AN ABSTRACT OF THE THESIS OF

Wangjin Mun for the degree of Doctor of Philosophy in Electrical and Computer Engineering presented on April 10, 1989.

Title: Pipelined Multiprocessor Computer Architecture and Fast Parallel Algorithms for Real-Time Robot Control.

Redacted for privacy

Abstract approved: _____
James H. Herzog

As a result of the automation revolution, robots are assuming ever more complex and demanding tasks. Robot system control schemes for fast and precise robot motion require utilization of the entirety of the robot dynamic formulations and the ability to evaluate these formulations in real-time. The dynamic formulations, which take into account robot nonlinearities and dynamic coupling, are computationally intensive. They are difficult to implement in real-time at high sampling rates due to the time required to compute the dynamic formulations. Reducing the computation time for practical implementation can be achieved by developing a computing algorithm for the efficient evaluation of

the dynamic formulations and by designing a dedicated computer architecture.

The proposed solution is a pipelined multiprocessor computer architecture and fast parallel algorithms for real-time control. The multiprocessor system can be utilized to concurrently perform pipelined parallel computations, thereby substantially increasing controller processing speed and CPU utilization.

Concurrent performance of pipelined parallel computations is based on consideration of the sequential dependencies of the dynamic formulations which are conducive to pipelining, and decomposition of the dynamic backward formulations for fast parallel computation. The decomposition of the backward formulations is based on computational simplification techniques.

The performance of the proposed algorithms, called "PAFP," is evaluated through analytic error analysis and experimental simulations, including motion simulations. It is compared to other approaches to the problem proposed by, respectively, Bejczy and Binder. Study results show that the errors introduced by decomposition are relatively small and compare very favorably to those obtained by other methods of computation simplification. The proposed computer architecture and the algorithms may be implemented with multiple low-cost microprocessors. This will allow a practical im-

plementation of a highly parallel structure to achieve real-time robot control with high sampling rates.

Pipelined Multiprocessor Computer Architecture and Fast Parallel Algorithms for Real-Time Robot Control

by

Wangjin Mun

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Completed April 10, 1989

Commencement June 1989

APPROVED:

Redacted for privacy

Associate Professor of Electrical and Computer Engineering in charge of major

Redacted for privacy

Head of Department of Electrical and Computer Engineering

Redacted for privacy

Dean of Graduate School

Date thesis is presented April 10, 1989

Typed by B. McMechan for Wangjin Mun

ACKNOWLEDGEMENTS

I would like to acknowledge the guidance, understanding, and patience given to me by my major professor, Dr. James Herzog, from the beginning through the completion of my graduate work at Oregon State University. I would also like to express my gratitude to my doctoral committee members, Drs. Eugene Fichter, John Murray, John Nabelek, and John Saugen, for reviewing this dissertation.

My special thanks to my wife, Byungjoo, for her encouragement and pursuit of our common goals and for her positive attitude to life through some trying times in our lives. My special thanks are also extended to my parents for their love and for supporting my education.

Dedicated to the memory of my father, Dogab Mun.

Table of Contents

Chapter		Page
1.	Introduction	1
	1.1 Research Scope and Objectives	1
	1.2 Organization of the Study	5
2.	Real-Time Industrial Robot Controls	7
	2.1 Introduction	7
	2.2 Industrial Robots	7
	2.2.1 Kinematics	8
	2.2.1.1 Robot Arm Kinematics	8
	2.2.1.2 Kinematics of the Stanford Robot Arm	9
	2.2.2 Robot Arm Dynamics	10
	2.2.2.1 Lagrange-Euler formulation	10
	2.2.2.2 Newton-Euler formulation	12
	2.3 Trajectory Planning	15
	2.4 Real-Time Robot Control	17
3.	Computer Architecture for Industrial Robot Control	24
	3.1 Introduction	24
	3.2 Computer Architecture of Industrial Robots	24
	3.2.1 Special-Purpose Single-Computer System	26
	3.2.2 Special-Purpose Parallel ComputerSystem	27
	3.3 Real-Time Industrial Robot Control Architecture	31
4.	Pipelined Multiprocessor Computer Architecture for Real-Time Robot Control	35
	4.1 Introduction	35
	4.2 Dynamics and Robot Arm Control	37
	4.3 Computational Simplification Schemes	39
	4.4 System Level Architecture Based on Multiprocessors	41
	4.5 Pipeline/Fast Parallel Algorithms	43
	4.5.1 Fast Parallel Computations Based on the Decomposed Newton-Euler Formulation	43
	4.5.2 Pipelined/Fast Parallel Computing Architecture for Real-Time Control	45
	4.6 Experimental Design	46

Table of Contents (continued)

Chapter		Page
5.	Parallel Computations and Error Analysis of Simplified Decomposition Techniques	55
	5.1 Introduction	55
	5.2 Parallel Computation Decomposition Schemes	55
	5.3 Fast Parallel Computations With Computational Simplification Techniques	57
	5.3.1 Fast Parallel Computations	57
	5.3.2 Dynamic Formulation Using Simplified Decomposition Techniques	63
	5.4 Error Analysis of Simplified Decomposition Techniques	70
	5.4.1 Scalar Case	71
	5.4.2 Vector Case	75
	5.5 Summary	79
6.	Performance Evaluation and Motion Simulations	84
	6.1 Introduction	84
	6.2 Development of the Simulation Model	85
	6.2.1 Trajectory Generation	85
	6.2.2 Simulated Control Methodology	89
	6.3 Torques/Forces Calculations	91
	6.3.1 Procedure	91
	6.3.2 Comparison of Algorithms	92
	6.3.3 Experimental Results, I	94
	6.3.4 Experimental Results, II	97
	6.4 Motion Simulations	98
	6.4.1 Procedure	98
	6.4.2 Experimental Results, III	99
	6.5 Summary	101
7.	Summary, Conclusions and Recommendations	127
	References	131

List of Figures

Figure		Page
2.1	Link Coordinate Systems and Link Parameters	20
2.2	Stanford Robot Arm	21
3.1	PUMA 560 Series Industrial Robot Arm Control Architecture Diagram	32
3.2	Systolic Architecture for the Robot Inertia Matrix	33
3.3	System Configuration of Multiprocessor System	34
4.1	System Level Multiprocessor Architecture	50
4.2	Timing Diagram of Pipelined N-E Computing Architecture	51
4.3	Timing Diagram of the PAFP Computing Architecture	52
4.4	Computing Structure with Buffer for Inverse Dynamics	53
4.5	Robot Joint Trajectories for Simulation	54
5.1	Four Possible Decompositions of the Newton-Euler Dynamic Formulation	80
5.2	Timing Diagrams of Decomposed Computations	81
5.3	Forces and Parameters for Link i	82
6.1	Reference Trajectories of Robot Joint for Motion Simulation Study	102
6.2	End Effector Trajectory of Trajectory 1	103
6.3	Block Diagram of Simulated Control Methodology	104
6.4	Comparison of Trajectory 1 Torque, 6-Second Duration	105

List of Figures (continued)

Figure		Page
6.5	Comparison of Trajectory 1 Torque, 2-Second Duration	106
6.6	Comparison of Trajectory 1 Torque, 1-Second Duration	107
6.7	Comparison of Relative Errors, Trajectory 1, 6-Second Duration	108
6.8	Comparison of Relative Errors, Trajectory 1, 2-Second Duration	109
6.9	Comparison of Relative Errors, Trajectory 1, 1-Second Duration	110
6.10	Comparison of Trajectory 2 Torque, 6-Second Duration	111
6.11	Comparison of Trajectory 2 Torque, 2-Second Duration	112
6.12	Comparison of Trajectory 2 Torque, 1-Second Duration	113
6.13	Comparison of Relative Errors, Trajectory 2, 6-Second Duration	114
6.14	Comparison of Relative Errors, Trajectory 2, 2-Second Duration	115
6.15	Comparison of Relative Errors, Trajectory 2, 1-Second Duration	116
6.16	Comparison of Trajectory 2 Torque, PAFP2, 2-Second Duration	117
6.17	Comparison of Trajectory 2 Torque, PAFP2, 1-Second Duration	118
6.18	Desired and Actual Trajectories, Closed Loop Simulation	119

List of Tables

Table		Page
2.1	Stanford Robot Arm Link Coordinate Parameters	22
2.2	Link Inertias for the Stanford Robot Arm	23
5.1	Comparison of Computation Accuracy Between Decomposition Schemes	83
6.1	Joint Positions for the Interpolation of the 2-Second Trajectory 1	120
6.2	Parameters of 5th Order Polynomial for the Interpolation of the 2-Second Trajectory	121
6.3	Comparison of Relative Root Mean Squared Errors, Trajectory 1, 6-Second Duration	122
6.4	Comparison of Relative Root Mean Squared Errors, Trajectory 1, 2-Second Duration	123
6.5	Comparison of Relative Root Mean Squared Errors, Trajectory 1, 1-Second Duration	124
6.6	Comparison of Relative Root Mean Squared Errors, Trajectory 2, 1-Second Duration	125
6.7	Comparison of Relative Position Errors, Trajectory 1	126

PIPELINED MULTIPROCESSOR COMPUTER ARCHITECTURE AND FAST PARALLEL ALGORITHMS FOR REAL-TIME ROBOT CONTROL

Chapter 1

Introduction

1.1 Research Scope and Objectives

The application of robot systems in the industrial world has increased dramatically during the past decade. As a result of the automation revolution, robots are assuming ever more complex and demanding tasks. However, the potentials of robot systems are presently under-realized due to current computational power limitations. Real-time robot control presents a great challenge to computer control systems. Therefore, the purpose of this research has been the design of a multiprocessor computer architecture associated with computing algorithms which have sufficient computing speed to effect real-time robot control.

Real-time systems differ from general multi-user systems in that absolute deadlines must be accommodated at each sample time [SHIN 87]. Therefore, the speed and reliability with which a robot can be controlled are of primary importance.

In general, high performance robot control requires utilization of the entirety of the robot dynamic equations and the ability to evaluate these equations in real-time. These dynamic equations, which take into account robot nonlinearities and dynamic coupling, are computationally intensive and difficult to implement in a real-time environment. Thus, a rapid and efficient computer structure for these dynamic equations is required in order to achieve real-time control with high sampling rates. The sampling rate directly affects both the speed and accuracy of the system. Higher sampling rates imply not only improved performance, they allow for greater robot arm stiffness. This is desirable in order to reduce the effect of unpredictable external disturbances on trajectory tracking performance [KANA 84, KHOS 87].

To increase computation speed, researchers have simplified the robot dynamic model computations [BEJC 79, BEJC 81, BEJC 83, PAUL 84], customized the equations for specific manipulators [KANA 84, HOLL 84, KHOS 86, KHOS 88a], and improved the computational architecture [LIN 83a, TURN 84, ORIN 84, LATH 85, KASA 85, NIGA 85, LEE 86b, WANA 86, BIND 86, ZHEN 86, LEUN 87, SHAN 88, KHOS 88b].

Simplification of dynamic model computations, when deduced from numerical comparisons for specific robot arms, is dependent on given speeds and motions and has

not been proven to be valid for fast conditions of motion. As a result, this method yields relatively large computational errors. Customizing the equations for specific manipulators requires a variety of robot arm physical simplifications and thus is not valid for very general use. To reduce the servo sampling period, most computational architectures are specific to the tasks necessary to control the robot arm. From a computational standpoint, computing the dynamic equations is the most demanding task. Thus, practical implementation of the dynamic equations to achieve real-time high sampling rates demands a more efficient computational mechanism. This includes an appropriate algorithm to utilize the resources of modern digital hardware.

Among two of the principal robot dynamic equations, the Lagrange-Euler and Newton-Euler dynamic equations [BEJC 79, LUH 80], the Newton-Euler dynamic equations are more efficient [HOLL 80]. The Newton-Euler dynamic equations will therefore be used to build a more efficient computational mechanism.

The Newton-Euler dynamic equations consist of a computationally efficient set of compact forward and backward recursive equations of motion, which are then sequentially applied to the robot links. Computing algorithms based on the Newton-Euler dynamic equations reflect a serially recursive process and the sequential dependencies of the dynamic equations are conducive to

pipelining. Thus, a pipelined computing architecture based on multiprocessors is a logical candidate for the efficient computation of real-time controls. The parallelism of recursive equations allows for multiple processing elements and their sequential nature serves to complement this type of architecture.

The primary purpose of this research is to design a pipelined multiprocessor computer architecture utilizing fast parallel algorithms for real-time control. The intent of this system is to provide concurrent pipelined parallel computations utilizing all of the processors, thereby allowing the practical implementation of a highly parallel structure to achieve real-time high sampling rates.

Concurrent performance of pipelined parallel computations is based on consideration of the sequential dependencies of the dynamic equations, which are conducive to pipelining, and decomposition of the dynamic backward equations for fast parallel computation. The decomposition of the backward equations is based on computational simplification techniques for very general case. These computation techniques have been shown to offer greater accuracy than other approaches discussed in the literature.

The second objective of this study has been the development of a software system capable of enabling computer simulation of the robot motion, a necessity

for the evaluation of the robot motion algorithms. Three criteria, including computation time, accuracy of torques/forces calculations with open-loop simulation, and the accuracy of path-trajectory tracking with closed-loop simulation, have been established for evaluation of the performance of the proposed computing architecture. For simulation purposes, the Stanford arm [BEJC 79, PAUL 84] was used as a test vehicle since it was currently in use in several other associated research projects and its mechanical parameters were readily availability.

1.2 Organization of the Study

Chapter 2 is a presentation of the mathematical computations related to industrial robot controls, accompanied by discussion of the proposed algorithm, which will be referred to as PAFP, Pipelined And Fast Parallel Algorithm.

Chapter 3 is a discussion of related computer architecture studies directed at industrial robot controls. It also includes a discussion of the design requirements of computer architecture for real-time industrial robot control systems.

In Chapter 4, a pipelined multiprocessor computing architecture is developed for this system in which all of the processors can be concurrently utilized to substantially increase controller processing speed.

In Chapter 5, four possible decomposition schemes for a pipelined computing architecture and three computational methods for each are considered in order to determine the most efficient means of parallel computation. Then, an analysis of the most efficient decomposition scheme for pipelined computing architecture, including discussion of three computational methods and an analysis of errors introduced into their algorithms, is presented.

In Chapter 6, the experimental research and the simulation results are presented.

In Chapter 7, conclusions drawn from this research and suggested directions for future research are offered.

Chapter 2

Real-Time Industrial Robot Controls

2.1 Introduction

An industrial robot is a computer-controlled, general-purpose mechanical manipulator consisting of several rigid bodies, called links, connected in series by revolute or prismatic joints. The most apparent robot control problem is to obtain dynamic models of the robot arm which can be used to determine control laws to achieve desired system response and performance.

In this chapter, the mathematical computations related to the dynamic modeling of industrial robots is presented, accompanied by a proposed control algorithm. This analysis is followed by discussions of trajectory planning for obstacle-free arm motion. The real-time control problems are also presented.

2.2 Industrial Robots

In this section, the mathematical computations related to industrial robots are discussed. A proposed control algorithm is presented.

2.2.1 Kinematics

Robot arm kinematics are geometric studies of robot arm motion with respect to fixed-reference coordinate systems. Motion is expressed as a function of time without regard to the forces/moments causing the motion. In this section, the robot arm kinematics used to derive the dynamic arm motion and trajectory planning equations are discussed. An example of the kinematics applied to the Stanford robot arm used in motion simulations for this study is provided. The Stanford robot arm was chosen for the motion simulations in this study because of its use in several other associated research projects and the ready availability of its mechanical parameters.

2.2.1.1 Robot Arm Kinematics

For robot arm kinematics, vector and matrix algebra are commonly utilized to develop a systematic and generalized approach representing and describing robot arm link locations with respect to a fixed-reference system. Since the robot arm links may be rotated and/or translated with respect to the reference coordinate system, a body-attached coordinate system is often established along the joint axis for each link. With the body-attached coordinate system associated with each link expressed in four geometric parameters, Θ_i, d_i, α_i and a_i, a homogeneous transformation matrix,

${}^{i-1}\mathbf{A}_i$, describing the geometric relationship between two adjacent links in space, as shown in Figure 2.1, can be expressed as follows [PAUL 84]:

$$ {}^{i-1}\mathbf{A}_i = \begin{bmatrix} c\Theta_i & -c\alpha_i s\Theta_i & s\alpha_i s\Theta_i & a_i c\Theta_i \\ s\Theta_i & c\alpha_i c\Theta_i & -s\alpha_i c\Theta_i & a_i s\Theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} , \qquad (2.2\text{-}1) $$

where $c\Theta_i \equiv \cos \Theta_i$, $s\Theta_i \equiv \sin \Theta_i$,

$c\alpha_i \equiv \cos \alpha_i$, $s\alpha_i \equiv \sin \alpha_i$,

$a_i \equiv$ the shortest distance between the z_{i-1} axis and the z_i axis,

$\alpha_i \equiv$ the angle between the z_{i-1} axis and the z_i axis about the x_i axis (using the right-hand rule),

$d_i \equiv$ the shortest distance between the x_{i-1} axis and the x_i axis, and

$\Theta_i \equiv$ the joint angle from the x_{i-1} axis to the x_i axis about the z_{i-1} axis (using the right-hand rule).

The homogeneous transformation matrix, ${}^{i-1}\mathbf{A}_i$, is used as the basic tool to obtain the robot arm kinematic equations.

2.2.1.2 Kinematics of the Stanford Robot Arm

The Stanford robot arm has seven links and six joints in addition to a gripper [BEJC 74, BEJC 79, PAUL 84]. An example of applying the approach given in section 2.2.1.1 to the Stanford robot arm is shown in Figure 2.2. Tables 2.1 and 2.2 include, respectively,

the link coordinate parameters and link inertias for the Stanford robot arm.

2.2.2 Robot Arm Dynamics

Robot arm dynamics are mathematical formulations of robot arm equations of motion. They are used to define the "inverse problem" of determining the control torques and forces that each joint motor must supply for the robot arm to follow the desired trajectory. This involves correct joint positions, velocities, and accelerations. Two major formulations, discussed in the following subsections, may be used to model the robot arm dynamics: Lagrange-Euler and Newton-Euler dynamic equations.

2.2.2.1 Lagrange-Euler formulation

In the Lagrange-Euler approach [BEJC 74, LEWI 74, BEJC 79, HOLL 80, BEJC 81, BEJC 83], the Lagrangian formulation for nonconservative systems is used. Through the use of the Lagrange-Euler equation and the robot arm kinematic equations, the Lagrange-Euler dynamics equations for a robot arm with n-joints and (n+1) links can be expressed as follows [BEJC 79]:

$$u_i = \sum_{j=i}^{n} \left[\sum_{k=1}^{j} \mathrm{Tr}[\mathbf{U}_{jk}\mathbf{J}_j(\mathbf{U}_{ji})']\ddot{q}_k \right] + \sum_{j=i}^{n} \left[\sum_{k=1}^{j} \sum_{m=1}^{j} \mathrm{Tr}[\mathbf{U}_{jkm}\mathbf{J}_j(\mathbf{U}_{ji})']\dot{q}_m \dot{q}_k \right] - \sum_{j=i}^{n} m_j \mathbf{g}'\mathbf{U}_{ji}\mathbf{r}_j , \quad \text{for } i = 1,2,\ldots n \qquad (2.2\text{-}2)$$

where Tr = trace operator,

$()'$ = transpose of (),

u_i = generalized input torque/force for joint i,

m_j = mass of link j,

$\mathbf{r}_j$ = a vector describing the center of mass of link j with respect to the j^{th} coordinate system,

$\mathbf{g}'$ = [0,0,9.8 m/s^2], a gravitational acceleration vector at sea level base,

$\mathbf{J}_j$ = inertia matrix for link j, and

$\mathbf{U}_{jk}$, $\mathbf{U}_{jkm}$ = 4 × 4 matrix transforming the vectors and matrices among the various joint coordinate systems.

The computational complexity of the Lagrange-Euler equations of motion (i.e., the multiplications and additions required to compute Equation (2.2-2) for every set point of the trajectory) is of the order, $O(n^4)$. J. M. Hollerbach [HOLL 80] has shown that for a six-link robot arm the number of multiplications and additions are, respectively, 66,271 and 51,548. Assuming the allocation of 2 milliseconds to calculate the equations for real-time control [KHOS 86], this corresponds to approximately 59 million arithmetic operations per second for the equations (assuming that multiplications and additions take approximately an equal amount of performance time, which is the case in many floating

point systems). For example, J. Y. S. Luh, M. W. Walker and R. P. Paul [LUH 80] have estimated that computing the torques for one nominal point in a robot arm trajectory requires 7.9 seconds on a PDP 11/45. Accordingly, it was determined that for all practical purposes it would not be possible to compute the real-time torques/forces for the robot arm trajectories using the Lagrange-Euler dynamic formulation.

Several simplifications to improve the computational feasibility of Lagrange-Euler dynamics for the robot dynamic equations of motion have been proposed [BEJC 81, LUH 81, BEJC 83, PAUL 84]. However, these solutions were deduced from numerical comparisons for specific robot arms, are dependent on given speeds and motions, and have not been proven to be valid for very general and fast conditions of motion.

