	59	54
Friction Model	$(\tau_{f_i})^J$ (Table 5)	15	5	10	7
	$(\tau_{f_i})^T$ (Table 6)	5(2*)	4(2*)	5(2*)	4(2*)
	Total	112(109*)	85(83*)	74(71*)	65(63*)

* - The transmission frictional torque is computed using a cubic spline approximation for the efficiency curve. If a linear interpolation is used then the computation will be only 2 multiplications and 2 additions as shown in figures within the brackets.

either a cubic spline interpolation or a linear interpolation. When anti-friction bearings such as ball or roller bearings are used in the joints, the coefficient of friction tends to be very low and it may be sufficient to consider the frictional torque in the transmission only. It can be seen from the Table 3.3, that if linear interpolation is used for the efficiency data, the additional computational count will be only 24 (15+5+2+2) floating point operations per joint, and if the coefficient of friction at the joint is considerably low (< 0.05), τ_f^J need not be computed and hence the additional load for frictional effects will be only 4 floating point operations.

3.5 Case Study

In order to demonstrate the significance of the frictional torque, the above formulation was used to generate the inverse dynamics problem of a PUMA-560 positioning system. The basic dynamic torque τ and the joint friction torque τ_f^J were computed and after a few steps the required nominal input torque τ_{in} was computed. The end-effector of the robot was moved along a straight line trajectory as shown in Fig. 3.6. The velocity profile in the global co-ordinates is shown in Fig. 3.7. Using the program given in Appendix D, the angular positions, velocity and accelerations of the three links were computed. The angular position and velocity of the links are shown in Figs. 3.8 and 3.9. The motor torques were computed using the procedure as shown in the flow-chart in Fig. 3.10. The parameters used in the friction model are given in Table 3.4.

The applied torque profiles with and without friction are shown in Figs. 3.13 to 3.15. At 0.6 secs for example, the contribution of frictional torque is as shown

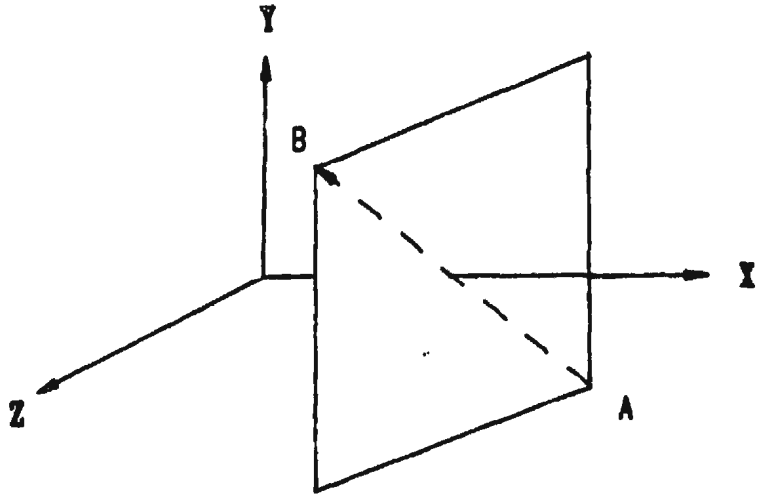


Figure 3.6: Trajectory in the Global Coordinate Frame

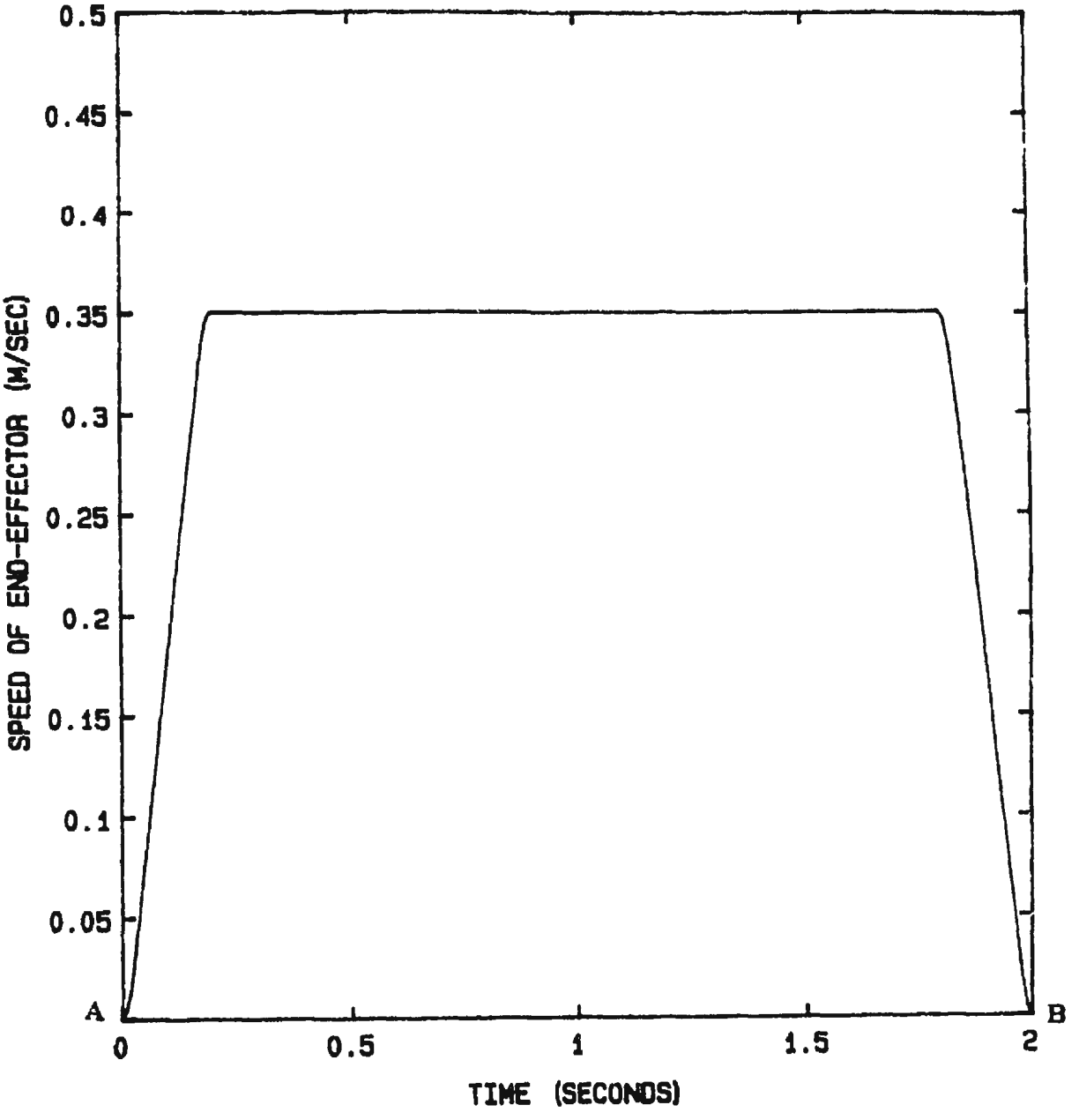


Figure 3.7: Velocity Profile in the Global Coordinate Frame

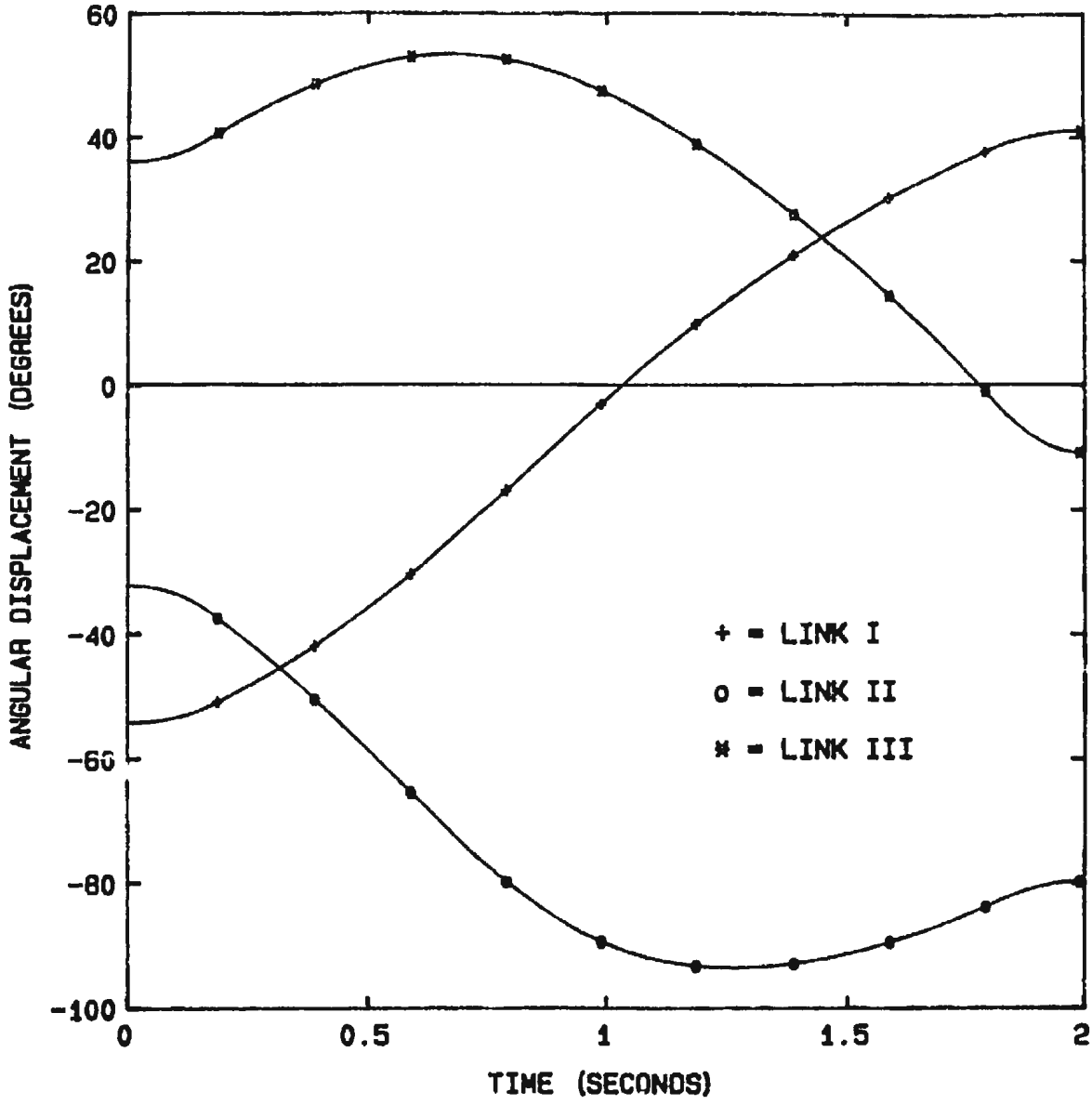


Figure 3.8: Angular Displacements in the Link Coordinate Frame

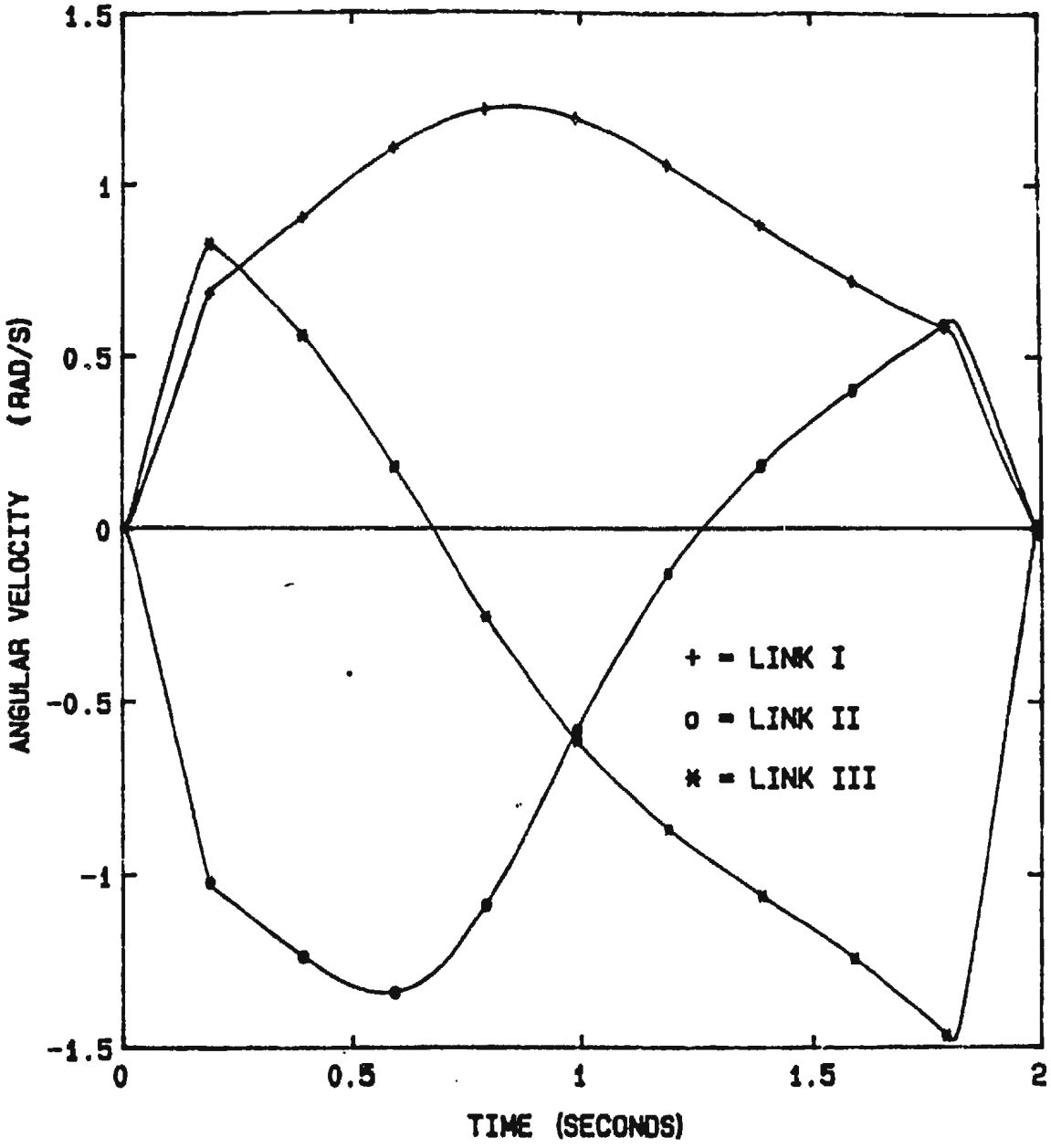


Figure 3.9: Angular Velocities in the Link coordinate Frame

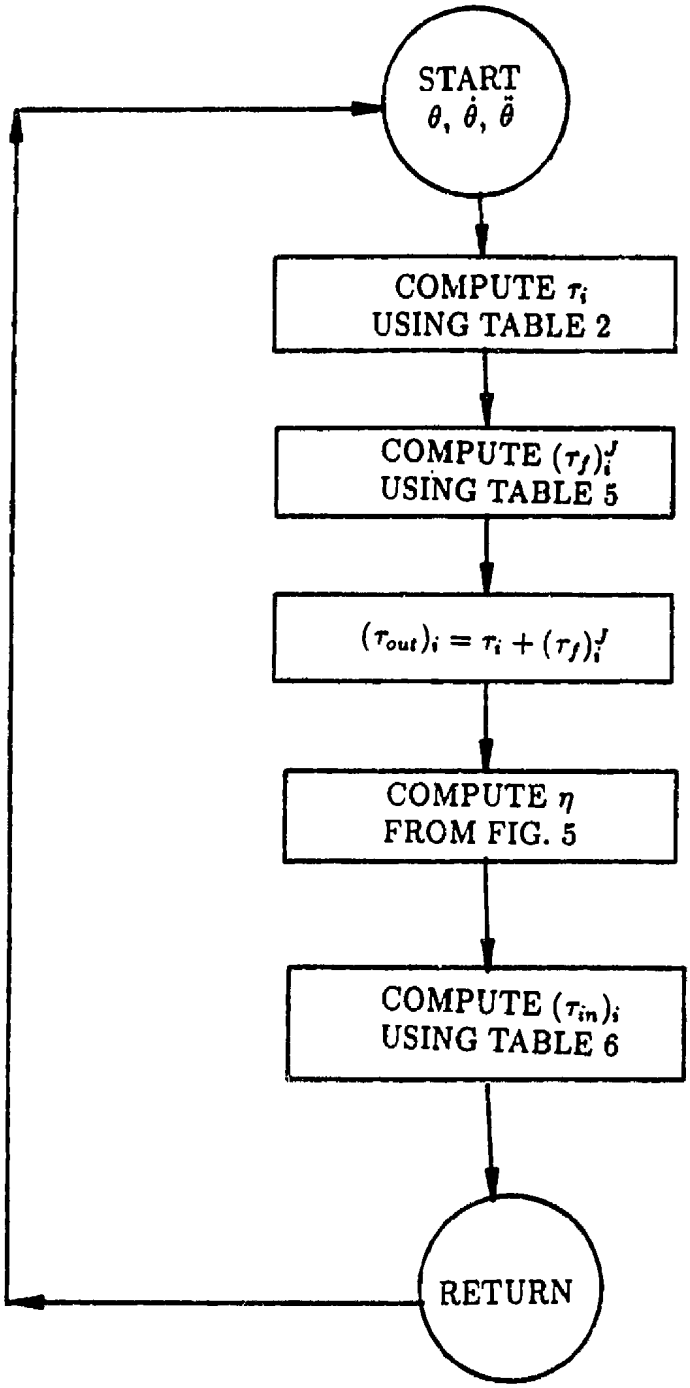


Figure 3.10: Flow-Chart for Computation of Frictional Torque

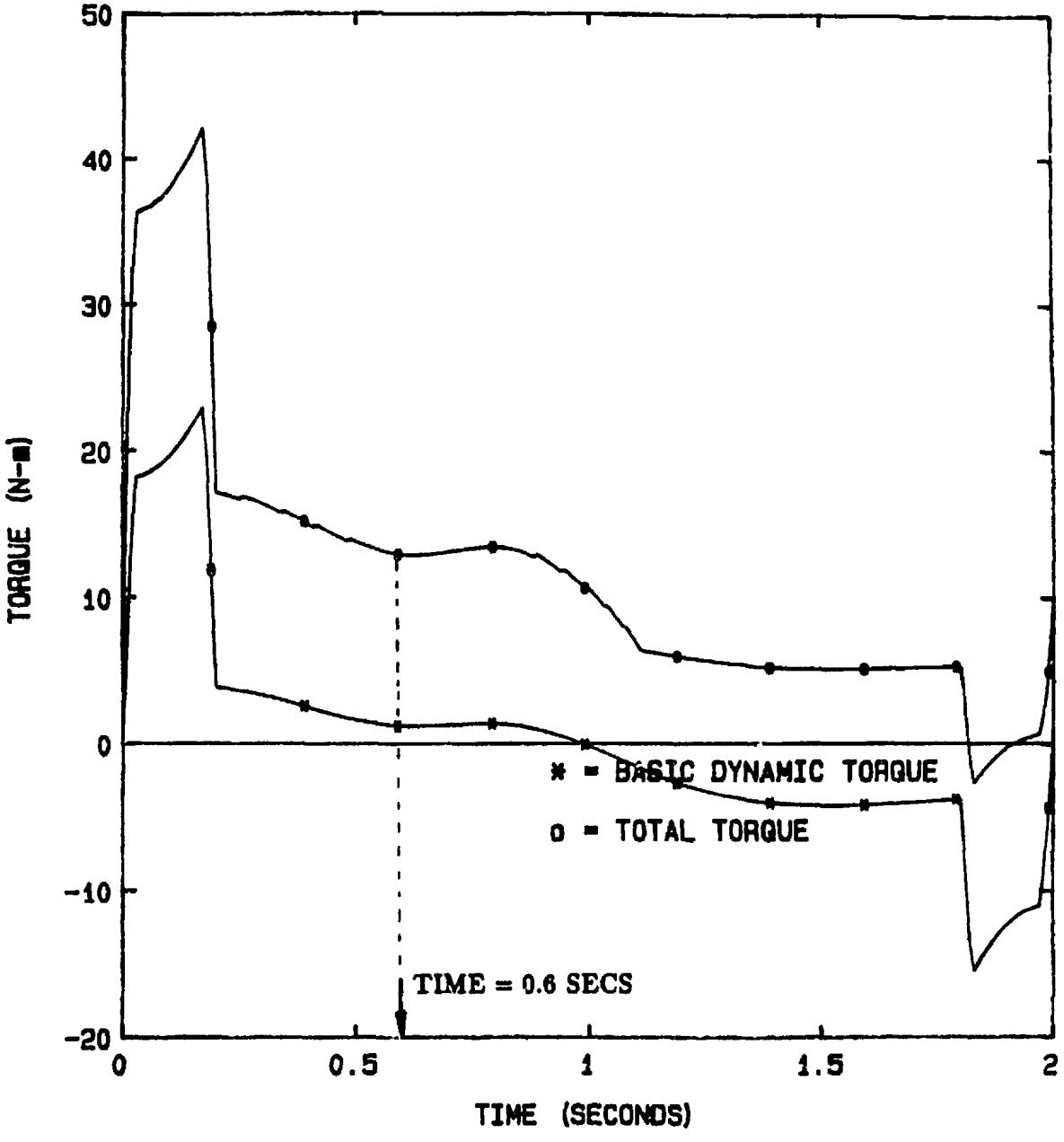


Figure 3.11: Torque Profile for the First Link

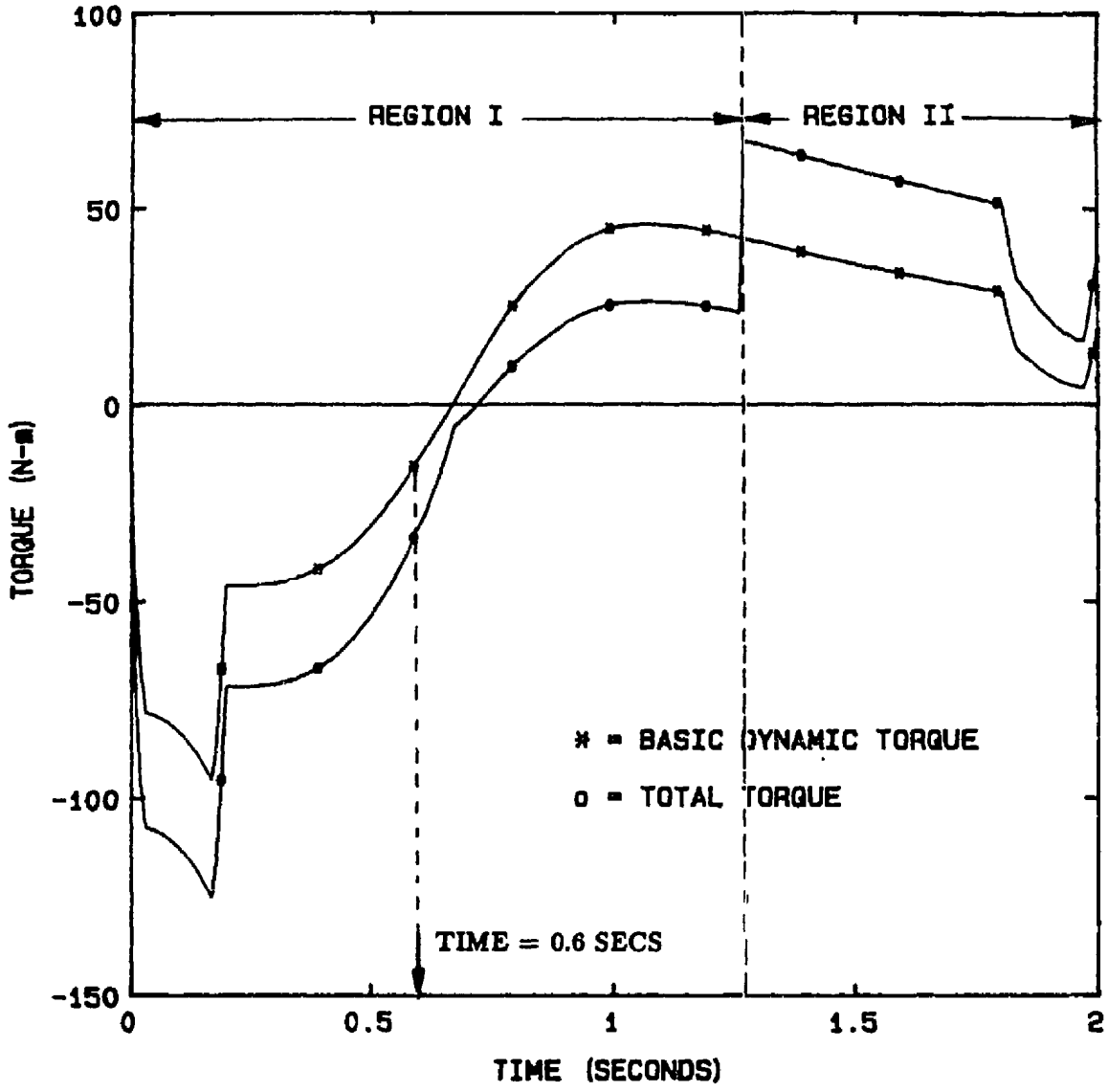


Figure 3.12: Torque Profile for the Second Link

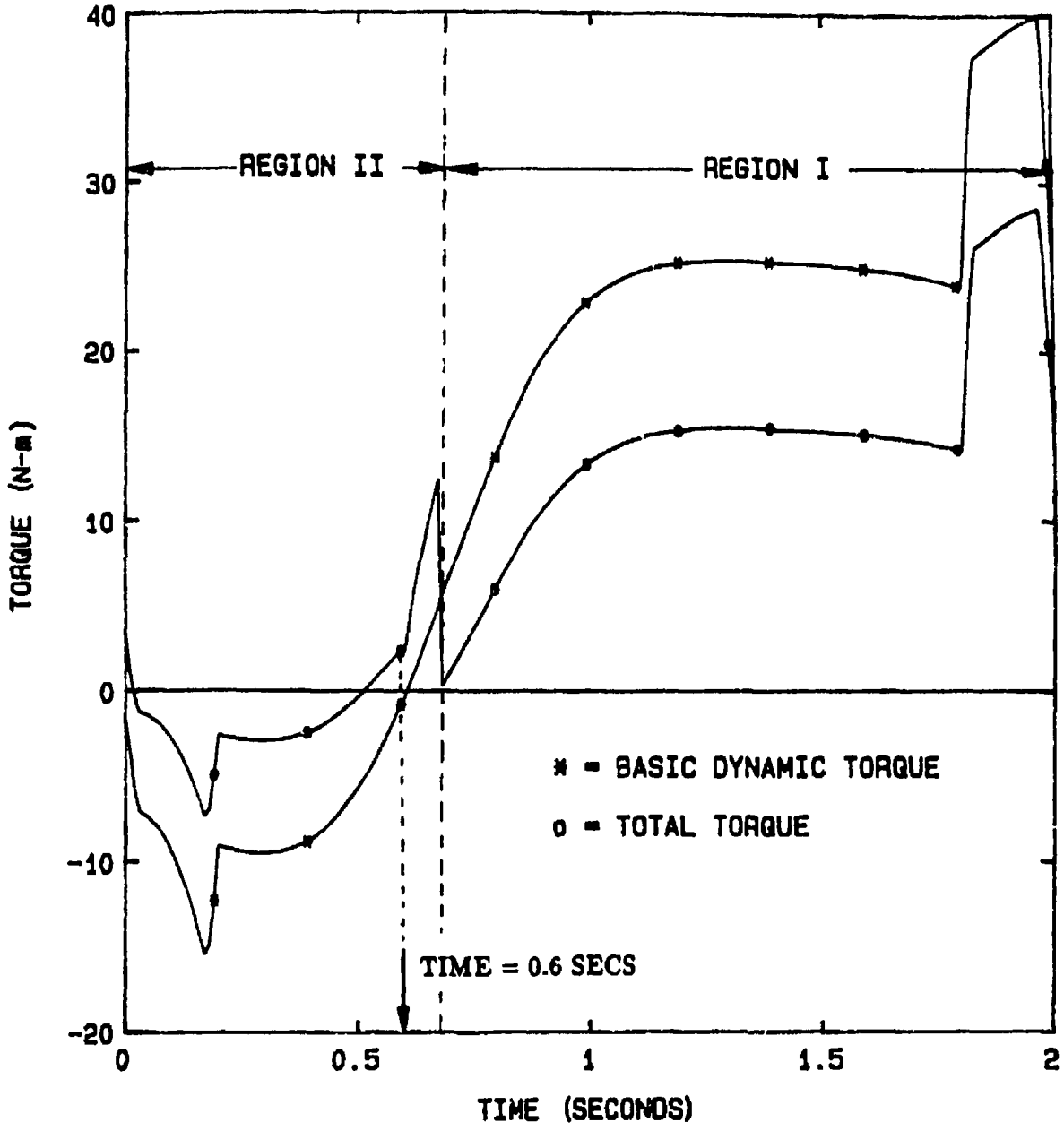


Figure 3.13: Torque Profile for the Third Link

Table 3.4: Friction Parameters for PUMA-560 Manipulator

Link No	Joint Friction		Transmission Friction	
	Friction Coeff. μ	Journal radius m	Break-away[1] Torque (τ_B) N-m	Maximum[1] Torque (τ_{max}) N-m
1	0.1	0.10	6.3	97.6
2	0.1	0.08	5.5	186.4
3	0.1	0.07	2.6	89.4

1 : Values taken from Armstrong, 1988.

in figure. The values of the joint frictional torque and the transmission frictional torque at this instant of time are given in Table 3.5.