2.2.2.2 Newton-Euler formulation

An alternative method to obtain the dynamic equations of robot arm motion based on the Newton-Euler equations has been proposed [ARMS 79, ORIN 79, LUH 80, WALK 82]. This technique results in a set of forward and backward recursive equations with multiple vector cross-product terms. The forward recursion propagates kinematic information for each link from the base reference frame to the end-effector. The backward recursion propagates the force exerted on each link from the end-effector of the robot arm to the base reference

frame. From these forces, the required joint torques may be computed [LUH 80]:

1) Forward equations (i=1,2,$\cdots$,n):

$$\boldsymbol{\omega}_i = \begin{cases} \boldsymbol{\omega}_{i-1} + \mathbf{z}_{i-1}\dot{q}_i & \text{if link i is rotational} \\ \boldsymbol{\omega}_{i-1} & \text{if link i is translational} \end{cases} \tag{2.2-3}$$

$$\dot{\boldsymbol{\omega}}_i = \begin{cases} \dot{\boldsymbol{\omega}}_{i-1} + \mathbf{z}_{i-1}\ddot{q}_i + \boldsymbol{\omega}_{i-1} \times \mathbf{z}_{i-1}\dot{q}_i & \text{if link i is rotational} \\ \dot{\boldsymbol{\omega}}_{i-1} & \text{if link i is translational} \end{cases} \tag{2.2-4}$$

$$\dot{\mathbf{v}}_i = \begin{cases} \dot{\boldsymbol{\omega}}_i \times \mathbf{p}_i^* + \boldsymbol{\omega}_i \times [\boldsymbol{\omega}_i \times \mathbf{p}_i^*] + \dot{\mathbf{v}}_{i-1} & \text{if link i is rotational} \\ \mathbf{z}_{i-1}\ddot{q}_i + \dot{\mathbf{v}}_{i-1} + \dot{\boldsymbol{\omega}}_i \times \mathbf{p}_i^* + 2\boldsymbol{\omega}_i \times \mathbf{z}_{i-1}\dot{q}_i + \boldsymbol{\omega}_i \times [\boldsymbol{\omega}_i \times \mathbf{p}_i^*] & \text{if link i is translational} \end{cases} \tag{2.2-5}$$

where $\dot{q}_i$ = joint velocity of joint i,

$\ddot{q}_i$ = joint acceleration of joint i,

$\boldsymbol{\omega}_i$ = angular velocity vector of link i,

$\dot{\boldsymbol{\omega}}_i$ = angular acceleration vector of link i,

$\dot{\mathbf{v}}_i$ = linear acceleration vector of link i,

$\mathbf{p}_i^*$ = origin of the coordinate system with respect to $(x_{i-1}, y_{i-1}, z_{i-1})$, and

$\mathbf{z}_{i-1}$ = components of a vector about the joint axis.

2) Backward Equations (i=n,n-1,$\cdots$,1):

$$\mathbf{f}_i = \mathbf{f}_{i+1} + m_i(\dot{\boldsymbol{\omega}}_i \times \mathbf{s}_{ci} + \boldsymbol{\omega}_i \times [\boldsymbol{\omega}_i \times \mathbf{s}_{ci}] + \dot{\mathbf{v}}_i) , \tag{2.2-6}$$

$$\mathbf{n}_i = \mathbf{n}_{i+1} + \mathbf{p}_i^* \times \mathbf{f}_{i+1} + (\mathbf{p}_i^* + \mathbf{s}_{ci}) \times (m_i [\dot{\boldsymbol{\omega}}_i \times \mathbf{s}_{ci} + \boldsymbol{\omega}_i \times [\boldsymbol{\omega}_i \times \mathbf{s}_{ci}] + \dot{\mathbf{v}}_i)] + \mathbf{I}_{ci}\boldsymbol{\omega}_i + \boldsymbol{\omega}_i \times (\mathbf{I}_{ci}\boldsymbol{\omega}_i) , \tag{2.2-7}$$

$$u_i = \begin{cases} \mathbf{n}^T_i \, \mathbf{z}_{i-1} + b_i \dot{q}_i \\ \qquad \text{if link } i \text{ is rotational} \\ \mathbf{f}^T_i \, \mathbf{z}_{i-1} + b_i \dot{q}_i \\ \qquad \text{if link } i \text{ is translational} \end{cases} \tag{2.2-8}$$

where m_i = total mass of link i,

b_i = the viscous damping coefficient for joint i,

$\mathbf{f}_i$ = force exerted on link i by link (i-1) at the $(x_{i-1}, y_{i-1}, z_{i-1})$ coordinate system,

$\mathbf{n}_i$ = moment exerted on link i by link (i-1) at the $(x_{i-1}, y_{i-1}, z_{i-1})$ coordinate system,

$\mathbf{s}_{ci}$ = center of mass of link i from the origin of the (x_i, y_i, z_i) coordinate system,

$\mathbf{I}_{ci}$ = inertia matrix of link i about its center of mass, referring to the (x_i, y_i, z_i) coordinate system,

u_i = input torque/force for joint i, and

T = transpose of a vector.

The resulting dynamic equations, excluding the dynamics of the control device, backlash, and gear friction, are a set of forward and backward recursive equations. These equations can be sequentially applied to the robot links, thereby avoiding the complicated calculations of Lagrange-Euler system equations. The most important result of this formulation is that the computation time for the generalized torques/forces is linearly proportional to the number of robot arm joints and is independent of the robot arm configuration.

Practical implementation of the Newton-Euler inverse dynamics formulation to achieve real-time sampling rates (i.e., of a few milliseconds) demands an efficient algorithm utilizing the capabilities of modern digital hardware. Methods proposed to reduce the computation-time required for the Newton-Euler formulation include customization of the equations for specific robot arms [KANA 84, HOLL 84, KHOS 85, KHOS 86, KHOS 88a] and the use of a prediction-algorithm for parallel computations [BIND 86].

In summary, the Lagrange-Euler dynamic equations are computationally inefficient for use as real-time controls unless the equations of motion can be simplified. Applying the Newton-Euler dynamic equations results in a more efficient equations of robot motion. P.K. Khosla and S. Ramos [KHOS 88b] have also noted that from the point of view of hardware, the Newton-Euler dynamic equations may also be more efficiently applied than the Lagrange-Euler dynamic equations.

2.3 Trajectory Planning

Trajectory planning interpolates or approximates desired paths through application of a class of polynomial functions. It generates a sequence of time-based robot arm control set points from the initial location to the destination of the arm [LIN 83b, CART 84, LEE 86a]. Trajectory planning accepts input variables

which indicate the constraints of the path, outputting a sequence of time-based intermediate robot arm configurations. These configurations are expressed either in joint or cartesian coordinates, from the initial location to the final location, and are used as joint variables for calculating the desired torques/forces that each joint motor must supply in order to determine desired robot arm trajectories.

Two major approaches, the joint interpolated approach and the Cartesian space approach, are used to plan robot arm trajectories [CRAI 86]. The joint interpolated approach plans the time-history of all joint variables, along with their first two time derivatives, to describe a desired robot arm motion [TAYL 79, LIN 83b]. Planning in the joint-variable space offers three advantages:

1) the trajectory is planned directly in terms of controlled variables during motion,
2) trajectory planning can be done in near real-time, and
3) joint trajectories are easier to plan.

The disadvantage of this method is the difficulty in determining the location of the various links and the end-effector during motion.

For Cartesian space planning, the time-history of the end-effector's position, velocity and acceleration

are planned, and the corresponding joint positions, velocities, and accelerations are derived from the end-effector information [PAUL 79, LEE 84]. The Cartesian space approach offers the advantage of straightforward methodology. However, since at this time no sensor exists which is capable of measuring the end-effector in Cartesian coordinates, Cartesian space-path planning requires a transformation between the Cartesian joint coordinate system and real-time application, a task that is computationally intensive. Furthermore, the transformation from the Cartesian coordinate system to a joint coordinate system is not a process of one-to-one mapping. Because of these disadvantages, the joint space-oriented method is more commonly used.

2.4 Real-Time Robot Control

Industrial robot arm control problems may be analyzed as two coherent sets of subproblems: (1) trajectory planning problems and (2) motion-control problems. Trajectory planning has been described in section 2.3. For motion-control problems, it is assumed that the desired motion is specified by a time-based trajectory of the robot arm in joint space. In general, the motion-control problem is to determine those control laws capable of achieving desirable system responses and performance.

Most current industrial approaches to industrial robot control system design treat each joint of the robot as a simple joint servomechanism. This approach models the varying dynamics of a robot arm inadequately since the motion and configuration of the entire arm mechanism are neglected. Changes in the parameters of the controlled system may be significant enough to render conventional feedback control strategies ineffective, resulting in reduced servo response speed and damping response, which limit the precision and speed of the end-effector. As a result, robot arms controlled in this manner may move with noticeable vibrations at high speeds.

One solution for this problem has been the use of real-time digital controls in which the applied robot arm torques are computationally derived from the complete dynamic model of the robot arm, called the model-based control law. For example, the computed torque method is a model-based control method which utilizes dynamic modeling of the feedforward path [LEE 82a, ORIN 84, AN 87, KHOS 88a]. This feedforward signal is augmented with a feedback signal derived from a linearly independent joint-controller, used to correct small deviations in trajectory tracking. The underlying concept is that the feedforward torques/forces provide gross signals and the independent joint-controllers

provide the correcting control signals for the rejection of unknown external disturbances.

This method is difficult to implement because for every sampling period, the dynamic equations of motion for the robot arm must be solved for every variable, including unknown loads, in order to determine appropriate torques/forces for each robot arm joint. This is the "inverse dynamics" problem [ORIN 84], which is the single greatest problem in achieving robot control. For most systems, model-based control laws require that the inverse dynamics be computed at real-time sampling intervals of a few milliseconds [KHOS 86, KHOS 88b]. This period is difficult to achieve due to the large number of vector and matrix operations assoiated with the inverse dynamics. Thus, for real-time robot controls, a method to quickly and efficiently evaluate the inverse dynamics is required. Increasing the sampling rate by reducing the computation time is of value since the real-time control sampling rate directly affects both the speed and accuracy of the system. A higher sampling rate implies not just improved performance, it would allow for greater robot arm stiffness. This is desirable to reduce the effect of unpredictable external disturbances on trajectory tracking performance [KANA 84, KHOS 87, KHOS 88a].

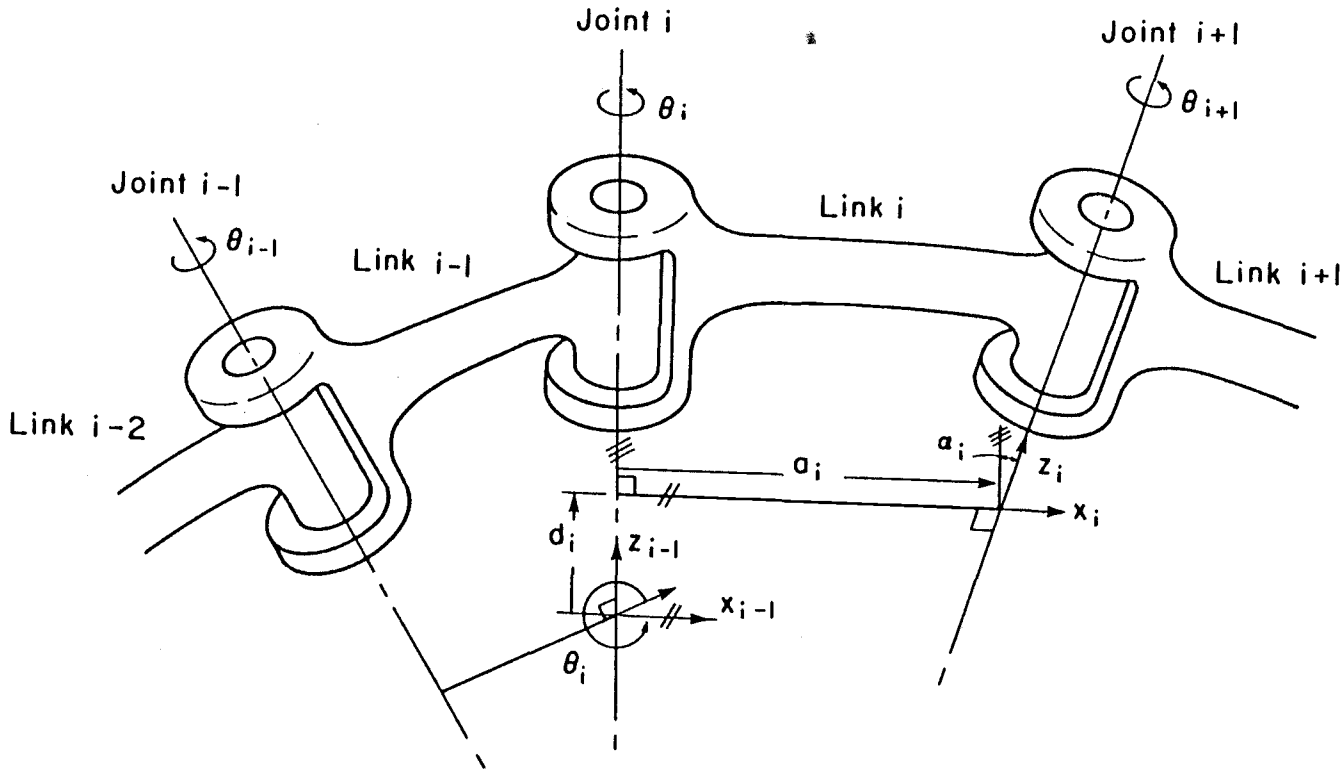

Fig. 2.1 Link Coordinate Systems and Link Parameters [PAUL 84].

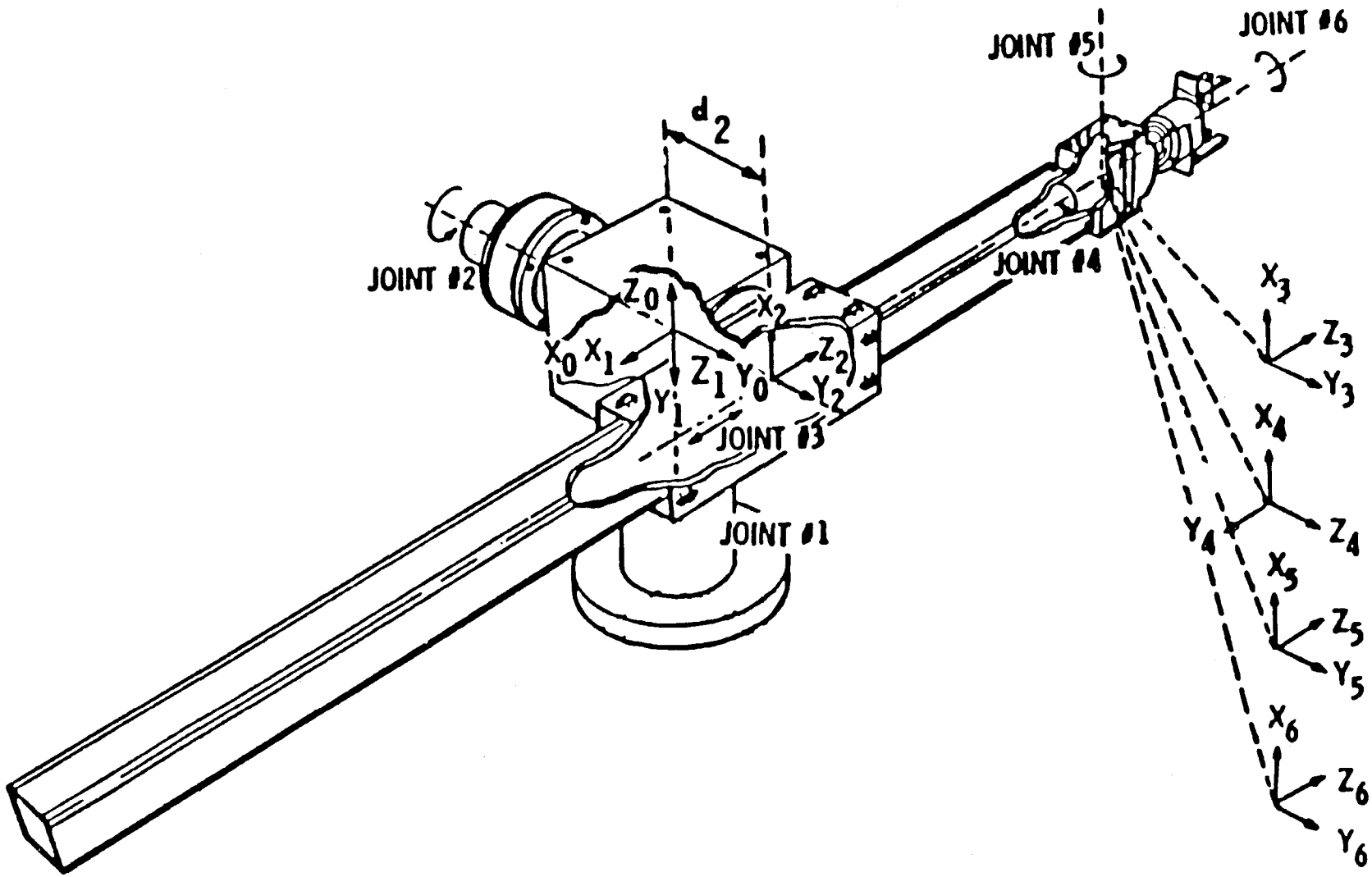

Figure 2.2 Stanford Robot Arm [modified BEJC 79].

Table 2.1 Stanford Robot Arm Link Coordinate Parameters.

Joint	Θ_i [deg]	d_i [m]	α_i [deg]	a_i [m]
1	$\Theta_1(t)$	0	-90	0
2	$\Theta_2(t)$	0.154	90	0
3	0	$d_3(t)$	0	0
4	$\Theta_4(t)$	0	-90	0
5	$\Theta_5(t)$	0	90	0
6	$\Theta_6(t)$	0	0	0

Table 2.2 Link Inertias for the Stanford Robot Arm [PAUL 84].

Link	Mass (kg)	mk_{xx}^2 (kgm^2)	mk_{yy}^2 (kgm^2)	mk_{zz}^2 (kgm^2)
1	9.29	0.276	0.255	0.071
2	5.01	0.108	0.018	0.100
3	4.25	2.510	2.510	0.006
4	1.08	0.002	0.001	0.001
5	0.63	0.003	0.003	0.0004
6	0.51	0.013	0.013	0.0003

Chapter 3

Computer Architecture for Industrial Robot Control

3.1 Introduction

Computer architecture for the control systems of industrial robots is reviewed in this chapter. A basic review of the architecture of industrial robot control and a discussion of some of the considerations involved in real-time industrial robot control are also included.

3.2 Computer Architecture of Industrial Robots

A basic approach to the design of industrial robot control systems has been to treat each robot joint as a simple servomechanism since present state-of-the-art robotic systems lack the computational power required for the real-time evaluation of complex control algorithms. In order to describe the control architecture of current industrial robots, the control architecture of a typical industrial robot, the PUMA 560 series, is described in this section.

Control structures of the PUMA 560 series, as shown in Figure 3.1, are hierarchically arranged and at the system hierarchy high level the host computer performs two main major functions [LEE 82b]:

1) On-line user interaction from the user's commands; and

(2) Subtask coordination to carry out the command, performed with six microprocessors.

Joint controllers for the direct control of each axis of motion, each with an integrated microprocessor, are at the low level of the system hierarchy. The main functions of the microprocessors include the following:

(1) Every 28 milliseconds, receive and acknowledge trajectory set points from the system computer and perform interpolation between the desired joint value and the joint value.

(2) Approximately every 1 millisecond, read the incremental values from the encoder mounted at each axis of rotation.

(3) Update the error between the joint-interpolated set points and the actual values and convert the error actuating signals to voltages using the DACs, sending the voltages to the analog servo board which moves the joint.

Basically, the control scheme in Figure 3.1 is a position plus derivative control method. One of main drawbacks of this control scheme is that the servomech-

anism approach used to model the varying dynamics of a robot arm is inadequate since the motion configuration of the whole arm mechanism is neglected. As a result, the robot arm moves at high speeds only with noticeable vibration [LEE 82b].

One possible solution for this problem is the use of a real-time control system in which the robot arm applied torques are obtained by a computer based on complete dynamic modeling of the arm, called model-based control laws. As discussed in Chapter 2, the principal problem in the use of this technique is the excessive computation time required for the formulation of dynamic equations. Consequently, the computing structure must be improved in order to achieve rapid and efficient evaluation of the dynamic equations in real-time.

3.2.1 Special-Purpose Single-Computer System

A number of efforts have been undertaken to relieve the host computer of some of the computational burden associated with industrial robot control. One option has been to use the host computer as a supervisory machine, incorporated with an attached special-purpose processor for performance of the bulk of the numerically intensive control algorithm computations [TURN 81, LEE 82b, LING 88]. The host computer performs trajectory planning, coordinate system transfor-

mations, and processes coordination with other robots. The special-purpose processor is a single-chip processor performing all of the vector and matrix operations required by the robotic control algorithm, thus relieving the host computer from a number of computation tasks. In practice, however, the quantity of data transferred back and forth between the attached processor and the host computer is often so great that an "I/O bottleneck" is created at the interface between the two, thereby diminishing a potentially substantial speed increase.

3.2.2 Special-Purpose Parallel Computer System

Another design approach has been the use of special-purpose computers with a parallel computing structure, based on a suitable algorithm, for the solution of specific problems. Parallel computers can be divided into three architectural configurations [HWAN 84]: the pipelined computer, Array Processors and multiprocessor systems.

Pipelined computers use the same hardware to perform overlapped computations, exploiting temporal parallelism. Array processors execute the same instructions on an array of processors used to achieve spatial parallelism. Multiprocessor systems achieve asynchronous parallelism through a set of interactive processors with shared resources. However, these three

structures are not mutually exclusive and may be used jointly to achieve a higher degree of multiplicity.

Pipelining is employed in almost all computer architectures because of its low hardware overhead and its ability to speed up the overall computation throughput by increasing the utilization of hardware resources [KOGG 81]. Pipelining may be exploited at various levels, ranging from pipelining within an arithmetic unit to the pipelining of multiple processors. D. E. Orin has proposed a processor pipelining scheme in which the data stream is processed by a cascade of processors [ORIN 84]. By executing some tasks concurrently on several processors, Orin has been able to reduce the total time of the overall problem of robot control by nearly one-third. However, with this system the inverse dynamics task was active for one-half the total real-time of execution. Reducing computation time for the inverse dynamics task has been identified as the only way to further increase throughput.

In order to reduce the inverse dynamics computation time, Nigam and Lee have proposed a multiprocessor architecture based on the functional decomposition of the robot arm equations of motion [NIGA 85]. The Newton-Euler formulation for inverse dynamics has been considered for this task. The recursive nature of the Newton-Euler equations of motion lend themselves to

decomposition in the terms used to generate the recursive forward and backward equations. The architecture is then tuned to the functional flow of the decomposed recursive Newton-Euler equation.

Array processors are very effective for certain regularly structured computational tasks, such as vector operations for the inertia matrix, as shown in Figure 3.2 [AMIN 87], and for the Jacobian [ORIN 87]. The linear speedup of array processors makes them most attractive [WAH 87]. However, their hardware requirements are linearly related to the amount of concurrency exploited. Furthermore, array processor performance is frequently dependent upon the performance of the communication structure used, which in turn is dependent upon the computational tasks executed with the processors. The lack of flexibility of array processors severely limits the number of applications for which they may be used.

Multiprocessor systems are good candidates as reliable and flexible system controllers [OZGU 85, CHEN 86]. Luh and Lin proposed a multiprocessor-based architecture for the execution of the inverse plant dynamic equations, using Newton-Euler dynamic equations divided into 18 subtasks for parallel processing [LUH 82]. In this architecture, a scheduling algorithm was proposed for the reduction of computation time. The

result, based on the use of 6 processors, was a reduction by a factor of 2.6 for the Stanford robot arm.

Kasahara and Narita proposed a different algorithm for the same purpose [KASA 85]. The system utilized off-the-shelf Intel 8086 16-bit microprocessors and 8087 coprocessors that jointly served as parallel processors. A total of seven processors (denoted OP), together with a common memory (CM), are connected to a common bus to form a typical multiprocessor system, as shown in Figure 3.3. However, in both of the multiprocessor systems, the important problem of interprocessor communication time and the pipelined nature of the Newton-Euler dynamic equations were ignored.

There are two basic architectural models for multiprocessor systems: (1) tightly coupled multiprocessors, in which all the processors communicate through shared main memory, and (2) loosely coupled multiprocessors, in which each processor has its own local memory and I/O channels. The distributed multiprocessor system is a special instance of the loosely coupled multiprocessor system.

A distributed multiprocessing system, proposed by Binder and Herzog as a means to reduce robot controller computation time, is a collection of nearly autonomous processing nodes that communicate to realize a coherent computing system [BIND 86]. They distributed the control law over multiple computers, one for each joint.

Each computer calculated its own dynamic equations for its joint torque and then serially controlled its own joint, which was directly connected to the computer. This serial control scheme with its distributed computing architecture calls for lengthy sampling periods since the computer also performs as the servo-controller for its own joint.

3.3 Real-Time Industrial Robot Control Architecture

Real-time systems are different from general multi-user systems since an absolute deadline must be dealt with at each sample time. Therefore, speed and reliability are generally of primary importance [SHIN 87, STAN 88]. Usually, in a real-time system such as a robot, a large number of vector and matrix operations must be computed with time limits of just a few milliseconds.

Most of the reviewed architectural implementations were specific to the control algorithm used, resulting in increased performance. Thus, the most attractive way of achieving significant industrial robot control real-time performance gains would appear to be through consideration of more efficient dynamic modeling and through the use dedicated computer architecture design incorporating both parallelism and special purpose hardware.

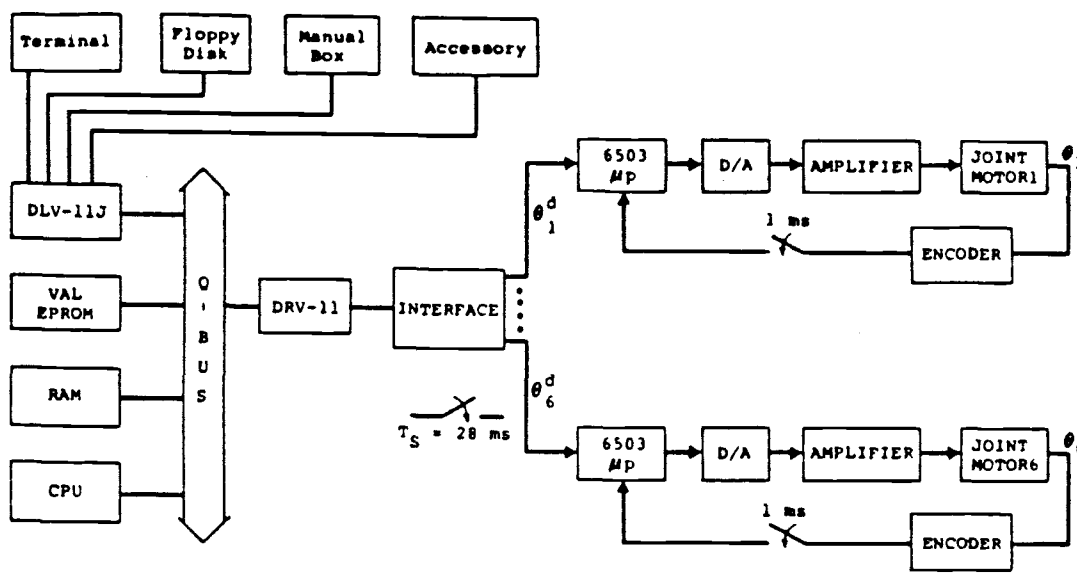

Figure 3.1 PUMA 560 Series Industrial Robot Arm Control Architecture Diagram [LEE 82b].

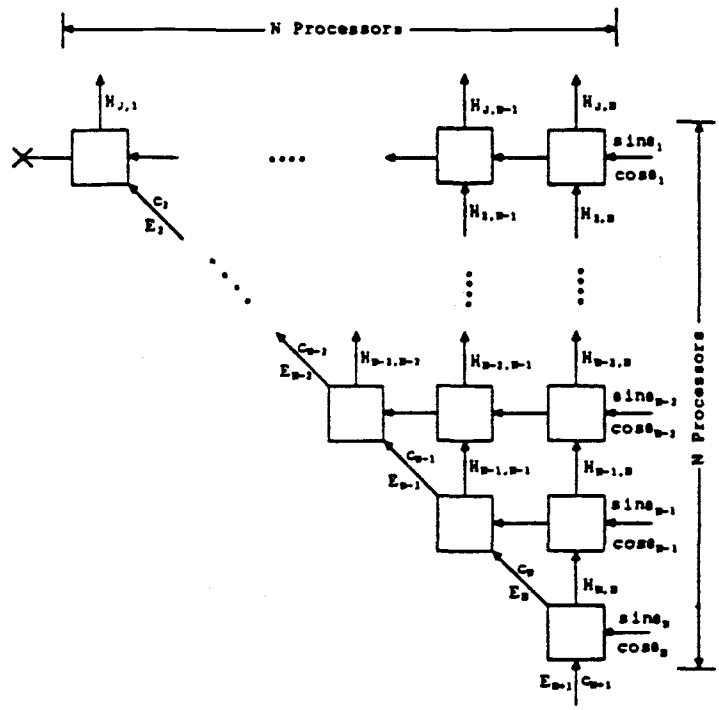

Figure 3.2 Systolic Architecture for the Robot Inertia Matrix [AMIN 87].

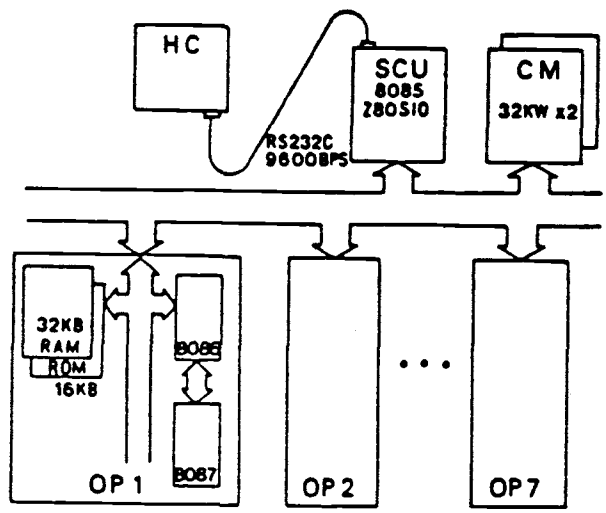

Figure 3.3 System Configuration of Multi-processor System [KASA 85].

Chapter 4

Pipelined Multiprocessor Computer Architecture for Real-Time Robot Control

4.1 Introduction

Robot manipulators are highly nonlinear systems, and their motion control laws involve computation of the appropriate torques/forces to servo each of the joints of the robot manipulator in order to track desired trajectories. These computations are based on dynamic equations for the robot manipulator and on feedback information describing actual motions.

The dynamic equations are computationally intensive and generally considered the principal challenge in the achievement of robot control. Because of this computational burden, the current generation of industrial robots do not consider robot dynamics, particularly the interaction forces among all the various joints. The result is that these systems offer limited performance.

Improved performance advanced control laws based on dynamic formulations have been developed for the computation of torques/forces to servo the robot

joints, but improved robot performance has not been achieved with respect to real-time controls. To obtain real-time dynamic controls, the robot system must provide sufficient computational speed for the dynamic equations since absolute deadlines must be met at each sampling time. For most systems, control laws require computation of the dynamic equations at real-time sampling periods of a few milliseconds [KHOS 86, KHOS 88b]. This is difficult to achieve since there are a large number of vector and matrix operations associated with the computation of dynamic equations. Thus, a fast and efficient evaluation of the dynamic equations and a dedicated computer architecture design for the algorithm are required in order to achieve real-time high sampling rates.

Several approaches have been proposed to reduce dynamics equation computation time, including simplification of dynamic model computations [BEJC 79, BEJC 81, BEJC 83, PAUL 84], customization of the dynamic equations for specific manipulators [KANA 84, HOLL 84, KHOS 86, KHOS 88a], and improvement of computing architecture [ORIN 84, KASA 85, NIGA 85, WANA 86, BIND 86, KHOS 88b]. In this chapter, a Pipelined Multiprocessor Robot Control architecture (denoted PMRC) is presented as a system for robot control. In addition, a computational method, using a pipelined and parallel algorithm based on decomposition of the Newton-Euler dynamic

equations, is included. This approach obtains sufficient improvement of computation time to achieve real-time control with high sampling rates.

4.2 Dynamics and Robot Arm Control

There are two principal formulations which can be used to model robot dynamics: the Lagrangian-Euler dynamic formulation [BEJC 79, BEJC 83] and the Newton-Euler dynamic formulation [LUH 80, WALK 82]. The Lagrangian-Euler dynamic formulation may be used to generate a set of closed form differential equations describing motions. However, this method is computationally inefficient, based upon an order, $O(n^4)$, where n is the number of degrees of freedom of the robot arm.

The Newton-Euler dynamic equation of robot arm motion may be used to obtain a computationally efficient set of compact forward and backward recursive equations of motion, which are then sequentially applied to the robot links. The forward recursion propagates kinematic information from the base coordinate frame to the hand coordinate frame. The backward recursion propagates the force and moments exerted on each link from the end-effector of the robot arm to the base reference frame. To compute the joint torques/forces, the Newton-Euler dynamic equation of motion has the order, $O(n)$.

The control problem of this dynamic equation is to determine the appropriate torques/forces which can be used to servo all the robot arm joints in real-time in order to track desired position trajectories. Several advanced control laws based on these equations, including a model-based control scheme for the robot arm, have been used to demonstrate the improved joint control performance [LEE 84, DESI 87, KHOS 88a]. The basic concept underlying the control laws is a feedforward control method utilizing dynamic modeling in the feedforward path. This feedforward component computes gross joint torques/forces along desired trajectories while compensating for robot nonlinear dynamics and dynamic coupling. The feedforward torques/forces signal is then augmented with a feedback signal derived from the linear independent joint controller, which can be assumed to correct for small deviations due to unknown external disturbances in trajectory tracking.

Controlling the robot manipulator reduces to computing control laws for every sampling period and robot system performance is directly influenced by the computational speed of the control laws. The higher the computation rate, the less sensitive the robot control system is to external disturbances. Thus, higher computation speeds make the system more robust with respect to the effect of unknown external disturbances on trajectory tracking performance [KANA 84, KHOS 87].

4.3 Computational Simplification Schemes

The inefficiency of control laws based on the complete Lagrangian-Euler dynamic formulations of motion prohibits their use in this instance. Simplified approximation models, based on a significance analysis of the terms, are more commonly used for the control of robot arms [BEJC 79, BEJC 81]. Paul [PAUL 84] used the same method, but also suggested some of the link coupling terms. Specifically, the simplified method presented by Bejczy and improved by Paul has been chosen for comparison with the performance of the algorithm proposed in this chapter.

Another and more efficient approach is to compute the inverse dynamics, using the Newton-Euler dynamic formulations of motion. Due to their relatively low computation requirements and their pipelined recursive structure, the Newton-Euler dynamic formulations are good candidates for implementation. However, practical implementation of the Newton-Euler inverse dynamic formulations to achieve real-time sampling rates (of a few milliseconds) demands an efficient algorithm utilizing the capabilities of modern digital hardware [KANA 84, KHOS 88a].

Some of the methods proposed to reduce computation time for the Newton-Euler dynamic formulations include

customization of the equations for specific robot arms [KANA 84, HOLL 84, KHOS 86, KHOS 88a] and use of a prediction algorithm for distributed computation [BIND 86]. Customization of the equations requires a variety of robot arm physical simplifications and thus is not valid for general use.

Binder [BIND 86] distributed the dynamic equations over multiple computers, one for each joint. In order to calculate the dynamic equations, he first divided every sampling interval, T, into 2n subintervals, T(1), T(2),... and T(2n), where n is the total number of robot arm joints. Then, one time slot was allocated to each step in the forward and reverse process. During the first n time slots, each computer, n, serially calculated its forward equations at each time slot. Then, during next n time slot, T(n+1), T(n+2),... and T(2n), the backward equations calculations were initiated from computer n, proceeding in reverse order to computer 1. In order to reduce the computation time required for the serial calculation method, Binder caused each computer to calculate its own dynamic equations for its joint torque while controlling the joint, which was directly connected to the computer. Computation of the dynamic equations for each joint was achieved by substituting predicted values, based on the values calculated during previous sampling intervals, for the actual values of all the forward and backward equation

variables. Based on this algorithm, computation time for the dynamic equations was reduced. However, the approach did not properly take into account the sequential dependencies of the dynamic equations. As a result, Binder's algorithm yielded errors in the calculation of the dynamic equations. Another problem was that the control scheme with this distributed computing architecture called for lengthy sampling periods since the computer also performed as a servo-controller for its own joint.

4.4 System Level Architecture Based on Multiprocessors

The Newton-Euler dynamic equation is a sequentially recursive process. The sequential process, which is also associated with one of the robot's joints, allows for multiple processing elements, while the sequential nature of the equations lends itself to a pipelined architecture. Therefore, a pipelined computing architecture based on multiprocessors is a logical candidate for the efficient computation of real-time controls.

One possible control system, shown in Figure 4.1, consists of three integral components: a system supervisor computer, a pipelined n-processor array, and microprocessor-based individual joint servo-controllers. The supervisor computer, capable of accommodating any user interface, controls the pipelined n-processor

array and the microprocessor-based joint controllers and shares a common memory segment with the microprocessor-based servo-controllers. Communications are established by reading/writing data to and from this common memory segment. In turn, the pipelined n-processor array calculates the desired torques/forces, passing results to the supervisor computer through one of the output ports. As illustrated in the pipeline algorithm shown in Figure 4.1, one processor is assigned to each stage of the pipeline and each processor i performs the computations associated with a particular link i.

For every sampling period in this system, the supervisor computer reads the joint positions and velocities from the shared memory and transfers the necessary parameters, including desired positions, velocities, and robot arm accelerations, to the n-processor array, receiving the results, which include the torques/forces of the joint actuators, from the processor array. All of the necessary data transferred between any two adjacent processors are also indicated in Figure 4.1.

Figure 4.2 shows the computation timing for the various processors. Each processor calculates its own joint variables, passing the computed results needed for performance of the recursive calculations associated with the Newton-Euler formulation to the adjacent processors. The joint torque u_i for a rotational joint

is just the component of the moment vector about the joint axis. On the basis of the timing chart provided in this figure, an initiation rate of 15(time unit)$^{-1}$ and a CPU utilization rate of approximately 33.3 percent are achieved.

4.5 Pipeline/Fast Parallel Algorithms

4.5.1 Fast Parallel Computations Based on the Decomposed Newton-Euler Formulation

In the architecture described in the previous section, following calculation of the forward equations, processor i remains idle until processor i+1 is ready to transmit the values of the backward variables of link i+1. As a result, the initiation rate and CPU utilization are decreased.

For a real-time control system, the sampling rate is directly influenced by the initiation rate of the pipelined computing architecture. In order to increase the initiation rate and CPU utilization, it is necessary for processor i to calculate its backward equations without idle periods following calculation of its forward equations. In the current sampling period, if processor i can obtain the backward variables, ${}^{i+1}\mathbf{f}_{i+1}(t)$ and ${}^{i+1}\mathbf{n}_{i+1}(t)$, of link i+1 without waiting for those values from processor i+1, then following calculation of its forward variable, ${}^{i}\dot{\mathbf{v}}_{i}(t)$, processor

i can calculate the backward variables, ${}^{i}\mathbf{f}_i(t)$ and ${}^{i}\mathbf{n}_i(t)$ without idle periods, as shown in Figure 4.3. In this case, the forward variables, ${}^{i}\boldsymbol{\omega}_i(t)$, ${}^{i}\dot{\boldsymbol{\omega}}_i(t)$ and ${}^{i}\dot{\mathbf{v}}_i(t)$, are exactly calculated and pipelined to the adjacent processor i+1.

If each processor is able to calculate its backward equations as shown in Figure 4.3, the processor array may be used to concurrently perform pipelined parallel computations, thereby greatly increasing controller processing speed and CPU utilization.

The problem which remains is how to properly calculate the backward variables, ${}^{i+1}\mathbf{f}_{i+1}(t)$ and/or ${}^{i+1}\mathbf{n}_{i+1}(t)$. Two simple decomposition schemes for the provision of acceptable backward variable values are (1) to use the immediately previous calculated value of the backward variable and (2) to consider the rate change of the previous values in addition to the method specified in (1). The second method may provide for better accuracy, but at the same time it requires a few more vector operations than the first method for determination of each backward variable .

In this study, the first method, the Pipelined And Fast Parallel algorithm (PAFP), has been used since it does not require extra mathematical computations for calculation of backward variables, ${}^{i+1}\mathbf{f}_{i+1}(t)$ and/or ${}^{i+1}\mathbf{n}_{i+1}(t)$, at the current sampling period. The underlying concept for this method is that over short

sampling periods of a few milliseconds, errors in calculation of the backward variables will be very small since the value of the recursive backward variables will have changed very little. Moreover, since the backward variables use the exact values of their forward variables, the effect on calculating the backward variables of link i, ${}^{i}\mathbf{f}_{i(t)}$ and/or ${}^{i}\mathbf{n}_{i(t)}$, due to the difference in ${}^{i+1}\mathbf{f}_{i+1(t)}$ and/or ${}^{i+1}\mathbf{n}_{i+1(t)}$ between successive sampling intervals, will be relatively small.

From the viewpoint of system control, if the torque values used for the feedforward gross signal are correct, with feedback signals used for correction of small deviations due to unknown external disturbances, including very small torque errors introduced into the algorithm, in trajectory tracking, then the PAFP can be used for the practical implementation of real-time controls with high sampling rates. To test the algorithm, a series of simulations have been undertaken to demonstrate performance (see section 4.6).

4.5.2 Pipelined/Fast Parallel Computing Architecture for Real-Time Control

A simple means of implementing the PAFP is to place a buffer between any two adjacent processors, as shown in Figure 4.4. The role of the buffer is to store the values of only the backward variables,

${}^{i+1}f_{i+1}(t)$ and ${}^{i+1}n_{i+1}(t)$, of the link i+1, calculated for the immediately previous sampling period, and to provide these values to an adjacent processor i for calculation of the backward equations of link i at the current sampling period. Intercommunication between the adjacent processors in the architecture described in Figure 4.1 may be controlled by utilization of precisely synchronized microprograms. Thus, since the time to transmit or receive data can be known, it is possible to use the buffers to provide the data to the adjacent processor i for calculation of the backward equations of link i.

Using this approach, the processor i calculates its torque/force without idle periods following calculation of its forward equations. The results, shown in Figure 4.3, are an initiation rate of 5(time unit)$^{-1}$ and approximately 100 percent CPU utilization. Therefore, the improvement in computational speed and CPU utilization for the PMRC is of the order of three times greater than that achieved in previously utilized architecture.

4.6 Experimental Design

In order to evaluate the performance of the PAFP in this chapter, two criteria, (1) accuracy of torque/force calculations with open loop simulation and (2) accuracy of path trajectory tracking with closed loop

simulation, were selected. In general, there are two principal approaches to system performance evaluation: the analytical approach and the simulation approach. A precise analytic analysis of the two criteria for the PAFP would be very complicated for reasons of calculation complexity, the number of computations involved, and time variances in the nonlinear dynamic equations. Therefore, a series of simulations have been used for the evaluation of the PAFP.

For simulation purposes, the Stanford robot arm described in Chapter 2 was used as a test vehicle because of the availability of its mechanical parameter specifications and since it was currently in use in several other associated research projects.

Experimental simulations involved evaluation of the torques/forces generated with the PAFP for different robot arm trajectories, each of which represented a different motion condition. A number of trajectories were used for evaluation of the accuracy of the algorithm. In this section, the results of one representative trajectory is presented. However, the results were similar to those obtained for other trajectories.

The representative trajectory of the end effector is a straight line path, parallel to the Cartesian Y-axis of the base coordinate system and divided into three segments. In the first segment, each joint was accelerated from rest to a constant velocity, maintain-

ing this velocity through the second segment. In the third segment the joints were decelerated to a rest condition. For smooth movements, a separate interpolation at every segment was performed in the joint coordinates and a fifth order polynomial was used to interpolate between the given points within the boundary conditions. Figure 4.5 shows plots for the six joint trajectories that satisfied the given constraints.