It can be seen that the joint frictional torque was much smaller than the transmission frictional torque and this was true for all times. The results in Figs. 3.13 to 3.15 show that the frictional torques are quite significant. An interesting point to note in Figs. 3.14 and 3.15 is that in region I, both $\dot{\theta}$ and τ being in the same direction, the total torque is less than the dynamic torque whereas in region II, the opposite is true.

3.6 Conclusion

An efficient scheme for dynamic modeling of the robotic manipulators including the non-linear frictional effects has been arrived at in this work. Based on the

work in this chapter, the following conclusions can be drawn.

1. Friction is significant in robotic mechanisms and should be included in the dynamic model for better accuracy.

2. An efficient algorithm, for modeling manipulator dynamics including friction can be developed.

3. The frictional effects are present in the joints as well as the transmissions in the robotic manipulators. These frictional effects should be modeled separately for greater accuracy since the frictional effects at the joints are much lower than than those in the transmissions.

4. The computational load to incorporate friction in the dynamic model is only marginally increased, when used along with the modified NE algorithm.

Table 3.5: Torque Values at Time = 0.6 secs

	Basic Dynamic Torque (N-m)	Joint Frictional Torque (N-m)	Total Torque at Joint (N-m)	Transmission Frictional Torque (N-m)	Total Input Torque (N-m)
Link I	1.2140	1.7803	2.9943	9.8968	12.8911
Link II	-15.6364	-0.3281	-15.9646	-17.9472	-33.9117
Link III	-0.8042	0.1241	-0.6801	3.0449	2.3648

Chapter 4

Parallel Processing of Inverse Dynamic Equations

4.1 Introduction

Increasingly robots are designed for high-precision and high-speed applications and these in turn demand a highly sophisticated control mechanism. The dynamics of these manipulators, as discussed in earlier chapters, is highly non-linear and demand a large number of computations for real-time control. Coupled with this, the demand for microprocessor based controllers, capable of attaining a sampling rate of over 1 KHz, has required research necessary for efficient algorithms which can be implemented in parallel architecture. Parallel computers are finding increasing applications, since they offer potential advantages of higher performance, lower cost to performance ratio, increased availability and easy portability of the controller. As discussed in Chapter 1, parallel processing has been a very attractive solution for modeling the inverse dynamics of robotic manipulators in real time control (Binder, 1985; Kasahara and Narita, 1984, 1985, 1988; Lee and Chang, 1988; Chen et al, 1988; Khosla and Ramos, 1988; Luh

and Lin, 1982; Nigam and Lee, 1985; Tonkinson and Donath, 1988; Vukobratovic and Kircanski, 1988). The modified NE Algorithm developed in Chapter 2 can be implemented as a parallel algorithm to achieve a high computational speed. The multiprocessor implementation of this algorithm using a "task streamlining approach" is discussed in this chapter. The task streamlining approach aims at

1. A systematic decomposition of the inverse dynamic problem of a robotic manipulator of arbitrary configuration to a finite number of subtasks of uniform computational load, and
2. A heuristic algorithm for scheduling these subtasks on a multiprocessor consisting of an arbitrary number of processors, thereby maximizing the speed-up as well as the processor utilization.

4.2 Multiprocessor Issues

4.2.1 Classification of Parallel Computers

A typical uniprocessor computer processes all instructions sequentially, one instruction at a time, and hence they are termed as Single-Instruction-Single-Data (SISD) systems. The schematic diagram of the organization of such a computer is shown in Fig. 4.1. The single control unit (CU) governs the instruction stream (IS) which flows to the processing unit (PU) and the data stream (DS) which flows from the memory module (MM) to the PU and vice versa. Parallel computers can process information and data in parallel through multiple PUs and use one or more of the CUs. Parallel computers, in general, can be grouped into two major families: (a) "vector" and (b) "multiprocessor" systems (Hwang and Briggs, 1967; Polychronopoulos, 1988). Vector processors are a set of identical processors which can process different data simultaneously and for this reason,

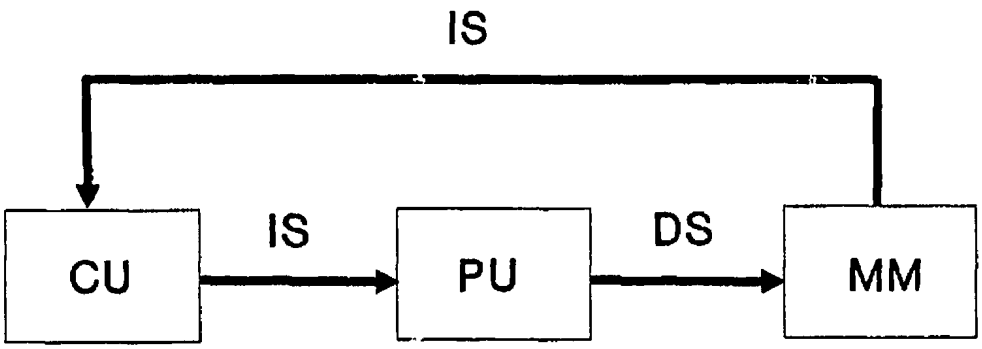


Figure 4.1: SISD Computer Organization

this reason, they are also called as Single-Instruction-Multiple-Data (SIMD) machines. The schematic diagram for such a computer is shown in Fig. 4.2. These vector processors can be further grouped into pipelined and array machines. A pipeline computer performs overlapped computations to exploit temporal parallelism, where component operations may be repeated many times, as in the case of matrix multiplication. Examples include the Cray 1, the CDC Cyber 205, the Fujitsu VP-100/200, the Hitachi S-810, and the Convex-1 computers. Array computers usually come with a number of identical arithmetic logic units (ALU) interconnected in some symmetric structure (e.g., linear array, mesh, ring). An array processor uses multiple synchronized arithmetic logic units to achieve spatial parallelism. Finite element equations and other partial differential equations are best-handled using an array processor (Ducksbury, 1986). Some existing array machines include the Goodyear MPP, ICL DAP, Illiac IV, and the Connection machine.

The "multiprocessor systems" are composed of a set of independent and autonomous processors that are fully or partially interconnected in some way. Multiprocessors can be synchronous or asynchronous where each processor is driven by its own clock. These can execute different instructions on different data and for this reason they are also called Multiple-Instruction-Multiple-Data (MIMD) systems. The organization for such a system is shown schematically in Fig. 4.3. Two major subfamilies of multiprocessor computers are the shared memory systems and message passing systems. In the former organization, all processors share the same memory address space, and are connected to a shared physical memory through a high bandwidth bus or a multistage interconnection

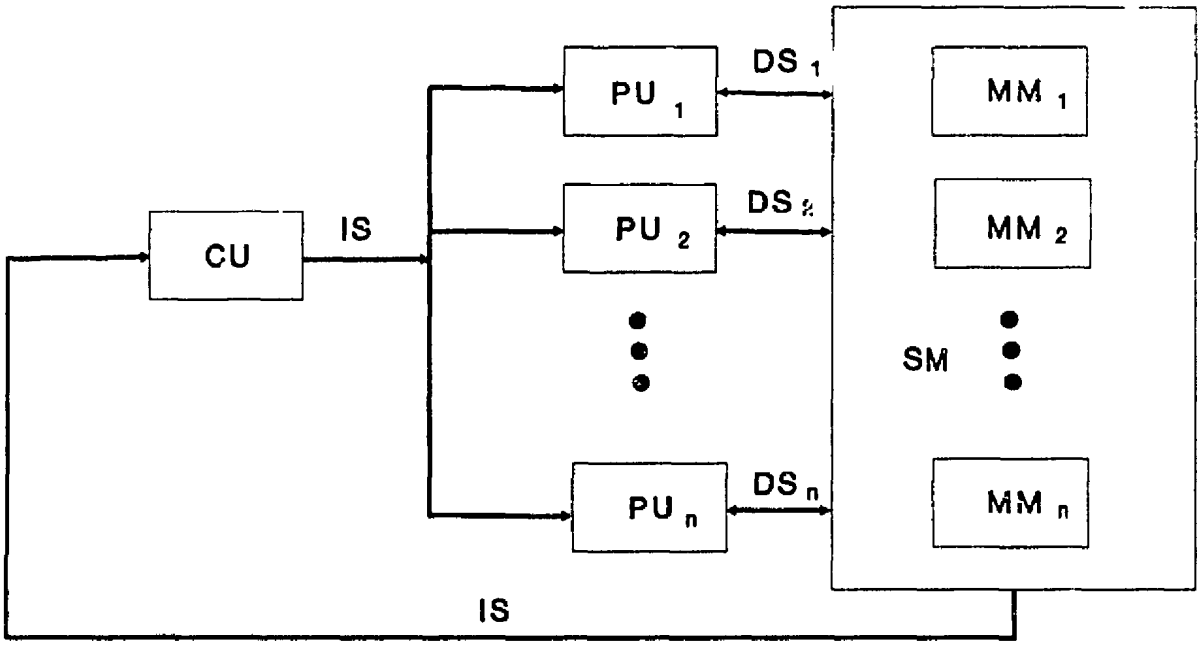


Figure 4.2: SIMD Computer Organization

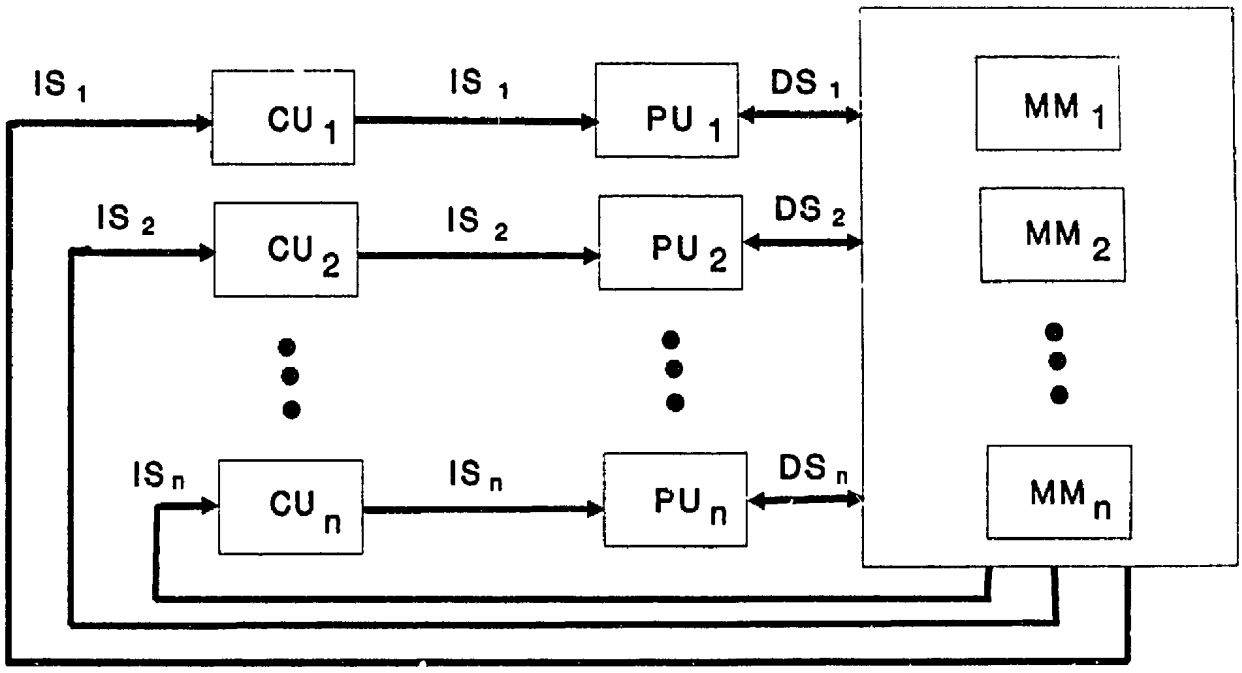


Figure 4.3: MIMD Computer Organization

network. Communication between processors is accomplished through the shared memory and hence they are also termed as 'tightly coupled processors'. Examples of shared memory systems include the Cray X-MP, Cray 2, ETA-10, Alliant FX/8, IBM 3090 and Sequent. In the message passing organization, each processor has its own private memory and there is no physical shared memory and hence they are also termed as loosely coupled systems or distributed systems. Processors communicate asynchronously using message passing mechanisms. The Intel hypercube, the Caltech cosmic cube, and the N-cube/10 are examples of message passing multiprocessors.

For implementation of the robot dynamics, it has been found that a shared memory multiprocessor would be an ideal configuration (Ramos, 1988). Researchers have taken two strategies - one using special purpose architecture (Chen et al, 1988; Lathrop, 1985) and the other using general purpose architecture (Kasahara and Narita, 1988). For the purpose of this work, a general purpose, shared memory multiprocessor as shown in Fig. 4.4 is considered.

4.2.2 Exploiting Parallelism in Algorithms, Synchronization and Uniformity of Subtasks

In most cases, parallelism is not explicit in computational algorithms which are designed for sequential execution and hence an algorithm which requires execution on a multiprocessor system must be decomposed into a set of processes or tasks to exploit the parallelism. Here, either a fine-grained approach or the coarse-grained approach can be taken. Calculations involving a number of nearly independent but communicating calculations such as Monte Carlo simulations, or database management systems, can be executed in parallel. This type of

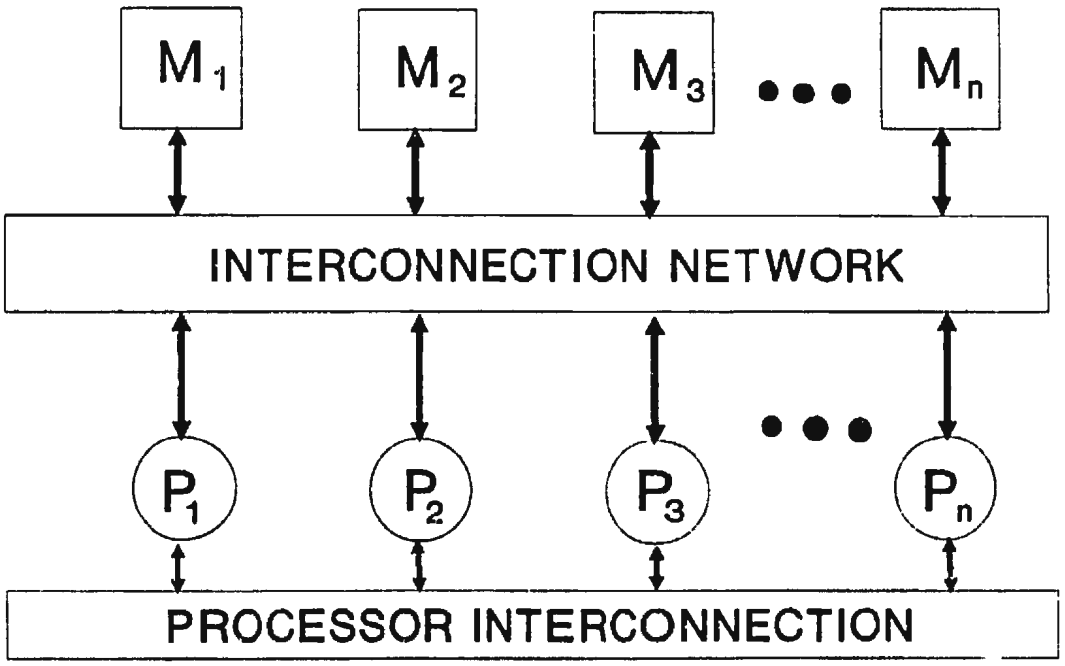


Figure 4.4: A Shared Memory Multiprocessor

parallelism that involves nearly independent tasks is termed as coarse-grained parallelism. On the other hand, if a normally indivisible calculation is partitioned among processors, this would be termed as fine-grained parallelism. An example of this would be computing different iterations of a 'do' loop in a program. Fine-grained parallel programming is generally more difficult to do than coarse-grained parallel programming, although both types depend on exactly the same principles. The fine-grained approach generally requires intensive scheduling strategy and also a high degree of interprocessor communication.

Efficient algorithms for solving the problems of arithmetic complexity are frequently based on a technique known as recursion. Recursion is an important algorithm design technique. It is a method of solving a problem by dividing it into a small number of smaller subproblems of the same type as the original problem. The subproblems are divided in the same way. Eventually the subproblems become small enough to be solved directly. The solution to the smaller subproblems are then combined to give solutions to the bigger subproblems, until the solution to the original problem is computed. A useful rule for recursively partitioning the problem is to create subproblems of approximately equal size; to be able to partition the job into subtasks of equal size is the most challenging work.

One of the critical issues for multiprocessor systems is synchronization, which is a fundamental problem with cooperating processes in a multiprocessor environment. The computation process in the case of a multiprocessor requires interprocessor communication. The parameters which are computed in one processor, say A, may be required for a subsequent computation in another processor,

say B. In such a case, the processor A after computation of these parameters should send them to the processor B, and the processor B should start the relevant computation only after receiving the parameter sent by the processor A. A synchronization mechanism is used to delay execution of a process in order to satisfy such data dependency constraints. Various synchronization mechanisms can be used, depending on the hardware (Oleinick, 1982). It has been noted that for a tightly coupled systems, a large variation in the size of the tasks composing the multiprocessor process requires significantly large synchronization time, which will slow down the overall performance (Kronsjo, 1985). Therefore, a zero variation, or tasks of uniform computational load, will be a very good choice for easy synchronization.

4.2.3 Multiprocessor Scheduling

For efficient implementation of an algorithm in a parallel computer, the tasks have to be scheduled on a finite number of processors to ensure maximum speed-up as well as high processor utilization. The speed-up is the ratio of the execution time on a uniprocessor over a parallel processor and the processor utilization indicates the idle conditions of the processor. There are many approaches to program scheduling. Most can be classified as static or dynamic. In static schemes, scheduling is done before program execution based on knowledge of global program information. The advantage of static scheduling is that the run-time overhead with respect to scheduling is minimal. Dynamic schemes are typically based on local information about the program. Scheduling decisions are made at run-time which incur a penalty or overhead. In other words, in

the static scheduling, the sequence of operations is decided beforehand whereas in the dynamic scheduling, decisions are made during the execution of the job. This overhead is the main disadvantage of dynamic scheduling. Considering the nature of the robotic problems, static scheduling has been found to be an ideal strategy (Kasahara and Narita, 1985).

A key issue in the study of processor scheduling is the amount of overhead or computation time needed to locate a suitable schedule. A scheduling algorithm is a procedure that produces a schedule for every given set of processes. An efficient scheduling algorithm is one that can locate a suitable schedule in an amount of time that is bounded in the length of the input by some polynomial. Construction of *optimal* schedules is NP-complete (NP stands for Non-Polynomial) in many cases[1]. NP-complete implies that an optimal solution may be very difficult to compute in the worst possible input case. However, construction of *suitable* schedules, that is, computing a reasonable answer making use of some heuristics for the typical input case, is not NP-complete. Therefore, suitable schedules can be obtained for concurrent processes. Various search schemes can be utilized to arrive at a sub-optimal schedule for the required multiprocessor configuration. Three characteristics of multiprocessor scheduling of the robot dynamics problem should be noted at this stage, namely

1. It is a deterministic problem (Coffman, 1975) since the tasks as well as the resources can be defined before solving the problem.

2. It is a non-preemptive scheduling problem (Coffman, 1975). With this restriction a task cannot be interrupted once it has begun execution; that is, it must be allowed to run to completion. In general, preemptive scheduling

[1] For an excellent background on the computational complexity of algorithms the reader is referred to Kronsjo, 1985 and Coffman, 1975.

permits a task to be interrupted and removed from the processor under the assumption that it will eventually receive all its required execution time, and there is no loss of execution time due to preemptions.

3. It falls under the category of list scheduling (Coffman, 1975). The sequence by which tasks are assigned to processors is then decided by a repeated scan of the list. The scheduling is done off-line.

Two principal measures of schedule performance are the *program speed-up*, S_p, defined as

$$S_p = \frac{T_1}{T_p} \tag{4.1}$$

and the *efficiency*, E_p, or processor utilization rate for a given program and a given number of processors, which is defined as

$$E_p = \frac{S_p}{p} \tag{4.2}$$

where p is the number of processors employed, T_1 is the the execution time in a uniprocessor machine and T_p is the execution time in a parallel processor with p processors. While we try to optimize on both these measures, the speed-up may be the dominant criteria for deciding the number of processors to be employed.

4.3 Task Streamlining Approach

As described in the previous section, in order to process the equations in parallel, the overall computational task should be split into a finite number of subtasks. Also, to achieve a high level of synchronization, the variation in the size of these tasks must be minimal. The Task Streamlining Approach, developed in this work aims at decomposing the inverse dynamic problem into a set of uniform subtasks and ordering them into a layered task graph and scheduling these subtasks on to the available processors. These steps are described in the following sections.

4.3.1 Task Decomposition Scheme

A common approach to the task decomposition of the inverse dynamic problem of manipulators has been one that was initiated by Luh and Lin (1982). The torques/forces at the joints were computed by using the set of nine equations of the modified NE algorithm given in Table 2.3. However, the level of computational complexity of these equations vary largely, from 0 to 24 flops, [3] (Luh and Lin, 1982) and hence it is necessary to split the larger tasks into subtasks to achieve better synchronization. From a careful analysis of these equations, which are mostly in vectorial form, it is to be noted that there are a number of explicit subtasks, requiring three floating point operations. In developing a parallel algorithm, no distinction need to be made between the computational load of a multiplication and that of an addition, since recent processors have almost the same execution time for both these operations (Liu and Chen, 1986). In the present work, the modified NE algorithm is taken as the base algorithm and each equation in the algorithm is analyzed and subdivided into subtasks of three floating point operations in an explicit way, so that these subtasks can be generated automatically using symbolic programming. The subtasks for the revolute joint is developed first and modifications are made subsequently to these subtasks for applying to a prismatic joint. The modified Newton-Euler algorithm, given in Table 2.4 is analyzed step by step and each step is decomposed into several subtasks of 3 floating point operations as shown in Table 4.1. The details of this decomposition process are given below.

Step 1 : The angular velocity is computed by the first equation in the algorithm,

[3] 'flops' is used to indicate the floating point arithmetic operations, such as multiplications and additions

which has the form

$$\{\omega\}_i = [R]_i^T \{\omega\}_{i-1} + \{z\}\dot{q}_i$$

This involves a matrix multiplication with a vector followed by vector addition. It should be noted that the rotation transformation matrix, $[R]_i$, is sparse and its structure is generally one of the two forms given below. If joint i is parallel to joint i-1, then

$$[R]_i = \begin{bmatrix} c_i & s_i & 0 \\ -s_i & c_i & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

If joint i is perpendicular to joint i-1, then

$$[R]_i = \begin{bmatrix} c_i & s_i & 0 \\ 0 & 0 & 1 \\ -s_i & c_i & 0 \end{bmatrix} \tag{4.4}$$

When joint i is parallel to joint i-1, then this multiplication results in equations as below

$$[R]_i\{\omega\}_{i-1} = \begin{Bmatrix} c_i * \omega_{i-1x} + s_i * \omega_{i-1y} \\ -s_i * \omega_{i-1x} + c_i * \omega_{i-1y} \\ \omega_{i-1z} \end{Bmatrix} \tag{4.5}$$

As we can see in these equations, the multiplication of the [R] matrix with the $\{\omega\}$ vector results in two expressions (ω_{ix} and ω_{iy}) each requiring three floating point operations (2 multiplications and 1 addition) and the third expression does not require any computation. When the axes are perpendicular, the expression for ω_{ix} and ω_{iz} will require three flops and the expression for ω_{iy} will require no computations. In either case the multiplication of the [R] matrix can be conveniently arranged in two subtasks. The addition of the relative velocity vector $\{\hat{z}\}\dot{q}$ affects only the ω_{iz} term, which requires one floating point operation (addition) which can be assigned to another independent subtask. It should also be noted that the addition of this vector to the result of previous multiplication

will affect only the z component since the relative velocity is along the z axis of the link frame. In essence, Step 1 can be computed through three subtasks, T_1, T_2 and T_3 as shown in Table 4.1.

Step 2 : The angular acceleration of the link is computed using the equation in Step 2 of the algorithm, which is written as

$$\{\alpha\}_i = [R]_i^T \{\alpha\}_{i-1} + \{z\}\ddot{q}_i + \dot{q}_i \begin{Bmatrix} \omega_{iy} \\ -\omega_{ix} \\ 0 \end{Bmatrix}$$

Here, we can define two intermediate parameters, $\{\alpha'\}$ and $\{c\}$ which can be written as

$$\{\alpha'\} = [R]_i^T \{\alpha\}_{i-1} \tag{4.6}$$

and

$$\begin{aligned} \{c\} &= \{z\}\ddot{q}_i + \dot{q}_i \begin{Bmatrix} \omega_{iy} \\ -\omega_{ix} \\ 0 \end{Bmatrix} \\ &= \begin{Bmatrix} \dot{q}_i\omega_{iy} \\ -\dot{q}_i\omega_{ix} \\ \ddot{q}_i \end{Bmatrix} \end{aligned} \tag{4.7}$$

It should be noted that $\{\alpha'\}$ involves multiplication of the rotational transformation matrix with a vector and as discussed earlier can be done in two subtasks. The parameter $\{c\}$ requires two flops and hence can be done with T_3 which has one flop computation from Step 1. Once the intermediate parameters, $\{\alpha'\}$ and $\{c\}$ are computed they can be added vectorially to yield $\{\alpha\}$, which again can form a subtask of three flops. Hence, Step 2 can be computed through the subtasks, T_3, T_4, T_5 and T_6 as shown in Table 4.1.

Step No	Subtask	Computations	Preceding Tasks
1	T_1, T_2 [a]	$\begin{Bmatrix} \omega_{ix} \\ \omega_{iy} \\ \omega'_{iz} \end{Bmatrix} = [R]_i^T \{\omega\}_{i-1}$	T_1^p, T_2^p, T_3^p [b]
2	T_3	$\omega_{iz} = \omega'_{iz} + \dot{q}_i$ $\{c\} = \begin{Bmatrix} \dot{q}_i\omega_{iy} \\ -\dot{q}_i\omega_{ix} \\ \ddot{q}_i \end{Bmatrix}$	T_1,T_2
	T_4, T_5 [a]	$\{\alpha'\} = [R]_i^T \{\alpha\}_{i-1}$	T_6^p [b]
	T_6	$\{\alpha\}_i = \{\alpha'\} + \{c\}$	T_3, T_4, T_5
2A	T_7	$k_x = \omega_y\omega_z$ $k_y = \omega_z\omega_x$ $k_z = \omega_x\omega_y$	T_1, T_2, T_3
	T_8	$sw_x = \omega_x^2$; $sw_y = \omega_y^2$; $sw_z = \omega_z^2$	T_1, T_2, T_3
	T_9	$\lambda_{11} = -(sw_y + sw_z)$ $\lambda_{22} = -(sw_x + sw_z)$ $\lambda_{33} = -(sw_x + sw_y)$	T_8

Table 4.1: Decomposition of Inverse Dynamic Tasks

[a] : The two subtasks indicate the two components of the vector each of which requires three flops. The third component does not require any computation as explained in the text.

[b] : Superscript 'p' indicates the subtasks of the previous link in the chain, which are required in the forward recursion. For example, the forward recursion of link 3 will require the subtasks corresponding to numbers 1,2,3 and 6 of link 2.

Step No	Subtask	Computations	Preceding Tasks
2A	T_{10}	$\lambda_{21} = k_x + \alpha_x$ $\lambda_{13} = k_y + \alpha_y$ $\lambda_{32} = k_z + \alpha_z$	T_6, T_7
	T_{11}	$\lambda_{23} = k_x - \alpha_x$ $\lambda_{31} = k_y - \alpha_y$ $\lambda_{12} = k_z - \alpha_z$	T_6, T_7
3	T_{12}, T_{13}, T_{14} [c]	$\{a'\} = [\lambda]_{i-1} . \{p\}_{i-1}$ (only multiplications)	T^p_9, T^p_{10}, T^p_{11}
	T_{15}, T_{16}, T_{17} [c]	$\{a''\} = \{a\}_{i-1} + \{a'\}$	T^p_{18}, T^p_{19}, T_{12}, T_{13}, T_{14}
	T_{18}, T_{19}	$\{a\}_i = [R]^T_i \{a''\}$	T_{15}, T_{16}, T_{17}
4	T_{20}, T_{21}, T_{22} [c]	$\{a'\}_{ci} = [\lambda]_i . \{s\}_i$ (only multiplications)	T_9, T_{10}, T_{11}
	T_{23}, T_{24}, T_{25} [c]	$\{a\}_{ci} = \{a\}_i + \{a\}'_{ci}$	T_{18}, T_{19}, T_{20}, T_{21}, T_{22}

Table 4.1: Decomposition of Inverse Dynamic Tasks (Contd.)

2

[2c] :The three subtasks correspond to the x,y,z components of the corresponding vector, each of which requires three flops.

Step No	Subtask	Computations	Preceding Tasks
5	T_{26}	$\{F\}_i = m_i\{a\}_{ci}$	T_{23}, T_{24}, T_{25}
6	T_{27}, T_{28}, T_{29}	$\{N\}_i = [I]_i\{\alpha\}_i + \left\{ \begin{array}{c} D_x.k_x \\ D_y.k_y \\ D_z.k_z \end{array} \right\}$	T_6, T_7
7	T_{30}, T_{31}	$\{f\}^i_{i+1} = [R]_{i+1}\{f\}_{i+1}$	T^s_{32} [d]
	T_{32}	$\{f\}_i = \{F\}_i + \{f\}^i_{i+1}$	T_{26}, T_{30}, T_{31}
8	T_{33}, T_{34}, T_{35}	$\{n'\}_i = \{p\}_i \times \{f\}^i_{i+1}$	T_{30}, T_{31}
	T_{36}, T_{37}, T_{38}	$\{n''\}_i = \{s\}_i \times \{F\}_i$	T_{26}
	T_{39}, T_{40}	$\{n'''\}_i = [R]_{i+1}\{n\}_{i+1}$	T^s_{41}, T^s_{42}, T^s_{43} [d]
	T_{41}, T_{42}, T_{43}	$\{n\}_i = \{N\}_i + \{n'\}_i + \{n''\}_i + \{n'''\}_i$	T_{27}, T_{28}, T_{29}, T_{40} to T_{40}

Table 4.1: Decomposition of Inverse Dynamic Tasks (Contd..)