For further performance evaluation of the PAFP, constraints on the total time allowed to complete a movement were given. By changing the total time required for completion of identical trajectories, the effect of different accelerations and velocities on the accuracy of the PAFP could be observed. The total elapsed time constraints were 6[s], 2[s] and 1[s] for, respectively, slow movement, fast movement, and very fast movement.

Within these constraints, the torques/forces were computed for each case, based upon a five-millisecond sampling period, and then compared, using four different methods of computation: exact computations, Bejczy's algorithm, Binder's algorithm and the PAFP. The exact computation, based on the full Newton-Euler dynamics equation, was taken as a reference for comparison with the other three methods. The relative torque errors along the trajectory were then calculated by dividing the torque error by the full scale torque or

force, in turn calculated as the difference between the maximum torque/force and the minimum torque/force for every plot.

The experimental results will be found in Chapter 6.

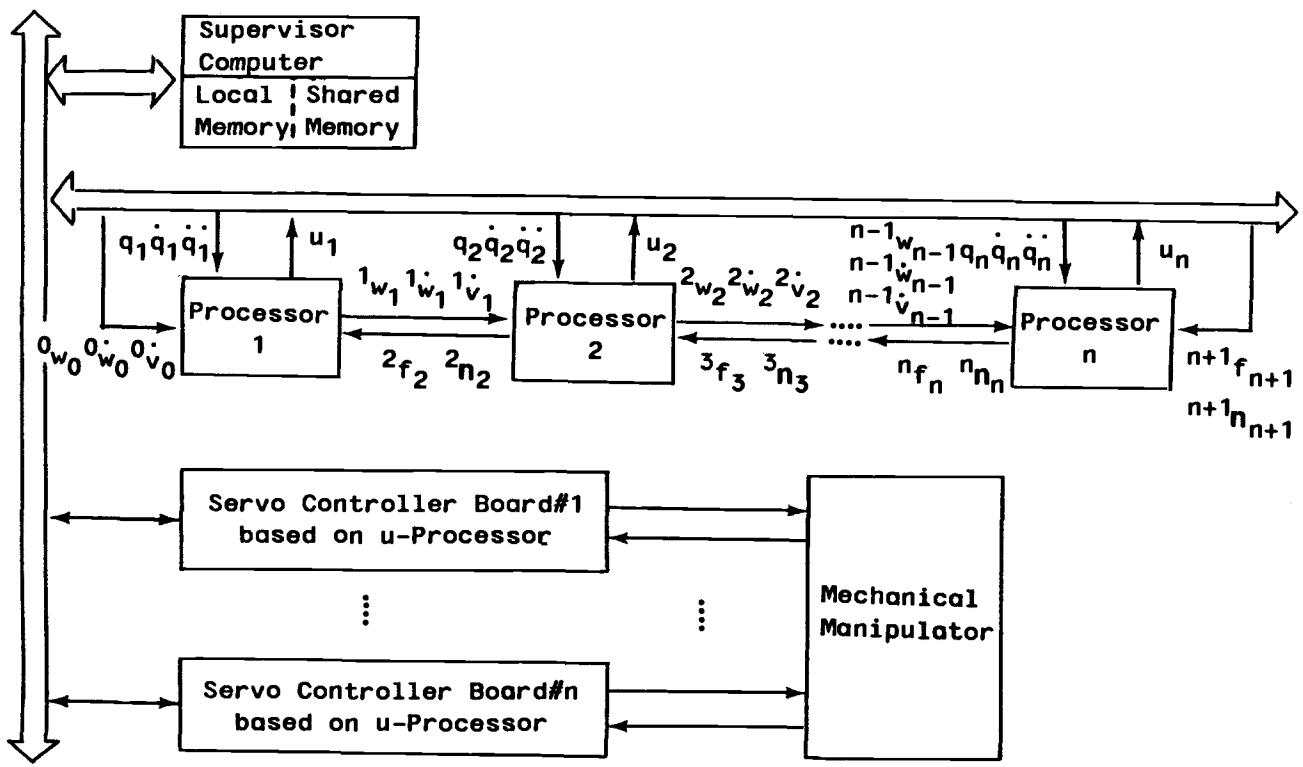

Figure 4.1 System Level Multiprocessor Architecture.

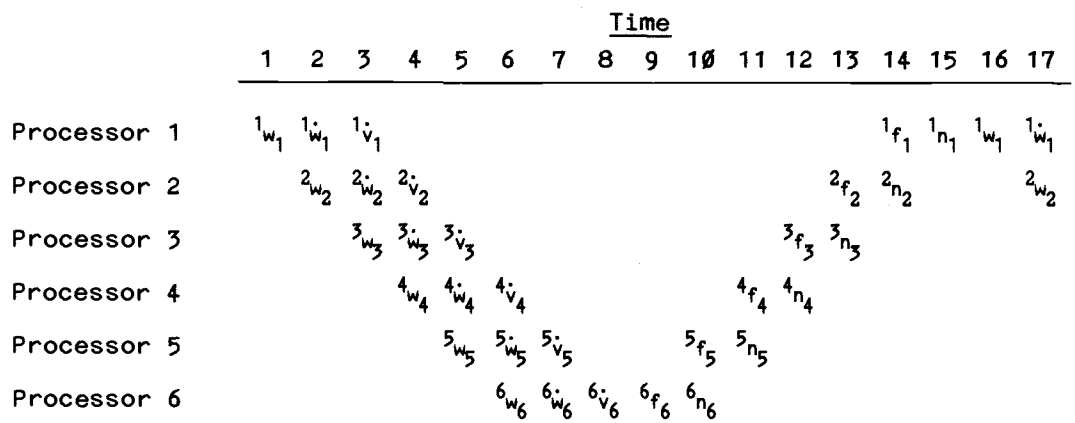

Figure 4.2 Timing Diagram of Pipelined N-E Computing Architecture (Initiation Rate = 1/15 & CPU Utilization ≈ 33.3%).

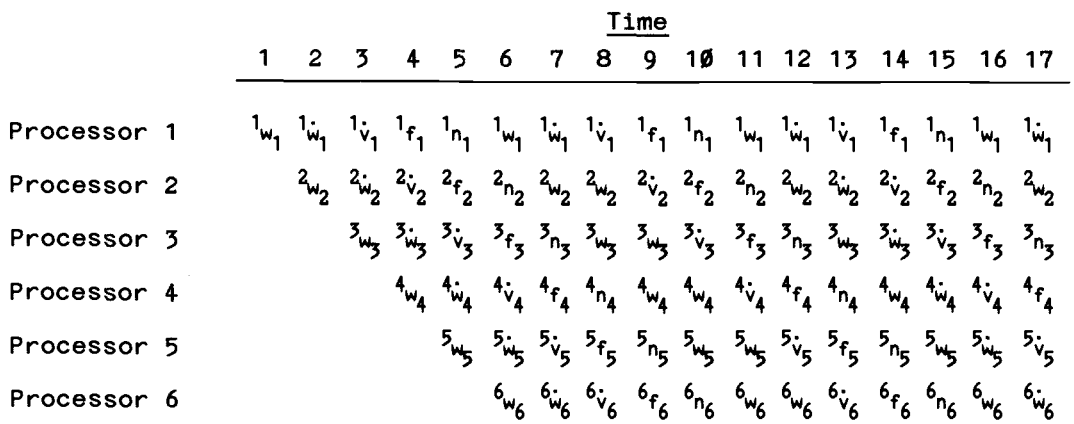

Figure 4.3 Timing Diagram of the Proposed Computing Architecture (Initiation Rate = 1/5 & CPU Utilization ≈ 100%).

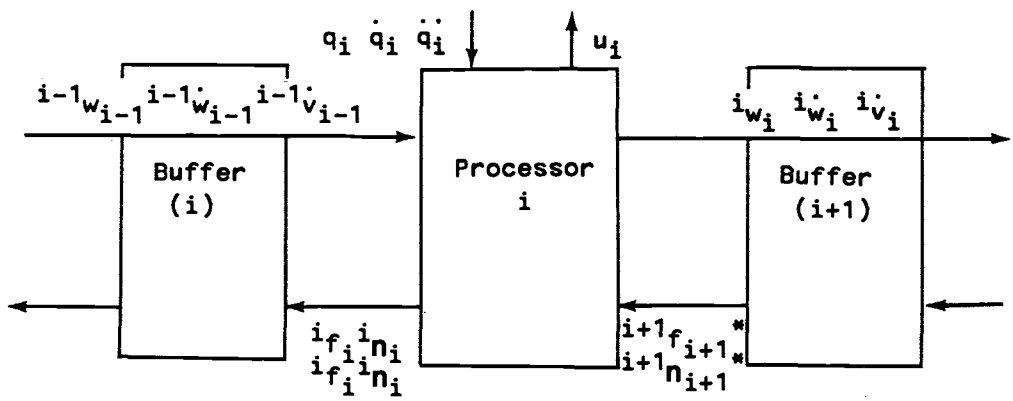

Figure 4.4 Computing Structure with Buffer for Inverse Dynamics.

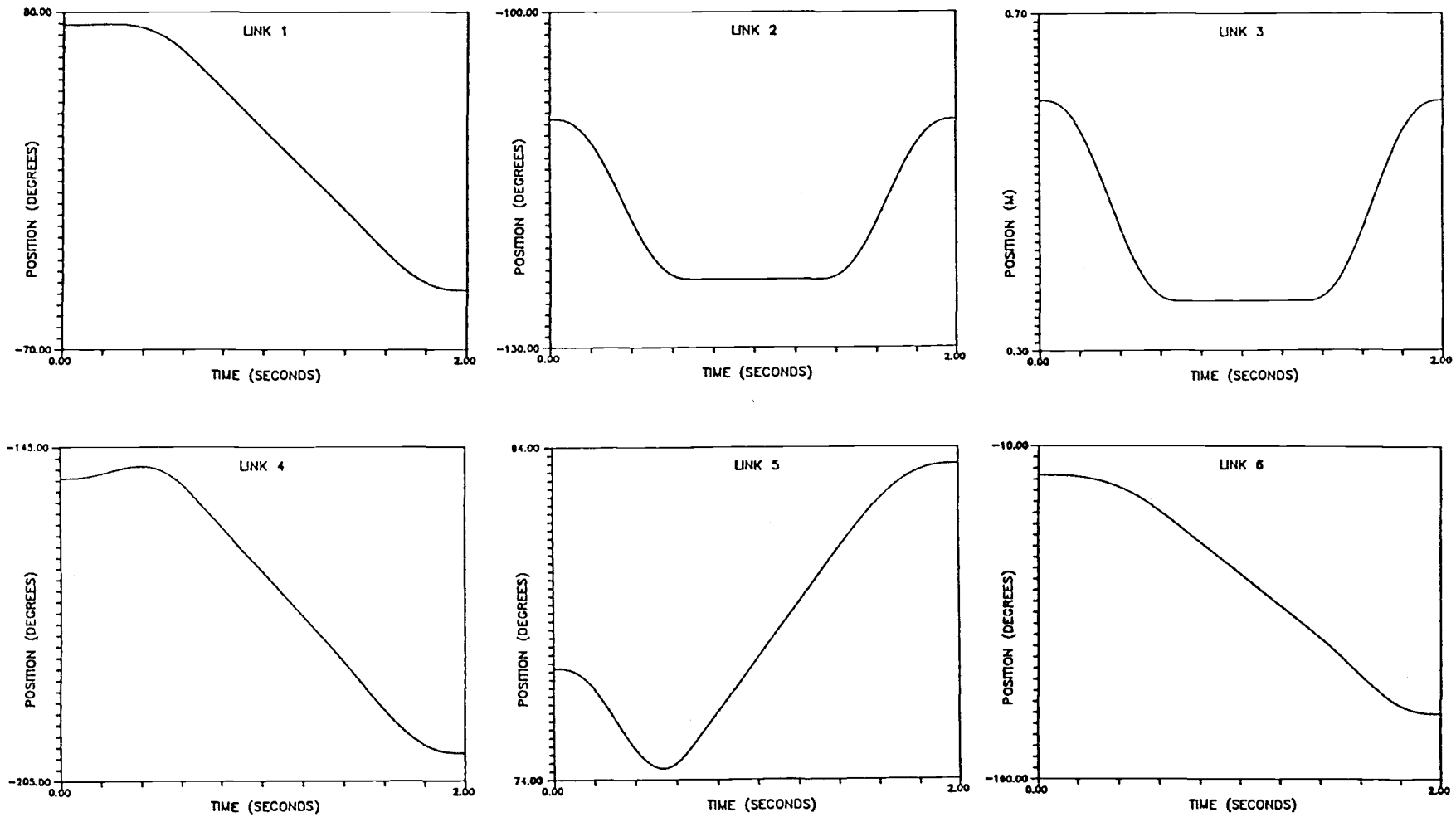

Figure 4.5 Robot Joint Trajectories for Simulation.

Chapter 5

Parallel Computations and Error Analysis of Simplified Decomposition Techniques

5.1 Introduction

In this chapter, four possible decomposition schemes for Newton-Euler dynamic formulations are considered in order to find the most efficient means of parallel computation for pipelined computing architecture. In addition, three computation methods for each of the decomposition schemes are also considered. Finally, the most efficient approaches to fast parallel computation of the dynamic formulations suitable for the pipelined computing architecture are presented. A discussion of the main principles underlying each method is included, with an analysis of errors introduced into the algorithms in each instance.

5.2 Parallel Computation Decomposition Schemes

Multiprocessor architecture design approaches for the computation of the recursive Newton-Euler dynamic formulations may differ on the means to split the dynamic formulations for pipelined computation. The

Newton-Euler dynamic formulations consist of a set of recursive forward and recursive backward equations of motion. Thus, as shown in Figure 5.1, it is possible to consider four decomposition schemes for pipelined computations.

The Newton-Euler robot dynamic equations reflect a serially recursive process. The sequential dependencies of the dynamic equations are conducive to pipelining. Thus, a pipelined computing architecture based on multiprocessors is a logical candidate for the efficient computation of real-time controls.

The proposal in Case 1 (Figure 5.2) involves the assignment of individual processors to n joints of the robot arm. Each processor i performs all of the computations associated with a particular link i, calculating its own joint variables and passing the computed partial results required for the performance of the recursive calculations associated with the Newton-Euler dynamic formulations to the adjacent processors. In Figure 5.1, the calculation of joint torque, u_i, is not included because, for example, the joint torque for a rotational joint is just the component of the moment vector about the joint axis. On the basis of the timing chart provided in Figure 5.2, the system performance results were an initiation rate of 15 (time unit)$^{-1}$ with approximately 33.3 percent CPU utilization.

Cases 2, 3, and 4 of the decomposition schemes are discussed in the following section.

5.3 Fast Parallel Computations With Computational Simplification Techniques

5.3.1 Fast Parallel Computations

In the computing architecture described in the previous section, following calculation of the forward equations, the processor i remains idle until the processor i+1 is ready to transmit the backward variable values of the link i+1. As a result, the initiation rate and CPU utilization are decreased.

For real-time control systems, the sampling rate is directly influenced by the initiation rate of the pipelined computing architecture. In order to increase the initiation rate, the processor i must calculate its backward equations without an idle period following the calculation of its forward equations. In order to implement this concept, Cases 2, 3, and 4 are considered (Figure 5.1).

For Case 2, if the processor i is able to obtain the backward variables, ${}^{i+1}\mathbf{f}_{i+1}(t)$ and ${}^{i+1}\mathbf{n}_{i+1}(t)$, of the link i+1 without waiting for those values from the processor i+1 in the current sampling period, then the processor i will be able to calculate the backward variables, ${}^{i}\mathbf{f}_{i}(t)$ and ${}^{i}\mathbf{n}_{i}(t)$, without idle periods following calculation of the forward variable,

${}^{i}\dot{\mathbf{v}}_{i}(t)$. If each processor is able to calculate its backward equations as so described, the pipelined processors may be used to concurrently perform pipelined parallel computations, thereby greatly increasing controller processing speed and CPU utilization. The algorithm described above is called the Pipelined and Fast Parallel Algorithm (PAFP).

The remaining problem is how to properly calculate the backward variables, ${}^{i+1}\mathbf{f}_{i+1}(t)$ and/or ${}^{i+1}\mathbf{n}_{i+1}(t)$, for Case 2 at the current sampling period. A computationally simple way for the provision of acceptable backward variables is to use the immediately previously calculated values of the backward variables. The underlying concept for this approach, PAFP1, is as follows.

For each link i, the results of calculating the terms in the dynamic equation at time t and time t+T, where T is a sampling interval, are slightly different. This difference, relatively short periods of only a few milliseconds, is due to changes in the position, velocity, and/or acceleration between points. The recursive backward variables, ${}^{i}\mathbf{f}_{i}(t)$ and ${}^{i}\mathbf{n}_{i}(t)$, of the link i are not only a function of the forward variables of the link i, they are also a function of the recursive backward variables, ${}^{i+1}\mathbf{f}_{i+1}(t)$ and ${}^{i+1}\mathbf{n}_{i+1}(t)$, of the link i+1. Therefore, over short sampling periods of a

few milliseconds, errors in calculating the backward variables may be relatively very small since the value of the recursive backward variables will have changed very little. Moreover, since the backward variables of the link use the exact values of its forward variables, the effect on calculating the backward variables of link i, ${}^{i}\mathbf{f}_i(t)$, and/or ${}^{i}\mathbf{n}_i(t)$, due to the difference in ${}^{i+1}\mathbf{f}_{i+1}(t)$ and/or ${}^{i+1}\mathbf{n}_{i+1}(t)$ between successive sampling intervals, can be expected to be very small.

In the PAFP1, expected errors in the torque calculations of the link i resulted from the difference in the backward variable values of the links i+1, ${}^{i+1}\mathbf{f}_{i+1}(t)$, and/or ${}^{i+1}\mathbf{n}_{i+1}(t)$ between the adjacent sampling intervals, which may include the rate of change of the values. Thus, it is a logical progression to consider the rate of change of the previous values to complement PAFP1, called PAFP2. In this manner, a greater degree of accuracy may be obtained than with the singular application of only the first method. One possible disadvantage of PAFP2 is that it requires more vector calculations for the backward variables of link i+1, ${}^{i+1}\mathbf{f}_{i+1}(t)$, and/or ${}^{i+1}\mathbf{n}_{i+1}(t)$ than PAFP1. For the above cases, the forward variables, ${}^{i}\boldsymbol{\omega}_i(t)$, ${}^{i}\dot{\boldsymbol{\omega}}_i(t)$, and ${}^{i}\dot{\mathbf{v}}_i(t)$, are exactly calculated for the backward variables, ${}^{i}\mathbf{f}_i(t)$ and ${}^{i}\mathbf{n}_i(t)$, and pipelined to the adjacent link i+1, as shown in Case 2 (Figure 5.2).

For a third computation method, the entire history of the backward variable values of the link i+1 were considered in order to calculate the approximated values of the backward variables, ${}^{i+1}\mathbf{f}_{i+1}(t)$ and ${}^{i+1}\mathbf{n}_{i+1}(t)$, of the link i+1 for the backward variables of the link i. The history of the backward variables was used to develop a stochastic model which was used for forecasting the approximated values of the backward variables. Additional computation was required to calculate parameters for the model and then to calculate the approximate values of the backward variables. However, the results of torque calculations from this method were not as good as the results of the previous two methods because the provided values of the backward variables reflected larger errors than those obtained by other methods. Moreover, this method could also require maintenance of a larger memory capacity for the momentary storage of the entire history of the backward variable values of the link i+1 in order to calculate the backward equations of the link, i. Therefore, the third computation method will not be given further consideration in this discussion.

The question which remains is the degree to which errors in the torques/forces calculations are introduced through use of the techniques described in this section. This issue is discussed in section 5.4.

Case 3 may use a method similar to that described for Case 2 since the recursive forward variables of the link i are also a function of the forward variables of the link i-1. Thus, in order to calculate the backward equations of the link i-1 without an idle period, following the calculation of its forward equations, the recursive forward equations are decomposed by application of the principle of previous computation methods. In this case, the forward variables of link i-1 use the immediately previous calculated values of the forward variables. As a result, the processor i was able to calculate the backward variables without an idle period following calculation of the forward variables. However, in this case the backward variables ${}^{i}\mathbf{f}_{i}(t)$ and ${}^{i}\mathbf{n}_{i}(t)$ were calculated by using the values of the decomposed forward variables and were pipelined for the backward variables of the adjacent link i-1. The torque computation time was close to that in Case 2, but as shown in Table 5.1, the torque errors of fast motion introduced into Case 3 using the above computation method were noticeably larger than the errors in Case 2, particularly for the last three links, where the torque errors in Case 2 were calculated from PAFP1.

Table 5.1 shows only the results of two representative trajectories, Traj 1 and Traj 2. However, the results presented for these trajectories were similar to those obtained for others. In comparing the torque

errors, one statistical measure, the root mean square torque error, was used. In order to calculate the root mean square relative torque error, the root mean square torque error was divided by the difference between the maximum and minimum torques for each plot.

When decomposition methods are applied to both the recursive forward and the recursive backward equations for Case 4, the results of a number of simulations showed the largest torque errors among all of the alternative cases for each decomposition method, as partially shown in Table 5.1, since Case 4 fails to consider the sequential dependencies of the dynamic equations. Therefore, on the basis of the above analysis and the simulation results, Case 2 was selected as the best decomposition scheme for the Newton-Euler dynamic equations suitable for the pipelined/fast parallel computing architecture. The results, shown in Case 2 of Figure 5.2, were an initiation rate of 5(time unit)$^{-1}$ with CPU utilization of approximately 100 percent. Therefore, the improvement in computational speed and CPU utilization for the Case 2 architecture is approximately three times greater than that achieved for Case 1.