3

[3d] : Superscript 's' indicates the tasks of the succeeding link in the chain which are required in the backward recursion. For example, while computing the backward recursion tasks of link 3, the subtasks corresponding to numbers, 32,41,42 and 43 of link 4 will be required.

Step 2A : The modified NE algorithm uses the $[\lambda]$ matrix to compute the acceleration difference vector, where

$$[\lambda]_i = \begin{bmatrix} -(\omega_y^2+\omega_z^2) & k_z-\alpha_z & k_y+\alpha_y \\ k_z+\alpha_z & -(\omega_z^2+\omega_x^2) & k_x-\alpha_x \\ k_y-\alpha_y & k_x+\alpha_x & -(\omega_x^2+\omega_y^2) \end{bmatrix}$$

where k_x, k_y, and k_z are given as

$$\begin{aligned} k_x &= \omega_y\omega_z \\ k_y &= \omega_z\omega_x \\ k_z &= \omega_x\omega_y \end{aligned}$$

The computation of the $[\lambda]$ matrix requires

1. product terms, k_x, k_y, and k_z
2. square terms, ω_x^2, ω_y^2 and ω_z^2
3. sum of square terms, $\omega_y^2+\omega_z^2$, $\omega_z^2+\omega_x^2$ and $\omega_x^2+\omega_y^2$
4. sum of $\{k\}$ and $\{\alpha\}$
5. difference of $\{k\}$ and $\{\alpha\}$

Each of the above tasks can be assigned as a subtask, requiring three flops each. Thus the matrix $[\lambda]$ can be computed using subtasks T_7, T_8, T_9, T_{10} and T_{11}.

Step 3 : The linear acceleration of the origin of the link co-ordinate frame is computed using the equation in Step 3, given as

$$\{a\}_i = [R]_i^T(\{a\}_{i-1} + [\lambda]_{i-1}.\{p\}_{i-1})$$

Here again, we can define intermediate parameters {a'} and {a"} so that

$$\{a'\} = [\lambda]_{i-1}.\{p\}_{i-1} \tag{4.8}$$

$$\{a''\} = \{a\}_{i-1} + \{a'\} \tag{4.9}$$

and

$$\{a\}_i = [R]_i^T \{a''\} \tag{4.10}$$

To compute the parameter {a'}, three multiplications and two additions are required for each component of the vector. The multiplications of each component can be assigned to a subtask and the two additions along with one addition required for {a"} are assigned to another subtask, such that Eqs. (4.8) and (4.9) can computed in six subtasks altogether. Eq. (4.10) requires the multiplication of the rotation transformation matrix $[R]^T$ with the vector {a"} and as discussed earlier this can be assigned to two subtasks. Thus, Step 3 can be computed in eight subtasks, namely, T_{12}, T_{13}, T_{14}, T_{15}, T_{16}, T_{17}, T_{18} and T_{19} as given in Table 4.1.

Step 4 : The linear acceleration of the CG of the link is computed in Step 4 as

$$\{a\}_{ci} = \{a\}_i + [\lambda]_i.\{s\}_i$$

Using arguments similar to those given for Step 3, we can split the task into six sub-tasks, namely, T_{20}, T_{21}, T_{22}, T_{23}, T_{24} and T_{25} as given in Table 4.1.

Step 5 : The forward recursion ends here with the computation of all velocities and accelerations. Steps 5 to 9 compute the backward recursion where the forces and moments are computed. The inertial force is computed in Step 5 as

$$\{F\}_i = m_i\{a\}_{ci}$$

which involves three multiplications and hence can be assigned as a subtask, T_{26}.

Step 6 : The inertial moment can be computed using the equation

$$\{N\}_i = [I]_i\{\alpha\}_i + \begin{Bmatrix} D_x.k_x \\ D_y.k_y \\ D_z.k_z \end{Bmatrix}$$

where the inertial constants are defined as

$$D_x = I_{zz} - I_{yy}$$
$$D_y = I_{xx} - I_{zz}$$
$$D_z = I_{yy} - I_{xx}$$

As discussed in Chapter 3, the inertia tensor is a diagonal matrix and results in equations as below

$$N_x = I_{xx}\alpha_x + D_x k_x \tag{4.11}$$

$$N_y = I_{yy}\alpha_y + D_y k_y \tag{4.12}$$

$$N_z = I_{zz}\alpha_z + D_z k_z \tag{4.13}$$

Hence this step can be computed in nine flops or in three subtasks, namely, T_{27}, T_{28} and T_{29}, assigning each equation to one subtask.

Step 7 : Having computed the inertial force, the joint force can be computed as

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

Here, the force at the previous joint, referred in the local co-ordinate can be treated as an intermediate variable as

$$\{f\}^i_{i+1} = [R]_{i+1}\{f\}_{i+1} \tag{4.14}$$

This requires the multiplication of the rotation transformation vector with a vector, and as discussed earlier, this can be done in two subtasks, namely, T_{30} and T_{31}. Subsequently, the vector addition can be treated as a subtask, T_{32}. Thus the joint forces at any joint can be computed in three subtasks.

Step 8 : The joint moment is computed in Step 8, using the equation

$$\{n\}_i = [R]_{i+1}\{n\}_{i+1} + \{N\}_i + \{s\}_i \times \{F\}_i + \{p\}_i \times \{f\}^i_{i+1}$$

At this stage, three intermediate parameters can be defined, namely

$$\{n'\}_i = \{p\}_i \times \{f\}_{i+1}^i \tag{4.15}$$

$$\{n''\}_i = \{s\}_i \times \{F\}_i \tag{4.16}$$

$$\{n'''\}_i = [R]_{i+1}\{n\}_{i+1} \tag{4.17}$$

All these three variables are vector variables which will have three components each. The first two variables, defined by Eqs. (4.15) and (4.16) are the results of cross-products which require 3 flops for each component of the vector. The third variable, defined by Eq. (4.17), is the result of the product of the rotation transformation matrix and a vector and as discussed earlier, this can be computed in two subtasks. Once all these three variables are computed, they can be added vectorially along with {N} computed in Step 6. This will require 3 flops for each component of the vector and hence the computation of each component can be assigned to a subtask. Thus, Step 8 can be computed in 11 steps, using subtasks, T_{33} to T_{43} as given in Table 4.1.

Step 9 : The applied torque at the joint is computed in Step 9 by taking the z component of the joint moment vector and hence there is no actual computation involved at this step, as indicated by

$$\tau_i = \{z\}\{n\}_i = n_{iz}$$

Thus the complete inverse dynamics of a manipulator can be written as a combination of 43 subtasks for each link in the manipulator. The subtasks T_1 to T_{25} are grouped under the forward recursion and the subtasks T_{26} to T_{43} are grouped under the backward recursion. All the subtasks are of equal computational load

of 3 flops. The symbolic computations is used at each task-level to arrive at the final equation. Thus the creation of these variables can be automated using the symbolic software such as REDUCE and hence this can be easily applied for manipulator of any arbitrary configuration. The precedence relationship among these subtasks can be understood either from Table 4.1 or from the task graphs given in Figs. 4.5 and 4.6. Referring to these figures, the following points must be noted:

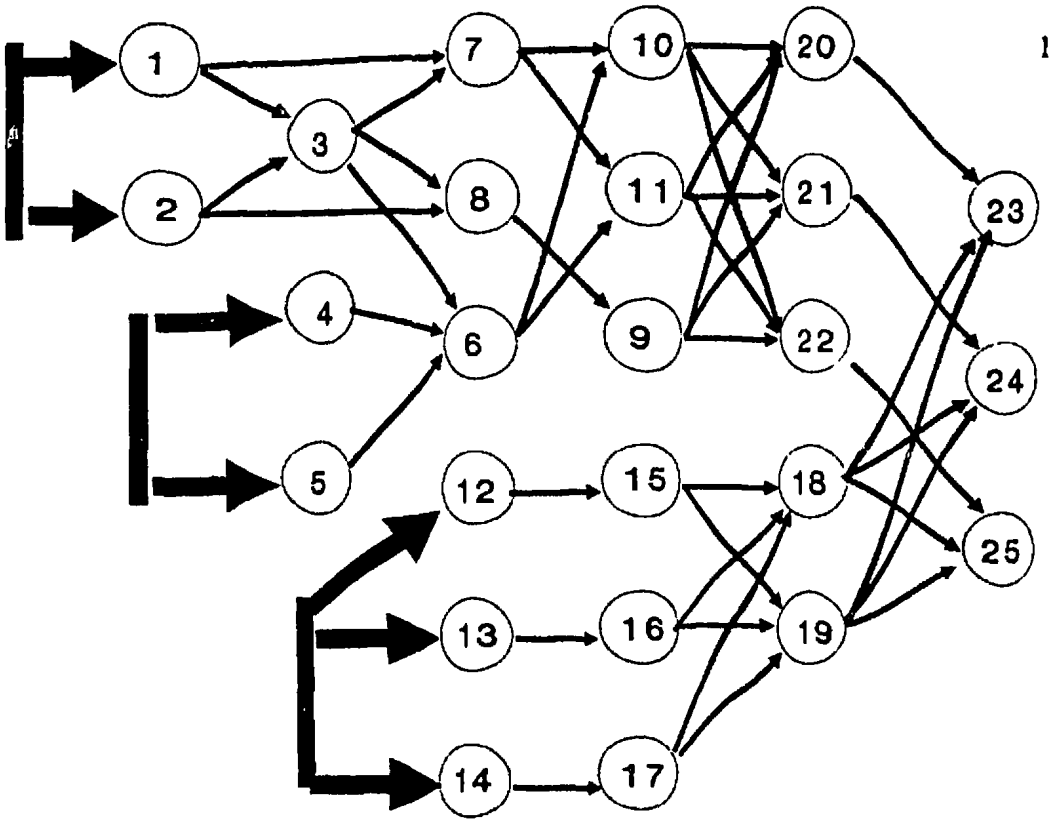
1. Fig. 4.5 indicates the subtasks corresponding to the forward recursion for any one link in the kinematic chain and Fig. 4.6 indicates the subtasks corresponding to the backward recursion for any one link in the kinematic chain.

2. The kinematics or the forward recursion of all the links should be completed before the backward recursion is started as mentioned in Chapter 2.

3. The kinematic computations of the first link are simpler than the other links since many of the parameters are either available readily or null. The external forces and moments on the end-effector, which forms the last link, are normally zero and this simplifies the backward recursion of last link. Table 4.2 indicates the list of subtasks that need not be executed for the first link in the forward recursion and for the last link in the backward recursion.

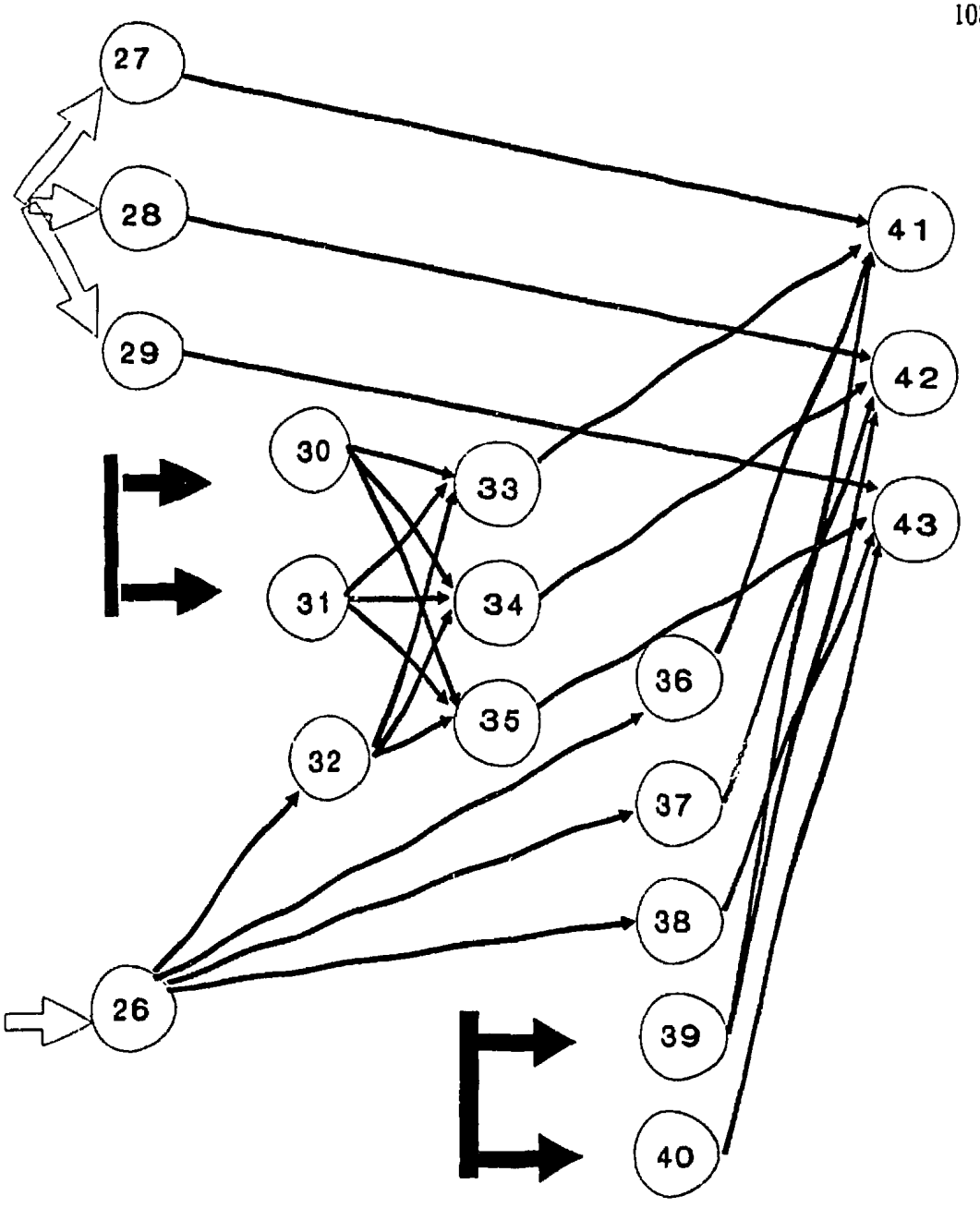
The above algorithm have been developed for a revolute joint. In case of a prismatic joint the computations are much simpler than those of a revolute joint, since the rotational parameters do not undergo a change in magnitude and the rotation transformation matrix consists of zeros and ones only. If joint i is parallel to joint i-1 ($\alpha_{i-1} = 0$), then

$$[R]_i = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.18}$$



Note : Thick arrows indicate input
from the dynamics of previous link

Figure 4.5: Task graph of Forward Recursion in Inverse Dynamics



Tasks 26,27,28 and 29 receive inputs from the forward of the same link and tasks 30,31,39 and 40 receive inputs from the backward recursion of the next link in the chain.

Figure 4.6: Task graph of Backward Recursion in Inverse Dynamics

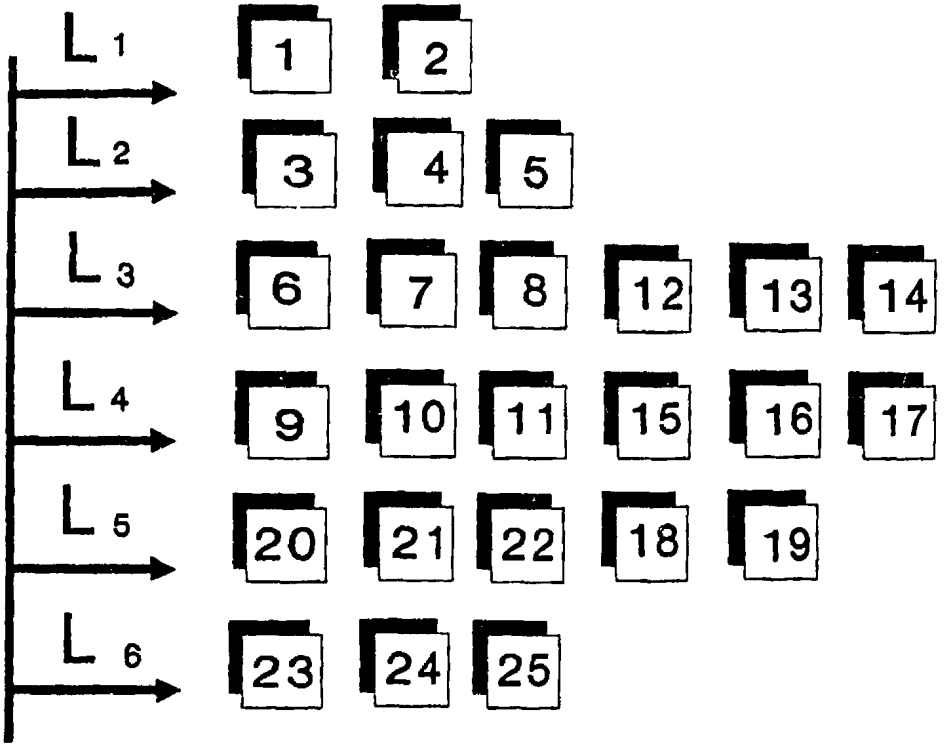
If joint i is perpendicular to joint i-1 ($\alpha_{i-1} = 90$ degrees), then

$$[R]_i = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{4.19}$$

The computations requiring multiplications with these rotation transformation matrices can be handled using symbolic programming avoiding the computational load in real-time. This is also given in Table 4.2, which indicates whether a subtask is required to be executed for a prismatic joint.

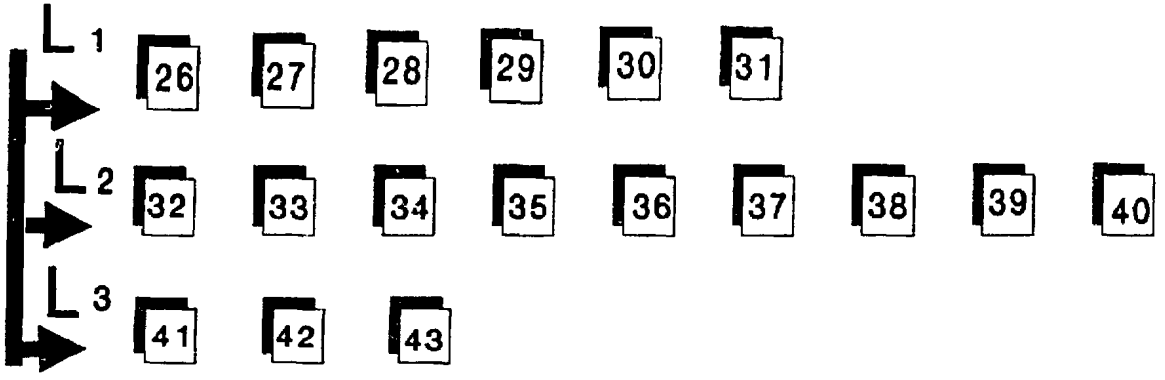
For an n link manipulator these task graphs can be combined to give the overall task graph. Such a procedure will result in a complex network, which will be difficult to handle for scheduling purposes. Also one likes to minimize all the computations in minimum amount of time. Assuming that there is no constraint on the number of processors these subtasks can be arranged into a layered task graph such that all the subtasks in any one layer can be executed simultaneously are arranged in one layer.

A layered task graph arranges the subtasks in disjoint layers such that a task in any one particular layer can be executed simultaneously without waiting for any other task in the same layer. By arranging the tasks in layers the scheduling problem can be solved efficiently. Since the tasks have been so designed so as to have the same number of computations, namely three floating point operations, they can be conveniently arranged into a layered task graph format (Polychronopoulos, 1988). The layered task graphs for the forward recursion and the backward recursion are shown in Figs. 4.7 and 4.8. These are shown separately since the algorithm requires the completion of forward recursion tasks for all the links before the backward recursion can be started.



Note : Precedence arrows are not shown
to maintain clarity

Figure 4.7: Layered Task Graph for the Forward Recursion



Note : Precedence arrows are not shown
to maintain clarity

Figure 4.8: Layered Task Graph for the Backward Recursion

Case Type	Nos. corresponding to subtasks that can be eliminated	Total No.
Prismatic Joint	1, 2,4,5,7,8,9,10,11	9
First Link	1,2,3,4,5,6,7,8,9,11,12, 13,14,15,16,17,18,26,27	19
Last Link	30,31,32,33,34,35,39,40	8

Table 4.2: Inverse Dynamic Tasks that can be eliminated for Special Cases

In order to generate the complete layered task graph of an n link robot, the graphs of forward recursion and backward recursion are stacked in tandem as shown in Fig. 4.9. The data dependency of the subtasks corresponding to one link on the subtasks of the previous link, in the case of forward recursion, dictates that the layered task graph of the adjacent links should be stacked in such a way that the layers corresponding to the ith link start after two layers of the i-1th link. This is clearly demonstrated in Fig. 4.9 where the layered task graph for a n-link manipulator has been assembled using the layered task graphs of the individual links. In other words, referring to Figs. 4.7 and 4.9, T_6 of the first link and T_1 of the second link can be done simultaneously. As shown in Fig. 4.9, the backward recursion of the link n, starts only after the completion of the computations of Layer 6 corresponding to the link n. Noting that the basic assumption has been that all the tasks have the same computational load, the total number of layers is an indication of the critical path for that particular task graph. For example, the critical path or the lower bound of the computational time in parallel implementation for the task graph defined in Fig. 4.9 can be computed by adding the number of layers for the forward recursion and the number of layers for the backward recursion. In the forward recursion, the six layers of each link are arranged in such a way that the tasks in the first layer can be executed simultaneously with the tasks in the third layer of the previous link. When the subtasks are arranged in this manner for n links, the total number of layers will be 6 + 2(n-2) = 2n + 2 layers. The subtasks of the first link do not affect the critical path, since they can be arranged in parallel with the subtasks of the second link as shown in Fig. 4.9. So they are not considered in deciding

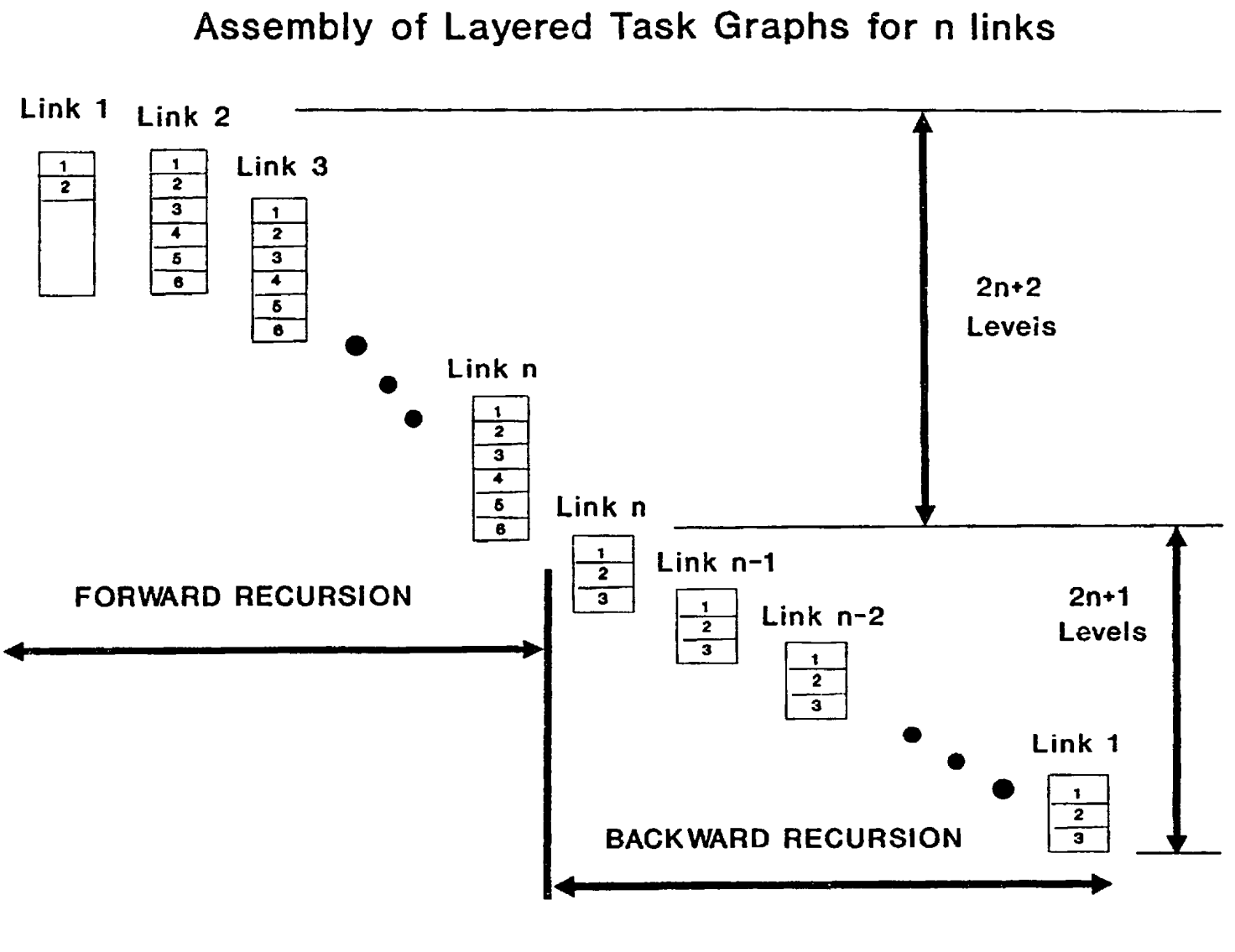


Figure 4.9: Task Graph Assembly for a six link manipulator

the total number of layers. Similary for the backward recursion, the total number of layers will be 3 + 2(n-1) = 2n + 1 layers. In total, for an n link manipulator, the total number of layers will be 4n+3. Each layer takes 3 flops for execution and hence the inverse dynamics of an n link manipulator can be computed in 3(4n+3)=12n+9 flops, when executed in parallel. This type of task streamlining, which can be called as the 'task streamlining approach' simplifies the scheduling problem as well as the synchronization in the actual implementation.

The computational tasks for the inverse dynamics of a six-link manipulator is detailed in Table 4.3. Here, the tasks are represented by a three digit number, wherein the first digit refers to the link number and the subsequent two digits refer to the subtask number. An empty box indicates that there is no subtask assigned to that particular layer, for that particular processor. As can be seen from the table, the total number of layers are 27 (n=6; 4n+3 = 27). This indicates the idle time of the processors. For example in the first cycle, processors 7 to 13 will be idle and for the 27th cycle, processors 4 to 13 will be idle. The number of columns indicate the concurrency of the algorithm. For example, referring to Table 4.3, the maximum number of tasks that can be executed simultaneously in any one cycle is 13 as indicated by the subtasks in layers 5, 7 and 9. The total number of layers indicate the critical path or the lower bound on the processing time for the parallel computation, which in the case of a six link robot is found to be 27 levels or 81 flops. For a uniprocessor implementation, it would take 231 levels or 693 flops and hence a speed-up of 8.55 is achieved. The number of processors required to achieve this speed-up would be as high as 13, and an efficiency of 65.81% is achieved.

Layer	Processor Loading												
	1	2	3	4	5	6	7	8	9	10	11	12	13
1	110	120	121	122	201	202							
2	123	124	125	203	204	205							
3	206	207	208	212	213	214	301	302					
4	209	210	211	215	216	217	303	304	305				
5	218	219	220	221	222	306	307	308	312	313	314	401	402
6	223	224	225	309	310	311	315	316	317	403	404	405	
7	318	319	320	321	322	406	407	408	412	413	414	501	502
8	323	324	325	409	410	411	415	416	417	503	504	505	
9	418	419	420	421	422	506	507	508	512	513	514	601	602
10	423	424	425	509	510	511	515	516	517	603	604	605	
11	518	519	520	521	522	606	607	608	612	613	614		
12	523	524	525	609	610	611	615	616	617				
13	618	619	620	621	622								
14	623	624	625										
15	626	627	628	629	630	631							
16	632	633	634	635	636	637	638	639	640				
17	641	642	643	526	527	528	529	530	531				
18	532	533	534	535	536	537	538	539	540				
19	541	542	543	426	427	428	429	430	431				
20	432	433	434	435	436	437	438	439	440				
21	441	442	443	326	327	328	329	330	331				
22	332	333	334	335	336	337	338	339	340				
23	341	342	343	226	227	228	229	230	231				
24	232	233	234	235	236	237	238	239	240				
25	241	242	243	126	127	128	129	130	131				
26	132	133	134	135	136	137	138	139	140				
27	141	142	143										

Table 4.3: Tasks for Inverse Dynamics of A Six-Link Manipulator
Maximum Concurrency = 13

This indicates that if one uses 13 processors for implementing the inverse dynamics of a six-link manipulator, the processors will be used for only 65.81% of the time or in other words they will be idling for 34.19% of the time. This may not be an optimal use for using the multiprocessor and hence one should vary the number of processors and decide the optimal number of processors that should be used to arrive at a good speed-up with a reasonable processor utilization rate.

4.3.2 Customization of Robot Dynamics

As detailed in Chapter 3, customization of the robot dynamics often leads to less computational load, since most of the position vectors are aligned along with one axis resulting in sparse vectors. For example, for the Stanford manipulator, Table 4.4 indicates the sparse elements in the position vectors s_i and p_i. s_i and p_i are the position vectors representing the the CG of the ith link and the origin of the i+1th link frame. As is evident, out of the 36 elements, 25 elements are zero and hence this need to be considered while formulating the dynamic equations for real time control.