5.3.2 Dynamic Formulation Using Simplified Decomposition Techniques

In the formulations described in Chapter 2, the equations of motion, as well as all of the inertial matrices and physical geometric parameters, are referenced to the base coordinate system. The result of this requirement is a complicated set of computations in which the inertia term of each link, with respect to the base coordinates, must be recomputed. Luh, Walker, and Paul [LUH 80] have noted that a more efficient method of calculation should have reference to its own link coordinate system. In order to develop a more efficient simplification technique for computation of the joint input forces/torques, each link's dynamics for referencing its own link coordinates is described as follows.

Let ${}^{i-1}R_i$ be a 3 x 3 rotation matrix which transforms any vector with reference to coordinate systems (x_i, y_i, z_i) and $(x_{i-1}, y_{i-1}, z_{i-1})$. Thus,

$$ {}^{0}R_i = {}^{0}R_1 \, {}^{1}R_2 \ldots {}^{i-1}R_i \, , \qquad (5.2\text{-}1) $$

where

$$ {}^{i-1}R_i = \begin{bmatrix} c\Theta_i & -c\alpha_i s\Theta_i & s\alpha_i s\Theta_i \\ s\Theta_i & c\alpha_i c\Theta_i & -s\alpha_i c\Theta_i \\ 0 & s\alpha_i & c\alpha_i \end{bmatrix} , $$

$$ c\Theta_i = \cos \Theta_i, \quad s\Theta_i = \sin \Theta_i, $$

$$ c\alpha_i = \cos \alpha_i, \quad s\alpha_i = \sin \alpha_i, $$

a_i = shortest distance between the z_{i-1} axis and the z_i axis,

α_i = angle between the z_{i-1} axis and the z_i axis about the x_i axis (using the right-hand rule),

d_i = distance from the origin of the $(i-1)^{th}$ coordinate system to the intersection of the z_{i-1} axis, with the x_i axis along the z_{i-1} axis, and

Θ_i = joint angle from the x_{i-1} axis to the x_i axis about the z_{i-1} axis (using the right-hand rule).

Since each coordinate system is orthonomal,

$$({}^{i-1}R_i)^{-1} = ({}^{i-1}R_i)^t = {}^iR_{i-1} , \qquad (5.2\text{-}2)$$

where $(\ )^t$ = the transpose of $(\ )$.

For the term $\omega_i(t)$, if link i is rotational, pre-multiplying both sides of the equation by ${}^iR_0$ gives

$$\begin{aligned} {}^iR_0\ \omega_i(t) &= {}^iR_0[\omega_{i-1}(t) + z_{i-1}\dot{q}_i(t)] \\ &= {}^iR_{i-1}[{}^{i-1}R_0\omega_{i-1}(t) + {}^{i-1}R_0 z_{i-1}\dot{q}_i(t)] \\ &= {}^iR_{i-1}[{}^{i-1}R_0\omega_{i-1}(t) + z_0\dot{q}_i(t)] . \qquad (5.2\text{-}3) \end{aligned}$$

Using simple notation, the above equation can be re-written as follows:

$${}^i\omega_i(t) = {}^iR_{i-1}[{}^{i-1}\omega_{i-1}(t) + z_0\dot{q}_i(t)] . \qquad (5.2\text{-}4)$$

Notice that

${}^i\omega_i(t)$ = angular velocity of link i with respect to its own coordinate system,

${}^{i-1}\omega_{i-1}(t)$ = angular velocity of link i-1 with respect to its own coordinate system, and

$$z_0 = [0,0,1]^t .$$

Similarly, if link i is rotational, the angular and linear accelerations become:

$$ {}^{i}\dot{\omega}_i = {}^{i}R_{i-1}[{}^{i-1}\dot{\omega}_{i-1} + z_0 \ddot{q}_i + {}^{i-1}\omega_{i-1} \times z_0\dot{q}_i] \qquad (5.2\text{-}5)$$

and

$$ {}^{i}\dot{v}_i = {}^{i}\dot{\omega}_i \times ({}^{i}R_0 p_i{}^*) + {}^{i}\omega_i \times [{}^{i}\omega_i \times ({}^{i}R_0 p_i{}^*) + {}^{i}R_{i-1}({}^{i-1}\dot{v}_{i-1}) , \qquad (5.2\text{-}6)$$

where

${}^{i}\dot{\omega}_i$ = angular acceleration of link i with respect to its own coordinate system.

${}^{i}\dot{v}_i$ = linear acceleration of link i with respect to its own coordinate system.

${}^{i}R_0 p_i{}^*$ = the location of coordinate system (x_i,y_i,z_i) from the origin of $(x_{i-1},y_{i-1}, z_{i-1})$ with respect to coordinate system (x_i,y_i,z_i).

$$\begin{bmatrix} a_i \\ d_i\sin\alpha_i \\ d_i\cos\alpha_i \end{bmatrix}$$

In order to derive equations of motion for the robot arm, D'Alembert's principle is applied to each link of the robot arm. When the viscous damping terms are omitted, force and moment equations are:

$$ {}^{i}R_0 F_i = m_i({}^{i}a_{ci}) = m_i[\, {}^{i}\dot{v}_i + {}^{i}\dot{\omega}_i \times ({}^{i}R_0 s_{ci}) + {}^{i}\omega_i \times [{}^{i}\omega_i \times ({}^{i}R_0 s_{ci})]] \qquad (5.2\text{-}7)$$

and

$$
\begin{aligned}
{}^{i}R_0 N_i &= {}^{i}R_0[I_{ci}\dot{\omega}_i + \omega_i \times (I_{ci}\,\omega_i)] \\
&= ({}^{i}R_0 I_{ci}\,{}^{0}R_i)({}^{i}R_0\dot{\omega}_i) + ({}^{i}R_0\omega_i) \\
&\quad \times [({}^{i}R_0 I_{ci}\,{}^{0}R_i)\,{}^{i}R_0\omega_i)] \\
&= ({}^{i}R_0 I_{ci}\,{}^{0}R_i)({}^{i}\dot{\omega}_i) + ({}^{i}\omega_i) \\
&\quad \times [({}^{i}R_0 I_{ci}\,{}^{0}R_i)\,{}^{i}\omega_i)] \,, \qquad (5.2\text{-}8)
\end{aligned}
$$

where ${}^{i}a_{ci}$ = linear acceleration of the center of mass of link i with respect to its own coordinate system,

${}^{i}R_0 I_{ci}\,{}^{0}R_i$ = inertia matrix of link i about its center of mass referred to its own coordinate system, (x_i, y_i, z_i),

${}^{i}R_0 F_i \equiv {}^{i}F_i$ = total external vector force exerted on link i at the center of mass with respect to its own coordinate system, and

${}^{i}R_0 N_i \equiv {}^{i}N_i$ = total external vector moment exerted on link i at the center of mass with respect to its own coordinate system.

The total external force, ${}^{i}F_i$, and moment, ${}^{i}N_i$, are those exerted on link i by gravity and its neighboring links, i-1 and i+1. With reference to Figure 5.4,

$$
{}^{i}R_0 F_i = {}^{i}R_0[f_i - f_{i+1}] \qquad (5.2\text{-}9)
$$

and

$$
\begin{aligned}
{}^{i}R_0 N_i &= {}^{i}R_0 n_i - {}^{i}R_0 n_{i+1} + ({}^{i}R_0 p_i - {}^{i}R_0 r_i) \times {}^{i}R_0 f_i \\
&\quad - ({}^{i}R_0 p_i - {}^{i}R_0 r_i) \times {}^{i}R_0 f_{i+1} \\
&= {}^{i}R_0 n_i - {}^{i}R_0 n_{i+1} + ({}^{i}R_0 p_i - {}^{i}R_0 r_i) \times {}^{i}R_0 F_i \\
&\quad - {}^{i}R_0\, p_i^{*} \times {}^{i}R_0 f_{i+1} \qquad (5.2\text{-}10)
\end{aligned}
$$

since $({}^{i}R_0 r_i - {}^{i}R_0 p_{i-1}) = ({}^{i}R_0 p_i^* - {}^{i}R_0 s_{ci})$. Then, from Eqs.(5.2-9) and (5.2-10) the following recursive equations can be obtained:

$$
\begin{aligned}
{}^{i}R_0 f_i &= {}^{i}R_0 f_{i+1} + {}^{i}R_0 F_i \\
&= {}^{i}R_{i+1}[{}^{i+1}R_0 f_{i+1}] + m_i[{}^{i}\dot{v}_i + {}^{i}\omega_i \times ({}^{i}R_0 s_{ci}) \\
&\quad + {}^{i}\omega_i \times [{}^{i}\omega_i \times ({}^{i}R_0 s_{ci})]]
\end{aligned}
\tag{5.2-11}
$$

and

$$
\begin{aligned}
{}^{i}R_0 n_i &= {}^{i}R_0 n_{i+1} + {}^{i}R_0\, p_i^* \times {}^{i}R_0 f_{i+1} + ({}^{i}R_0 p_i^* \\
&\quad + {}^{i}R_0 s_{ci}) \times {}^{i}R_0 F_i + {}^{i}R_0 N_i \\
&= {}^{i}R_{i+1}[{}^{i+1}R_0 n_{i+1} + {}^{i}R_0\, p_i^* \times {}^{i+1}R_0 f_{i+1}] \\
&\quad + ({}^{i}R_0 p_i^* + {}^{i}R_0 s_{ci}) \times m_i({}^{i}R_0 a_{ci}) \\
&\quad + ({}^{i}R_0 I_{ci}\, {}^{0}R_i)({}^{i}R_0 \dot{\omega}_i) \\
&\quad + ({}^{i}R_0 \omega_i) \times [({}^{i}R_0 I_{ci}\, {}^{0}R_i)\, {}^{i}R_0 \omega_i)] \ .
\end{aligned}
\tag{5.2-12}
$$

The above equations can again be rewritten with a simple notation as follows:

$$
\begin{aligned}
{}^{i}f_i &= {}^{i}R_{i+1}[{}^{i+1}f_{i+1}] + m_i[{}^{i}\dot{v}_i + {}^{i}\omega_i \times ({}^{i}R_0 s_{ci}) \\
&\quad + {}^{i}\omega_i \times [{}^{i}\omega_i \times ({}^{i}R_0 s_{ci})]]
\end{aligned}
\tag{5.2-13}
$$

and

$$
\begin{aligned}
{}^{i}n_i &= {}^{i}R_{i+1}[{}^{i+1}n_{i+1} + {}^{i+1}R_0\, p_i^* \times {}^{i+1}f_{i+1}] \\
&\quad + ({}^{i}R_0 p_i^* + {}^{i}R_0 s_{ci}) \times m_i[{}^{i}\dot{v}_i + {}^{i}\omega_i \times ({}^{i}R_0 s_{ci}) \\
&\quad + {}^{i}\omega_i \times [{}^{i}\omega_i \times ({}^{i}R_0 s_{ci})]] + ({}^{i}R_0 I_{ci}\, {}^{0}R_i)({}^{i}\omega_i) \\
&\quad + ({}^{i}\omega_i) \times [({}^{i}R_0 I_{ci}\, {}^{0}R_i)\, {}^{i}\omega_i)] \ ,
\end{aligned}
\tag{5.2-14}
$$

where

${}^{i}f_i(t)$ = force exerted on link i by link i-1 with respect to its own coordinate system at coordinate system $(x_{i-1}, y_{i-1}, z_{i-1})$ and

${}^{i}\mathbf{n}_{i}(t)$ = moment exerted on link i by link i-1 with respect to its own coordinate system at coordinate system $(x_{i-1}, y_{i-1}, z_{i-1})$.

The above inertia matrix, ${}^{i}R_{0}I_{ci}{}^{0}R_{i}$, is then no longer dependent on the orientation of link i, even should the robot arm move. Much simpler computations are the result.

In direct application of Eqs.(5.2-13) and (5.2-14), after calculation of the forward equations, each processor i must wait for the receipt of the backward variable values from its adjacent processor i+1. Therefore, in order to increase the computation speed and CPU utilization, a simplified decomposition technique based upon provision of acceptable values of the backward variables is proposed in order for each processor i to calculate its backward equations without idle periods following calculation of the forward equations. Two computationally simple decomposition methods are proposed: (1) Using the immediately previous calculated values of the backward variables and (2) considering the first method and the rate of change of previous values as follows:

$$ {}^{i}\mathbf{f}_{i}{}^{*} = {}^{i}R_{i+1}[{}^{i+1}\mathbf{f}_{i+1}{}^{*}] + m_{i}[{}^{i}\dot{\mathbf{v}}_{i} + {}^{i}\boldsymbol{\omega}_{i} \times ({}^{i}R_{0}\mathbf{s}_{ci}) + {}^{i}\boldsymbol{\omega}_{i} \times [{}^{i}\boldsymbol{\omega}_{i} \times ({}^{i}R_{0}\mathbf{s}_{ci})]] \qquad (5.2\text{-}15) $$

and

$$
\begin{aligned}
{}^{i}\mathbf{n}_i^{*} = {}&{}^{i}R_{i+1}[{}^{i+1}\mathbf{n}_{i+1}^{*} + {}^{i}R_0\ \mathbf{p}_i^{*} \times {}^{i+1}\mathbf{f}_{i+1}^{*}] \\
&+ ({}^{i}R_0\mathbf{p}_i^{*} + {}^{i}R_0\mathbf{s}_{ci}) \times m_i[{}^{i}\dot{\mathbf{v}}_i + {}^{i}\boldsymbol{\omega}_i \times ({}^{i}R_0\mathbf{s}_{ci}) \\
&+ {}^{i}\boldsymbol{\omega}_i \times [{}^{i}\boldsymbol{\omega}_i \times ({}^{i}R_0\mathbf{s}_{ci})]] + ({}^{i}R_0\mathbf{I}_{ci}{}^{0}R_i)({}^{i}\dot{\boldsymbol{\omega}}_i) \\
&+ ({}^{i}\boldsymbol{\omega}_i) \times [({}^{i}R_0\mathbf{I}_{ci}{}^{0}R_i){}^{i}\boldsymbol{\omega}_i)] \ , \qquad (5.2\text{-}16)
\end{aligned}
$$

where

$${}^{i+1}\mathbf{f}_{i+1}^{*}(t) = {}^{i+1}\mathbf{f}_{i+1}^{*}(t-1) + {}^{i+1}\mathbf{df}_{ri+1}(t-1),$$

$${}^{i+1}\mathbf{n}_{i+1}^{*}(t) = {}^{i+1}\mathbf{n}_{i+1}^{*}(t-1) + {}^{i+1}\mathbf{dn}_{ri+1}(t-1),$$

${}^{i+1}\mathbf{df}_{ri+1}(t-1)$ = the rate of change of the previous values of ${}^{i+1}\mathbf{f}_{i+1}$ between the previous sampling intervals,

${}^{i+1}\mathbf{dn}_{ri+1}(t-1)$ = the rate of change of the previous values of ${}^{i+1}\mathbf{n}_{i+1}$ between the previous sampling intervals, and

$(t-i)$ = i^{th} previous sampling interval with respect to the current sampling interval, and the rate of changes of values, ${}^{i+1}\mathbf{df}_{ri+1}(t-1)$ and ${}^{i+1}\mathbf{dn}_{ri+1}(t-1)$, are equal to zero for the first method.

See section 5.4 for ${}^{i+1}\mathbf{df}_{ri+1}(t-1)$ and ${}^{i+1}\mathbf{dn}_{ri+1}(t-1)$.

From the kinematic relationship between the neighboring links and the establishment of coordinate systems $\mathbf{q}_i$[rad] in $(x_{i-1}, y_{i-1}, z_{i-1})$ coordinates actually rotates about the z_{i-1} axis if joint i is rotational. Therefore, the input torque, u_i, at the joint is the sum of the projection of $\mathbf{n}_i$ onto z_{i-1} and the viscous damping moment at those coordinates. If, however,

joint i (and hence link i) is translational, then its actual displacement is q_i relative to coordinates $(x_{i-1}, y_{i-1}, z_{i-1})$ along the z_{i-1} axis. The input force, u_i, at that joint is then the sum of the projection of f_i^* onto z_{i-1} and the viscous damping force at those coordinates. Thus, for the backward equations with reference to their own coordinate systems,

$$u_i = \begin{cases} ({}^i n_i^*)^t({}^i R_{i-1} z_0) + b_i \dot{q}_i & \text{if link i is rotational} \\ ({}^i f_i^*)^t({}^i R_{i-1} z_0) + b_i \dot{q}_i & \text{if link i is translational} \end{cases} \quad (5.2\text{-}17)$$

The result is that after calculation of the for ward equations, each processor i will be able to calculate its backward equations without idle periods. In the following section, errors derived from the use of this decomposition technique are analyzed.

5.4 Error Analysis of Simplified Decomposition Techniques

The decomposed scheme described in the previous section uses exact values of all the variables involved in torque calculations, with the exception of ${}^i f_i(t)$ and/or ${}^i n_i(t)$. In order to analyze errors derived from use of the decomposed ${}^i f_i(t)$ and/or ${}^i n_i(t)$, the scalar case of force, $f_i(t)$, is first analyzed. This concept is then expanded to the general case of three dimensional motion.

5.4.1 Scalar Case

For the case of planar motion in a reference coordinate system, the general recursive form of Equation (5.2-13) can be expressed in a scalar form.

$$f_i(t) = f_{i+1}(t) + F_i(t) \qquad i=n,n-1,\ldots 1. \qquad (5.2\text{-}18)$$

where $F_i = m_i a_{ci}$. If n is equal to 6 and no external force is exerted on link n, then this equation can be rewritten as follows:

$$f_6(t) = F_6(t)$$

$$f_5(t) = F_6(t)+F_5(t)$$

$$f_4(t) = F_6(t)+F_5(t)+F_4(t) \ ,$$

in general form,

$$f_i(t) = \sum_{j=i}^{n} F_j(t) \qquad (5.2\text{-}19)$$

where n = the number of degrees of freedom of the robot arm and

i = the index of link i.

If the value of $f_{i+1}(t-1)$ from the immediately previous sample interval is used in place of the current value, $f_{i+1}(t)$, then the above equations become:

$$f_6^*(t) = F_6(t)$$

$$f_5^*(t) = F_6(t-1)+F_5(t)$$

$$f_4^*(t) = F_6(t-2)+F_5(t-1)+F_4(t)$$

or

$$f_i^*(t) = \sum_{j=i}^{n} F_j(t-m) \ , \qquad (5.2\text{-}20)$$

where m = j-i. Let $f_{ei}(t) = f_i(t)-f_i^*(t)$. From Equations (5.2-19) and (5.2-20), the error, $f_{ei}(t)$, can be expressed as follows since the recursive backward equations use the exact values of their forward variables:

$$f_{ei}(t)= \sum_{j=i}^{n} [F_j(t)-F_j(t-m)] \ , \qquad (5.2\text{-}21)$$

or, let

$$dF_i(t-m) = F_i(t-m)-F_i(t-(m-1)),$$

$$f_{e6}(t) = 0,$$

$$f_{e5}(t) = F_6(t) - F_6(t-1) = dF_6(t),$$

$$\begin{aligned} f_{e4}(t) &= F_6(t) - F_6(t-2) + F_5(t) - F_5(t-1), \\ &= F_6(t)-F_6(t-1) + F_6(t-1) - F_6(t-2) \\ &\quad + F_5(t) - F_5(t-1) \\ &= dF_6(t-1) + dF_6(t) + dF_5(t), \end{aligned}$$

$$\begin{aligned} f_{e3}(t) &= F_6(t) - F_6(t-3) + F_5(t) - F_5(t-2) \\ &\quad + F_4(t) - F_4(t-1), \\ &= F_6(t) - F_6(t-1) + F_6(t-1) - F_6(t-2) \\ &\quad + F_6(t-2) - F_6(t-3) + F_5(t) - F_5(t-1) \\ &\quad + F_5(t-1)-F_5(t-2) \\ &\quad + F_4(t) - F_4(t-1) \\ &= dF_6(t-2) + dF_6(t-1) + dF_5(t-1) + dF_6(t) \\ &\quad + dF_5(t) + dF_4(t) \ , \end{aligned}$$

in general form,

$$f_{ei}(t)= \sum_{j=i+1}^{n} \sum_{k=0}^{j-i-1} dF_j(t-k) \ . \qquad (5.2\text{-}22)$$

If $j > n$, then $f_{ei}(t) = 0$.

Equation (5.2-22) can again be expressed in the following recursive form:

$$f_{e6}(t) = 0,$$

$$f_{e5}(t) = dF_6(t),$$

$$f_{e4}(t) = dF_6(t-1) + dF_6(t) + dF_5(t)$$
$$= f_{e5}(t-1) + dF_6(t) + dF_5(t),$$

$$f_{e3}(t) = dF_6(t-2) + dF_6(t-1) + dF_5(t-1) + dF_6(t) + dF_5(t) + dF_4(t),$$
$$= f_{e4}(t-1) + dF_6(t) + dF_5(t) + dF_4(t),$$

and in the recursive form,

$$f_{ei}(t) = f_{ei+1}(t-1) + \sum_{j=i+1}^{n} dF_j(t) \ . \qquad (5.2\text{-}23)$$

On the basis of the above equations, it can be said that error introduced into the above method results from the difference of $f_{i+1}(t)$, $dF_{i+1}(t)$ between adjacent sampling intervals at a given time, which is then passed to the adjacent link. Thus, in order to reduce the errors from the adjacent link, it is logical to consider the rate of change of $f_{i+1}(t)$ for the link i in addition to the first method.