In the modified NE algorithm, the operations involving the position vectors appear in steps 3, 4 and 8. The corresponding subtasks are summarized in Table 4.5. Depending on the number of zero elements in the position vector, the computations can be reduced resulting in less number of subtasks. For example, in Step 3, if $\{p_i\}$ has one zero element, the three multiplications and three additions corresponding to that zero element can be cut down, resulting in cutting down two subtasks and if it has two zero elements, four subtasks can be cut down and if the complete vector is zero than all the six subtasks can be cut

down. This can be done in a systematic way and Table 4.6 indicates the subtasks that can be cut down from the computational load corresponding to one, two and three zero elements in the position vectors. For example, applying this to the Stanford manipulator, the subtasks that can be eliminated owing to the sparsity in position vectors are shown in Table 4.6. The overall computational load for the Stanford Manipulator is shown in Table 4.8. The maximum concurrency level for the computation of the inverse dynamics of the Stanford Manipulator is 9. If one uses 9 processors, then the algorithm can be implemented as it exists but it may often be required to limit the number of processors to a lower number, in which case, an efficient scheduling strategy need to be developed.

4.3.3 Scheduling Strategy

With the streamlined list of tasks arranged in layers, the scheduling of the jobs can be easily handled by a heuristic algorithm. The jobs in each level are scheduled on the available processors in a systematic manner until all the processors are engaged and if there are pending jobs after engaging all the processors they are scheduled in the subsequent cycle. If there are processors available in any cycle after scheduling all the jobs in one layer, a check is made in the subsequent layers beginning with the most immediate layer for jobs which can be scheduled, i.e. jobs for which the precedants have already been scheduled in earlier layers. Where there are no ready-to-execute tasks, the available processors are left idle. This process is continued until all the tasks are allocated. The scheduling strategy is summarized in Fig. 4.10.

Link No.	Position vector of CG $\{s\}_i$			Position vector of Origin $\{p\}_i$		
	s_x	s_y	s_z	p_x	p_y	p_z
1	0	√	√	0	√	0
2	0	0	√	0	√	0
3	0	0	√	0	0	0
4	0	√	√	0	0	0
5	0	√	√	0	0	0
6	0	0	√	0	0	0

Table 4.4: Sparsity in Position Vectors

Step No	Computations	Total subtasks	subtasks to be executed for		
			one zero element	two zero elements	three zero elements
3	$a_{i-1} + [\lambda]_{i-1}.\{p\}_{i-1}$	T_{12}, T_{13}, T_{14} T_{15}, T_{16}, T_{17}	T_{14}, T_{15} T_{16}, T_{17}	T_{16} T_{17}	- -
4	$\{a\}_i + [\lambda]_i.\{s\}_i$	T_{20}, T_{21}, T_{22} T_{23}, T_{24}, T_{25}	T_{22}, T_{23} T_{24}, T_{25}	T_{24} T_{25}	- -
8	$\{p\}_i \times \{f\}_{i+1}^i$	T_{33}, T_{34}, T_{35}	T_{34}, T_{35}	T_{35}	-
8	$\{s\}_i \times \{F\}_i$	T_{36}, T_{37}, T_{38}	T_{37}, T_{38}	T_{38}	-

Table 4.5: Subtasks Eliminated for Sparsity in Position Vectors

Link No	No. of zeros in vectors		Subtask Nos. to be eliminated	Total No. eliminated
	$\{s\}_i$	$\{p\}_i$		
1	1	2	112, 113, 114, 115, 120, 121, 136, 133, 134	9
2	2	2	212, 213, 214, 215, 220, 221, 222, 223, 233, 234, 236, 237	10
3	2	3	312, 313, 314, 315, 316, 317, 320, 321, 322, 323, 333, 334, 335, 336, 337	15
4	1	3	412, 413, 414, 415, 416, 417, 420, 421, 433, 434, 435, 436	12
5	1	3	512, 513, 514, 515, 516, 517, 520, 521, 533, 534, 535, 536	12
6	2	3	612, 613, 614, 615, 616, 617, 620, 621, 622, 623, 633, 634, 635, 636, 637	15

Table 4.6: Subtasks Eliminated in Stanford Manipulator for Sparsity

	Processor Loading								
Layer	1	2	3	4	5	6	7	8	9
1	110	120	121	122	201	202	0	0	0
2	123	124	125	203	204	205	0	0	0
3	206	207	208	0	0	0	0	0	0
4	209	210	211	216	217	303	0	0	0
5	218	219	306	401	402	0	0	0	0
6	224	225	403	404	405	0	0	0	0
7	318	319	406	407	408	501	502	0	0
8	324	325	409	410	411	503	504	505	0
9	418	419	422	506	507	508	601	602	0
10	423	424	425	509	510	511	603	604	605
11	518	519	522	606	607	608	0	0	0
12	523	524	525	609	610	611	0	0	0
13	618	619	0	0	0	0	0	0	0
14	624	625	0	0	0	0	0	0	0
15	626	627	628	629	0	0	0	0	0
16	638	0	0	0	0	0	0	0	0
17	641	642	643	526	527	528	529	530	531
18	532	537	538	539	540	0	0	0	0
19	541	542	543	426	427	428	429	430	431
20	432	437	438	439	440	0	0	0	0
21	441	442	443	326	327	328	329	330	331
22	332	338	339	340	0	0	0	0	0
23	341	342	343	226	227	228	229	230	231
24	232	233	235	238	239	240	0	0	0
25	241	242	243	126	127	130	131	0	0
26	132	135	137	138	139	140	0	0	0
27	141	142	143	0	0	0	0	0	0

Table 4.7: Tasks for Customized Inverse Dynamics of Stanford Manipulator
Maximum Concurrency = 9

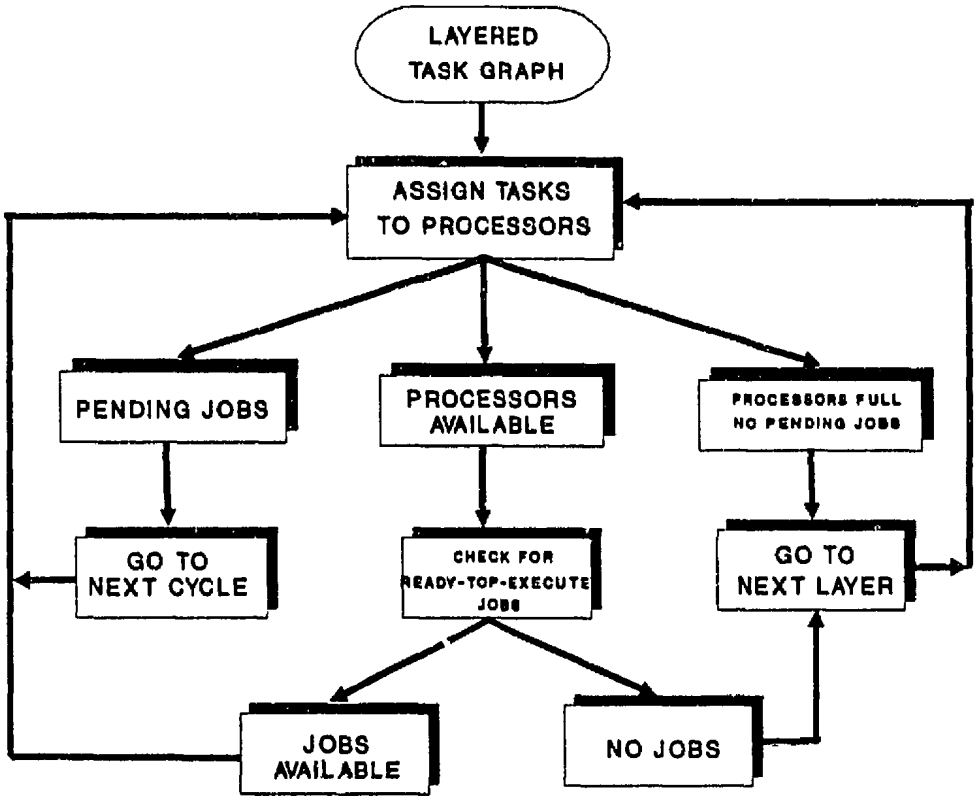


Figure 4.10: Scheduling Strategy

4.4 Case Study - Stanford Manipulator

The inverse dynamics of Stanford manipulator, whose kinematic and dynamic parameters are given in Tables 2.2, 2.7 and 2.8, was analyzed for parallel implementation. The choice of Stanford Manipulator was made since published results of earlier researchers (Luh and Lin, 1982; Kasahara and Narita, 1985; and Chen et. al, 1988) were available in the literature, for comparison of results of this task streamlining approach. The layered task graph, in the form of a table, for Stanford manipulator is given in Table 4.8. As indicated earlier, the subtasks are represented by a three digit number, the first digit representing the link number and the subsequent two digits representing the subtask number. As can be seen from a comparison of Table 4.3 and Table 4.8, the computational load for the six-link Stanford Manipulator is only 156 subtasks (468 flops) compared to the load for the generalized six-link manipulator in Table 4.8, namely 231 subtasks (693 flops). This indicates a reduction of about 25%. This has been achieved by going through the procedure outlined in Section 4.3.2. The tasks that can be eliminated for the zero elements in the position vectors were identified in Table 4.7 and deleted from the generalized six-link manipulator load given in Table 4.3. Also, since joint 3 of the manipulator is a prismatic joint, the tasks as indicated in Table 4.2 for a prismatic joint were also eliminated. This has resulted in Table 4.8, which can be scheduled on the required number of processors.

Using the scheduled strategy outlined in Section 4.3.3, the computational load given in Table 4.4 was scheduled using this algorithm. The task schedules for two to nine processor configurations are given in Tables C.1 to C.8 in Appendix C. This was repeated for the Stanford manipulator using the layered task graph

in Table 4.8. The tasks were scheduled on two to eight processor configurations and the corresponding schedules are shown in Tables C.9 to C.15 in Appendix C.

4.5 Results and Discussion

The Task-streamlining approach results in a simplified and an efficient approach to scheduling the inverse dynamic tasks in a parallel processor. The speed-up and the efficiency of the Task Streamlined approach for a six link manipulator with varying number of processors is shown in Figs. 4.11 and 4.12. It can be seen that upto six processors, an efficiency of over 98% is achieved along with an excellent speed-up, which is almost equal to the number of processors employed. Beyond six processors, the increase in the speed-up is only marginal, whereas the efficiency falls drastically. Hence a six processor configuration would be ideal for implementation of the inverse dynamic computations.

The effect of varying the number of processors on the speed-up and efficiency, for the customized inverse dynamics of the six-link Stanford manipulator, is also shown in Figs. 4.11 and 4.12. Since the number of tasks have been reduced by over 25%, the concurrency level has dropped from 13 to 9, and this is reflected in the speed-up and efficiency curves. In the case of the customized equations, the efficiency is more than 98% upto four processors, and beyond this level, it starts coming down drastically. So a four processor configuration would be ideal for the customized dynamics of the Stanford manipulator.

The processing time for the inverse dynamic tasks for Stanford manipulator, using a 16-bit microprocessor (Intel 8086), is compared with those previously published by other researchers in Table 4.8. For a uniprocessor implementation,

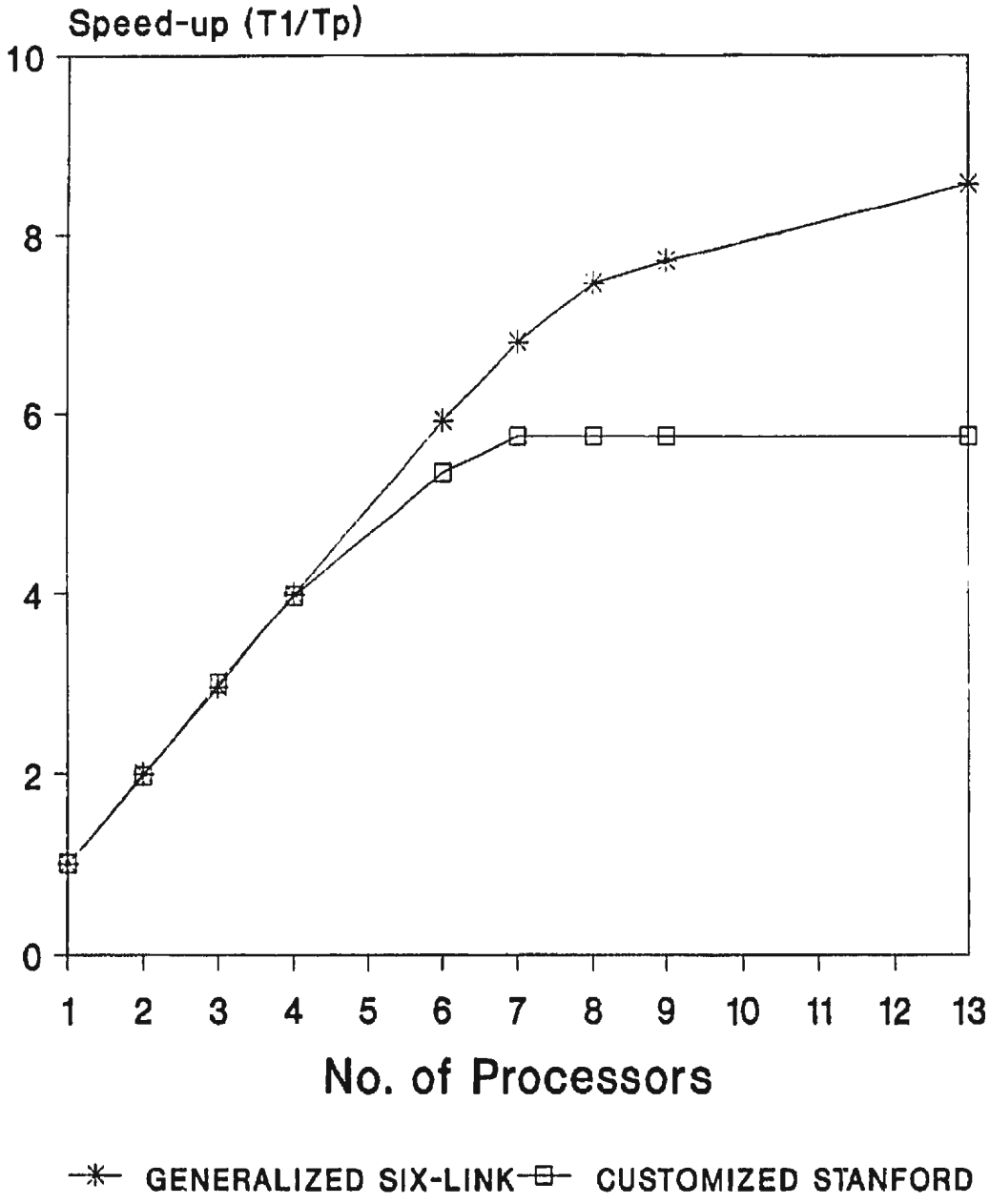


Figure 4.11: Speed-up vs No. of Processors for inverse dynamic computation for a six-link manipulator

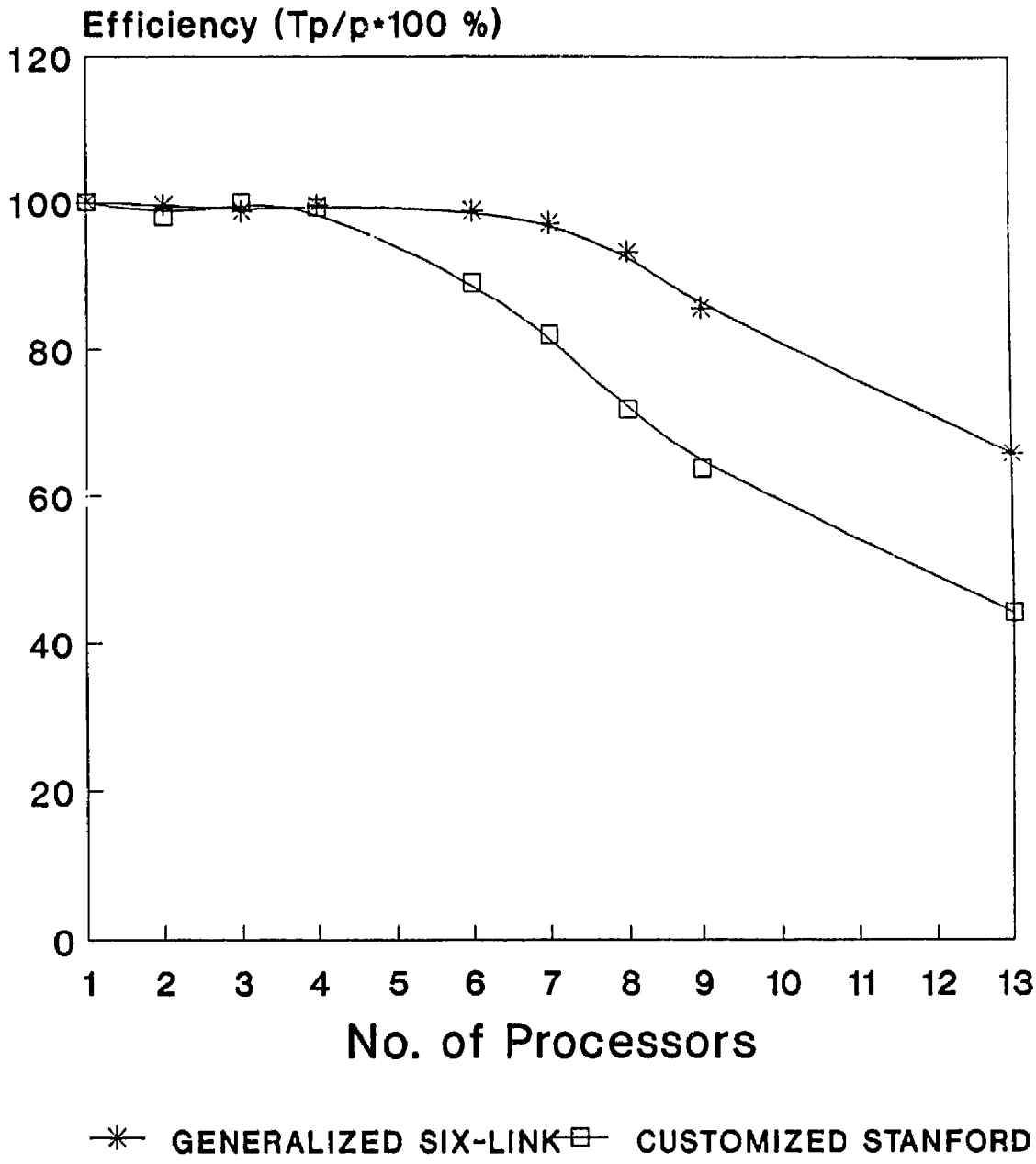


Figure 4.12: Efficiency vs No. of Processors for inverse dynamic computation for a six-link manipulator

the time required by the scheme presented in this work is much shorter than those of others. Compared to a processing time of 24.83 ms as proposed by others this work requires only 21.06 ms using the modified algorithm. Also as can be seen from Fig. 4.13 which presents the same information in a graphical form, this work achieves a reduced processing time owing to the increased concurrency as well as the efficiency of the algorithm. For a four processor implementation, Kasahara's approach and Chen's approach requires about 6.59 ms, whereas the task streamlining approach using the modified NE algorithm requires only 5.26 ms. Also the minimum time has been reduced from 5.60 ms to 3.65 ms for a seven-processor configuration.

It should also be noted here that since all the subtasks are of the same size, the synchronization overheads will be minimum. Moreover, the scheduling of the subtasks is also simplified due to this uniformity in the tasks. Also the process of mapping the tasks into the scheduling problem can be automated using symbolic programming.

The friction model developed in Chapter 3 can be included in these task graphs as additional subtasks. The friction computations given by Table 3.1 and 3.2 can be decomposed in a similar way and incorporated into the total task graph. For manipulators using anti-friction bearings, only the transmission friction has to be computed. For clarity of the work, only the frictionless dynamics was parallelized here. This approach can be extended for including the friction tasks in the computational load.

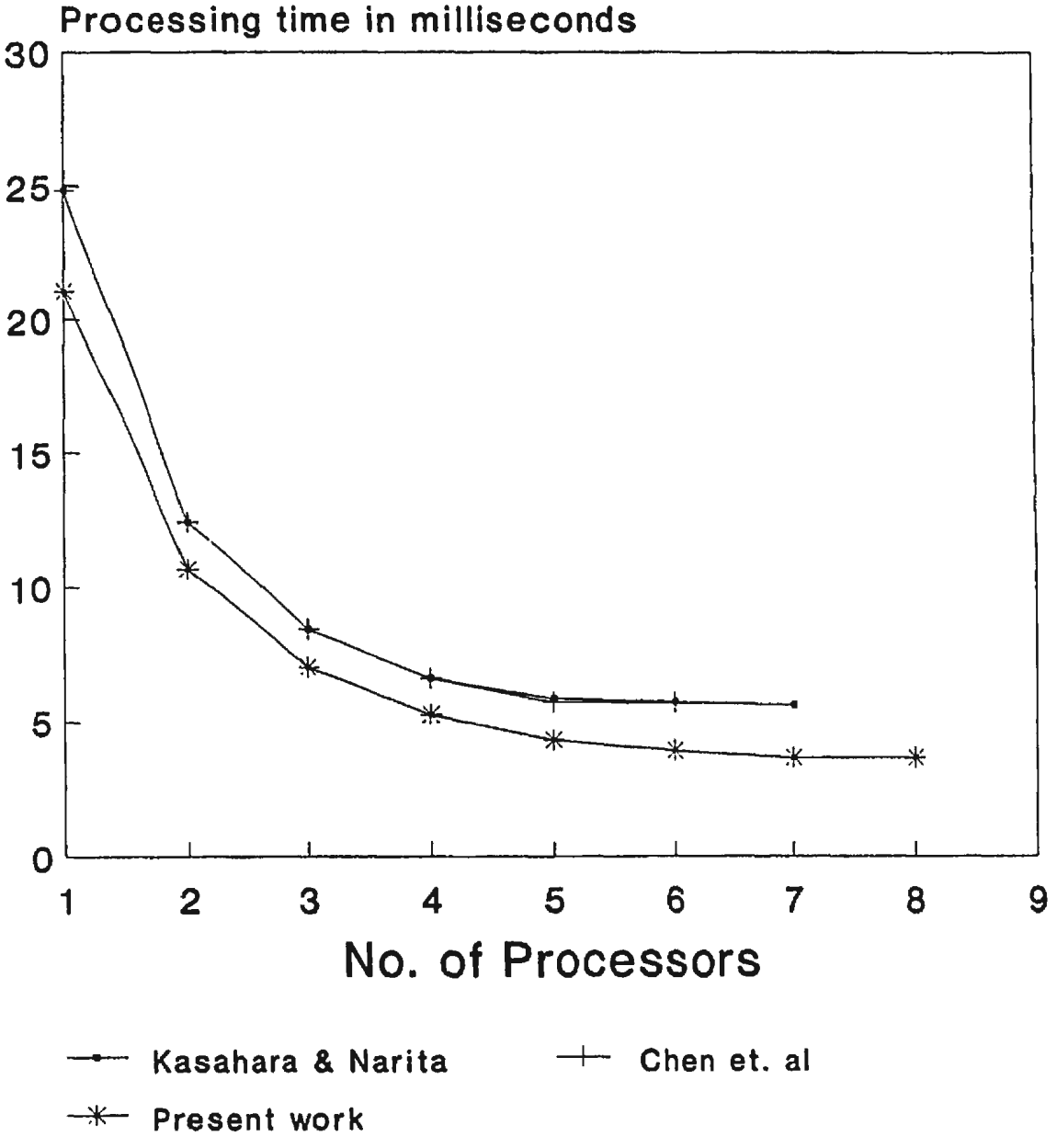


Figure 4.13: Comparison of Processing Time for inverse dynamic computation of Stanford Manipulator

4.6 Conclusion

An efficient computational algorithm for inverse dynamic problems of robotic manipulators has been presented in this work utilizing the modified NE algorithm developed in Chapter 2. Based on the work in this chapter the following conclusions can be drawn.

1. The modified NE algorithm using the task streamlined approach for decomposition of the tasks, results in increased concurrency.
2. A six processor configuration would be ideal for implementing the generalized inverse dynamics of manipulators.
3. A computational count of 4n+3 operations, where n is the number of links in the manipulator, can be achieved for the inverse dynamic problem.
4. Customization of the algorithm can bring down the number of processors required and the processing time. Also a four-processor configuration would be optimal for implementing the customized dynamics of the Stanford manipulator.
5. The modified NE algorithm, developed in this work, when used with the task streamlining approach, decreases the synchronization time and scheduling time.

No. of Proc.	Luh & Lin (1982)	Kasahara & Narita (1985)	Chen Lee & Hou (1988)	Present Work
1	24.8	24.83	24.83	21.06
2	-	12.42	12.42	10.67
3	-	8.43	8.44	7.02
4	-	6.59	6.59	5.26
5	-	5.86	5.72	4.32
6	9.67	5.73	5.70	3.92
7	-	5.60	-	3.65
8	-	-	-	3.65

Table 4.8: Comparison of processing time for Stanford Manipulator dynamics

Chapter 5

Summary, Contributions and Recommendations

5.1 Summary of the Work

A computationally efficient and accurate solution, for solving the inverse dynamic problem in real-time, was developed in this thesis. The conventional NE algorithm was modified using symbolic computations and a [λ] matrix. The modified algorithm, before customization, results in a reduction of 30-40% of the computational load, over the conventional NE algorithm. Customization of the algorithm for specific manipulators was suggested using iterative symbolic programming and this approach was demonstrated for some standard manipulators. The resulting computational load is further reduced by customization, and the number of floating point operations was also considerably less than the results published earlier, especially for manipulators with more than three links.

Modeling of friction for robotic mechanisms was suggested by modeling of the joint friction using Coulomb's law and the transmission friction using an input-output function. This was demonstrated for PUMA-560 manipulator with

three links, and it was shown that the total friction is quite significant in robotic mechanisms. It was also shown that the joint friction is much smaller than the transmission friction and they should be modeled separately for better accuracy. It was also shown that the computational load for including friction in the dynamic model of the manipulator, is only marginally increased, when used along with either the NE algorithm or the modified NE algorithm developed in this work.

Finally, the modified NE algorithm was parallelized using a 'task streamlining approach'. The algorithm was decomposed into subtasks of uniform computational load of three floating point operations. These subtasks were arranged optimally in a layered task graph and assembled for a given number of links. The resulting task graph was used to schedule the tasks on the required number of processors using a simplified bin-packing algorithm. The speed-up and efficiency of the algorithm for a six link manipulator was demonstrated and it was concluded that a configuration consisting of six processors would be ideal for implementing the inverse dynamic problems. Customization procedure was also discussed and this was demonstrated for the Stanford manipulator. It was also shown that the minimum processing time for the computation of the inverse dynamics is only 3.65 ms, which is lower than the results published by earlier researchers in this field.

5.2 Contributions of this Work

The problem of real-time computation of the inverse dynamics of manipulators has been addressed by a wide body of researchers. The uniqueness of this work is in the way in which vector mechanics, symbolic programming and parallel pro-

cessing are combined to yield in an efficient algorithm. The specific contributions of this work can be listed as follows.

1. The existing NE algorithm has been modified using the [λ] matrix for the minimization of computations.
2. An analytical friction model has been developed, which can be used to predict the inverse dynamic torques/forces more accurately.
3. Due to (a) the use of the [λ] matrix approach, (b) the fine decomposition of tasks, and (c) the optimal layering of the tasks in a parallel algorithm, minimum processing time was achieved which was better than the results published by other researchers.

The incorporation of friction and the low processing time suggests a promise for implementation of this algorithm in the real-time control of industrial manipulators.

5.3 Recommendations for Future Work

As a follow-up of this research, two significant avenues are open for further work. The first is to develop an expert system using an expert shell such as VP-Expert, to integrate the various pieces of this work, so as to get a design tool for the robot designer. The generation of inverse kinematic and inverse dynamic equations and parallelizing them on a given number of processors can be handled in an excellent way. The second is to extend this work to incorporate the flexibility of the links. The real-time computation of the inverse dynamics incorporating flexibility would be a feasible job with parallel implementation. This would be worth attempting for application to space robotics or underwater robotics.

REFERENCES

Albus, J. S., (1975), *A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)*, Journal of Dynamic Systems, Measurements and Control, pp. 220-227.

Albus, J. S., (1981), *Brains, Behavior and Robotics*, Byte Publication Inc., Massachussetts.

Armstrong, B., Khatib, O., Burdick, J., (1986), *The Explicit Dynamic Model and Inertial Parameters of the PUMA 560 arm*, Proc. of IEEE Int. Conf. on Robotics and Automation, pp.510-518.

Armstrong, B., (1988), *Dynamics for Robot Control: Friction Modeling and Ensuring Excitation During Parameter Identification, Ph.D. Thesis*, Stanford University.

Balafoutis, C.A. et al., (1988), *Efficient Modeling and Computation of Manipulator Dynamics Using Orthogonal Cartesian Tensors*, IEEE Journal of Robotics and Automation, Vol.4, No.6, December (1988).

Binder, E. E., (1985), *Distributed Architecture and Fast Parallel Algorithms in Real-time Robot Control*, Ph.D. Thesis, Oregon State University.

Brawer, S., (1989), *Introduction to Parallel Programming*, Academic Press, Inc., Boston.

Burdick, J.W., (1986), *An algorithm for Generation of Efficient Manipulator Dynamic Equations*, Proc. of IEEE Int. Conf. on Robotics and Automation, pp.212-218.

Canudas De Wit, C., Noel, P., Aubin, A., Brogliato, B., Drevet, P., (1989), *Adaptive Friction Compensation in Robot Manipulators: Low-velocities*, Proc. of IEEE Int. Conf. on Robotics and Automation, pp.1352-1357.

Canudas, C., Astrom, K.J., and Braun, K., (1986), *Adaptive Friction Compensation in DC Motor Drives*, Proc. of IEEE Int. Conf. on Robotics and Automation, pp.1556-1561.