As a second method, the value of $f_{i+1}(t-1)$ is taken from the immediately previous sample interval and the values of the rate of change are used in place of the current value, $f_{i+1}(t)$, as follows. From Equation (5.2-23), let

$$df_i(t) = \sum_{j=i+1}^{n} dF_j(t) \ , \text{ then,}$$

$$df_6(t) = 0,$$

$$df_5(t) = dF_6(t),$$

$$df_4(t) = dF_6(t) + dF_5(t),$$

$$= df_5(t)+dF_5(t)$$

in a recursive form,

$$df_i(t) = df_{i+1}(t) + dF_{i+1}(t) \ . \qquad (5.2\text{-}24)$$

When this step is completed by adding the difference, $df_i(t)$, to the value of $f_{i+1}(t-1)$, the force exerted on the link i, $f_i(t)$, can be calculated with precision. However, the processor i must wait for the completion of the processor i+1 calculation of $f_{i+1}(t)$ for determination of the difference, $df_{i+1}(t)$ and $dF_{i+1}(t)$, before it can initiate its own calculation, $f_i(t)$. In order to reduce the waiting time for the processor i+1, the rate of change of the previous values can be used in place of the difference for concurrent computations as follows:

$$df_i(t) = df_{i+1}(t-1) + dF_{i+1}(t-1) \ , \qquad (5.2\text{-}25)$$

where $df_{i+1}(t-1)$ and $dF_{i+1}(t-1)$, which were already calculated at the previous sampling interval and stored in the buffer i+1, are passed from the buffer i+1 to the processor i to calculate $df_i(t)$ at the current sampling interval. Then, the processor i can calculate $f_i^*(t)$ by merely adding the difference to $f_i^*(t)$. As a result, the calculation error will be reduced. However, unlike the case in which the exact terms, Equation (5.2-24), have been added, the error is not

completely eliminated. With an argument identical to that used for the first method, the error can be reduced to the following equation:

$$f_{ei}(t) = f_{ei+1}(t-1) + \sum_{j=i+1}^{n} dF_{ej}(t) \ , \qquad (5.2\text{-}26)$$

where $dF_{ej}(t) = dF_j(t) - dF_j(t-1)$.

The error now becomes a function of $dF_{ej}(t)$, which is smaller than before. From Equation (5.2-26), the error still seems to have been passed from the adjacent link. However, in a real case each joint of the robot arm is accelerated from rest to a selected velocity, maintaining this velocity through a given segment and then decelerated to a rest condition. Even when a joint is accelerated to specific velocity, joint acceleration is increased and then decreased, or vice versa, and in turn the force exerted on the link i is increased and then decreased over the trajectory segment. Therefore, the average magnitude of $dF_i(t)$ through a trajectory is close to zero.

5.4.2 Vector Case

For the general motion of robot arm, Equation (5.2-13) becomes:

$$^{i}\mathbf{f}_i(t) = {}^{i}R_{i+1}[{}^{i+1}\mathbf{f}_{i+1}(t)] + {}^{i}\mathbf{F}_i(t) \ , \qquad (5.2\text{-}28)$$

where

$$^{i}\mathbf{F}_i(t) = m_i\,{}^{i}\mathbf{a}_{ci} \text{ and}$$

$$^{i}R_{i+1} = \begin{bmatrix} c\Theta_{i+1} & -c\alpha_i s\Theta_{i+1} & s\alpha_i s\Theta_{i+1} \\ s\Theta_{i+1} & c\alpha_i c\Theta_{i+1} & -s\alpha_i c\Theta_{i+1} \\ 0 & s\alpha_i & c\alpha_i \end{bmatrix}$$

from Equation (5.2-1). Then, Equation (5.2-19) becomes:

$$^{6}\mathbf{f}_{6}(t) = {}^{6}\mathbf{F}_{6}(t),$$

$$^{5}\mathbf{f}_{5}(t) = {}^{5}\mathbf{R}_{6}{}^{6}\mathbf{f}_{6}(t) + {}^{5}\mathbf{F}_{5}(t) = {}^{5}\mathbf{R}_{6}{}^{6}\mathbf{F}_{6}(t) + {}^{5}\mathbf{F}_{5}(t),$$

$$^{4}\mathbf{f}_{4}(t) = {}^{4}\mathbf{R}_{5}{}^{5}\mathbf{f}_{5}(t) + {}^{4}\mathbf{F}_{4}(t)$$

$$= {}^{4}\mathbf{R}_{6}{}^{6}\mathbf{F}_{6}(t) + {}^{4}\mathbf{R}_{5}{}^{5}\mathbf{F}_{5}(t) + {}^{4}\mathbf{F}_{4}(t),$$

in general form,

$$^{i}\mathbf{f}_{i}(t) = \sum_{j=i}^{n} {}^{i}\mathbf{R}_{j}\ {}^{j}\mathbf{F}_{j}(t)\ . \qquad (5.2\text{-}29)$$

With a similar argument for the scalar case, the value of $^{i+1}\mathbf{f}_{i+1}(t-1)$ from the immediately previous sample intervals may be used in place of the current values if it can be assumed that the matrix, $^{i}\mathbf{R}_{i+1}$, is constant around a position in space for a short sampling interval of a few milliseconds. Then, Equations (5.2-20) and (5.2-21) can be expressed, respectively, as follows:

$$^{i}\mathbf{f}_{i}{}^{*}(t) = \sum_{j=i}^{n} {}^{i}\mathbf{R}_{j}\ {}^{j}\mathbf{F}_{j}(t-m) \qquad (5.2\text{-}29)$$

and

$$^{i}\mathbf{f}_{ei}(t) = {}^{i}\mathbf{f}_{i}{}^{*}(t) - {}^{i}\mathbf{f}_{i}(t)$$

$$= \sum_{j=i}^{n} {}^{i}\mathbf{R}_{j}\ [{}^{j}\mathbf{F}_{j}(t) - {}^{j}\mathbf{F}_{j}(t-m)]\ , \qquad (5.2\text{-}30)$$

where m = j-i. Again, the error, $^{i}\mathbf{f}_{ei}(t)$, between $^{i}\mathbf{f}_{i}{}^{*}(t)$ and $^{i}\mathbf{f}_{i}(t)$, can be expressed in a recursive form. Then, for greater accuracy the rate of change of previous values is also used in addition to

${}^{i+1}f_{i+1}{}^{*}(t)$. The error equations for general motion are

$$ {}^{i}f_{ei}(t) = {}^{i}R_{i+1}\,{}^{i+1}f_{ei+1}(t-1) + \sum_{j=i+1}^{n} {}^{i}R_{j}\,d^{j}F_{j}(t) , \qquad (5.2\text{-}31) $$

where $d^{j}F_{j}(t) = [{}^{j}F_{j}(t) - {}^{j}F_{j}(t-1)]$.

The absolute values of the matrix, ${}^{i}R_{i+1}$, are equal or are less than one. Therefore, the effect of the first term of Equation (5.2-31) on ${}^{i}f_{ei}(t)$ may be less than the scalar case. Again, by letting

$$ d^{i}f_{i}(t) = \sum_{j=i+1}^{n} {}^{i}R_{j}\,d^{j}F_{j}(t) , $$

Equation (5.2-25) for fast computation becomes for general motion:

$$ \begin{aligned} d^{i}f_{i}(t) &= {}^{i}R_{i+1}[d^{i+1}f_{i+1}(t-1) + d^{i+1}F_{i+1}(t-1)] . \\ &= {}^{i}R_{i+1}\,{}^{i+1}f_{ri+1}(t-1) , \end{aligned} \qquad (5.2\text{-}32) $$

where

${}^{i+1}f_{ri+1}(t-1) = d^{i+1}f_{i+1}(t-1) + d^{i+1}F_{i+1}(t-1)$.

For analyzing errors derived from use of the decomposed, ${}^{i}n_{i}(t)$, Equation (5.2-14) can be rewritten as follows:

$$ \begin{aligned} {}^{i}n_{i}(t) &= {}^{i}R_{i+1}[{}^{i+1}n_{i+1}(t) + {}^{i+1}p_{i}{}^{*} \times {}^{i+1}f_{i+1}(t)] \\ &\quad + {}^{i}K_{i}(t), \\ &= {}^{i}R_{i+1}\,{}^{i+1}n_{i+1}(t) + {}^{i}K_{i}(t) \\ &\quad + {}^{i}R_{i+1}[{}^{i+1}p_{i}{}^{*} \times {}^{i+1}f_{i+1}(t)], \\ &= {}^{i}n_{1i}(t) + {}^{i}n_{2i}(t) , \end{aligned} \qquad (5.2\text{-}33) $$

where

$$
\begin{aligned}
{}^{i}\mathbf{K}_i(t) = & ({}^{i}R_0\mathbf{p}_i^{*} + {}^{i}R_0\mathbf{s}_{ci}) \times m_i[{}^{i}\dot{\mathbf{v}}_i(t) + {}^{i}\dot{\boldsymbol{\omega}}_i(t) \\
& \times ({}^{i}R_0\mathbf{s}_{ci}) + {}^{i}\boldsymbol{\omega}_i(t) \times [{}^{i}\boldsymbol{\omega}_i(t) \times ({}^{i}R_0\mathbf{s}_{ci})]] \\
& + ({}^{i}R_0\mathbf{I}_{ci}{}^{0}R_i){}^{i}\dot{\boldsymbol{\omega}}_i(t) + {}^{i}\boldsymbol{\omega}_i(t) \\
& \times [({}^{i}R_0\mathbf{I}_{ci}{}^{0}R_i){}^{i}\boldsymbol{\omega}_i(t)], \\
{}^{i}\mathbf{n}_{1i}(t) = & {}^{i}R_{i+1}{}^{i+1}\mathbf{n}_{i+1}(t) + {}^{i}\mathbf{K}_i(t), \text{ and} \\
{}^{i}\mathbf{n}_{2i}(t) = & {}^{i}R_{i+1}[{}^{i+1}\mathbf{p}i^{*} \times {}^{i+1}\mathbf{f}_{i+1}(t)].
\end{aligned}
$$

From Equation (5.2-33), errors from the decomposed ${}^{i}\mathbf{n}_i(t)$ are the sum of errors from ${}^{i}\mathbf{n}_{1i}(t)$ and ${}^{i}\mathbf{n}_{2i}(t)$. Therefore an arguement similiar to that presented above can be applied to the moment exerted on link i by link i+1 with respect to its own coordinate system. The rate of change used for fast calculation of the moment can be expressed as follows:

$$
\begin{aligned}
d{}^{i}\mathbf{n}_i(t) &= {}^{i}R_{i+1}[d{}^{i+1}\mathbf{n}_{i+1}(t-1) + d{}^{i+1}\mathbf{K}_{i+1}(t-1)] \\
&= {}^{i}R_{i+1}{}^{i+1}\mathbf{n}_{ri+1}(t-1) , \qquad (5.2\text{-}34)
\end{aligned}
$$

where

$d{}^{i+1}\mathbf{K}_{i+1}(t) = [{}^{i+1}\mathbf{K}_{i+1}(t) - {}^{i+1}\mathbf{K}_{i+1}(t-1)]$ and

${}^{i+1}\mathbf{n}_{ri+1}(t-1) = d{}^{i+1}\mathbf{n}_{i+1}(t-1) + d{}^{i+1}\mathbf{K}_{i+1}(t-1)$.

In general terms, error analysis for the simplification technique presented in this section is extremely complicated since some terms in the nonlinear dynamic equations are position and velocity dependent. Thus, motion simulation should be performed in order to accurately evaluate the torque/force errors introduced by the proposed algorithm.

5.5 Summary

Four decomposition schemes for Newton-Euler formulations and three computation methods for each of the decomposition schemes were considered. The second decomposition scheme was selected for use with the Newton-Euler formulations suitable for pipelined/fast parallel computing architecture. Among the three computation methods considered, the first two methods were selected as the most appropriate for the proposed architecture The results were an initiation rate of $5(\text{time unit})^{-1}$ with approximately 100 percent CPU utilization.

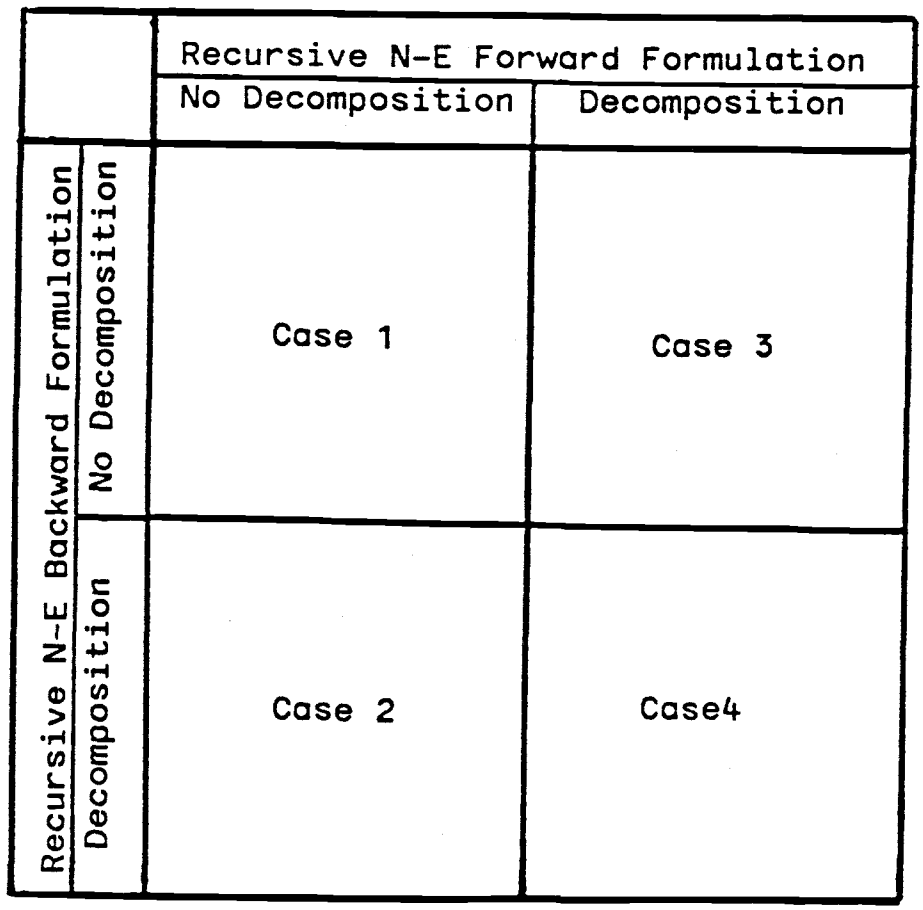

Figure 5.1 Four Possible Decompositions of the Newton-Euler Dynamic Formulation.

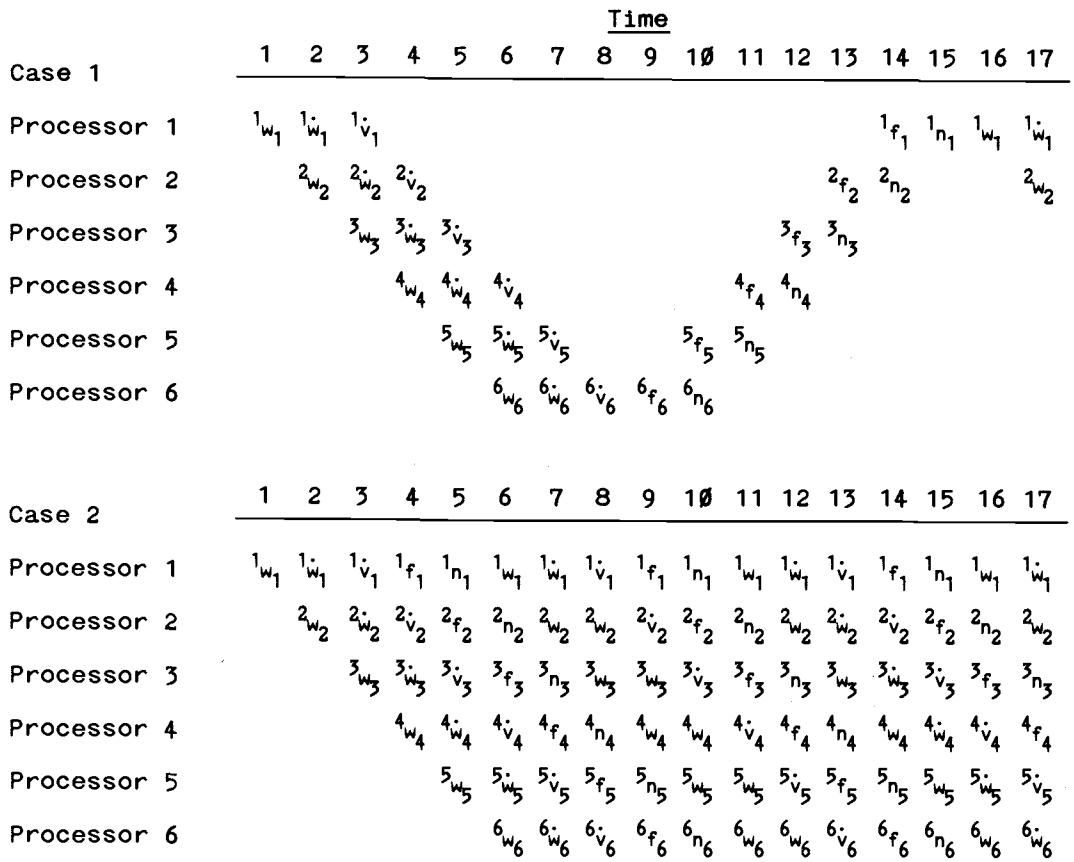

Figure 5.2 Timing Diagrams of Decomposed Computations.

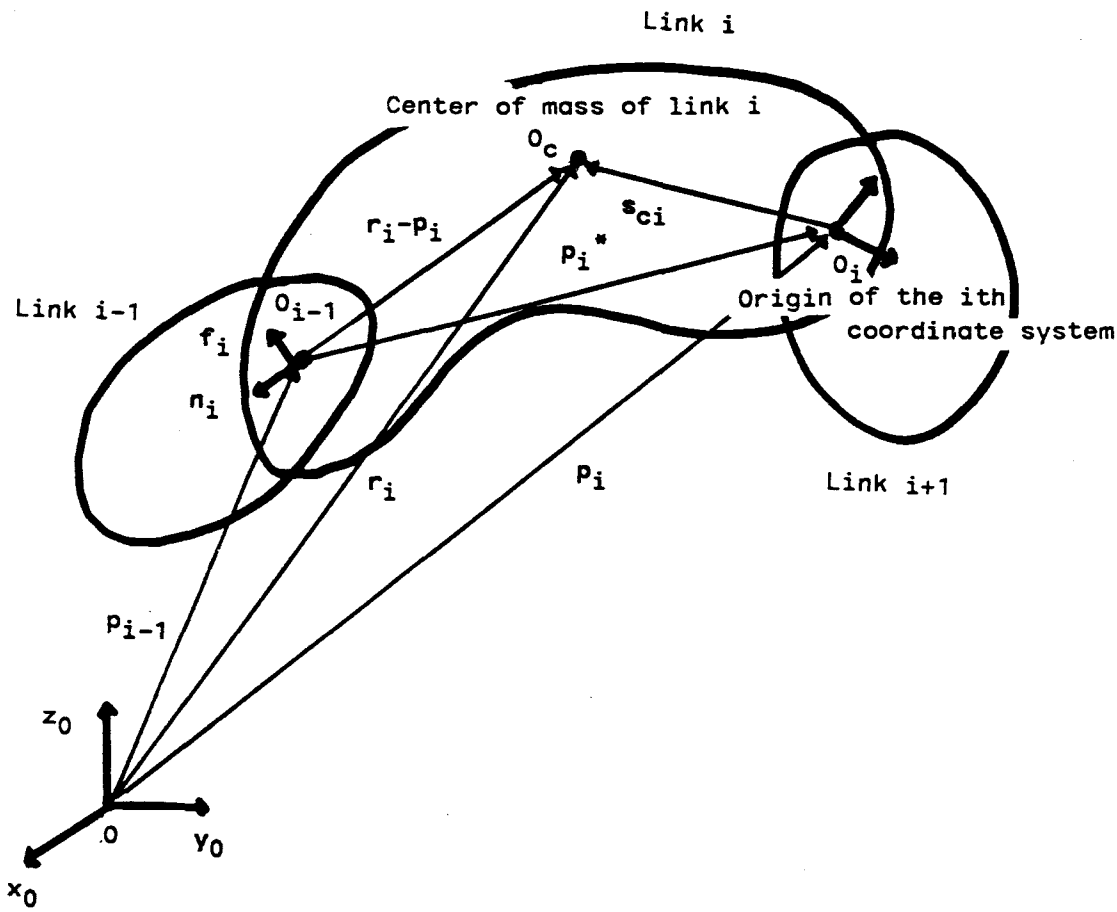

Figure 5.3 Forces and Parameters for Link i.

Table 5.1 Comparison of Computation Accuracy Between Decomposition Schemes.

		Link1 [%]	Link2 [%]	Link3 [%]	Link4 [%]	Link5 [%]	Link6 [%]
Traj 1	Case 2	1.1	Ø.9	Ø.4	Ø.5	Ø.4	Ø
	Case 3	1.6	1.1	Ø.5	1.2	1.9	18.Ø
	Case 4	3.2	2.2	1.1	2.3	3.3	25.7
Traj 2	Case 2	1.6	Ø.6	Ø.4	Ø.8	Ø.5	Ø
	Case 3	1.9	Ø.8	Ø.7	1.8	1.9	26.8
	Case 4	4.6	1.6	1.3	3.6	3.5	38.3

Chapter 6

Performance Evaluation and Motion Simulations

6.1 Introduction

In order to evaluate the performance of the proposed technique, two of the evaluation criteria considered in this study, accuracy of torques/forces calculation with open-loop simulation and accuracy of path-trajectory tracking with closed-loop simulation, are discussed in this chapter.

Generally, there are two principal approaches to performance evaluation: the analytical approach and the simulation approach [SAUE 81]. Precise analysis of the two criteria for the proposed algorithm is complicated because of the complexity and number of computations involved and time variances in the nonlinear dynamic equations. Therefore, a series of simulations was developed for evaluation of the proposed algorithms. Following discussion of the simulation, experimental results are compared with a class of alternative simplified methods in performance of calculations suggested in other studies.

6.2 Development of the Simulation Model

For simulation purposes, the Stanford robot arm described in Chapter 2 was used as a test vehicle because of its use in several other associated research projects and the availability of its mechanical parameters.