Cesareo, G., Nicolo, F., and Nicosia, S., (1984), *DYMIR:A Code for Generating Dynamic Model of Robots*, Proc. of the First International Conf. on Robotics, Paul, R.P., Ed., Atlanta, GA, pp. 115 - 120.

Chen, C.L., Lee, C.S.G., Hou, E.S.H, (1988), *Efficient Scheduling Algorithms for Robot Inverse Dynamics Computation on a Multiprocessor System*, Proc. of the IEEE Conf. on Robotics and Automation, pp. 1146 - 1151.

Chironis, N.P., (1967), *Gear Design and Application*, McGraw-Hill, Inc., New York, pp. 120-122.

Coffman, E.G. (Ed.), (1975), *Computer and Job-shop Scheduling Theory*, John Wiley & Sons, New York.

Craig, J.J., (1986), *Introduction to Robotics: Mechanics and Control*, Stanford University, Addison-Wesley.

Denavit, J. and Hartenberg, R.S., (1955), *A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices*, J. of Applied Mechanics,Vol.77, No.2, pp. 215 - 221.

Dhanaraj, C., and Sharan, A. M., (1990), *On Efficient Modeling of Rigid Link Robot Dynamic Problems*, Proc. of IASTED Conf. on Modeling, Simulation and Optimization, Montreal, 23-25 May, pp.

Dhanaraj, C., and Sharan, A.M., (1990), *Friction Modeling in Robot Dynamics - A Case Study*, To be presented in Int. Conf. Automation, Robotics and Computer Vision, Singapore, 24-27 September, 1990.

Ducksbury, P.G., (1986), *Parallel Array Processing*, Ellis Horwood Ltd., Chichester.

Dudley, D.W., Ed. (1962), *Gear Handbook: The Design, Manufacture, and Applications of Gears*, McGraw-Hill, Inc., New York, pp. 3-35 to 3-43.

Fu, K.S., Gonzalez, R.C., Lee, C.S.G., (1987), *Robotics: Control, Sensing, Vision and Intelligence*, McGraw-Hill, New York.

Gogoussis, A. and Donath, M., (1988), *Coulomb Friction Effects on the Dynamics of Bearings and Transmissions in Precision Robot Mechanisms*, Proc. of IEEE Int. Conf. on Robotics and Automation, pp.1440 - 1446.

Gogoussis, A. and Donath, M., (1987), *Coulomb Friction Joint and Drive Effects in Robot Mechanisms*, Proc. of IEEE Int. Conf. on Robotics and Automation, pp.828 - 836.

Hollerbach, J.M., (1980), *A recursive Lagrangian formulation of manipulator dynamics and a comparative study of dynamics formulation complexity*, IEEE Transactions on Systems, Man and Cybernetics, Vol.SMC-10, No.11, pp.730-736.

Hwang, K. and Briggs, F.A., (1984)., *Computer Architecture and Parallel Processing*, McGraw-Hill, Inc., New York.

Izaguirre, A., Paul, R., (1986), *Automatic Generation ofthe Dynamic Equations of the Robot Manipulators Using a Lisp Program*, Proc. of International conf. on Robotics and Automation, pp.220-226.

Kane, T.R. and Levinson, D.A., (1983), *The Use of Kane's dynamical equations in robotics*, Int. Journal of Robotics Research, Vol.2, No.3, pp.3-21.

Kane, T.R., and Levinson, D.A., (1985), *Dynamics: Theory and Applications*, McGraw-Hill Book Company, New York.

Kasahara, H., Iwata, M., Narita, S. (1988), *Parallel Processing of Robot Dynamics Simulation Using Optimal Multiprocessor Scheduling Algorithms*, Journal of Systems and Computers in Japan, Vol.19, No.10, pp.45-54.

Kasahara, H., Narita, S., (1984), *Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing*, IEEE Trans. on Computers, Vol.C-33, No.11.

Kasahara,H., Narita,S., (1985), *Parallel Processing of robot arm control computation on a multimicroprocessor system*, IEEE Journal of Robotics and Automation, Vol.1, No.2, pp. 104 -113.

Khalil, W., Kleinfinger, J.F., Gautier, M, (1986), *Reducing the Computational Burden of the Dynamic Models of Robots*, Proc. of IEEE Int. Conf. on Robotics and Automation, pp.525-531.

Khosla, P.K., Ramos,S. (1988), *A comparative Analysis of the Hardware Requirements for the Lagrange-Euler and Newton-Euler Dynamic Formulations*, Proc. of the IEEE Int. conf. on Robotics and Automation, pp.291-296.

Khosla,P.K., Kanade,T., (1988), *Experimental Evaluation of Nonlinear Feedback and Feed-forward Control Schemes for Manipulators*, The Int. Journal of Robotics Research, Vol.7, No.1, pp. 18 - 28.

Kronsjo, L., (1985), *Computational Complexity of Sequential and Parallel Algorithms*, John Wiley & Sons, New York.

Lathrop, R.H., (1985), *Parallelism in Manipulator Dynamics*, Int. Journal of Robotics Research, Vol.4, No.2, pp. 80-102.

Lee, C.S.G., Chang, P.R., (1988), *Efficient Parallel Algorithms for Robot Forward Dynamics Computation*, IEEE Trans. on Systems, Man, and Cybernetics, vol.18, No.2, pp.238 - 251.

Leu, M.C., N.Hemati, (1986), *Automated Symbolic Derivation of Dynamic Equations of Motion for Robotic Manipulators*, ASME Journal of Dynamic Systems, Measurement ad Control, Vol.108, pp.172 - 179.

Liu, C.H. and Chen, Y.M., (1986), *Multi-Microprocessor-Based Cartesian-Space Control Techniques for a Mechanical Manipulator*, IEEE J. Robotics and Automation, Vol.RA-2, No.2, pp. 110-115.

Luh, J.Y.S. and Lin, C.S., (1981), *Automatic Generation of Dynamic Equations for Mechanical Manipulators*, Proc. of the 1981 Joint Automatic Control Conference, Charlottesville, VA, pp. TA-20/1-5.

Luh, J.Y.S., Walker, M.W., and Paul R.P.C., (1980), *On-line computational scheme for mechanical manipulators*, ASME Journal of Dynamic Systems, Measurement and Control, Vol.102, pp.69-76.

Luh, J.Y.S., and Lin, C.S., (1982), *Scheduling of parallel computation for a computer-controlled mechanical manipulator*, IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-12, No.2, pp.214-234.

MACSYMA Reference Manual, Mathlab Group, Lab. for Comp. Sc., MIT, (1983).

Murray, J.J. and Neuman, C.P., (1988), *Organizing customized robot dynamics algorithms for efficient numerical evaluation*, IEEE Transactions on Systems, Man and Cybernetics, Vol.18, No.1, pp.115-125.

Neuman, C.P. and Murray, J.J., (1985), *Computational Robot Dynamics: Foundations and Applications*, Journal of Robotic Systems, Vol.2, No.4, pp.425-452.

Neuman, C.P. and Murray, J.J., (1987a), *Customized computational robot dynamics*, Journal of robotic systems, Vol.4, No.4, pp.503-526.

Neuman, C.P., and Murray, J.J., (1987b), *Symbolically efficient formulations for computational robot dynamics*, Journal of Robotic Systems, Vol.4, No.6, pp.743-769.

Nigam, R. and Lee, C.S.G., (1985), *A multiprocessor-based controller for control of mechanical manipulators*, IEEE Journal of Robotics and Automation, Vol. RA-1, No.4. pp.173-182.

Oleinick, P.N., (1982), *Parallel Algorithms on a Multiprocessor*, UMI Research Press, Ann Arbor, Michigan.

Orin, D.E., McGhee, R. B., Vukobratovic, M., and Hartoch, G., 1979, *Kinematic and Kinetic Analysis of Open-Chain Linkages Utilizing Newton Euler Methods*, Mathematical Biosciences, Vol.43, pp. 107-130.

Paul, R.P., (1981), *Robot Manipulators: Mathematics, Programming and Control*, MIT Press , Massachussets.

Polychronopoulos, C.D., (1988), *Parallel Programming and Compilers*, Kluwer Academic Publishers, Boston.

REDUCE Users' Manual, Version 3.3, The Rand Corporation, Santa Monica, CA 90406-2138, (1987).

Ramos, S., (1988), *Parallelism in Manipulator Dynamics: Analysis and Implementational Issues for High-Speed Control, M.Eng. Thesis*, Carnegie Mellon University, Pittsburgh, USA.

Rayna, G., (1987), *REDUCE: Software for Symbolic Computation*, Springer-Verlag, New York.

Silver, W.M., (1982), *On the equivalence of Lagrangian and Newton-Euler Dynamics for Manipulators*, Int. Journal of Robotics Research, Vol.1, No.2, pp.118-128.

Stepanenko, Y. and Vukobratovic, M., (1976), *Dynamics of Articulated Open-Chain Active Mechanisms*, Mathematical Biosciences, Vol.28, pp.137-170.

Tonkinson, J. and Donath, M., (1988), *Scheduling robot inverse dynamics computation for multiprocessor based control*, Proc. USA-Japan symposium on Flexible Automation - Crossing Bridges - Advances in Flexible Automation and Robotics, pp.107-113.

Toogood, R.W., (1989), *Efficient Robot Inverse and Direct Dynamics Algorithms Using Micro-Computer Based Symbolic Generation*, Proc. of IEEE Int. conf. on Robotics and Automation, pp.1827-1833.

Uicker, J. J., (1965), *On the Dynamic Analysis of Spatial Linkages Using 4 x 4 Matrices, Ph. D. Thesis*, North-Western University, IL, USA.

Vecchio, L., Nicosia, S., Nicolo, F., and Lentini, D. *Automatic Generation of Dynamical Models of Manipulators*, Proc. of the Tenth Int. Symposium on Industrial Robots, Milan, Italy, March 5-7, 1980, pp.293-301.

Vukobratovic, M. and Kircanski, N., (1984), *A Method for Computer-Aided Construction of Analytical Models of Robotic Manipulators*, Proc. of the First Int. Conf. on Robotics, Paul, R.P., Ed., Atlanta, GA, pp. 519 - 528.

Vukobratovic, M., Kircanski, N., Li, S.G., (1988) *An approach to Parallel Processing of Dynamic Robot Models*, Int. Journal of Robotics Research, Vol.7, No.2, pp.64 - 71.

Yang, D.C.H. and Tzeng, S.W., (1986), *Simplification and linearization of manipulator dynamics by the design of inertia distribution*, Int. Journal of Robotics Research, Vol.5, No.3, pp.120-128.

Yin, S and Yuh, J., (1989), *An Efficient Algorithm for Automatic Generation of Manipulator Dynamic Equations*, Proc. of IEEE Int. Conf. on Robotics and Automation, pp.1812-1817.

APPENDIX

Appendix A

Lagrange Equations of Motion

A.1 Closed Form Equations

The closed form Lagrange Equations [1], which are widely used for simulation and control applications are given as

$$\tau_i = \sum_{j=i}^{n} \sum_{k=1}^{j} \mathbf{Tr}\ [U]_{ij}[J]_i[U]_{ik}^T \ddot{q}_k + \sum_{j=i}^{n} \sum_{k=1}^{j} \sum_{m=1}^{j} \mathbf{Tr}\ [U]_{jkm}[J]_i[U]_{ji}^T \dot{q}_k \dot{q}_m - \sum_{j=i}^{n} m_j g [U]_{ji}. \qquad for \quad i = 1, 2, ..., n. \tag{A.1}$$

where **Tr** denotes the trace operator and $[U]_{ij}$ denotes the partial derivative of the transformation matrix T_i^0 with respect to q_j and $[U]_{ijk}$ denotes the partial derivative of $[U]_{ij}$ with respect to q_k and ${}^j\bar{r}_j$ is the position vector of the CG of the jth link projected in the link frame. J_i is the pseudo-inertia tensor about the origin of the link co-ordinate frame, given as

$$[J]_i = \begin{bmatrix} \frac{-I_{xx}+I_{yy}+I_{zz}}{2} & I_{xy} & I_{xz} & m_i\bar{x}_i \\ I_{xy} & \frac{I_{xx}-I_{yy}+I_{zz}}{2} & I_{yz} & m_i\bar{y}_i \\ I_{xz} & I_{yz} & \frac{I_{xx}+I_{yy}-I_{zz}}{2} & m_i\bar{z}_i \\ m_i\bar{x}_i & m_i\bar{y}_i & m_i\bar{z}_i & m_i \end{bmatrix} \tag{A.2}$$

[1] For the derivation of these equations, refer to Fu et. al. (1985).

This is generally written as a second order matrix differential equation, given as

$$\{\tau\} = [D(q)]\{\ddot{q}\} + \{h(q,\dot{q})\} + \{c(q)\} \qquad \text{(A.3)}$$

where

$\{\tau\} = \{\tau(t)\}$ = n x 1 generalised torque vector applied at joints

$\{q\} = \{q(t)\}$ = n x 1 vector of the joint variables of the robot arm

$\{\dot{q}\} = \{\dot{q}(t)\}$ = nx1 vector of the joint velocities of the robot arm

$\{\ddot{q}\} = \{\ddot{q}(t)\}$ =nx1 vector of the joint accelerations of the robot arm

$\{D(q)\}$ = n x n inertial acceleration related symmetric matrix, where

$$[D]_{ik} = \sum_{j=max(i,k)}^{n} Tr\,([U]_{jk}[J]_j[U]_{ji}^T) \begin{cases} i = 1 \ to \ n \\ k = 1 \ to \ n \end{cases}$$

$\{h(q,\dot{q})\}$ = n x 1 nonlinear coriolis and centrifugal force vector, where

$$h_i = \sum_{k=1}^{n}\sum_{m=1}^{n} h_{ikm}\dot{q}_k\dot{q}_m \quad i = 1,2,....n$$

where

$$h_{ikm} = \sum j = max(i,k,m)^n \; Tr\,([U]_{jkm}[J]_j[U]_{ji}^T) \quad i,k,m = 1,2,....n$$

$\{c(q)\}$ = n x 1 gravity loading force vector where,

$$\{c\}_i = \sum_{j=i}^{n}(-m_j g[U]_{ji}.{}^j\bar{r}_j) \quad i = 1,2,....n$$

A.2 Recursive Lagrange Equations Using 4×4 D-H Transformation Matrices

Hollerbach introduced recursion [2] in the Lagrange Equations using 4 × 4 D-H transformation matrices and the equations are given as a set of forward and backward recursion equations, as below

[2] For derivation of these equations refer to Hollerbach (1980).

FORWARD RECURSION

$$[T]_0 = Identity; \quad [\dot{T}]_0 = [0]; \quad [\ddot{T}]_0 = [0]$$

$${}^0[T]_i = {}^0[T]_{i-1}.{}^{i-1}[T]_i$$

$${}^0[\dot{T}]_i = {}^0[\dot{T}]_{i-1}.{}^{i-1}[T]_i + {}^0[T]_{i-1}.\frac{\partial^{i-1}[T]_i}{\partial q_i}\dot{q}_i$$

$${}^0[\ddot{T}]_i = {}^0[\ddot{T}]_{i-1}.{}^{i-1}[T]_i + 2{}^0[\dot{T}]_{i-1}.\frac{\partial^{i-1}[T]_i}{\partial q_i}\dot{q}_i + {}^0[T]_{i-1}.\frac{\partial^2\,{}^{i-1}[T]_i}{\partial q_i^2}\dot{q}_i^2 + {}^0[T]_{i-1}.\frac{\partial^{i-1}[T]_i}{\partial q_i}\ddot{q}_i$$

BACKWARD RECURSION

$$[D]_{n+1} = [0]; \quad [c]_{n+1} = [0]$$

$$[D]_i = [J]_i\,{}^0[\ddot{T}]_i^T + {}^0[T]_{i+1}.[D]_{i+1}$$

$$[c]_i = m_i.{}^0[T]_i + {}^0[T]_{i+1}.[c]_{i+1}$$

$$\{\tau\}_i = \mathrm{Tr}\left(\frac{\partial^0[T]_i}{\partial q_i}[D]_i\right) - \{g\}^T\frac{\partial^0[T]_i}{\partial q_i}[c]_i$$

A.3 Recursive Lagrange Equations Using 3×3 Rotation Transformation Matrices

The recursive equations can also be written using the 3 × 3 rotation transformation matrices and the position vector p_i, to improve the computational efficiency of the algorithm (Hollerbach 1980). The forward recursion is similar to that of the previous formulation, except that, in this case, the $[R]_{3\times3}$ matrices are used instead of the $[T]_{4\times4}$ matrices as given below

$$[R]_i^0 = [R]_{i-1}^0.[R]_i^{i-1}$$

$$[\dot{R}]_i^0 = [\dot{R}]_{i-1}^0 . [R]_i^{i-1} + [R]^0 i - 1 . \frac{\partial^{i-1}[R]_i}{\partial q_i} \dot{q}_i$$

$$[\ddot{R}]_i^0 = [\ddot{R}]_{i-1}^0 . [R]_i^{i-1} + 2[\dot{R}]_{i-1}^0 . \frac{\partial^{i-1}[R]_i}{\partial q_i} \dot{q}_i + {}^0[R]_{i-1} . \frac{\partial^{2\ i-1}[R]_i}{\partial q_i^2} \dot{q}_i^2 + {}^c[R]_{i-1} . \frac{\partial^{i-1}[R]_i}{\partial q_i} \ddot{q}_i$$

The second derivative of the position vector ($\{r\}_i^0$) of the origin of the link frame from the origin of the ba·e frame is also computed recursively as

$$\{\ddot{r}\}_i^0 = \{\ddot{r}\}_{i-1} + [\ddot{R}]_i^0 \{p\}_{i-1}^i$$

If $\{s\}_i$ is the position vector of the CG of the ith link in its own link co-ordinate frame, than the backward recursion is written as follows.

$$[D]_{n+1} = [0]; \quad [c]_{n+1} = [0]$$

$$[D]_i = [J]_i \, [\ddot{R}]_i^{0T} + [R]_{i+1}^0 [D]_{i+1} + \{r\}_{i+1}^i \{e\}_{i+1} + \frac{\{s\}_i}{m_i} \{\ddot{r}\}_i^{0T}$$

where $\{e\}_i$ is written as

$$\{e\}_i = [R]_{i+1}^i . \{e\}_{i+1} + m_i \{\ddot{r}\}_i^{0T} + \frac{\{s\}_i}{m_i} \{\ddot{r}\}_i^{0T}$$

The recursion for $[c]_i$ is the same as the previous one but using the 3x3 rotation matrix given as

$$[c]_i = m_i . {}^0[R]_i + {}^0 \, [R]_{i+1} . [c]_{i+1}$$

The torque vector is now written as

$$\{\tau\}_i = \mathrm{Tr} \left(\frac{\partial [R]_i^0}{\partial q_i} [D]_i \right) - \{g\}^T \frac{\partial [R]_i^0}{\partial q_i} [c]_i$$

Appendix B

Derivation of Newton-Euler Algorithm

The Newton-Euler formulation for robot dynamic problems, assumes the links to be rigid bodies obeying Newton's equation and the Euler's equation. The kinematic parameters, such as the position vector of the joint locations and the center of gravity (CG) of the links, and the dynamic parameters, namely the moment of inertia (MI) tensor about the CG, are assumed to be known. For computational efficiency all the vectors and the inertia tensors are referred to the corresponding link coordinate frame.

Fig. 2.6, shows three consecutive links in the kinematic chain of an arbitrary manipulator. Referring to the figure, $\{F\}_i$ and $\{N\}_i$ are the inertial forces and moments acting at the CG of the link i and $\{f\}_i$ and $\{n\}_i$ are the reaction forces and moments acting at the joint i. $\{s_i\}$ is the position vector of the CG of the ith link and the $\{p\}_{i-1}$ is the position vector of O_i, origin of the ith frame, referred in the i-1th frame, as defined in Eq. (2.8). The problem of inverse dynamics is to compute the joint torques/forces (torque for a revolute joint and force for a prismatic joint) given the relative position, velocity and acceleration of each link

($q_i, \dot{q}_i$ and $\ddot{q}_i$ for i=1 to n). For a revolute joint these are angular parameters and for a prismatic joint they refer to linear displacement, linear velocity and linear acceleration of the CG of the link. The local link coordinate frames are assigned in such a way that the local z axis is aligned with the direction of the relative motion at the joint. Hence for a revolute joint, the relative angular velocity and the relative angular acceleration of the ith link with respect to the previous link will be in the z direction and hence these vectors can be written as

$$\{\omega\}_{i/i-1} = \begin{Bmatrix} 0 \\ 0 \\ \dot{\theta} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix} \dot{\theta} = \{\hat{z}\}\dot{\theta} \tag{B.1}$$

$$\{\alpha\}_{i/i-1} = \begin{Bmatrix} 0 \\ 0 \\ \ddot{\theta} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix} \ddot{\theta} = \{\hat{z}\}\ddot{\theta} \tag{B.2}$$

where $\theta, \dot{\theta}, \ddot{\theta}$ refer to the rotational parameters. In case of a prismatic joint, the relative linear velocity and the relative linear acceleration of the CG of the link can be written as

$$\{\omega\}_{i/i-1} = \begin{Bmatrix} 0 \\ 0 \\ \dot{q} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix} \dot{q} = \{\hat{z}\}\dot{q} \tag{B.3}$$

$$\{\alpha\}_{i/i-1} = \begin{Bmatrix} 0 \\ 0 \\ \ddot{q} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix} \ddot{q} = \{\hat{z}\}\ddot{q} \tag{B.4}$$

In order to make the algorithm, a general purpose one, we will denote the relative joint displacements, rotational or linear, by q and similarly its first and second derivatives.

For a revolute joint, the angular velocity of one link can be computed with the knowledge of the angular velocity of the previous link. If the angular velocity of the i-1th link is known then the angular velocity of the ith link can be written as

$$\{\omega\}_i = [R]_i^T \{\omega\}_{i-1} + \{\omega\}_{i/i-1} \tag{B.5}$$

Here the first term on the left hand side of the equation refer to the angular velocity of the base projected into the coordinate frame of link 1 and the second term refer to the relative angular velocity. Substituting Eq. (B.1), we can write this as

$$\{\omega\}_i = [R]_i^T \{\omega\}_{i-1} + \{\hat{z}\}\dot{q} \tag{B.6}$$

where $\{\hat{z}\}$ denotes the unit vector along the z axis. Differentiating (B.6), we can write the expression for the angular acceleration of the link i. Noting that $\{\omega\}$ is a rotating vector (Shames 1967), we can write

$$\{\alpha\}_i = [R]_i^T \{\alpha\}_i + \{\hat{z}\}\ddot{q} + [R]_i^T \{\omega\}_{i-1} \times \{\hat{z}\}\dot{q} \tag{B.7}$$

The last term in the left hand side of the above equation refers to the coriolis acceleration. If the joint is prismatic, then there is no relative rotation between the two adjacent links and hence the angular velocity and the angular acceleration remain the same in magnitude. The these vectors projected in the ith link frame can be written as

$$\{\omega\}_i = [R]_i^T \{\omega\}_{i-1} \tag{B.8}$$

$$\{\alpha\}_i = [R]_i^T \{\alpha\}_i \tag{B.9}$$

In order to compute the forces and moments acting at the CG of the link, we need to determine the linear acceleration of the CG. This is determined by first computing the linear acceleration of the origin of the link coordinate frame and subsequently computing the linear acceleration of the CG. Referring to Fig. 2.8, the acceleration of a point can be written as (Shames 1967)

$$\{a\}_P = \{a\}_O + \{\alpha\} \times \{r\}_{OP} + \{\omega\} \times \{\omega\} \times \{r\}_{OP} + \{\ddot{r}\}_{P/O} + 2\{\omega\} \times \{\dot{r}\}_{P/O} \tag{B.10}$$

where $\{a\}_P$ and $\{a\}_O$ are the linear accelerations of points P and O, $\{r\}_{OP}$ is the position vector of P, $\{\omega\}$ and $\{\alpha\}$ are the angular velocity and the angular acceleration of $\{r\}_{OP}$, $\{\dot{r}\}_{P/O}$ and $\{\ddot{r}\}_{P/O}$ refer to the linear velocity and acceleration of point P with respect to O. Applying this to a prismatic joint, we can write as

$$\begin{aligned}\{a\}_i^{i-1} &= \{a\}_{i-1} + \{\alpha\}_{i-1} \times \{p\}_{i-1} + \{\omega\}_{i-1} \times \{\omega\}_{i-1} \times \{p\}_{i-1} + \{\hat{z}\}\ddot{q}_i \\ &\quad +2\,[R]_i^T \{\omega\}_{i-1} \times \{\hat{z}\}\dot{q}_i\end{aligned} \tag{B.11}$$

For a revolute joint, there is no relative translation between the origin of the ith link and that of the i-1th link and hence the above expression will reduce to

$$\{a\}_i^{i-1} = \{a\}_{i-1} + \{\alpha\}_{i-1} \times \{p\}_{i-1} + \{\omega\}_{i-1} \times \{\omega\}_{i-1} \times \{p\}_{i-1} \tag{B.12}$$

These can be projected to the the ith frame by pre-multiplying with the rotation transformation matrix, $[R]_i^T$, and written as

$$\begin{aligned}\{a\}_i &= [R]_i^T(\{a\}_{i-1} + \{\alpha\}_{i-1} \times \{p\}_{i-1} + \{\omega\}_{i-1} \times \{\omega\}_{i-1} \times \{p\}_{i-1} + \{\hat{z}\}\ddot{q}_i \\ &\quad +2\,[R]_i^T \{\omega\}_{i-1} \times \{\hat{z}\}\dot{q}_i)\end{aligned} \tag{B.13}$$

for a prismatic joint and

$$\{a\}_i = [R]_i^T(\{a\}_{i-1} + \{\alpha\}_{i-1} \times \{p\}_{i-1} + \{\omega\}_{i-1} \times \{\omega\}_{i-1} \times \{p\}_{i-1}) \tag{B.14}$$

for a revolute joint.

Similarly the linear acceleration of the CG can be computed. Noting that there is no relative motion between the CG and the origin of the ith link, we can write, for revolute as well as prismatic joint

$$\{a\}_{ci} = \{a\}_i + \{\alpha\}_i \times \{s\}_i + \{\omega\}_i \times \{\omega\}_i \times \{s\}_i) \tag{B.15}$$

Eqs. (B.6) to (B.15) compute the kinematic parameters of a link with the knowledge of those of the previous link. Since the velocity and acceleration of the base is commonly known these equations can be used recursively to compute the kinematics of the other links in the chain by starting with link1 and moving successively, link by link, outward to link n. This is termed as the forward recursion in the NE algorithm. Since the kinematics of the links are known, the inertial forces and moments can be computed using the Newton's equation and the Euler's equation respectively.

$$\{F\}_i = m_i\{a\}_{ci} \tag{B.16}$$

$$\{N\}_i = [I]\{\alpha\}_i + \{\omega\}_i \times [I]\{\omega\}_i \tag{B.17}$$

Note that the second term of Eq. (B.17) refers to the gyroscopic moment of the link, which does not appear for a two dimensional problem.

At this stage we can apply the equilibrium conditions for each link starting with the outermost link (i=n), and move inward, link by link to compute the joint forces and moments. Considering the equilibrium of link i shown in Fig. 2.6 we can write the equilibrium equations as

$$\sum\{F\} = m_i\{a\}_{ci} \tag{B.18}$$

$$\sum\{N\} = [I]_i\{\alpha\}_i + \{\omega\}_i \times [I]_i\{\omega\}_i \tag{B.19}$$

where {F} and {N} denote the external forces and moments acting on the link. This directly yields the solution for the joint force $\{f\}_i$ and the joint moment $\{n\}_i$ as

$$\{f\}_i = \{F\}_i + [R]_{i+1}\{f\}_{i+1} \tag{B.20}$$

$$\{n\}_i = [R]_{i+1}\{n\}_{i+1} + \{N\}_i + \{s\}_i \times \{F\}_i + \{p\}_i \times ([R]_{i+1}\{f\}_{i+1}) \tag{B.21}$$

The forces and moments exerted at the end-effector are normally equal to zero and this can be used to compute the joint forces at the nth joint. For n-1th link, the joint forces and moments at the end where the nth joint is located, will be equal and opposite to the joint forces and moments computed for the nth link. Using this knowledge, the joint forces and moments at the other end can be computed using Eqs. (B.20) and (B.21) Thus we can proceed down the chain successively from i=n to i=1 and determine the force and moment acting at each joint.

The alignment of the local z axis along the direction of motion of the link facilitates easy computation of the actuator torques/forces which are the z components of the joint moments/forces. For rotational joints, the vector moment is projected along the axis of rotation to yield the joint torque. For sliding joints, the vector force is projected along the sliding axis to yield the joint force. The other components of the force and moment are generated by the structure and bearings of the device. Thus we can write

$$\tau_i = \{\hat{z}\}.\{f_i\} \qquad for\ a\ prismatic\ joint \qquad (B.22)$$

$$\tau_i = \{\hat{z}\}.\{n_i\} \qquad for\ a\ revolute\ joint \qquad (B.23)$$

Appendix C

Schedules for Inverse Dynamics Computation

The schedule for computation of the inverse dynamic subtasks are given in this appendix. Tables C.1 to C.8 give the schedule for the computation of the inverse dynamic tasks of a six-link manipulator given in Table 4.3. As mentioned in Chapter 4, the subtasks are denoted by a three digit number. The first number corresponds to the link number and the subsequent two numbers correspond to the subtask number as given by Table 4.2.

The computation of customized inverse dynamics of Stanford manipulator given in Table 4.9 is scheduled in Tables C.9 to C.15.

	Processor	
Level	1	2
1	110	229
2	121	231
3	201	232
4	123	234
5	125	236
6	204	238
7	206	241
8	208	243
9	213	129
10	301	131
11	209	132
12	211	134
13	216	136
14	303	138
15	305	141
16	218	143
17	220	221
18	222	306
19	307	308
20	312	313
21	314	402
22	223	224
23	225	309
24	310	311
25	315	316
26	317	403
27	404	405
28	318	319
29	320	321
30	322	406
31	407	408
32	412	413
33	414	501
34	502	323
35	324	325
36	409	410
37	411	415
38	416	417
39	503	504

	Processor	
Level	1	2
40	505	418
41	419	420
42	421	422
43	506	507
44	508	512
45	513	514
46	601	602
47	423	424
48	425	509
49	510	511
50	515	516
51	517	603
52	604	605
53	518	519
54	520	521
55	522	606
56	607	608
57	612	613
58	614	523
59	524	525
60	609	610
61	611	615
62	616	617
63	618	619
64	620	621
65	622	623
66	624	625
67	626	627
68	628	629
69	636	637
70	638	526
71	641	642
72	643	527
73	528	529
74	530	531
75	532	533
76	534	535
77	536	537
78	538	539

	Processor	
Level	1	2
79	540	426
80	541	542
81	543	427
82	428	429
83	430	431
84	432	433
85	434	435
86	436	437
87	438	439
88	440	326
89	441	442
90	443	327
91	328	329
92	330	331
93	332	333
94	334	335
95	336	337
96	338	339
97	340	226
98	341	342
99	343	226
100	227	228
101	229	230
102	231	239
103	232	233
104	234	235
105	236	237
106	238	240
107	241	242
108	243	126
109	129	130
110	131	139
111	132	133
112	134	135
113	136	137
114	138	140
115	141	142
116	143	0

Table C.1: Two Processor Schedule of Inverse Dynamics of a Six-Link Manipulator

Level	Processor		
	1	2	3
1	110	120	121
2	122	201	202
3	123	124	125
4	203	204	205
5	206	207	208
6	212	213	214
7	301	302	209
8	210	211	215
9	216	217	303
10	304	305	218
11	219	220	221
12	222	306	307
13	308	312	313
14	314	401	402
15	223	224	225
16	309	310	311
17	315	316	317
18	403	404	405
19	318	319	320
20	321	322	406
21	407	408	412
22	413	414	501
23	502	323	324
24	325	409	410
25	411	415	416
26	417	503	504

Level	Processor		
	1	2	3
27	505	418	419
28	420	421	422
29	506	507	508
30	512	513	514
31	601	602	423
32	424	425	509
33	510	511	515
34	516	517	603
35	604	605	518
36	519	520	521
37	522	606	607
38	608	612	613
39	614	523	524
40	525	609	610
41	611	615	616
42	617	620	621
43	618	619	622
44	623	624	625
45	626	627	628
46	629	636	637
47	638	526	527
48	641	642	643
49	526	527	528
50	529	530	531
51	532	533	534
52	535	536	537

Level	Processor		
	1	2	3
53	538	539	540
54	541	542	543
55	426	427	428
56	429	430	431
57	432	433	434
58	435	436	437
59	438	439	440
60	441	442	443
61	326	327	328
62	329	330	331
63	332	333	334
64	335	336	337
65	338	339	340
66	341	342	343
67	226	227	228
68	229	230	231
69	232	233	234
70	235	236	237
71	238	239	240
72	241	242	243
73	126	0	0
74	129	130	131
75	132	133	134
76	135	136	137
77	138	139	140
78	141	142	143

Table C.2: Three Processor Schedule of Inverse Dynamics of a Six-Link Manipulator

Level	Processor Loading			
	1	2	3	4
1	110	120	121	122
2	201	202	123	124
3	125	203	204	205
4	206	207	208	212
5	213	214	301	302
6	209	210	211	215
7	216	217	303	304
8	305	218	219	220
9	221	222	306	307
10	308	312	313	314
11	401	402	223	224
12	225	309	310	311
13	315	316	317	403
14	404	405	318	319
15	320	321	322	406
16	407	408	412	413
17	414	501	502	323
18	324	325	409	410
19	411	415	416	417
20	503	504	505	418
21	419	420	421	422
22	506	507	508	512
23	513	514	601	602
24	423	424	425	509
25	510	511	515	516
26	517	603	604	605
27	518	519	520	521
28	522	606	607	608
29	612	613	614	523

Level	Processor Loading			
	1	2	3	4
30	524	525	609	610
31	611	615	616	617
32	618	619	620	621
33	622	526	527	528
34	623	624	625	529
35	626	627	628	629
36	636	637	638	530
37	641	642	643	531
38	532	533	534	535
39	536	537	538	539
40	540	427	428	429
41	541	542	543	426
42	430	431	436	437
43	432	433	434	435
44	438	439	440	327
45	441	442	443	326
46	328	329	330	331
47	332	333	334	335
48	336	337	338	339
49	340	227	228	229
50	341	342	343	226
51	230	231	236	237
52	232	233	234	235
53	238	239	240	126
54	241	242	243	136
55	129	130	131	137
56	132	133	134	135
57	138	139	140	0
58	141	142	143	0

Table C.3: Four Processor Schedule of Inverse Dynamics of a Six-Link Manipulator

Level	Processor Loading				
	1	2	3	4	5
1	110	120	121	122	201
2	202	123	124	125	203
3	204	205	301	302	212
4	206	207	208	213	214
5	209	210	211	215	216
6	217	303	304	305	401
7	218	219	220	221	222
8	306	307	308	312	313
9	314	402	223	224	225
10	309	310	311	315	316
11	317	403	404	405	501
12	318	319	320	321	322
13	406	407	408	412	413
14	414	502	323	324	325
15	409	410	411	415	416
16	417	503	504	505	601
17	418	419	420	421	422
18	506	507	508	512	513
19	514	602	423	424	425
20	509	510	511	515	516
21	517	603	604	605	520
22	518	519	521	522	606
23	607	608	612	613	614
24	523	524	525	609	610

Level	Processor Loading				
	1	2	3	4	5
25	611	615	616	617	528
26	618	619	620	621	622
27	623	624	625	529	427
28	626	627	628	629	428
29	636	637	638	530	531
30	641	642	643	526	527
31	532	533	534	535	536
32	537	538	539	540	430
33	541	542	543	426	429
34	431	327	328	329	227
35	432	433	434	435	436
36	437	438	439	440	330
37	441	442	443	326	331
38	332	333	334	335	336
39	337	338	339	340	228
40	341	342	343	226	229
41	230	231	236	237	238
42	232	233	234	235	239
43	240	126	129	130	131
44	241	242	243	0	0
45	132	133	134	135	136
46	137	138	139	140	0
47	141	142	143	0	0

Table C.4: Five Processor Schedule of Inverse Dynamics of a Six-Link Manipulator

Level	Processor Loading					
	1	2	3	4	5	6
1	110	120	121	122	201	202
2	123	124	125	203	204	205
3	206	207	208	212	213	214
4	301	302	209	210	211	215
5	216	217	303	304	305	401
6	218	219	220	221	222	306
7	307	308	312	313	314	402
8	223	224	225	309	310	311
9	315	316	317	403	404	405
10	318	319	320	321	322	406
11	407	408	412	413	414	501
12	502	323	324	325	409	410
13	411	415	416	417	503	504
14	505	418	419	420	421	422
15	506	507	508	512	513	514
16	601	602	423	424	425	509
17	510	511	515	516	517	603
18	604	605	518	519	520	521
19	522	606	607	608	612	613
20	614	523	524	525	609	610
21	611	615	616	617	527	528
22	618	619	620	621	622	529
23	623	624	625	427	428	429
24	626	627	628	629	327	328
25	636	637	638	326	329	226
26	641	642	643	526	530	531
27	532	533	534	535	536	537
28	538	539	540	426	430	431
29	541	542	543	436	437	438
30	432	433	434	435	439	440
31	441	442	443	330	331	227
32	332	333	334	335	336	337
33	338	339	340	228	229	230
34	341	342	343	231	236	237
35	232	233	234	235	238	239
36	240	126	129	130	131	0
37	241	242	243	136	137	138
38	132	133	134	135	139	140
39	141	142	143	0	0	0

Table C.5: Six Processor Schedule of Inverse Dynamics of a Six-Link Manip-

Level	Processor Loading						
	1	2	3	4	5	6	7
1	110	120	121	122	201	202	129
2	123	124	125	203	204	205	126
3	206	207	208	212	213	214	301
4	302	209	210	211	215	216	217
5	303	304	305	218	219	220	221
6	222	306	307	308	312	313	314
7	401	402	223	224	225	309	310
8	311	315	316	317	403	404	405
9	318	319	320	321	322	406	407
10	408	412	413	414	501	502	323
11	324	325	409	410	411	415	416
12	417	503	504	505	420	421	422
13	418	419	506	507	508	512	513
14	514	601	602	423	424	425	509
15	510	511	515	516	517	603	604
16	605	518	519	520	521	522	614
17	606	607	608	612	613	523	524
18	525	609	610	611	615	616	617
19	618	619	620	621	622	526	527
20	623	624	625	528	529	536	537
21	626	627	628	629	538	426	427
22	636	637	638	530	531	428	429
23	641	642	643	436	437	438	326
24	532	533	534	535	539	540	327
25	541	542	543	430	431	328	329
26	432	433	434	435	439	440	336
27	441	442	443	330	331	337	338
28	332	333	334	335	336	339	340
29	341	342	343	226	227	228	229
30	230	231	236	237	238	136	137
31	232	233	234	235	239	240	138
32	241	242	243	130	131	0	0
33	132	133	134	135	139	140	0
34	141	142	143	0	0	0	0

Table C.6: Seven Processor Schedule of Inverse Dynamics of a Six-Link Manipulator

Level	Processor Loading							
	1	2	3	4	5	6	7	8
1	110	120	121	122	201	202	129	0
2	123	124	125	203	204	205	0	0
3	206	207	208	212	213	214	301	302
4	209	210	211	215	216	217	303	304
5	305	218	219	220	221	222	306	307
6	308	312	313	314	401	402	223	224
7	225	309	310	311	315	316	317	403
8	404	405	318	319	320	321	322	126
9	406	407	408	412	413	414	501	502
10	323	324	325	409	410	411	415	416
11	417	503	504	505	418	419	420	421
12	422	506	507	508	512	513	514	601
13	602	423	424	425	509	510	511	515
14	516	517	603	604	605	612	613	614
15	518	519	520	521	522	606	607	608
16	609	610	611	615	616	617	226	227
17	618	619	620	621	622	523	524	525
18	623	624	625	228	229	136	137	138
19	626	627	628	629	526	527	528	529
20	636	637	638	536	537	538	426	326
21	641	642	643	530	531	436	336	226
22	532	533	534	535	539	540	236	0
23	541	542	543	427	428	429	430	431
24	432	433	434	435	437	438	439	440
25	441	442	443	327	328	329	330	331
26	332	333	334	335	337	338	339	340
27	341	342	343	227	228	229	230	231
28	232	233	234	235	237	238	239	240
29	241	242	243	126	129	130	131	0
30	132	133	134	135	139	140	0	0
31	141	142	143	0	0	0	0	0

Table C.7: Eight Processor Schedule of Inverse Dynamics of a Six-Link Manipulator

Level	Processor Loading								
	1	2	3	4	5	6	7	8	9
1	110	120	121	122	201	202	0	0	0
2	123	124	125	203	204	205	0	0	0
3	206	207	208	212	213	214	301	302	0
4	209	210	211	215	216	217	303	304	305
5	218	219	220	221	222	306	307	308	312
6	313	314	401	402	223	224	225	309	310
7	311	315	316	317	403	404	405	320	321
8	318	319	322	406	407	408	412	413	414
9	501	502	323	324	325	409	410	411	415
10	416	417	503	504	505	420	421	422	512
11	418	419	506	507	508	513	514	601	602
12	423	424	425	509	510	511	515	516	517
13	603	604	605	518	519	520	521	522	0
14	606	607	608	612	613	614	0	0	0
15	523	524	525	609	610	611	615	616	617
16	618	619	620	621	622	0	0	0	0
17	623	624	625	0	0	0	0	0	0
18	626	627	628	629	0	0	0	0	0
19	636	637	638	0	0	0	0	0	0
20	641	642	643	526	527	528	529	530	531
21	532	533	534	535	536	537	538	539	540
22	541	542	543	426	427	428	429	430	431
23	432	433	434	435	436	437	438	439	440
24	441	442	443	326	327	328	329	330	331
25	332	333	334	335	336	337	338	339	340
26	341	342	343	226	227	228	229	230	231
27	232	233	234	235	236	237	238	239	240
28	241	242	243	126	127	128	129	130	131
29	132	133	134	135	136	137	138	139	140
30	141	142	143	0	0	0	0	0	0

Table C.8: Nine Processor Schedule of Inverse Dynamics of a Six-Link Manipulator

Level	Processor	
	1	2
1	110	120
2	121	122
3	201	202
4	123	124
5	125	203
6	204	205
7	206	207
8	208	216
9	209	210
10	211	217
11	303	218
12	219	306
13	401	402
14	224	225
15	403	404
16	405	318
17	319	406
18	407	408
19	501	502
20	324	325
21	409	410
22	411	503
23	504	505
24	418	419
25	422	506
26	507	508

Level	Processor	
	1	2
27	601	602
28	423	424
29	425	509
30	510	511
31	603	604
32	605	518
33	519	522
34	606	607
35	608	523
36	524	525
37	609	610
38	611	618
39	619	528
40	624	625
41	626	627
42	628	629
43	638	529
44	641	642
45	643	526
46	530	531
47	527	532
48	537	538
49	539	540
50	541	542
51	543	426
52	427	428

Level	Processor	
	1	2
53	429	430
54	431	439
55	432	437
56	438	440
57	441	442
58	443	326
59	327	328
60	330	331
61	329	332
62	338	339
63	340	226
64	341	342
65	343	227
66	228	229
67	230	231
68	232	233
69	235	238
70	239	240
71	241	242
72	243	126
73	130	131
74	127	132
75	135	137
76	138	139
77	140	0
78	141	142
79	143	0

Table C.9: Two Processor Schedule for Customized Inverse Dynamics of Stanford Manipulator

	Processor Loading		
Level	1	2	3
1	110	120	121
2	122	201	202
3	123	124	125
4	203	204	205
5	206	207	208
6	209	210	211
7	216	217	303
8	218	219	306
9	401	402	224
10	225	403	404
11	405	318	319
12	406	407	408
13	501	502	324
14	325	409	410
15	411	503	504
16	505	418	419
17	422	506	507
18	508	601	602
19	423	424	425
20	509	510	511
21	603	604	605
22	518	519	522
23	606	607	608
24	523	524	525
25	609	610	611
26	618	619	526

	Processor Loading		
Level	1	2	3
27	624	625	527
28	626	627	628
29	629	528	529
30	638	426	326
31	641	642	643
32	530	531	427
33	532	537	538
34	539	540	428
35	541	542	543
36	429	430	431
37	432	437	438
38	439	440	327
39	441	442	443
40	329	330	331
41	328	332	338
42	339	340	226
43	341	342	343
44	227	228	229
45	230	231	126
46	232	233	235
47	238	239	240
48	241	242	243
49	127	130	131
50	132	135	137
51	138	139	140
52	141	142	143

Table C.10: Three Processor Schedule for Customized Inverse Dynamics of Stanford Manipulator

	Processor Loading			
Level	1	2	3	4
1	110	120	121	122
2	201	202	123	124
3	125	203	204	205
4	206	207	208	126
5	209	210	211	216
6	217	303	401	402
7	218	219	306	403
8	224	225	404	405
9	318	319	406	407
10	408	501	502	324
11	325	409	410	411
12	503	504	505	418
13	419	422	506	507
14	508	601	602	423
15	424	425	509	510
16	511	603	604	605
17	518	519	522	606
18	607	608	523	524
19	525	609	610	611
20	618	619	526	527

	Processor Loading			
Level	1	2	3	4
21	624	625	528	529
22	626	627	628	629
23	638	537	538	426
24	641	642	643	530
25	531	427	428	429
26	532	539	540	326
27	541	542	543	430
28	431	438	327	328
29	432	437	439	440
30	441	442	443	329
31	330	331	226	227
32	332	338	339	340
33	341	342	343	228
34	229	230	231	129
35	232	233	235	238
36	239	240	130	131
37	241	242	243	137
38	135	138	139	140
39	141	142	143	0

Table C.11: Four Processor Schedule for Customized Inverse Dynamics of Stanford Manipulator

	Processor Loading				
Level	1	2	3	4	5
1	110	121	122	201	202
2	120	203	204	205	129
3	206	207	208	123	124
4	209	210	211	216	217
5	303	125	218	219	401
6	306	402	224	225	227
7	403	404	405	318	319
8	406	407	408	501	502
9	324	325	409	410	411
10	503	504	505	418	419
11	506	507	508	601	602
12	509	510	511	603	422
13	604	605	518	519	423
14	522	606	607	608	424
15	524	525	609	610	611
16	618	619	523	425	526
17	624	625	527	528	529
18	626	627	628	629	426
19	638	427	428	428	326
20	327	328	329	226	126
21	229	238	138	228	0
22	641	642	643	530	531
23	532	537	538	539	540
24	541	542	543	430	431
25	432	437	438	439	440
26	441	442	443	330	331
27	332	338	339	340	0
28	341	342	343	230	231
29	232	233	235	239	240
30	241	242	243	130	131
31	132	135	137	139	140
32	141	142	143	0	0

Table C.12: Five Processor Schedule for Customized Inverse Dynamics of Stanford Manipulator

	Processor Loading					
Level	1	2	3	4	5	6
1	110	120	121	122	201	202
2	123	124	125	203	204	205
3	206	207	208	0	0	0
4	209	210	211	216	217	303
5	218	219	306	401	402	0
6	224	225	403	404	405	0
7	318	319	406	407	408	501
8	502	324	325	0	0	0
9	409	410	411	503	504	505
10	418	419	422	506	507	508
11	601	602	423	424	425	0
12	509	510	511	603	604	605
13	518	519	522	606	607	608
14	523	524	525	609	610	611
15	618	619	526	527	528	426
16	624	625	427	428	326	327
17	626	627	628	629	328	0
18	638	226	227	228	126	0
19	641	642	643	529	530	531
20	532	537	538	539	540	0
21	541	542	543	429	430	431
22	432	437	438	439	440	0
23	441	442	443	329	330	331
24	332	338	339	340	0	0
25	341	342	343	229	230	231
26	232	233	235	238	239	240
27	241	242	243	127	130	131
28	132	135	137	138	139	140
29	141	142	143	0	0	0

Table C.13: Six Processor Schedule for Customized Inverse Dynamics of Stanford Manipulator

	Processor Loading						
Level	1	2	3	4	5	6	7
1	110	120	121	122	201	202	0
2	123	124	125	203	204	205	0
3	206	207	208	0	0	0	0
4	209	210	211	216	217	303	0
5	218	219	306	401	402	0	0
6	224	225	403	404	405	0	0
7	318	319	406	407	408	501	502
8	325	409	410	411	503	504	505
9	418	419	506	507	508	601	602
10	509	510	511	603	604	605	0
11	518	519	522	606	607	608	324
12	523	524	525	609	610	611	422
13	618	619	423	424	425	526	527
14	624	625	426	427	326	327	226
15	626	627	628	629	227	0	0
16	638	0	0	0	0	0	0
17	641	642	643	528	529	530	531
18	532	537	538	539	540	0	0
19	541	542	543	428	429	430	431
20	432	437	438	439	440	0	0
21	441	442	443	328	329	330	331
22	332	338	339	340	0	0	0
23	341	342	343	228	229	230	231
24	232	233	235	238	239	240	0
25	241	242	243	126	127	130	131
26	132	135	137	138	139	140	0
27	141	142	143	0	0	0	0

Table C.14: Seven Processor Schedule for Customized Inverse Dynamics of Stanford Manipulator

	Processor Loading							
Level	1	2	3	4	5	6	7	8
1	110	120	121	122	201	202	129	0
2	123	124	125	203	204	205	0	0
3	206	207	208	126	0	0	0	0
4	209	210	211	216	217	303	137	138
5	218	219	306	401	402	0	0	0
6	224	225	403	404	405	0	0	0
7	318	319	406	407	408	501	502	226
8	324	325	409	410	411	503	504	505
9	418	419	422	506	507	508	601	602
10	424	425	509	510	511	603	604	605
11	518	519	522	606	607	608	423	0
12	523	524	525	609	610	611	0	0
13	618	619	526	426	326	0	0	0
14	624	625	0	0	0	0	0	0
15	626	627	628	629	0	0	0	0
16	638	0	0	0	0	0	0	0
17	641	642	643	527	528	529	530	531
18	532	537	538	539	540	0	0	0
19	541	542	543	427	428	429	430	431
20	432	437	438	439	440	0	0	0
21	441	442	443	327	328	329	330	331
22	332	338	339	340	0	0	0	0
23	341	342	343	227	228	229	230	231
24	232	233	235	238	239	240	0	0
25	241	242	245	130	131	0	0	0
26	132	135	139	140	0	0	0	0
27	141	142	143	0	0	0	0	0

Table C.15: Eight Processor Schedule for Customized Inverse Dynamics of Stanford Manipulator

Appendix D

Program Listing

D.1 Numeric Programs for Inverse Dynamics

The inverse dynamics of PUMA-560 (3 DOF) manipulator is computed numerically using two approaches:

1. Lagrange formulation (D.1.1)

2. Newton-Euler formulation (D.1.2)

These programs are written for MATLAB software. The position, velocity and acceleration of each link can be defined in the input section and the program computes the torques/forces at the joints.

In order to compute the torque profiles for a given trajectory, the inverse kinematics has to be solved first. A program using MATLAB to compute the inverse kinematics for the 3 DOF PUMA-560 manipulator is given in Section D.1.3.

D.1.1 Inverse Dynamics using Lagrange Equations

```
%general parameters for puma 560

m2=17.4
m3=6.05

j1=[0.0058 0 0 0;0 0.0058 0 0;0 0 -0.0058 0;0 0 0 0]
j2=[0.6984 0 0 -1.1832;0 0.0340 0 0;0 0 0.2416 0;-1.1832 0 0 17.4]
j3=[0.0039 0 0 0;0 0.0335 0 -0.4235;0 0 0.1432 0;0 -0.4235 0 6.05]

p1=[0;0;0]
p2=[0;0.2435;0]
p3=[0.4318;0;-0.0934]

g=[0 0 -9.8 0]'

cg2=[-0.068;0;0;1]
cg3=[0;-0.07;0;1]

%rotation matrices are computed

s1=sin(q1)
c1=cos(q1)
s2=sin(q2)
c2=cos(q2)
s3=sin(q3)
c3=cos(q3)

r1=[c1 -s1 0;s1 c1 0;0 0 1]
r2=[c2 -s2 0;0 0 1;-s2 -c2 0]
r3=[c3 -s3 0;s3 c3 0;0 0 1]

r1=[[r1] [p1];0 0 0 1]
r2=[[r2] [p2];0 0 0 1]
r3=[[r3] [p3];0 0 0 1]

qq=[0 -1 0 0;1 0 0 0;0 0 0 0;0 0 0 0]

dr1=r1*qq
dr2=r2*qq
dr3=r3*qq

ddr1=dr1*qq
ddr2=dr2*qq
ddr3=dr3*qq

t1=r1
```

```
t2=t1*r2
t3=t2*r3

ftt0=[0 0 0 0;0 0 0 0;0 0 0 0;0 0 0 0]
ftt1=ftt0*r1+dr1*v1
ftt2=ftt1*r2+t1*dr2*v2
ftt3=ftt2*r3+t2*dr3*v3


t0=[1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1]
stt0=ftt0
stt1=stt0*r1+2*ftt0*dr1*v1+t0*ddr1*v1^2+t0*dr1*a1
stt2=stt1*r2+2*ftt1*dr2*v2+t1*ddr2*v2^2+t1*dr2*a2
stt3=stt2*r3+2*ftt2*dr3*v3+t2*ddr3*v3^2+t2*dr3*a3


dt1=dr1
dt2=t1*dr2
dt3=t2*dr3
d3=j3*(stt3')+[0 0 0 0;0 0 0 0;0 0 0 0;0 0 0 0]
d2=j2*(stt2')+r3*d3
d1=j1*(stt1')+r2*d2


b1=(dt1)*d1
b2=(dt2)*d2
b3=(dt3)*d3
trc1=trace(b1)
trc2=trace(b2)
trc3=trace(b3)

cc3=m3*cg3+[0;0;0;0]
cc2=m2*cg2+r3*cc3
cc1=r2*cc2

g1=-g'*dt1*cc1
g2=-g'*dt2*cc2
g3=-g'*dt3*cc3


toq1=trc1+g1
toq2=trc2+g2
toq3=trc3+g3
```

D.1.2 Inverse Dynamics using Newton-Euler Equations

```
%general parameters for PUMa 560


m2=17.4;
m3=6.05;
m4=0.0;
;
i1=[0 0 0;0 0 0;0 0 0.35+1.14];
i2=[0.130 0 0;0 0.524 0; 0 0 0.539+4.71];
i3=[0.192 0 0; 0 0.0154 0; 0 0 0.212+0.83];
;
p1=[0;0;0];
p2=[0;0.2435;0];
p3=[0.4318;0;-0.0934];
p4=[0;-0.4318;0];
;
cg2=[-0.068;0;0];
cg3=[0;-0.07;0];
y3new=(6.05*0.07+m4*0.4318)/(m4+6.05)
cg3=[0;-y3new;0]
m3=m4+m3;
%increments in inertia due to payload
dxx=m4*(p4(2)^2+p4(3)^2);
dyy=m4*(p4(1)^2+p4(3)^2);
dzz=m4*(p4(1)^2+p4(2)^2);
di3=[dxx 0 0;0 dyy 0;0 0 dzz];
i3=i3+di3
%rotation matrices are computed;
;
s1=sin(q1);
c1=cos(q1);
s2=sin(q2);
c2=cos(q2);
s3=sin(q3);
c3=cos(q3);

r1=[c1 -s1 0;s1 c1 0;0 0 1];
r2=[c2 -s2 0;0 0 1;-s2 -c2 0];
r3=[c3 -s3 0;s3 c3 0;0 0 1];


%Forward recursion starts here
;
av0=[0;0;0];
aa0=[0;0;0];
```

```
la0=[0;0;9.8];
;
% link no: 1
r1'*av0;
av1=r1'*av0+[0;0;v1];
a=av1;
b=[0;0;v1];
c=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)];
aa1=r1'*(aa0)+[0;0;a1]+c;
a=aa0;
b=p1;
c=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)];
a=av0;
b=p1;
ccc=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)] ;
b=ccc;
cc=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)]
la0+c+cc;
la1=r1'*(la0+c+cc);
%cg acceleration is not calculated as this is not required.

% link no: 2
r2'*av1;
av2=r2'*av1+[0;0;v2];
a=av2;
b=[0;0;v2];
c=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)];
aa2=r2'*(aa1)+[0;0;a2]+c;
a=aa1;
b=p2;
c=[a(2,1)*b(3,1)-b(2,1)*a(3.1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)];
a=av1;
b=p2;
ccc=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)] ;
b=ccc;
cc=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)] ;
la1+c+cc;
la2=r2'*(la1+c+cc);
a=aa2;
b=cg2;
```

```
c=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)];
a=av2;
b=cg2;
ccc=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)] ;
b=ccc;
cc=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)] ;
lc2=la2+c+cc;

% link no: 3
r3'*av2;
av3=r3'*av2+[0;0;v3];
a=av3;
b=[0;0;v3];
c=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)];
aa3=r3'*(aa2)+[0;0;a3]+c;
a=aa2;
b=p3;
c=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)];
a=av2;
b=p3;
ccc=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)] ;
b=ccc;
cc=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)] ;
la2+c+cc;
la3=r3'*(la2+c+cc);
a=aa3;
b=cg3;
c=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)];
a=av3;
b=cg3;
ccc=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)] ;
b=ccc;
cc=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)] ;
lc3=la3+c+cc;

%Forward Recursion ends here

%Backward Recursion starts here
```

```
%External forces are defined zero

f4=[0;0;0];
n4=[0;0;0];
r4=[1 0 0;0 1 0;0 0 1];
p4=[0;0;0];

%link 3

r4*f4;
m3*lc3;
f3=r4*f4+m3*lc3;
mom=i3*av3;
a=av3;
b=mom;
c=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)];
a=cg3;
b=m3*lc3;
cc=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)];
a=p4;
b=r4*f4;
ccc=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)] ;
mom=i3*aa3;
dd=r4*n4;
i3*aa3;
n3=(i3*aa3+c)+cc+ccc;


%link 2
r3*f3;
m2*lc2;
f2=r3*f3+m2*lc2;
mom=i2*av2;
a=av2;
b=mom;
c=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)];
a=cg2;
b=m2*lc2;
cc=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)];
a=p3;
b=r3*f3;
ccc=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
```

```
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)] ;
r3*n3;
i2*aa2;
n2=r3*n3+(i2*aa2+c)+cc+ccc;


%link 1

f1=r2*f2;
mom=i1*av1;
a=av1;
b=mom;
c=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)];
a=p2;
b=r2*f2;
ccc=[a(2,1)*b(3,1)-b(2,1)*a(3,1);
b(1,1)*a(3,1)-a(1,1)*b(3,1);a(1,1)*b(2,1)-b(1,1)*a(2,1)] ;
i1*aa1;
r2*n2;
n1=r2*n2+(i1*aa1+c)+ccc;

%end of backward recursion

%the torques are given by the z components of n, the joint moments.

t1=n1(3,1)
t2=n2(3,1)
t3=n3(3,1)
```