6.2.1 Trajectory Generation

The experimental simulations involved calculation of torques/forces with the proposed algorithm for different robot arm trajectories, each selected as the representation of different motion conditions. These trajectories allowed evaluation of the effect of different motion conditions on the accuracy of the proposed algorithm.

A number of trajectories were used in experimentation to achieve this purpose. In this section, only two of the representative trajectories are presented. However, the results presented for these trajectories were similar to those obtained from other trajectories.

The first trajectory (trajectory 1) was characterized by two segments of accelerations (or decelerations), and one segment of constant velocity (i.e, zero acceleration). It was selected in order to examine the effects of different acceleration (velocity) profiles on the calculations. The effect of constant velocity

was of special interest in trajectory 1, and it was maintained in passage through the segment between two intermediate points. The simulation of trajectory 1 was generated as follows.

Suppose that the path traveled by the end effector is given by the starting and final positions in Cartesian space subject to the following four constraints:

i) Two via points between the starting and final position are used to bound the deviation in Cartesian space;

ii) A time limit is imposed on completion of movement along every segment;

iii) The end effector has zero acceleration and zero velocity at both the starting and final positions; and

iv) The end effector also has constant velocity in the joint space between the two via points.

Each of the path points was specified in terms of a desired position and the orientation of the end effector, {T}, relative to the base coordinate system as follows:

$$T_s(t_0) = \begin{bmatrix} 0 & 0.86 & 0.5 & -0.3 \\ 1 & 0 & 0 & -0.5 \\ 0 & 0.5 & -0.86 & -0.2 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad t_0 = 0.0[\text{sec}]$$

$$T_1(t_1) = \begin{bmatrix} 0 & 0.86 & 0.5 & -0.3 \\ 1 & 0 & 0 & -0.15 \\ 0 & 0.5 & -0.86 & -0.2 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad t_1 = 0.7[\text{sec}]$$

$$T_2(t_2) = \begin{bmatrix} 0 & 0.86 & 0.5 & -0.3 \\ 1 & 0 & 0 & 0.15 \\ 0 & 0.5 & -0.86 & -0.2 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad t_2 = 1.3[\text{sec}]$$

and

$$T_f(t_3) = \begin{bmatrix} 0 & 0.86 & 0.5 & -0.3 \\ 1 & 0 & 0 & 0.5 \\ 0 & 0.5 & -0.86 & -0.2 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad t_3 = 2.0[\text{sec}]$$

where $T_s(t_0)$ = the starting point,

$T_f(t_3)$ = the final point, and

$T_1(t_1)$, $T_2(t_2)$ = the via points.

In order to perform the interpolation in the joint space, each of these points was converted into a set of desired joint angles, $q_i(t)$, by application of inverse kinematics. This is shown in Table 6.1, in which the four (six-dimensional) sets of joint positions corresponding to the four Cartesian positions, $T_s(t_0)$, $T_1(t_1)$, $T_2(t_2)$, and $T_f(t_3)$, respectively, are denoted by $J_s(t_0)$, $J_1(t_1)$, $J_2(t_2)$, and $J_f(t_3)$. The mechanical parameters of the Stanford robot arm were used in the calculations. Then, a 5th order polynomial, $q(t) = p_5t^5 + p_4t^4 + p_3t^3 + p_2t^2 + p_1t^1 + p_0$, was used to interpolate joint values in the joint space. The parameters, p_5, p_4, p_3, p_2, p_1 and p_0, of the 5th order polynomial, which satisfied the boundary conditions of each path segment, were calculated and are shown in Table 6.2, where $q(t)$ and $\dot{q}(t)$ denote, respectively, the time function of the joint position and velocity. The final trajectory of each joint of the Stanford robot arm is shown in Figure 6.1. Figure 6.2 indicates

the trajectory traveled by the end effector of the manipulator in Cartesian space.

The secondary trajectory (trajectory 2) was characterized by two high speed acceleration segments without constant velocity. This trajectory was selected to examine the effect of high speeds and acceleration on the calculations. It used the same end points as trajectory 1, but was restricted to only one intermediate point, at which it was stopped. These restrictions increased accelerations and velocities along the trajectory.

For further performance evaluation of the proposed algorithm, constraints on the total time allowed to complete movements were given. By changing the total time required for completion of identical trajectories, the effect of different accelerations and velocities on the accuracy of the proposed algorithm could be observed. The given total elapsed time constraints were slow movement, fast movement, and very fast movement at, respectively, 6[s], 2[s] and 1[s].

Within these constraints, the torques/forces were computed for each case at five milliseconds sampling intervals. This period was determined by the criterion that the controls would be updated at a rate of about 15 times the structural resonance frequency of the robot arm [PAUL 84].

6.2.2 Simulated Control Methodology

A block diagram of the dynamic computer simulation is shown in Figure 6.3. For the robot arm model, the general form of the Lagrangian-Euler dynamic formulations, Equation (6.2-1), was used for motion simulation in the manipulator.

$$\mathbf{u}(t) = \mathbf{D}(\mathbf{q}(t))\ddot{\mathbf{q}}(t) + \mathbf{H}(\mathbf{q}(t),\dot{\mathbf{q}}(t)) + \mathbf{C}(\mathbf{q}(t)) \quad , \qquad (6.2\text{-}1)$$

where $\mathbf{u}$ = 6 × 1 desired torque/force vector,

$\mathbf{D}$ = 6 × 6 inertial acceleration-related matrix,

$\mathbf{H}$ = 6 × 1 nonlinear Coriolis and centrifugal force vector, and

$\mathbf{C}$ = 6 × 1 gravity loading force vector.

From Equation (6.2-1), the desired torque/force, $\mathbf{u}(t)$, for every joint can be calculated (see section 6.3 below), using the proposed algorithms.

For simulation purposes, the actual acceleration of the joint variables can be obtained by solving the following equation from Equation(6.2-1):

$$\ddot{\mathbf{q}}_a(t) = \mathbf{D}^{-1}[\mathbf{u}(t) - (\mathbf{H}+\mathbf{C})] \quad , \qquad (6.2\text{-}2)$$

where $\ddot{\mathbf{q}}_a(t)$ = actual joint acceleration. The $(\mathbf{H}+\mathbf{C})$ term can be easily be computed from the Newton-Euler equations of motion by setting the joint acceleration vector $\ddot{\mathbf{q}}(t) = \mathbf{0}$. Then, only the matrix $\mathbf{D}$ is calculated

from the Lagrangian-Euler formulation. Once the actual joint acceleration is calculated, the next actual velocity and position of the joint can be calculated by integrating, respectively, the actual joint acceleration and velocity. This procedure was repeated from the initial point through all the sample points of the trajectory. For the initial point, the actual and the desired trajectory points, $\mathbf{q}(t)$, $\dot{\mathbf{q}}(t)$ and $\ddot{\mathbf{q}}(t)$, are identical. The integration utilizes a numerical integration, the Euler technique, as follows;

$$\dot{\mathbf{q}}(t+dt) = \dot{\mathbf{q}}(t) + \ddot{\mathbf{q}}(t)*dt \qquad (6.2\text{-}3)$$

and

$$\dot{\mathbf{q}}(t+dt) = \dot{\mathbf{q}}(t) + (\ddot{\mathbf{q}}(t) + 0.5*\ddot{\mathbf{q}}(t)*dt)*dt \ , \qquad (6.2\text{-}4)$$

where dt = sampling interval of the simulation. The sampling interval should be less than, or at most equal to, the sampling interval of the torque calculation.

The computed torque technique [LEE 82a, ORIN 84, AN 87, KHOS 88a] is a basic scheme of control, i.e., a feedback control with feedforward and feedback components. The feedforward component computes the normal joint torque along the desired trajectory to compensate for the interaction forces among all the various joints; the feedback component computes the necessary correction torques to compensate for any deviations from the desired trajectory. For the purpose of motion simulation, the computed torque technique, based on a

feedback position gain matrix, $\mathbf{K}_p$, and a feedback velocity gain matrix, $\mathbf{K}_v$, was used.

In this technique, terms in the dynamic torques/forces calculations are computed from the actual joint positions and velocities. The accelerations are updated from the difference between the actual and the desired path. This generates accelerations, and thus torque, tending to the decrease of errors. The acceleration was computed from the following equation:

$$\ddot{\mathbf{q}}(t) = \mathbf{K}_p(\mathbf{q}_d(t) - \mathbf{q}_a(t)) + \mathbf{K}_v(\dot{\mathbf{q}}_d(t) - \dot{\mathbf{q}}_a(t)) + \ddot{\mathbf{q}}_d(t) \quad , \qquad (6.2\text{-}5)$$

where $\mathbf{K}_p$ and $\mathbf{K}_v$, representing motion simulation, are described in section 6.4.

6.3 Torques/Forces Calculations

6.3.1 Procedure

The general experimental procedure involved torques/forces calculations for the algorithms which were evaluated. Calculations with open loop simulation were performed for the different robot motion trajectories. The torques/forces for the same trajectories were also calculated with the exact recursive fashion. Finally, the relative torques/forces errors between the exact calculations and those for the evaluated algorithms were determined for comparing the performance of the algorithms. Comparison of the relative errors is of value since the magnitude of the errors is relative

tothe calculated values of the torques/forces for a specific trajectory. Several different methods of computation were used: the Bejczy [BEJC 79, 81, PAUL 84] algorithm and the Binder [BIND 86] algorithm. Each is described in the following section. Relative torque errors along the trajectory were calculated by dividing the torque errors by the full scale torque or force, in turn calculated as the difference between the maximum torque and the minimum torque for every plot.

Two main experimental comparisons were performed. The purposes of the first experiment were to evaluate the torque errors in the first proposed algorithm, PAFP1, in comparison to errors developed with other algorithms. The objectives of the second experiment were to evaluate the errors in the second proposed algorithm, PAFP2, comparing them with errors occurring with use of PAFP1.

Each experiment was performed with different robot motion trajectories, chosen to represent different motion conditions as described in previous section of this study. This allowed for the evaluation of the effect of different velocity and acceleration conditions on the accuracy of the algorithms.

6.3.2 Comparison of Algorithms

The proposed algorithms were compared with a class of simplified methods for calculation performance,

i.e., the Bejczy [BEJC 79, 81, PAUL 84] algorithm and the Binder [BIND 86] algorithm. Bejczy derived the complete dynamic equations for the Stanford manipulator. By eliminating less significant terms, and precalculating terms which were not position dependent, a simplified and efficient algorithm for this manipulator was derived. In the simplification of the equations for link 4, it was suggested that one of the gravity terms in **C**(4) should be neglected. It was found that the addition of this term could improve the accuracy of the simplified algorithm without necessitating significant additional computation time and this was done for purposes of experimental comparison. The simplified method presented by Bejczy, as amended by Paul, with the added term described above, will be referred to as Bejczy's algorithm in this chapter.

Binder [BIND 86] distributed the dynamic equations over multiple computers, one for each joint. Each computer calculated its own dynamic equations for its joint torque and then serially served as the servo controller for each joint. Computation of the dynamic equations for each joint was achieved by substituting predicted values, based on previously calculated values, for the actual values of all the forward and backward equation variables. Based upon this algorithm, computation time required for the dynamic equations was reduced. However, this approach failed to properly

consider the sequential dependencies of the dynamic equations, which are conducive to pipelining between adjacent links. As a result, this algorithm generated errors in the calculation of the inverse dynamics.

Programs for the implementation of the above algorithms have been prepared and were used in the experimentation in this study.

6.3.3 Experimental Results, I

The purpose of the first experiment was to evaluate torque errors resulting from the PAFP1, in comparison to errors developed with the application of other algorithms. Results of the torque computations for each joint using PAFP1 are shown in Figures 6.4 to 6.15 and Tables 6.3 to 6.6 for trajectories 1 and 2. These tables compare the relative root mean squared torque errors for each joint introduced by use of the three different computational methods. Each torque calculation was based on a five-millisecond sampling period. In order to investigate the effect of using a high sampling rate for torque calculations in PAFP1, relative torque errors were also calculated with a high sampling rate of one-millisecond. Figures 6.4 to 6.6 for trajectory 1 show torque comparisons between PAFP1 and exact calculations. In addition, Figures 6.7 to 6.9 compare relative torque errors for each joint introduced by use of the three different computational

methods. Figures 6.10 to 6.15 compare the results for trajectory 2. A solid straight line, parallel to the time-axis (Figures 6.7 to 6.9 and 6.13 to 6.15) was used as a reference line representing zero percent of the relative torque error. Elapsed time is labeled on the horizontal axis of each plot.

The slow motion 6-second duration was characterized by slow velocity and acceleration, and the very fast motion 1-second duration was characterized by very fast velocity and acceleration. Sections with constant torques in the Figures 6.4 to 6.6 correspond to path segments at constant velocities. Torque errors along the sections of constant velocities in Bejczy's algorithm were caused by the neglected Coriolis and centrifugal terms. Errors in the sections with acceleration result from neglecting the coupling terms in the inertia matrix, as well as neglecting the Coriolis and centrifugal terms. The coupling terms are related to the relative accelerations between the links. The effects of these terms are demonstrated in the very fast motion plots. For fast acceleration, the coupling terms are more significant, therefore the errors in the algorithm became noticeably large, particularly for the last three links. From Tables 6.3 to 6.6 it may be seen that the relative torque error of link 1 at each trajectory did not change, even when the speed of the robot arm was increased. From this we may conclude that

there was little interaction between adjacent links and the first link.

Torque errors in Binder's algorithm increased as total elapsed time decreased. The errors were caused by the difference between the predicted values and the exact values of the variables involved in all the forward and backward recursive equations. This occurred because Binder's algorithm fails to properly consider the sequential dependencies of the pipelined nature of the dynamic formulations. Increasing the speed of motion increases the difference values for all forward and backward variables. As a result, when motion speed is increased, the torque errors in Binder's algorithm become large, particularly for the last three links.

As shown in the figures, the PAFP1 performance was much better than the performances of the other two algorithms for the calculation of torques. Trajectory 2 was characterized by higher speeds and acceleration than trajectory 1. Even for very fast motion for both trajectories, the torque errors in the last three links, as well as in the first three links, were still relatively small. In particular, errors in link 6 were always zero since link 6 used the exact values of the recursive backward equation variables. Torque errors introduced into the PAFP1 resulted from the effect of calculating the backward equation of link i, ${}^{i}\mathbf{f}_{i}(t)$ and/or ${}^{i}\mathbf{n}_{i}(t)$, due to differences in force and/or

moment exerted on link i+1 by link i with respect to its own coordinate system between successive sampling intervals. Thus, if a smaller sampling interval of 1-millisecond could be achieved with PAFP1, the torque errors shown in Figures 6.7 to 6.9 would become much smaller.

6.3.4 Experimental Results, II

The purpose of this experiment was to evaluate the errors resulting from the use of PAFP2 and to compare them with errors resulting from the use of PAFP1 (Experimental Results, I, in the previous section). From the results of the first experiment, the torque errors were large for the fast trajectories, particularly for trajectory 2, which was characterized by higher speeds and greater acceleration than trajectory 1. In this section, the fast motion and the very fast motion speeds for trajectory 2 were selected for the evaluation of torque errors along the trajectory, shown in Figures 6.16 and 6.17. Thus, the effect of using PAFP2 can be clearly observed. Solid lines in the figures represent the exact calculations, the dashed lines represent PAFP1, and the dotted lines represent PAFP2. The results of the calculation patterns achieved by the use of PAFP2 are similar to results achieved with the use of PAFP1, but in every case the magnitude of the errors was reduced. The results presented for the

PAFP2 trajectory were also similar to those obtained for the other trajectories.

Tables 6.3 to 6.6 summarize the results of Experiments I and II, as well as trajectories 1 and 2. In every case, using PAFP2, the relative root mean squared torque errors are less than the errors generated with the application of PAFP1, and the errors are less than 1.2 percent even for the very fast motion in trajectory 2. Furthermore, when a high sampling rate of 1-millisecond was used, the errors produced by application of the proposed methods become even smaller, as shown in Figures 6.7 to 6.9 and 6.13 to 6.15. From this we may conclude that a higher sampling rate directly affects the accuracy of system performance and is desirable.

6.4 Motion Simulations

The objective of this experiment was to evaluate the performance of the proposed algorithms and the accuracy of their path trajectory tracking by comparing the actual trajectories with the desired trajectories. The results of the these motion simulations are related to the feedback technique applied.

6.4.1 Procedure

From the block diagram of the dynamic computer simulation shown in Figure 6.3, the motion simulation

procedure was initiated by computing the torques/forces required to maintain the desired trajectory. Then, the torque calculations were applied to the actual position and velocity of the robot arm joints, and actual acceleration was calculated from Equation (6.2-2), as explained in section 6.2.2. Unlike open loop simulation, the simulation procedure for closed loop motion simulation uses the feedback gain matrices $\mathbf{K}_p$ and $\mathbf{K}_v$. In general, it was difficult to determine the optimal value of these matrices for the best path trajectory tracking since $\mathbf{K}_p$ and $\mathbf{K}_v$ are nonlinear, position dependent matrices. However, the matrices can be obtained by ignoring the coupling terms and by assuming constant matrices. Based upon this assumption, diagonal and constant matrices can be obtained by trying and tuning the diagonal terms until stable motion is achieved.

6.4.2 Experimental Results, III

When open loop simulation was performed, it was observed that a condition of instability (a continually diverging response) was exhibited. This occurred because of accumulated errors with time.

For closed loop simulation, the matrices $\mathbf{K}_p$ and $\mathbf{K}_v$, based upon the assumption stated in the previous section, were experimentally determined from a number of motion simulations though they may not provide optimal results for numerical comparison. Nonetheless,

from the simulation results achieved from application of the feedback gain matrices, the following general observations may be offered.

When the speed of the robot arm was increased, using Bejczy's and Binder's algorithm, the last three links were more difficult to control than the first three because of large errors at the former. As foreseen, the position errors in path trajectory tracking with the proposed methods were very small, even at the very fast motion indicated in Table 6.7, which shows the relative root mean squared position errors for fast and very fast motion of trajectory 1 at 5- and 1-millisecond sampling rates with PAFP1. From this table it may also be observed that the position errors for the 1-millisecond sampling rate were much smaller than those for the 5-millisecond sampling rate. From this we may conclude that the increased sampling rate used with the proposed algorithms has a substantial impact on robot arm control. The relative root mean squared position errors were calculated by dividing the root mean squared position errors by the difference between the maximum position and the minimum position for every plot. In addition, Figure 6.18 shows the path tracking errors of the entire trajectory as an example of motion simulation. In the plot, the solid line represents the desired trajectory and the dotted line represents the actual trajectories, Actual1 and Actual2, for the fast

motion case using PAFP1. The 5-millisecond sampling rate with PAFP1 and the feedback gain matrices, $\mathbf{k}_p$ and $\mathbf{k}_v$, were used for the plots, where diagonal terms of

$$\mathbf{k}_p = [3.8,3.2,0.5,6,6,6]$$

and

$$\mathbf{k}_v = [39,36,14,49,49,49]$$

for Actual1, and diagonal terms of

$$\mathbf{k}_p = [11.4,9.6,1.5,12.0,18.0,18.0]$$

and

$$\mathbf{k}_v = [117.0,108.0,42.0,98.0,147.0,147.0]$$

for Actual2.

6.5 Summary

In order to evaluate the performance of the proposed methods, accuracy of torque calculations and accuracy of path trajectory tracking were considered as the evaluation criteria. A series of simulations were developed to test performance results. Following simulation, the experimental results were compared with a class of alternative methods and comparative results indicated that the performance of the proposed methods was superior to those of the alternative methods.

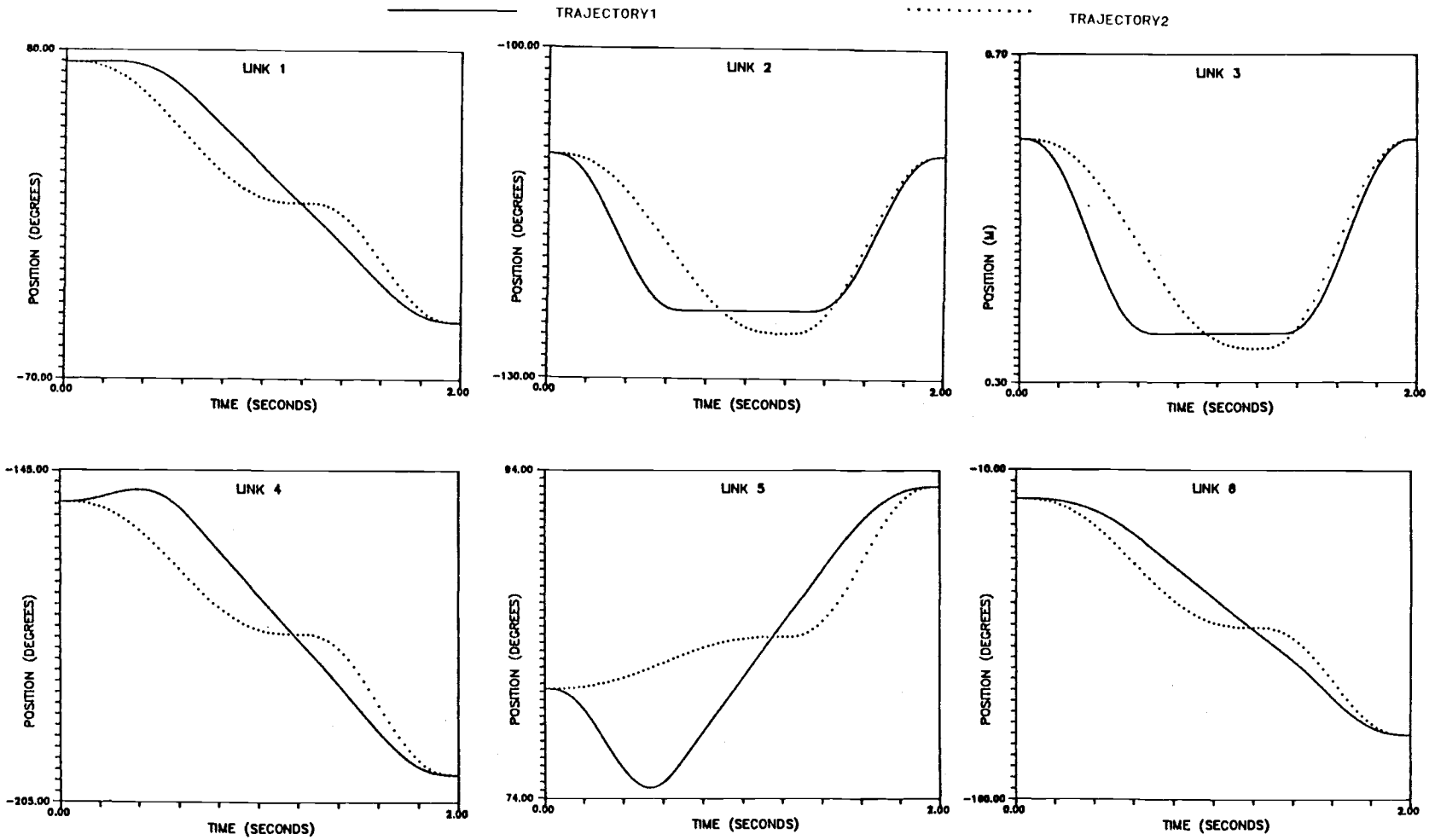

Figure 6.1 Reference Trajectories of Robot Joint for Motion Simulation Study.