D.1.3 Inverse Kinematics Program for PUMA-560 (3 DOF)

```
%Program pumakin.m
%
% Inverse Kinematics for PUMA-560 Manipulator
%
%h=distance of the first joint from base =0.0(can be changed)
%e=distance of the third joint from second joint along 2nd link=0.4318
%g=distance of the third joint from first joint along 1st link=0.1270
%f=distance of the wrist from the third joint along 3rd link=0.4521
%d=distance of end effector from the third joint=0.0
load traj2
h=0.0;
e=0.4318;
f=0.4318;
g=0.1270;
d=0.0;
px=0.3
for i=1:201,
pz=ppp(i,2);
py=pz;
q1f=atan2(-px,py)+atan2(sqrt(px^2+py^2-g^2),g);
q1s=atan2(-px,py)+atan2(-sqrt(px^2+py^2-g^2),g);
%choose q1;
if i > 1,
df=abs(q1-q1f);
ds=abs(q1-q1s);
if df < ds, q1=q1f; end;
if df > ds, q1=q1s; end;
if df == 0, q1=q1s; end;
end;
if i == 1, q1=q1f; end;
q3f=atan2(e^2+f^2+g^2-px^2-py^2-(pz-h)^2,
sqrt(4*e^2*f^2-(e^2+f^2+g^2-px^2-py^2-(pz-h)^2)^2));
q3s=atan2(e^2+f^2+g^2-px^2-py^2-(pz-h)^2,
-sqrt(4*e^2*f^2-(e^2+f^2+g^2-px^2-py^2-(pz-h)^2)^2));
%choose q3;
if i > 1,
df=abs(q3-q3f);
ds=abs(q3-q3s);
if df < ds, q3=q3f; end;
if df > ds, q3=q3s; end;
if df == 0, q3=q3s; end;
end;
if i == 1, q3=q3f; end;
%q3
s3=sin(q3);
```

```
c3=cos(q3);
c1=cos(q1);
s1=sin(q1);
q2=atan2(-(px*c1+py*s1)*f*c3-(pz-h)*(e-f*s3),
(px*c1+py*s1)*(e-f*s3)-(pz-h)*f*c3);
theta(i,1)=q1
theta(i,2)=q2
theta(i,3)=q3

y1=theta(i,1);
y2=theta(i,2);
y3=theta(i,3);
d3=5.*.0254;
d4=17.*.0254;
a2=17.*.0254;
a3=0;
pp(i,1)=a3*cos(y1)*cos(y2+y3)-d4*cos(y1)*sin(y2+y3)
+a2*cos(y2)*cos(y1)-d3*sin(y1);;
pp(i,2)=a3*sin(y1)*cos(y2+y3)-d4*sin(y1)*sin(y2+y3)
+a2*cos(y2)*sin(y1)+d3*cos(y1);;
pp(i,3)=-a3*sin(y2+y3)-d4*cos(y2+y3)-a2*sin(y2);;

vz=ppp(i,3);
vy=vz;
vx=0.0;
rv=[vx;vy;vz];

con11=-a3*sin(y1)*cos(y2+y3)+d4*sin(y1)*sin(y2+y3)
-a2*sin(y1)*cos(y2)-d3*cos(y1);
con12=-a3*cos(y1)*sin(y2+y3)-d4*cos(y1)*cos(y2+y3)
-a2*sin(y2)*cos(y1);
      con13=-a3*cos(y1)*sin(y2+y3)-d4*cos(y1)*cos(y2+y3);

con21=a3*cos(y1)*cos(y2+y3)-d4*cos(y1)*sin(y2+y3)
+a2*cos(y1)*cos(y2)-d3*sin(y1);;
con22=-a3*sin(y1)*sin(y2+y3)-d4*sin(y1)*cos(y2+y3)
-a2*sin(y2)*sin(y1);;
con23=-a3*sin(y1)*sin(y2+y3)-d4*sin(y1)*cos(y2+y3);;
;
con31=0.0;
con32=-a3*cos(y2+y3)+d4*sin(y2+y3)-a2*cos(y2);
con33=-a3*cos(y2+y3)+d4*sin(y2+y3);
con=[con11 con12 con13;con21 con22 con23; con31 con32 con33] ;
vq=inv(con)*rv;
;
v(i,1)=vq(1)
v(i,2)=vq(2)
v(i,3)=vq(3)
```

```
;
v1=vq(1);
v2=vq(2);
v3=vq(3);

vv(i,1)=con11*v1+con12*v2+con13*v3;
vv(i,2)=con21*v1+con22*v2+con23*v3;
vv(i,3)=con31*v1+con32*v2+con33*v3;


az=ppp(i,4);
ay=az;
ax=0.0;
ra=[ax;ay;az];
;
d3=5.*.0254;
d4=17.*.0254;
a2=17.*.0254;
a3=0.0;
;
s1=sin(y1);
s2=sin(y2);
s3=sin(y3);
c1=cos(y1);
c2=cos(y2);
c3=cos(y3);
s23=sin(y2+y3);
c23=cos(y2+y3);


con11=-a3*sin(y1)*cos(y2+y3)+d4*sin(y1)*sin(y2+y3)
-a2*sin(y1)*cos(y2)-d3*cos(y1);
con12=-a3*cos(y1)*sin(y2+y3)-d4*cos(y1)*cos(y2+y3)
-a2*sin(y2)*cos(y1);
      con13=-a3*cos(y1)*sin(y2+y3)-d4*cos(y1)*cos(y2+y3);
;
tx11=(-a3*c1*c23+d4*c1*s23-a2*c2*c1+d3*s1)*v1^2;
tx22=(-a3*c1*c23+d4*c1*s23-a2*c2*c1)*v2^2;
tx33=(-a3*c1*c23+d4*c1*s23)*v3^2;
tx12=(a3*s1*s23+d4*s1*c23+a2*s2*s1)*2.*v1*v2;
tx23=(-a3*c1*c23+d4*c1*s23)*v2*v3*2.;
tx13=(a3*s1*s23+d4*s1*c23)*2.*v1*v3;
sra1=ra(1)-(tx11+tx22+tx33+tx12+tx13+tx23);
ss1=(tx11+tx22+tx33+tx12+tx13+tx23) ;

con21=a3*cos(y1)*cos(y2+y3)-d4*cos(y1)*sin(y2+y3)
+a2*cos(y1)*cos(y2)-d3*sin(y1);
con22=-a3*sin(y1)*sin(y2+y3)-d4*sin(y1)*cos(y2+y3)
```

```
-a2*sin(y2)*sin(y1);
con23=-a3*sin(y1)*sin(y2+y3)-d4*sin(y1)*cos(y2+y3);
ty11=(-a3*s1*c23+d4*s1*s23-a2*c2*s1-d3*c1)*v1^2;
ty22=(-a3*s1*c23+d4*s1*s23-a2*c2*s1)*v2^2;
ty33=(-a3*s1*c23+d4*s1*s23)*v3^2;
ty12=(-a3*c1*s23-d4*c1*c23-a2*s2*c1)*v1*v2*2.;
ty23=(-a3*s1*c23+d4*s1*s23)*v3*v2*2.;
ty13=(-a3*c1*s23-d4*c1*c23)*v3*v1*2.;
sra2=ra(2)-(ty11+ty22+ty33+ty12+ty13+ty23);
ss2=(ty11+ty22+ty33+ty12+ty13+ty23);

con31=0.0;
con32=-a3*cos(y2+y3)+d4*sin(y2+y3)-a2*cos(y2);
con33=-a3*cos(y2+y3)+d4*sin(y2+y3);
tz11=0.0;
tz22=(a3*s23+d4*c23+a2*s2)*v2^2;
tz33=(a3*s23+d4*c23)*v3^2;
tz23=(a3*s23+d4*c23)*v2*v3*2.;
sra3=ra(3)-(tz11+tz22+tz33+tz23);
ss3=(tz11+tz22+tz33+tz23);
;
con=[con11 con12 con13;con21 con22 con23;con31 con32 con33];
sra=[sra1;sra2;sra3];
qa=inv(con)*sra ;
al(i,1)=qa(1);
al(i,2)=qa(2);
al(i,3)=qa(3);

a1=qa(1);
a2=qa(2);
a3=qa(3);

aa(i,1)=con11*a1+con12*a2+con13*a3+ss1;
aa(i,2)=con21*a1+con22*a2+con23*a3+ss2;
aa(i,3)=con31*a1+con32*a2+con33*a3+ss3;

end;

t=ppp(1:201,1);
```

D.2 Program in REDUCE for generating the Inverse Dynamics

The symbolic program for generating the inverse dynamics of manipulators is given here. This can be executed using the symbolic package, REDUCE, in VAX/7000 system. The program will generate the inverse dynamic equations in FORTRAN which can be used in the control software. The kinematic and dynamic parameters used for generating the program are given in Tables 2.5 to 2.8. The program given in this section is for a PUMA-560 manipulator with 6 DOF. This can be modified with new input data for any other manipulator.

```
"======================"$
"PROGRAM INITIALIZATION"$
"======================"$

ON FLOAT$
OFF EXP;
ON FORT;
OFF PERIOD;
NLINK:=6$
ARRAY THETA(NLINK),ALPA(NLINK) ,A(NLINK), D(NLINK), JT(NLINK)$
ON NUMVAL$
MATRIX R,R1,R2,R3,RR4,R5,R6,R7,R8,R9$
ORDER
DXX2,DXX3,DXX4,DXX5,DXX6,DYY2,DYY3,DYY4,DYY5,DYY6,DZZ2,
DZZ3,DZZ4,DZZ5,DZZ6$

"====================="$
"OUTPUT FILE IS OPENED"$
"====================="$

OUT PUMTOR$

"=================================================="$
"DENAVIT HARTENBERG MATRIX PARAMETERS ARE ENTERED HERE"$
"=================================================="$

THETA(3):=Q3$
THETA(1):=Q1$
THETA(2):=Q2$
THETA(4):=Q4$
THETA(5):=Q5$
THETA(6):=Q6$

ALPA(1):=0$
ALPA(2):=-PI/2$
ALPA(3):=0$
ALPA(4):=PI/2$
ALPA(5):=-PI/2$
ALPA(6):=PI/2$

A(1):=0$
A(2):=0$
A(3):=0.4318$
A(4):=-0.0203$
A(5):=0$
A(6):=0$
```

```
D(1):=0$
D(2):=0.2435$
D(3):=-0.0934$
D(4):=0.4331$
D(5):=0$
D(6):=0$

"================================="$
"INERTIA MATRICES ARE ENTERED HERE"$
"================================="$

IXX2 :=0.1351$
IYY2 :=0.6089$
IZZ2 :=5.3301$

IXX3 :=0.066$
IYY3 :=0.0134$
IZZ3 :=0.9395$

IXX4 :=0.0021$
IYY4 :=0.0021$
IZZ4 :=0.2013$

IXX5 :=0.3E-03$
IYY5 :=0.3E-03$
IZZ5 :=0.1794$

IXX6 :=0.2422E-03$
IYY6 :=0.2422E-03$
IZZ6 :=0.1930$

I1:=MAT((0,0,0),(0,0,0),(0,0,1.490))$
I2:=MAT((IXX2,0,0),(0,IYY2,0),(0,0,IZZ2))$
I3:=MAT((IXX3,0,0),(0,IYY3,0),(0,0,IZZ3))$
I4:=MAT((IXX4,0,0),(0,IYY4,0),(0,0,IZZ4))$
I5:=MAT((IXX5,0,0),(0,IYY5,0),(0,0,IZZ5))$
I6:=MAT((IXX6,0,0),(0,IYY6,0),(0,0,IZZ6))$

"==============================="$
"LINK MASS DATA IS ENTERED HERE"$
"==============================="$

M2 :=17.4$
M3 :=4.8$
M4 :=0.82$
M5 :=0.34$
M6 :=0.09$
```

```
"====================================="$
"LINK CG CO-ORDINATES ARE ENTERED HERE"$
"====================================="$

CG2:=MAT((-0.068),(0),(0))$
CG3:=MAT((0),(-0.070),(0))$
CG4:=MAT((0),(0),(0))$
CG5:=MAT((0),(0),(0))$
CG6:=MAT((0),(0),(0.032))$

"============================"$
"DEFINITION OF CROS OPERATOR"$
"============================"$

OPERATOR X3,Y3,Z3$
FOR ALL X1,Y1,Z1,X2,Y2,Z2 let X3(X1,Y1,Z1,X2,Y2,Z2)=Y1*Z2-Z1*Y2$
FOR ALL X1,Y1,Z1,X2,Y2,Z2 let Y3(X1,Y1,Z1,X2,Y2,Z2)=Z1*X2-X1*Z2$
FOR ALL X1,Y1,Z1,X2,Y2,Z2 let Z3(X1,Y1,Z1,X2,Y2,Z2)=X1*Y2-X2*Y1$

"================================================"$
"ROTATION TRANSFORMATION MATRICES ARE COMPUTED HERE"$
"================================================"$

FOR I:=1:NLINK DO <<
R:= MAT((COS(THETA(I)), -SIN(THETA(I)),0),(SIN(THETA(I))*COS(ALPA(I)),
COS(THETA(I))*COS(ALPA(I)),
-SIN(ALPA(I))),(SIN(THETA(I))*SIN
(ALPA(I)),COS(THETA(I))*SIN(ALPA(I)),COS(ALPA(I))))$;
P:=MAT((A(I)),(-D(I)*SIN(ALPA(I))),(D(I)*COS(ALPA(I))))$;
IF I=1 THEN  R1:=R ;
IF I=2 THEN  R2:=R ;
IF I=3 THEN  R3:=R ;
IF I=4 THEN  RR4:=R ;
IF I=5 THEN  R5:=R ;
IF I=6 THEN  R6:=R ;
IF I=2 THEN  P1:=P ;
IF I=3 THEN  P2:=P ;
IF I=4 THEN  P3:=P ;
IF I=5 THEN  P4:=P ;
IF I=6 THEN  P5:=P ;>>$
P6:=MAT((0),(0),(0))$

"=================================================="$
"ALTERNATIVELY, ROTATION MATRICES CAN BE DIRECTLY ENTERED"$
"TRANSFORMATION MATRICES ARE ENTERED HERE"$
"=================================================="$

R1:=MAT((C1,-S1,0),(S1,C1,0),(0,0,1))$
```

```
R2:=MAT((C2,-S2,0),(0,0,1),(-S2,-C2,0))$
R3:=MAT((C3,-S3,0),(S3,C3,0),(0,0,1))$
RR4:=MAT((C4,-S4,0),(0,0,-1),(S4,C4,0))$
R5:=MAT((C5,-S5,0),(0,0,1),(-S5,-C5,0))$
R6:=MAT((C6,-S6,0),(0,0,(-1)),(S6,C6,0))$

"=============================="$
"RELATIVE VELOCITIES ARE ASSIGNED HERE"$
"=================================="$

RV1:=MAT((0),(0),(V1))$
RV2:=MAT((0),(0),(V2))$
RV3:=MAT((0),(0),(V3))$
RV4:=MAT((0),(0),(V4))$
RV5:=MAT((0),(0),(V5))$
RV6:=MAT((0),(0),(V6))$

"========================================"$
"RELATIVE ACCELERATIONS ARE ASSIGNED HERE"$
"========================================"$

RA1:=MAT((0),(0),(A1))$
RA2:=MAT((0),(0),(A2))$
RA3:=MAT((0),(0),(A3))$
RA4:=MAT((0),(0),(A4))$
RA5:=MAT((0),(0),(A5))$
RA6:=MAT((0),(0),(A6))$

"=================================="$
"ANGULAR VELOCITIES ARE CALCULATED HERE"$
"=================================="$

AV1:=RV1$
AV2:=TP(R2)*(AV1)+RV2;
AV2:=MAT((AV2X),(AV2Y),(AV2Z))$
AV3:=TP(R3)*(AV2)+RV3;
AV3:=MAT((AV3X),(AV3Y),(AV3Z))$
AV4:=TP(RR4)*(AV3)+RV4;
AV4:=MAT((AV4X),(AV4Y),(AV4Z))$
AV5:=TP(R5)*(AV4)+RV5;
AV5:=MAT((AV5X),(AV5Y),(AV5Z))$
AV6:=TP(R6)*(AV5)+RV6;
AV6:=MAT((AV6X),(AV6Y),(AV6Z))$

"========================================"$
"ANGULAR ACCELERATIONS ARE CALCULATED HERE"$
"========================================"$
```

```
AA1:=RA1$
CROS2:=MAT((V2*AV2Y),(-V2*AV2X),(0))$
AA2:=TP(R2)*(AA1)+RA2+CROS2;
AA2:=MAT((AA2X),(AA2Y),(AA2Z))$
CROS3:=MAT((V3*AV3Y),(-V3*AV3X),(0))$
AA3:=TP(R3)*(AA2)+RA3+CROS3;
AA3:=MAT((AA3X),(AA3Y),(AA3Z))$
CROS4:=MAT((V4*AV4Y),(-V4*AV4X),(0))$
AA4:=TP(RR4)*(AA3)+RA4+CROS4;
AA4:=MAT((AA4X),(AA4Y),(AA4Z))$
CROS5:=MAT((V5*AV5Y),(-V5*AV5X),(0))$
AA5:=TP(R5)*(AA4)+RA5+CROS5;
AA5:=MAT((AA5X),(AA5Y),(AA5Z))$
CROS6:=MAT((V6*AV6Y),(-V6*AV6X),(0))$
AA6:=TP(R6)*(AA5)+RA6+CROS6;
AA6:=MAT((AA6X),(AA6Y),(AA6Z))$

"================================="$
"GRAVITY EFFECTS ARE INCLUDED HERE"$
"================================="$

GG:=MAT((0),(0),(G))$
LET G=9.8;

"================================="$
"LAMBDA MATRICES ARE DEFINED HERE"$
"================================="$

LAM1:=MAT((0,0,0),(0,0,0),(0,0,V1**2-A1))$
LAM2:=MAT((RSX2,RZM2,RYP2),(RZP2,RSY2,RXM2),(RYM2,RXP2,RSZ2))$
LAM3:=MAT((RSX3,RZM3,RYP3),(RZP3,RSY3,RXM3),(RYM3,RXP3,RSZ3))$
LAM4:=MAT((RSX4,RZM4,RYP4),(RZP4,RSY4,RXM4),(RYM4,RXP4,RSZ4))$
LAM5:=MAT((RSX5,RZM5,RYP5),(RZP5,RSY5,RXM5),(RYM5,RXP5,RSZ5))$
LAM6:=MAT((RSX6,RZM6,RYP6),(RZP6,RSY6,RXM6),(RYM6,RXP6,RSZ6))$

"================================================="$
"LINEAR ACCELERATION OF THE ORIGINS ARE CALCULATED HERE"$
"================================================="$

LA1:=TP(R1)*GG$
LA2P:=(LA1)+LAM1*P1;
LA2:=TP(R2)*(LA2P)$
LA3P:=(LA2)+LAM2*P2;
LA3P:=MAT((LA3PX),(LA3PY),(LA3PZ))$
LA3:=TP(R3)*LA3P;
LA3:=MAT((LA3X),(LA3Y),(LA3Z))$
LA4P:=(LA3)+LAM3*P3;
LA4P:=MAT((LA4PX),(LA4PY),(LA4PZ))$
```

```
TPRR4:=TP(RR4)$
LA4:=TPRR4*LA4P;
LA4:=MAT((LA4X),(LA4Y),(LA4Z))$
LA5:=TP(R5)*((LA4)+LAM4*P4);
LA5:=MAT((LA5X),(LA5Y),(LA5Z))$
LA6:=TP(R6)*((LA5)+LAM5*P5);
LA6:=MAT((LA6X),(LA6Y),(LA6Z))$

"=============================================="$
"LINEAR ACCN OF CGS OF LINKS ARE CALCULATED HERE"$
"=============================================="$

LC2:= LA2+LAM2*CG2$
LC3:= LA3+LAM3*CG3$
LC4:= LA4+LAM4*CG4$
LC5:= LA5+LAM5*CG5$
LC6:= LA6+LAM6*CG6$

"=========================================="$
"INERTIAL FORCES OF LINKS ARE CALCULATED HERE"$
"=========================================="$

IF2:= M2*LC2;
IF3:= M3*LC3;
IF4:= M4*LC4;
IF5:= M5*LC5;
IF6:= M6*LC6$

IF2:= MAT((IF2X),(IF2Y),(IF2Z))$
IF3:= MAT((IF3X),(IF3Y),(IF3Z))$
IF4:= MAT((IF4X),(IF4Y),(IF4Z))$
IF5:= MAT((IF5X),(IF5Y),(IF5Z))$

"======================================="$
"JOINT FORCES ARE CALCULATED HERE"$
"COMPUTE JF(I) IN THE PREVIOUS CO-ORD FRAME"$
"======================================="$

JF6:=IF6;
JF6:=MAT((JF6X),(JF6Y),(JF6Z))$
JFP6:=R6*JF6;
JFP6:=MAT((JFP6X),(JFP6Y),(JFP6Z))$
JF5:=JFP6+IF5;
JF5:=MAT((JF5X),(JF5Y),(JF5Z))$
JFP5:=R5*JF5;
JFP5:=MAT((JFP5X),(JFP5Y),(JFP5Z))$
JF4:=JFP5+IF4;
JF4:=MAT((JF4X),(JF4Y),(JF4Z))$
```

```
JFP4:=RR4*JF4;
JFP4:=MAT((JFP4X),(JFP4Y),(JFP4Z))$
JF3:=JFP4+IF3;
JF3:=MAT((JF3X),(JF3Y),(JF3Z))$
JFP3:=R3*JF3;
JFP3:=MAT((JFP3X),(JFP3Y),(JFP3Z))$
JF2:=JFP3+IF2;
JF2:=MAT((JF2X),(JF2Y),(JF2Z))$
JFP2:=R2*JF2;
JFP2:=MAT((JFP2X),(JFP2Y),(JFP2Z))$
JF1:=JFP2$

"====================================="$
"INERTIAL MOMENTS ARE CALCULATED HERE"$
"====================================="$

IN1:=I1*AA1$
IN2:=I2*AA2+MAT((DXX2*KX2),(DYY2*KY2),(DZZ2*KZ2))$
IN3:=I3*AA3+MAT((DXX3*KX3),(DYY3*KY3),(DZZ3*KZ3))$
IN4:=I4*AA4+MAT((DXX4*KX4),(DYY4*KY4),(DZZ4*KZ4))$
IN5:=I5*AA5+MAT((DXX5*KX5),(DYY5*KY5),(DZZ5*KZ5))$
IN6:=I6*AA6+MAT((DXX6*KX6),(DYY6*KY6),(DZZ6*KZ6))$

"================================================="$
"JOINT MOMENTS ARE CALCULATED HERE"$
"TWO CROS PRODUCTS ARE REQUIRED"$
"ONE IS CG(I) X IF(I) (DENOTED BY CN)"$
"OTHER ONE IS P(I) X R(I+1)*JF(I+1) (DENOTED BY DN)"$
"================================================="$

CN2:=MAT((X3(CG2(1,1),CG2(2,1),CG2(3,1),IF2(1,1),IF2(2,1),IF2(3,1))),
(Y3(CG2(1,1),CG2(2,1),CG2(3,1),IF2(1,1),IF2(2,1),IF2(3,1))),
(Z3(CG2(1,1),CG2(2,1),CG2(3,1),IF2(1,1),IF2(2,1),IF2(3,1))))$

CN3:=MAT((X3(CG3(1,1),CG3(2,1),CG3(3,1),IF3(1,1),IF3(2,1),IF3(3,1))),
(Y3(CG3(1,1),CG3(2,1),CG3(3,1),IF3(1,1),IF3(2,1),IF3(3,1))),
(Z3(CG3(1,1),CG3(2,1),CG3(3,1),IF3(1,1),IF3(2,1),IF3(3,1))))$

CN4:=MAT((X3(CG4(1,1),CG4(2,1),CG4(3,1),IF4(1,1),IF4(2,1),IF4(3,1))),
(Y3(CG4(1,1),CG4(2,1),CG4(3,1),IF4(1,1),IF4(2,1),IF4(3,1))),
(Z3(CG4(1,1),CG4(2,1),CG4(3,1),IF4(1,1),IF4(2,1),IF4(3,1))))$

CN5:=MAT((X3(CG5(1,1),CG5(2,1),CG5(3,1),IF5(1,1),IF5(2,1),IF5(3,1))),
(Y3(CG5(1,1),CG5(2,1),CG5(3,1),IF5(1,1),IF5(2,1),IF5(3,1))),
(Z3(CG5(1,1),CG5(2,1),CG5(3,1),IF5(1,1),IF5(2,1),IF5(3,1))))$

CN6:=MAT((X3(CG6(1,1),CG6(2,1),CG6(3,1),JF6(1,1),JF6(2,1),JF6(3,1))),
(Y3(CG6(1,1),CG6(2,1),CG6(3,1),JF6(1,1),JF6(2,1),JF6(3,1))),
```

```
(Z3(CG6(1,1),CG6(2,1),CG6(3,1),JF6(1,1),JF6(2,1),JF6(3,1)))$

"======================================="$
"EVALUATE CROS PRODUCT OF P(I) X JFP(I)"$
"======================================="$

DN2:=MAT((X3(P1(1,1),P1(2,1),P1(3,1),JFP2(1,1),JFP2(2,1),JFP2(3,1))),
(Y3(P1(1,1),P1(2,1),P1(3,1),JFP2(1,1),JFP2(2,1),JFP2(3,1))),
(Z3(P1(1,1),P1(2,1),P1(3,1),JFP2(1,1),JFP2(2,1),JFP2(3,1)))$

DN3:=MAT((X3(P2(1,1),P2(2,1),P2(3,1),JFP3(1,1),JFP3(2,1),JFP3(3,1))),
(Y3(P2(1,1),P2(2,1),P2(3,1),JFP3(1,1),JFP3(2,1),JFP3(3,1))),
(Z3(P2(1,1),P2(2,1),P2(3,1),JFP3(1,1),JFP3(2,1),JFP3(3,1)))$

DN4:=MAT((X3(P3(1,1),P3(2,1),P3(3,1),JFP4(1,1),JFP4(2,1),JFP4(3,1))),
(Y3(P3(1,1),P3(2,1),P3(3,1),JFP4(1,1),JFP4(2,1),JFP4(3,1))),
(Z3(P3(1,1),P3(2,1),P3(3,1),JFP4(1,1),JFP4(2,1),JFP4(3,1)))$

DN5:=MAT((X3(P4(1,1),P4(2,1),P4(3,1),JFP5(1,1),JFP5(2,1),JFP5(3,1))),
(Y3(P4(1,1),P4(2,1),P4(3,1),JFP5(1,1),JFP5(2,1),JFP5(3,1))),
(Z3(P4(1,1),P4(2,1),P4(3,1),JFP5(1,1),JFP5(2,1),JFP5(3,1)))$

DN6:=MAT((X3(P5(1,1),P5(2,1),P5(3,1),JFP6(1,1),JFP6(2,1),JFP6(3,1))),
(Y3(P5(1,1),P5(2,1),P5(3,1),JFP6(1,1),JFP6(2,1),JFP6(3,1))),
(Z3(P5(1,1),P5(2,1),P5(3,1),JFP6(1,1),JFP6(2,1),JFP6(3,1)))$

"================================================="$
"NOW COMPUTE THE JOINT MOMENTS IN BACKWARD ITERATION"$
"================================================="$

JM6:= IN6+CN6;
JM6:=MAT((JM6X),(JM6Y),(JM6Z))$

JM5:=R6*JM6+DN6+IN5+CN5;
JM5:=MAT((JM5X),(JM5Y),(JM5Z))$

JM4:=R5*JM5+DN5+IN4+CN4;
JM4:=MAT((JM4X),(JM4Y),(JM4Z))$

JM3:=RR4*JM4+DN4+IN3+CN3;
JM3:=MAT((JM3X),(JM3Y),(JM3Z))$

JM2:=R3*JM3+DN3+IN2+CN2;
JM2:=MAT((JM2X),(JM2Y),(JM2Z))$

JM1:=R2*JM2+DN2+IN1;

"============================================="$
```

```
"JOINT TORQUES EXTRACTED FROM THE JOINT MOMENTS"$
"=============================================="$

T1:=JM1Z;
T2:=JM2Z;
T3:=JM3Z;
T4:=JM4Z;
T5:=JM5Z;
T6:=JM6Z;

"======================"$
"OUTPUT FILE IS CLOSED"$
"======================"$

SHUT PUMTOR;

"=============="$
"END OF PROGRAM"$
"=============="$

BYE;
```

D.3 Inverse Dynamic Equations of Standard Manipulators

The inverse dynamics equations of Stanford manipulator and PUMA-560 manipulator have been generated using symbolic program given in Appendix D.2. Four programs are given in the subsequent sections, for the following manipulators.

1. Stanford Manipulator - 3 DOF system
2. PUMA-560 Manipulator - 3 DOF system
3. Stanford Manipulator - 6 DOF system
4. PUMA-560 Manipulator - 6 DOF system

The inputs to the program are the position, velocity and acceleration supplied at the joints and the torque/force for each actuator is computed by using the program. These programs can be compiled and used in the controller program for computing the input torque signal.

Following each program the number of multiplications and additions required for computing the inverse dynamic torques/forces are also given.

D.3.1 Stanford Manipulator - 3 DOF System

```
      AV2X=-S2*V1
      AV2Y=-C2*V1
      AV2Z=V2
      AA2X=-(S2*A1-V2*AV2Y)
      AA2Y=-(C2*A1+V2*AV2X)
      AA2Z=A2
KX2=AV2Y*AV2Z
KY2=AV2X*AV2Z
KZ2=AV2X*AV2Y
RX2=AV2X*AV2X
RY2=AV2Y*AV2Y
RZ2=AV2Z*AV2Z
RSY2=-(RX2+RZ2)
RSZ2=-(RY2+RX2)
RXM2=KX2-AA2X
RXP2=KX2+AA2X
RYP2=KY2+AA2Y
RZM2=KZ2-AA2Z
      IF2X=-49.098*(S2+0.0107551*RYP2)
      IF2Y=-49.098*(C2+0.0107551*RXM2)
      IF2Z=-0.528054*RSZ2
      JF3X=-63.406*((S2-0.2040816*V3*AV3Y)+0.0657857*
     . RZM2)
      JF3Y=-12.94*(V3*AV3X-0.32235*RXP2)
      JF3Z=63.406*((C2+0.1020408*A3)-0.0657857*RSY2)
      JF2X=IF2X+JF3X
      JF2Y=IF2Y-JF3Z
      JFP2X=C2*JF2X-S2*JF2Y
      JM3X=2.51*((AA2X-KX3)+0.2568526*JF3Y)
      JM3Y=2.51*((AA2Z+KY3)-0.2568526*JF3X)
      JM3Z=0
      JM2X=0.108*((AA2X+19.54629*KX2)+0.9759259*IF2Y+
     . 9.259259*JM3X)
      JM2Y=0.1*((AA2Y-21.03*KY2)-1.054*IF2X-10*JM3Z)
      JM2Z=2.211*((AA2Z-0.0036183*KZ2)+0.452284*JM3Y)
      JM1Z=-(C2*JM2Y+S2*JM2X-1.208*A1+0.1524*JFP2X)
      T1=JM1Z
      T2=JM2Z
      T3=JF3Z
```

No. of Multiplications = 48
No. of Additions = 33
Total = 81

D.3.2 Stanford Manipulator - 6 DOF System

```
      AV2X=-S2*V1
      AV2Y=-C2*V1
      AV2Z=V2
      AV4X=C4*AV2X+S4*AV2Z
      AV4Y=C4*AV2Z-S4*AV2X
      AV4Z=V4-AV2Y
      AV5X=C5*AV4X-S5*AV4Z
      AV5Y=-(C5*AV4Z+S5*AV4X)
      AV5Z=V5+AV4Y
      AV6X=C6*AV5X+S6*AV5Z
      AV6Y=C6*AV5Z-S6*AV5X
      AV6Z=V6-AV5Y
      AA2X=-(S2*A1-V2*AV2Y)
      AA2Y=-(C2*A1+V2*AV2X)
      AA2Z=A2
      AA4X=C4*AA2X+S4*AA2Z+V4*AV4Y
      AA4Y=C4*AA2Z-S4*AA2X-V4*AV4X
      AA4Z=A4-AA2Y
      AA5X=C5*AA4X-S5*AA4Z+V5*AV5Y
      AA5Y=-(C5*AA4Z+S5*AA4X+V5*AV5X)
      AA5Z=A5+AA4Y
      AA6X=C6*AA5X+S6*AA5Z+V6*AV6Y
      AA6Y=C6*AA5Z-S6*AA5X-V6*AV6X
      AA6Z=A6-AA5Y

KX2=AV2Y*AV2Z
KY2=AV2X*AV2Z
KZ2=AV2X*AV2Y
KX4=AV4Y*AV4Z
KY4=AV4X*AV4Z
KZ4=AV4X*AV4Y
KX5=AV5Y*AV5Z
KY5=AV5X*AV5Z
KZ5=AV5X*AV5Y
KX6=AV6Y*AV6Z
KY6=AV6X*AV6Z
KZ6=AV6X*AV6Y
RX2=AV2X*AV2X
RY2=AV2Y*AV2Y
RZ2=AV2Z*AV2Z
RSX2=-(RY2+RZ2)
RSY2=-(RX2+RZ2)
RSZ2=-(RY2+RX2)
RXM2=KX2-AA2X
RXP2=KX2+AA2X
```

```
RYM2=KY2-AA2Y
RYP2=KY2+AA2Y
RZM2=KZ2-AA2Z
RZP2=KZ2+AA2Z

RX4=AV4X*AV4X
RY4=AV4Y*AV4Y
RSZ4=-(RY4+RX4)
RXM4=KX4-AA4X
RXP4=KX4+AA4X
RYP4=KY4+AA4Y
RZM4=KZ4-AA4Z

RX5=AV5X*AV5X
RZ5=AV5Z*AV5Z
RSY5=-(RZ5+RX5)
RXP5=KX5+AA5X
RZM5=KZ5-AA5Z

RX6=AV6X*AV6X
RY6=AV6Y*AV6Y
RSZ6=-(RY6+RX6)
RXM6=KX6-AA6X
RYP6=KY6+AA6Y
      LA3X=-9.8*(S2-0.2040816*V3*AV3Y)
      LA3Y=-2*V3*AV3X
      LA3Z=9.8*C2+A3
      LA4X=(C4*RZM2-S4*RXP2)*Q3+LA3X*C4+LA3Y*S4
      LA4Y=-((C4*RXP2+S4*RZM2)*Q3+LA3X*S4-LA3Y*C4)
      LA4Z=LA3Z+Q3*RSY2
      LA5X=LA4X*C5-LA4Z*S5
      LA5Y=-(LA4X*S5+LA4Z*C5)
      LA5Z=LA4Y
      LA6X=LA5X*C6+LA5Z*S6
      LA6Y=-(LA5X*S6-LA5Z*C6)
      LA6Z=-LA5Y
      IF2X=-49.098*(S2+0.0107551*RYP2)
      IF2Y=-49.098*(C2+0.0107551*RXM2)
      IF2Z=-0.528054*RSZ2
      IF3X=4.25*(LA3X-0.6447*RZM2)
      IF3Y=4.25*(LA3Y+0.6447*RXP2)
      IF3Z=4.25*(LA3Z-0.6447*RSY2)
      IF4X=1.08*(LA4X-0.0054*RZM4-0.0092*RYP4)
      IF4Y=1.08*(LA4Y-0.0054*RSY4-0.0092*RXM4)
      IF4Z=1.08*(LA4Z-0.0054*RXP4-0.0092*RSZ4)
      IF5X=0.63*(LA5X-0.0566*RZM5)
      IF5Y=0.63*(LA5Y-0.0566*RSY5)
      IF5Z=0.63*(LA5Z-0.0566*RXP5)
```

```
      JF6X=0.51*(LA6X+0.1554*RYP6)
      JF6Y=0.51*(LA6Y+0.1554*RXM6)
      JF6Z=0.51*(LA6Z+0.1554*RSZ6)
      JFP6X=C6*JF6X-S6*JF6Y
      JFP6Y=-JF6Z
      JFP6Z=C6*JF6Y+S6*JF6X
      JF5X=IF5X+JFP6X
      JF5Y=IF5Y+JFP6Y
      JF5Z=IF5Z+JFP6Z
      JFP5X=C5*JF5X-S5*JF5Y
      JFP5Y=JF5Z
      JFP5Z=-(C5*JF5Y+S5*JF5X)
      JF4X=IF4X+JFP5X
      JF4Y=IF4Y+JFP5Y
      JF4Z=IF4Z+JFP5Z
      JFP4X=C4*JF4X-S4*JF4Y
      JFP4Y=C4*JF4Y+S4*JF4X
      JFP4Z=JF4Z
      JF3X=IF3X+JFP4X
      JF3Y=IF3Y+JFP4Y
      JF3Z=IF3Z+JFP4Z
      JFP3X=JF3X
      JFP3Y=-JF3Z
      JFP3Z=JF3Y
      JF2X=IF2X+JFP3X
      JF2Y=IF2Y+JFP3Y
      JFP2X=C2*JF2X-S2*JF2Y
      JM6X=0.013*((AA6X+0.5615385*KX6)-11.95384*JF6Y)
      JM6Y=0.013*((AA6Y-0.5615385*KY6)+11.95384*JF6X)
      JM6Z=0.0203*AA6Z
      JM5X=0.003*((AA5X+33.2*KX5)-18.86666*IF5Z)
     .+JM6X*C6-JM6Y*S6
      JM5Y=0.0004*(AA5Y-242.5*KY5)-JM6Z
      JM5Z=JM6X*S6+JM6Y*C6+0.1*AA5Z+0.0566*IF5X-0.0026
     . *KZ5
      JM4X=JM5X*C5-JM5Y*S5+0.002*AA4X+0.0092*IF4Y-
     . 0.0054*IF4Z+0.106*KX4
      JM4Y=0.001*((AA4Y-105*KY4)-9.2*IF4X)+JM5Z
      JM4Z=-(JM5X*S5+JM5Y*C5-0.107*AA4Z-0.0054*IF4X+
     . 0.001*KZ4)
      JM3X=JM4X*C4-JM4Y*S4-Q3*JFP4Y+2.51*AA2X+0.6447*
     . IF3Y-2.51*KX3
      JM3Y=JM4X*S4+JM4Y*C4+Q3*JFP4X+2.51*AA2Z-0.6447*
     . IF3X+2.51*KY3
      JM3Z=JM4Z
      JM2X=0.108*((AA2X+19.54629*KX2)+9.259259*JM3X+
     . 0.9759259*IF2Y)
      JM2Y=0.1*((AA2Y-21.03*KY2)-1.054*IF2X)-JM3Z
```

```
JM2Z=2.211*(AA2Z-0.0036183*KZ2)+JM3Y
JM1Z=-(JM2X*S2+JM2Y*C2-1.208*A1+0.1524*JFP2X)
T1=JM1Z
T2=JM2Z
T3=JF3Z
T4=JM4Z
T5=JM5Z
T6=JM6Z
```

No. of Multiplications = 183
No. of Additions = 140
Total = 323

D.3.3 PUMA-560 Manipulator - 3 DOF System

```
      AV2X=-S2*V1
      AV2Y=-C2*V1
      AV2Z=V2
      AV3X=C3*AV2X+S3*AV2Y
      AV3Y=C3*AV2Y-S3*AV2X
      AV3Z=V3+AV2Z
      AA2X=-(S2*A1-V2*AV2Y)
      AA2Y=-(C2*A1+V2*AV2X)
      AA2Z=A2
      AA3X=C3*AA2X+S3*AA2Y+V3*AV3Y
      AA3Y=C3*AA2Y-S3*AA2X-V3*AV3X
      AA3Z=A3+AA2Z
KX2=AV2Y*AV2Z
KY2=AV2X*AV2Z
KZ2=AV2X*AV2Y
KX3=AV3Y*AV3Z
KY3=AV3X*AV3Z
KZ3=AV3X*AV3Y
RY2=AV2Y*AV2Y
RZ2=AV2Z*AV2Z
RSX2=-(RY2+RZ2)
RYM2=KY2-AA2Y
RZP2=KZ2+AA2Z
RX3=AV3X*AV3X
RZ3=AV3Z*AV3Z
RSY3=-(RX3+RZ3)
RXP3=KX3+AA3X
RZM3=KZ3-AA3Z
      LA3PX=-9.8*(S2-0.0440612*RSX2+0.0095306*RYP2)
      LA3PY=-9.8*(C2-0.0440612*RZP2+0.0095306*RXM2)
      LA3PZ=0.4318*(RYM2-0.2163038*RSZ2)
      LA3X=C3*LA3PX+S3*LA3PY
      LA3Y=C3*LA3PY-S3*LA3PX
      LA3Z=LA3PZ
      IF2X=-170.52*(S2+0.0069388*RSX2)
      IF2Y=-170.52*(C2+0.0069388*RZP2)
      IF2Z=-1.1832*RYM2
      JF3X=-0.336*RZM3+LA3X)
      JF3Y=-0.336*RSY3+LA3Y)
      JF3Z=-0.336*RXP3+LA3Z)
      JFP3X=C3*JF3X-S3*JF3Y
      JFP3Y=C3*JF3Y+S3*JF3X
      JFP3Z=JF3Z
      JF2X=IF2X+JFP3X
      JF2Y=IF2Y+JFP3Y
      JFP2X=C2*JF2X-S2*JF2Y
```

```
      JM3X=0.066*((AA3X+14.03181*KX3)-1.060606*JF3Z)
      JM3Y=0.0134*(AA3Y-65.18656*KY3)
      JM3Z=0.9395*((AA3Z-0.0559872*KZ3)+0.0745077*JF3X)
      JM2X=C3*JM3X-S3*JM3Y+0.1351*AA2X+0.0934*JFP3Y+
     . 4.7212*KX2
      JM2Y=C3*JM3Y+S3*JM3X+0.6089*AA2Y+0.068*IF2Z-
     . 0.0934*JFP3X-0.4318*JFP3Z-5.195*KY2
      JM2Z=5.3301*((AA2Z+0.0888914*KZ2)-0.0127577*IF2Y
     . +0.0810116*JFP3Y)+JM3Z
      JM1Z=-(C2*JM2Y+S2*JM2X-1.49*A1+0.2435*JFP2X)
      T1=JM1Z
      T2=JM2Z
      T3=JM3Z
```

No. of Multiplications = 80
No. of Additions = 55
Total = 135

D.3.4 PUMA-560 Manipulator - 6 DOF System

```
      AV2X=-S2*V1
      AV2Y=-C2*V1
      AV2Z=V2
      AV3X=C3*AV2X+S3*AV2Y
      AV3Y=C3*AV2Y-S3*AV2X
      AV3Z=V3+AV2Z
      AV4X=C4*AV3X+S4*AV3Z
      AV4Y=C4*AV3Z-S4*AV3X
      AV4Z=V4-AV3Y
      AV5X=C5*AV4X-S5*AV4Z
      AV5Y=-(C5*AV4Z+S5*AV4X)
      AV5Z=V5+AV4Y
      AV6X=C6*AV5X+S6*AV5Z
      AV6Y=C6*AV5Z-S6*AV5X
      AV6Z=V6-AV5Y
      AA2X=-(S2*A1-V2*AV2Y)
      AA2Y=-(C2*A1+V2*AV2X)
      AA2Z=A2
      AA3X=C3*AA2X+S3*AA2Y+V3*AV3Y
      AA3Y=C3*AA2Y-S3*AA2X-V3*AV3X
      AA3Z=A3+AA2Z
      AA4X=C4*AA3X+S4*AA3Z+V4*AV4Y
      AA4Y=C4*AA3Z-S4*AA3X-V4*AV4X
      AA4Z=A4-AA3Y
      AA5X=C5*AA4X-S5*AA4Z+V5*AV5Y
      AA5Y=-(C5*AA4Z+S5*AA4X+V5*AV5X)
      AA5Z=A5+AA4Y
      AA6X=C6*AA5X+S6*AA5Z+V6*AV6Y
      AA6Y=C6*AA5Z-S6*AA5X-V6*AV6X
      AA6Z=A6-AA5Y
KX2=AV2Y*AV2Z
KY2=AV2X*AV2Z
KZ2=AV2X*AV2Y
KX3=AV3Y*AV3Z
KY3=AV3X*AV3Z
KZ3=AV3X*AV3Y
KX4=AV4Y*AV4Z
KY4=AV4X*AV4Z
KZ4=AV4X*AV4Y
KX5=AV5Y*AV5Z
KY5=AV5X*AV5Z
KZ5=AV5X*AV5Y
KX6=AV6Y*AV6Z
KY6=AV6X*AV6Z
KZ6=AV6X*AV6Y
RX2=AV2X*AV2X
```

```
RY2=AV2Y*AV2Y
RZ2=AV2Z*AV2Z
RSX2=-(RY2+RZ2)
RSZ2=-(RX2+RY2)
RXM2=KX2-AA2X
RYM2=KY2-AA2Y
RYP2=KY2+AA2Y
RX3=AV3X*AV3X
RY3=AV3Y*AV3Y
RZ3=AV3Z*AV3Z
RSX3=-(RY3+RZ3)
RSY3=-(RX3+RZ3)
RXP3=KX3+AA3X
RYM3=KY3-AA3Y
RX4=AV4X*AV4X
RY4=AV4Y*AV4Y
RZM3=KZ3-AA3Z
RZP3=KZ3+AA3Z
RSZ4=-(RX4+RY4)
RXM4=KX4-AA4X
RYP4=KY4+AA4Y
RX6=AV6X*AV6X
RY6=AV6Y*AV6Y
RSZ6=-(RX6+RY6)
RXM6=KX6-AA6X
RYP6=KY6+AA6Y
      LA3PX=-9.8*(S2-0.0440612*RSX2+0.0095306*RYP2)
      LA3PY=-9.8*(C2-0.0440612*RZP2+0.0095306*RXM2)
      LA3PZ=0.4318*(RYM2-0.2163038*RSZ2)
      LA3X=C3*LA3PX+S3*LA3PY
      LA3Y=C3*LA3PY-S3*LA3PX
      LA3Z=LA3PZ
      LA4PX=LA3X-0.0203*RSX3-0.4331*RZM3
      LA4PY=LA3Y-0.0203*RZP3-0.4331*RSY3
      LA4PZ=LA3Z-0.0203*RYM3-0.4331*RXP3
      LA4X=C4*LA4PX+S4*LA4PZ
      LA4Y=C4*LA4PZ-S4*LA4PX
      LA4Z=-LA4PY
      LA5X=LA4X*C5-LA4Z*S5
      LA5Y=-(LA4X*S5+LA4Z*C5)
      LA5Z=LA4Y
      LA6X=LA5X*C6+LA5Z*S6
      LA6Y=-(LA5X*S6-LA5Z*C6)
      LA6Z=-LA5Y
      IF2X=-170.52*(S2+0.0069388*RSX2)
      IF2Y=-170.52*(C2+0.0069388*RZP2)
      IF2Z=-1.1832*RYM2
      IF3X=4.8*(LA3X-0.07*RZM3)
```

```
      IF3Y=4.8*(LA3Y-0.07*RSY3)
      IF3Z=4.8*(LA3Z-0.07*RXP3)
      IF4X=0.82*(LA4X-0.019*RYP4)
      IF4Y=0.82*(LA4Y-0.019*RXM4)
      IF4Z=0.82*(LA4Z-0.019*RSZ4)
      IF5X=0.34*LA5X
      IF5Y=0.34*LA5Y
      IF5Z=0.34*LA5Z
      JF6X=0.09*(LA6X+0.032*RYP6)
      JF6Y=0.09*(LA6Y+0.032*RXM6)
      JF6Z=0.09*(LA6Z+0.032*RSZ6)
      JFP6X=C6*JF6X-S6*JF6Y
      JFP6Y=-JF6Z
      JFP6Z=C6*JF6Y+S6*JF6X
      JF5X=IF5X+JFP6X
      JF5Y=IF5Y+JFP6Y
      JF5Z=IF5Z+JFP6Z
      JFP5X=C5*JF5X-S5*JF5Y
      JFP5Y=JF5Z
      JFP5Z=-(C5*JF5Y+S5*JF5X)
      JF4X=IF4X+JFP5X
      JF4Y=IF4Y+JFP5Y
      JF4Z=IF4Z+JFP5Z
      JFP4X=C4*JF4X-S4*JF4Y
      JFP4Y=-JF4Z
      JFP4Z=C4*JF4Y+S4*JF4X
      JF3X=IF3X+JFP4X
      JF3Y=IF3Y+JFP4Y
      JF3Z=IF3Z+JFP4Z
      JFP3X=C3*JF3X-S3*JF3Y
      JFP3Y=C3*JF3Y+S3*JF3X
      JFP3Z=JF3Z
      JF2X=IF2X+JFP3X
      JF2Y=IF2Y+JFP3Y
      JFP2X=C2*JF2X-S2*JF2Y
      JM6X=0.0002422*((AA6X+795.862*KX6)-132.1222*JF6Y)
      JM6Y=0.0002422*((AA6Y-795.862*KY6)+132.1222*JF6X)
      JM6Z=0.193*AA6Z
      JM5X=JM6X*C6-JM6Y*S6+0.0003*AA5X+0.1791*KX5
      JM5Y=0.0003*(AA5Y-597.0*KY5)-JM6Z
      JM5Z=JM6X*S6+JM6Y*C6+0.1794*AA5Z
      JM4X=JM5X*C5-JM5Y*S5+0.0021*AA4X+0.019*IF4Y+
     . 0.1992*KX4
      JM4Y=0.0021*(AA4Y-94.85714*KY4-9.047619*IF4X)+JM5Z
      JM4Z=-(JM5X*S5+JM5Y*C5-0.2013*AA4Z)
      JM3X=JM4X*C4-JM4Y*S4+0.066*AA3X-0.07*IF3Z-0.4331
     . *JFP4Z+0.9261*KX3
      JM3Y=0.0134*((AA3Y-65.18656*KY3)+1.514925*JFP4Z)-JM3Z
```

```
      JM3Z=JM4X*S4+JM4Y*C4+0.9395*AA3Z+0.07*IF3X+
     . 0.4331*JFP4X-0.0203*JFP4Y-0.0526*KZ3
      JM2X=JM3X*C3-JM3Y*S3+0.1351*AA2X+0.0934*JFP3Y+
     . 4.7212*KX2
      JM2Y=JM3X*S3+JM3Y*C3+0.6089*AA2Y+0.068*IF2Z-
     . 0.0934*JFP3X-0.4318*JFP3Z-5.195*KY2
      JM2Z=5.3301*((AA2Z+0.0888914*KZ2)
     . -0.0127577*IF2Y+0.0810116*JFP3Y)+JM3Z
      JM1Z=-(JM2X*S2+JM2Y*C2-1.49*A1+0.2435*JFP2X)
      T1=JM1Z
      T2=JM2Z
      T3=JM3Z
      T4=JM4Z
      T5=JM5Z
      T6=JM6Z
```

No. of Multiplications = 208
No. of Additions = 152
Total = 360

D.4 Robot Simulation Program

A FORTRAN program for simulating a robot is given in this appendix. The dynamic equation of the robot is written as

$$[D]\{\ddot{q}\} = \{\tau\} - \{h\}(q, \dot{q})$$

where [D] is the matrix of inertial coefficients, and {h} is the vector of velocity and gravitational terms and $\{\tau\}$ is the vector of applied torques. The velocity and gravitational vectors are obtained from the symbolic program by setting the joint acceleration strings to zero. The inertial coefficients are evaluated using the symbolic programming of the modified NE algorithm, by making following assumptions:

1. The velocity and acceleration strings are set to zero.

2. For the jth column of the [D] matrix, the acceleration of the jth joint is set to one and the acceleration of all the other joints are set to zero. The corresponding torque vector computed by the modified NE algorithm yields the jth column.

The resulting second order differential equation is solved by fourth order Runge-Kurta method.

```
c=================================================================
c
c       ROBOT SIMULATION PROGRAM - 3 DOF PUMA-560 MANIPULATOR
c
c=================================================================
c
c Program uses Runge-Kurta fourth order method to
c solve the differential equation and
c cholesky decomposition to find the inverse of
c the inertia matrix.
c
c=================================================================
PARAMETER(NN=6,mm=3)
DIMENSION F(NN),y(nn),pp(mm),tq(201,3),tor(3)
open (unit=10, file='outtorq.dat',type='old')
open (unit=16, file='disp.dat',type='new')
open (unit=17, file='velo.dat',type='new')
open (unit=18, file='accn.dat',type='new')

DATA T,TLIM,H,M/0.0,2.0,0.02,0/
DATA Y/3.6997,2.3083,0.5951,0.0,0.0,0.0/
LL=1
N=NN
do i=1,201
read(10,*) (tq(i,j),j=1,3)
c write(*,*) (tq(i,j),j=1,3)
end do
8 IF (T-TLIM) 6,6,7
6 CALL RUNGE(N,F,Y,T,H,M,K)
GO TO (10,20),K
10 F(1)=Y(4)
F(2)=Y(5)
F(3)=Y(6)
jj=(t/0.01)+1
write(*,*) t,jj
tor(1)=tq(jj,1)
tor(2)=tq(jj,2)
tor(3)=tq(jj,3)
write(*,*) tor
call interpol(jj,tor,y,pp)
F(4)=PP(1)
F(5)=PP(2)
F(6)=PP(3)
GO TO 6
20 WRITE(16,*) (Y(J),J=1,3)
write(17,*) (Y(j),j=4,6)
write(18,*) (pp(j),j=1,3)
GO TO 8
```

```
7 STOP
END

SUBROUTINE RUNGE(N,F,Y,T,H,M,K)
DIMENSION F(6),Y(6),Q(6)
write(*,*) 'm',m,'t',t
M=M+1
GO TO (1,4,5,3,7),M
1 DO 2 J=1,N
2 Q(J)=0.0
A=0.5
GO TO 9
3 A=1.707107
4 T=T+0.5*H
5 DO 6 J=1,N
C PRINT *,F
Y(J)=Y(J)+A*(F(J)*H-Q(J))
6 Q(J)=2.*A*H*F(J)+(1.-3.*A)*Q(J)
A=0.2928932
GO TO 9
7 DO 8 J=1,N
8 Y(J)=Y(J)+H*F(J)/6.-Q(J)/3.
C PRINT *,Y
M=0
K=2
GO TO 10
9 K=1
10 RETURN
END

subroutine interpol(jj,tor,y,pp)
dimension y(6),pp(3),tq(201,3),vc(3),ac(3,3),toind(3),tor(3)
dimension la(3),lb(3,2),s(3),x(3)
call velcoeff(y,vc)
write(*,*) 'vc',vc
write(*,*) 'tor in sub',tor
toind(1)=0.0
toind(2)=0.0
toind(3)=0.0
if(jj.eq.1) go to 55
toind(1)=tor(1)-vc(1)
toind(2)=tor(2)-vc(2)
toind(3)=tor(3)-vc(3)
55 write(*,*) 'toind', toind
call acccoeff(y,ac)
write(*,*) 'ac values'
write(*,*) ac
WRITE(*,*) 'TOIND', TOIND
```

```
call lsarg(3,ac,3,toind,1,x)
WRITE(*,*) 'SOLN',X
c call simul(ac,toind,3,2,la,lb,s)
do i=1,3
pp(i)=x(i)
end do
WRITE(*,*) 'PP IN SUB',PP
return
end


subroutine acccoeff(y,ac)
dimension y(6),ac(3,3)
q1=y(1)
q2=y(2)
q3=y(3)
s2=sin(q2)
c2=cos(q2)
s3=sin(q3)
c3=cos(q3)
      ac(1,1)=6.881538*(c2*s2*c3*s3+0.5946234*c2*s2*c3-s2**2*
     . s3**2-0.5946234*s2**2*s3-0.0035408*s2**2+0.5*s3**2+
     . 0.5946234*s3+0.9436266)
      ac(1,2)=-0.7112062*(c2*c3-s2*s3-1.056647*s2)
      ac(1,3)=-0.7112062*(c2*c3-s2*s3)
      ac(2,1)=-0.7112062*(c2*c3-s2*s3-1.056647*s2)
      ac(2,2)=4.091923*(s3+3.085665)
      ac(2,3)=2.045961*(s3+2.104716)
      ac(3,1)=-0.7112062*(c2*c3-s2*s3)
      ac(3,2)=2.045961*(s3+2.104716)
      ac(3,3)=4.306169
write(56,*) ac
return
end


subroutine velcoeff(y,vc)
dimension y(6),vc(3)
q1=y(1)
q2=y(2)
q3=y(3)
v1=y(4)
v2=y(5)
v3=y(6)
s2=sin(q2)
c2=cos(q2)
s3=sin(q3)
c3=cos(q3)
```

```
      vc(1)=-13.76656*(c2*s2*s3**2*v1*v2+c2*s2*s3**2*v1*v3+
     . 0.5948432*c2*s2*s3*v1*v2+0.2974216*c2*s2*s3*v1*v3+
     . 0.0036841*c2*s2*v1*v2-0.5*c2*s2*v1*v3-0.0516941*c2*
     . s3*v2**2-0.1033881*c2*s3*v2*v3-0.0516941*c2*s3*v3**
     . 2-0.0546354*c2*v2**2+s2**2*c3*s3*v1*v2+s2**2*c3*s3*
     . v1*v3+0.5948432*s2**2*c3*v1*v2+0.2974216*s2**2*c3*
     . v1*v3-0.0516941*s2*c3*v2**2-0.1033881*s2*c3*v2*v3-
     . 0.0516941*s2*c3*v3**2-0.5*c3*s3*v1*v2-0.5*c3*s3*v1*
     . v3-0.2974216*c3*v1*v2-0.2974216*c3*v1*v3)
      vc(2)=6.883283*(c2*s2*s3**2*v1**2+0.5948432*c2*s2*s3*v1
     . **2+0.0036841*c2*s2*v1**2-6.750189*c2*s3-8.182499*
     . c2+s2**2*c3*s3*v1**2+0.5948432*s2**2*c3*v1**2-
     . 6.750189*s2*c3-0.5*c3*s3*v1**2-0.2974216*c3*v1**2+
     . 0.5948432*c3*v2*v3+0.2974216*c3*v3**2)
      vc(3)=6.883283*(c2*s2*s3**2*v1**2+0.2974216*c2*s2*s3*v1
     . **2-0.5*c2*s2*v1**2-6.750189*c2*s3+s2**2*c3*s3*v1**
     . 2+0.2974216*s2**2*c3*v1**2-6.750189*s2*c3-0.5*c3*s3
     . *v1**2-0.2974216*c3*v1**2-0.2974216*c3*v2**2)
c write(*,*) vc
return
end

subroutine simul(a,b,n,ind,la,lb,s)
dimension a(n,n),b(n),la(n),lb(n,2),s(n)
do 100 i=1,n
100  la(i)=0
do 250 k=1,n
z=0.0
do 150 i=1,n
if (la(i).eq.1) go to 150
do 140 j=1,n
if (la(j)-1) 130,140,300
130 if (abs(z).ge.abs(a(i,j))) go to 140
ia=i
ib=j
z=a(i,j)
140 continue
150 continue
la(ib)=la(ib)+1
if (ia.eq.ib) go to 190
do 160 i=1,n
z=a(ia,i)
a(ia,i)=a(ib,i)
160 a(ib,i)=z
if (ind.eq.0) go to 190
z=b(ia)
b(ia)=b(ib)
b(ib)=z
```

```
190 lb(k,1)=ia
lb(k,2)=ib
s(k)=a(ib,ib)
a(ib,ib)=1.0
do 200 i=1,n
200 a(ib,i)=a(ib,i)/s(k)
if (ind.eq.0) go to 220
b(ib)=b(ib)/s(k)
220 do 250 i=1,n
if (i.eq.ib) go to 250
z=a(i,ib)
a(i,ib)=0.0
do 230 j=1,n
230 a(i,j)=a(i,j)-a(ib,j)*z
if (ind.eq.0) go to 250
b(i)=b(i)-b(ib)*z
250 continue
do 270 i=1,n
j=n-i+1
if (lb(j,1).eq.lb(j,2)) go to 270
ia=lb(j,1)
ib=lb(j,2)
do 260 k=1,n
z=a(k,ia)
a(k,ia)=a(k,ib)
a(k,ib)=z
260 continue
270 continue
300 return
end
```