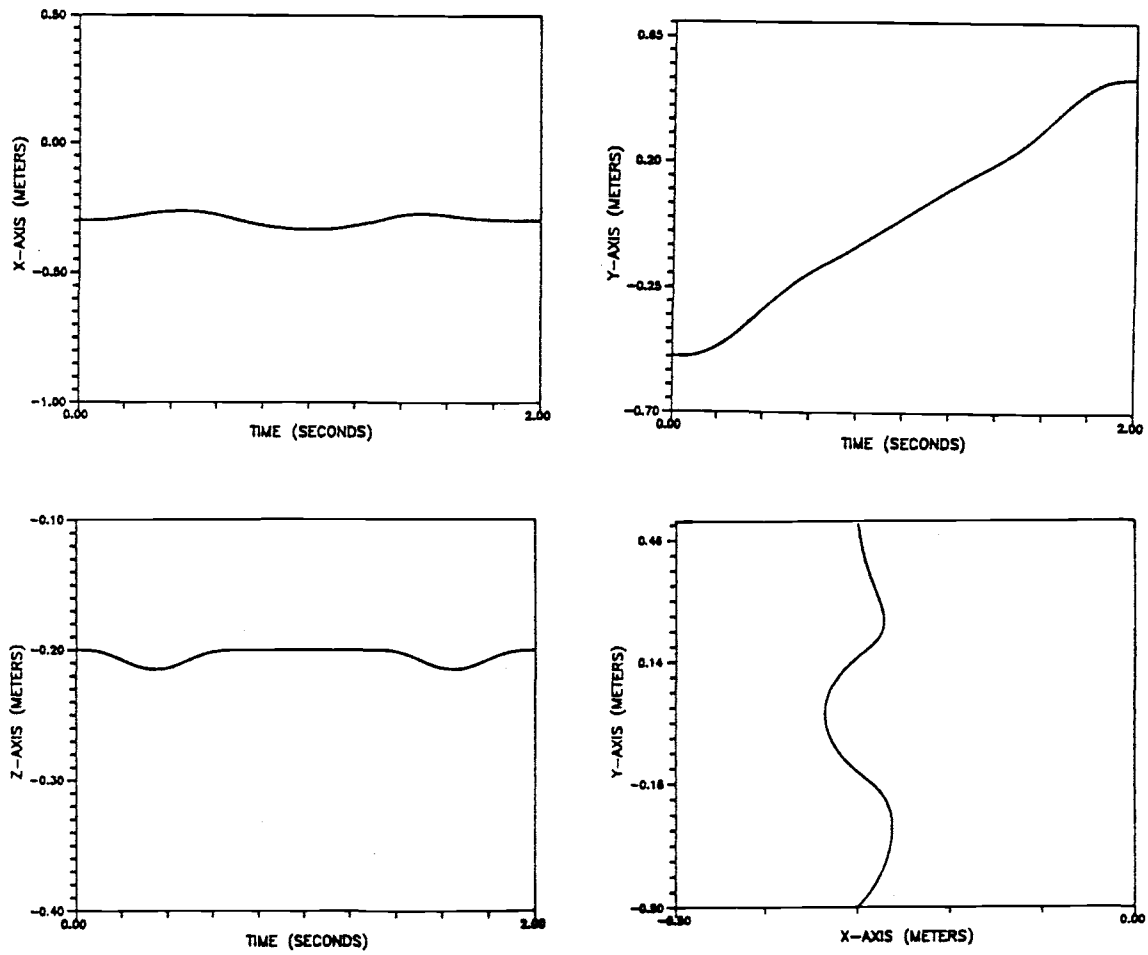

Figure 6.2 End Effector Trajectory of Trajectory 1.

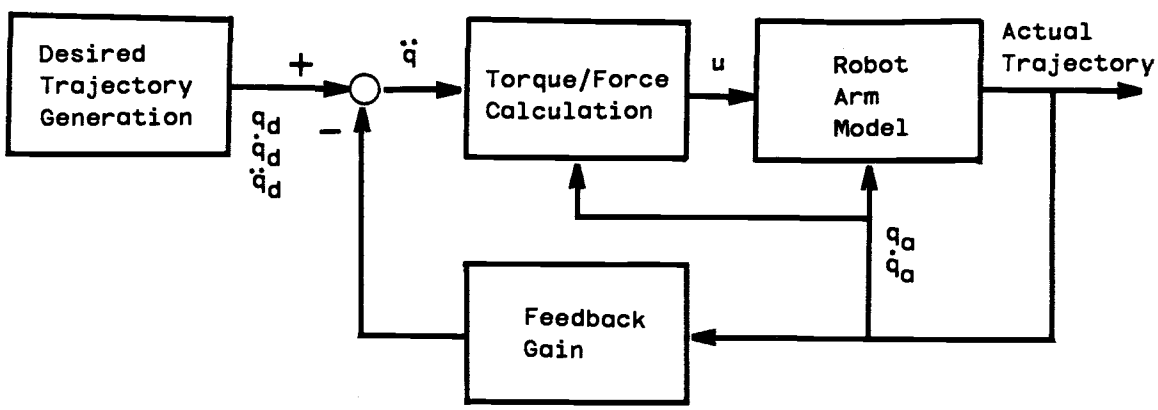

Figure 6.3 Block Diagram of Simulated Control Methodology.

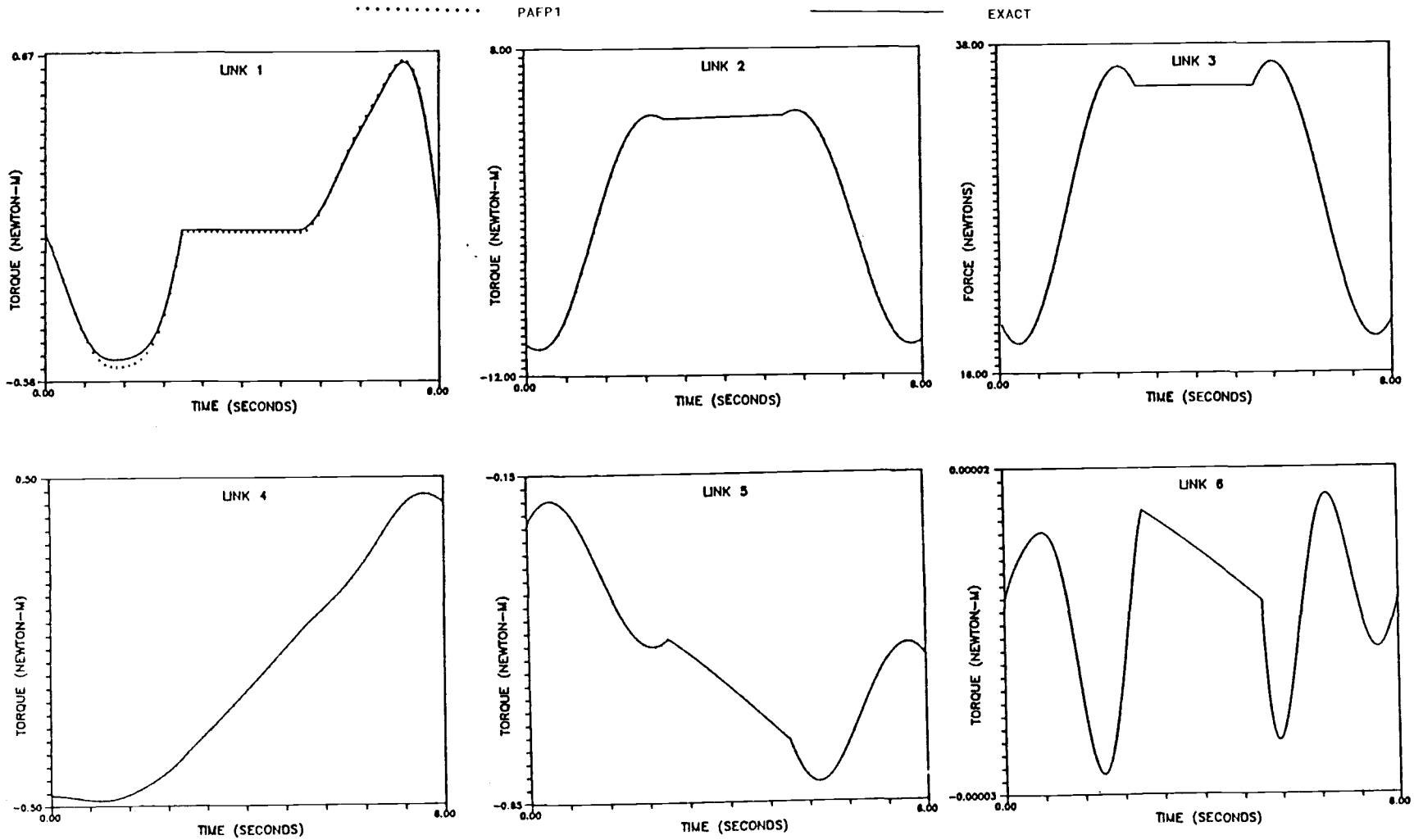

Figure 6.4 Comparison of Trajectory 1 Torque, 6-Second Duration.

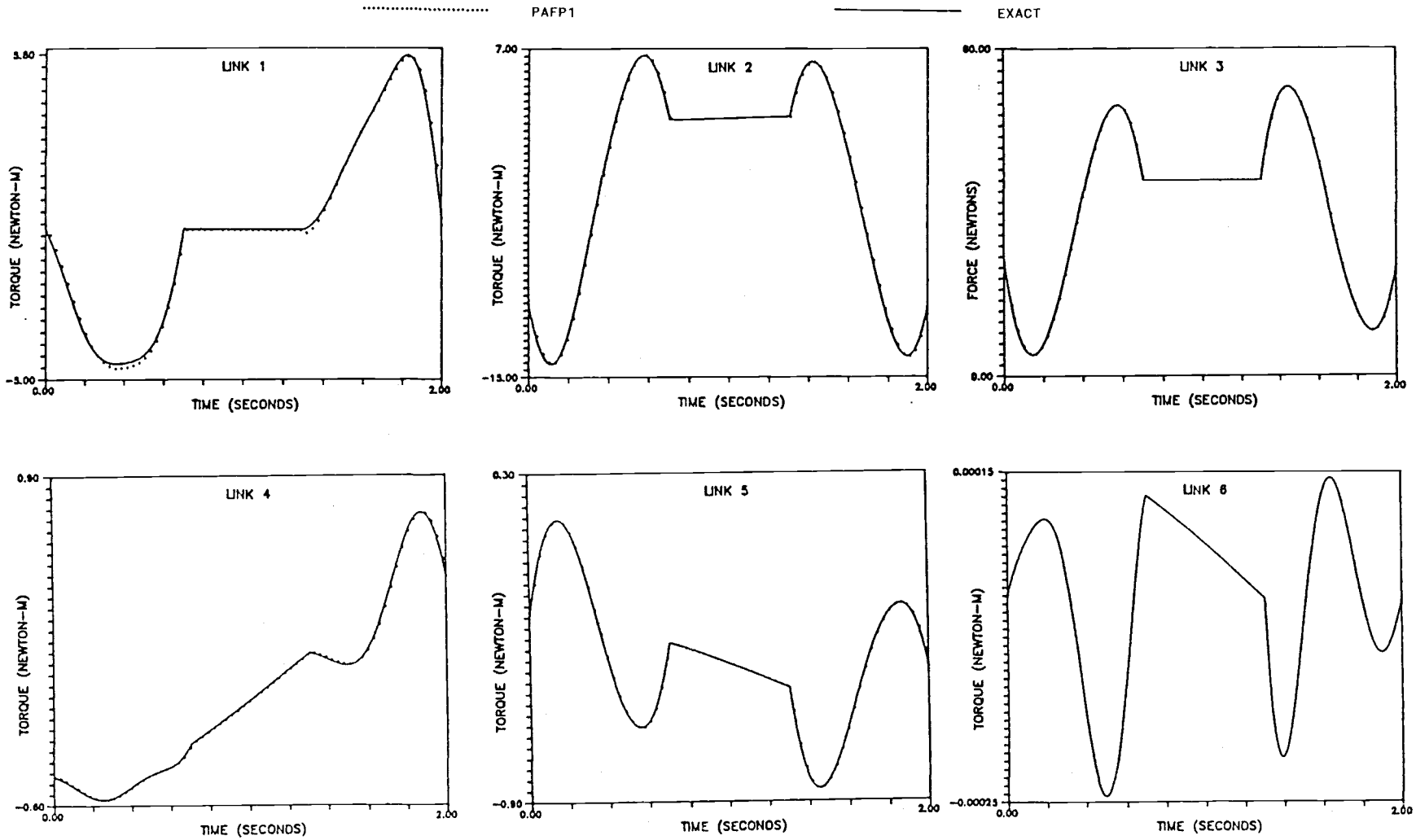

Figure 6.5 Comparison of Trajectory 1 Torque, 2-Second Duration.

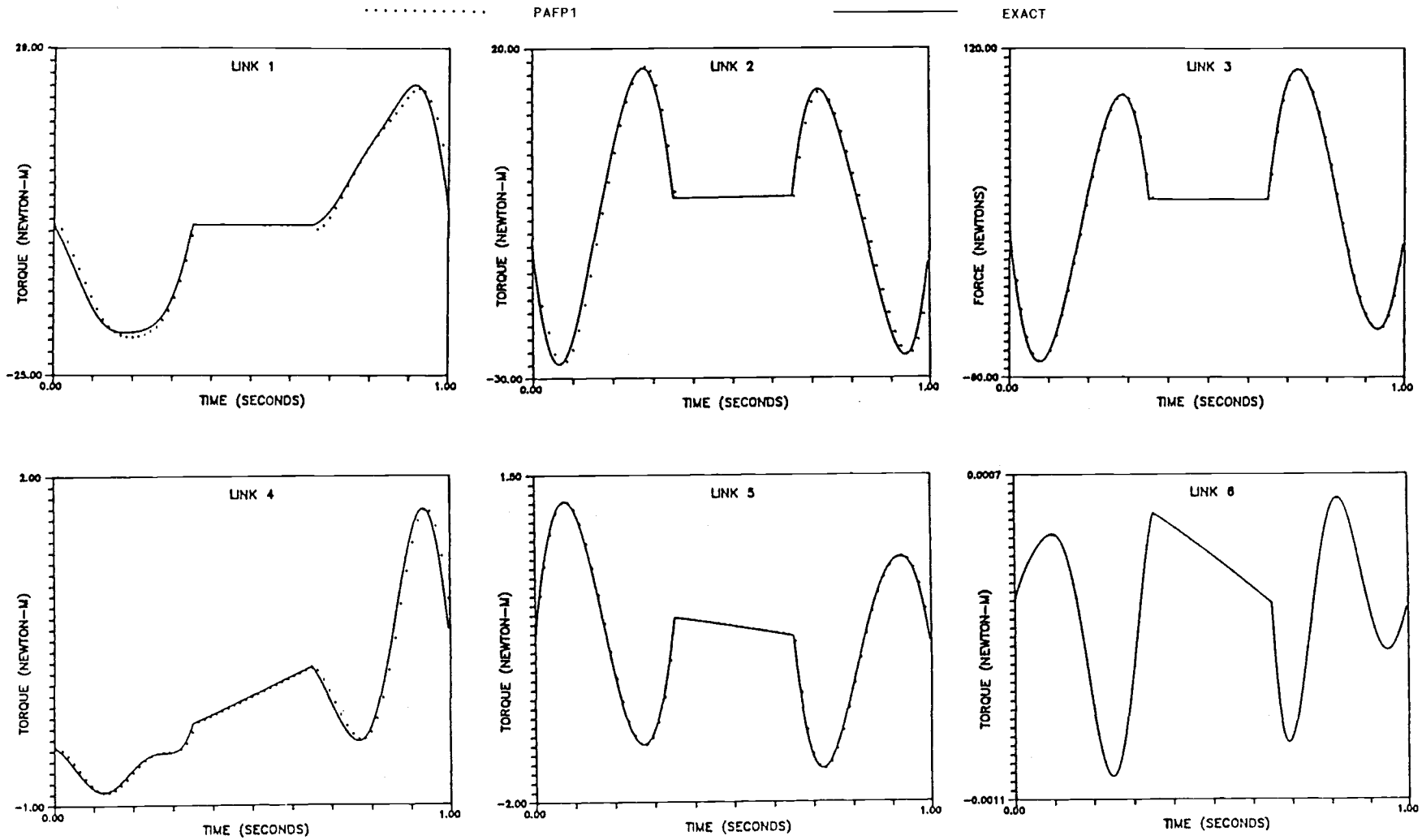

Figure 6.6 Comparison of Trajectory 1 Torque, 1-Second Duration.

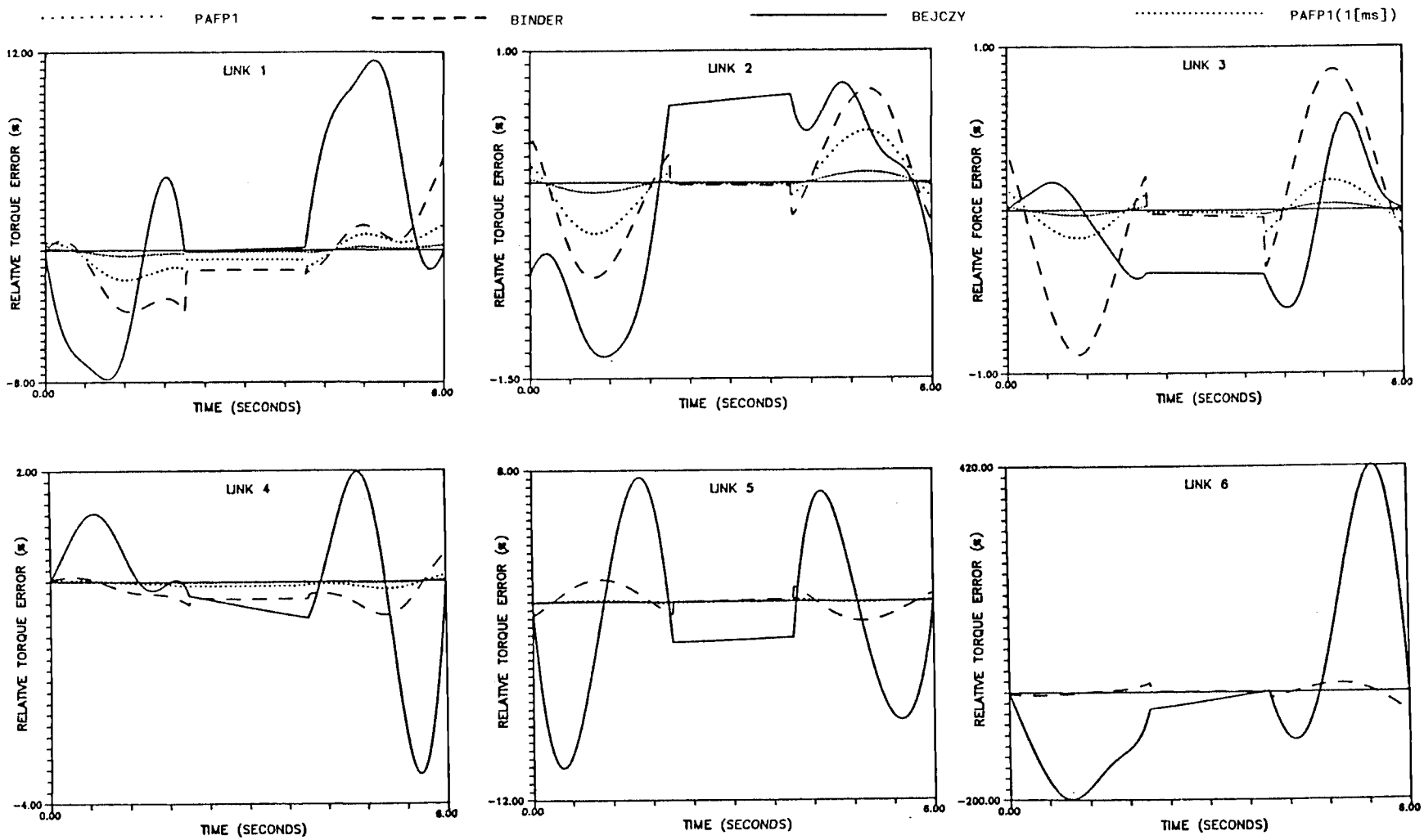

Figure 6.7 Comparison of Relative Errors,
Trajectory 1, 6-Second Duration.

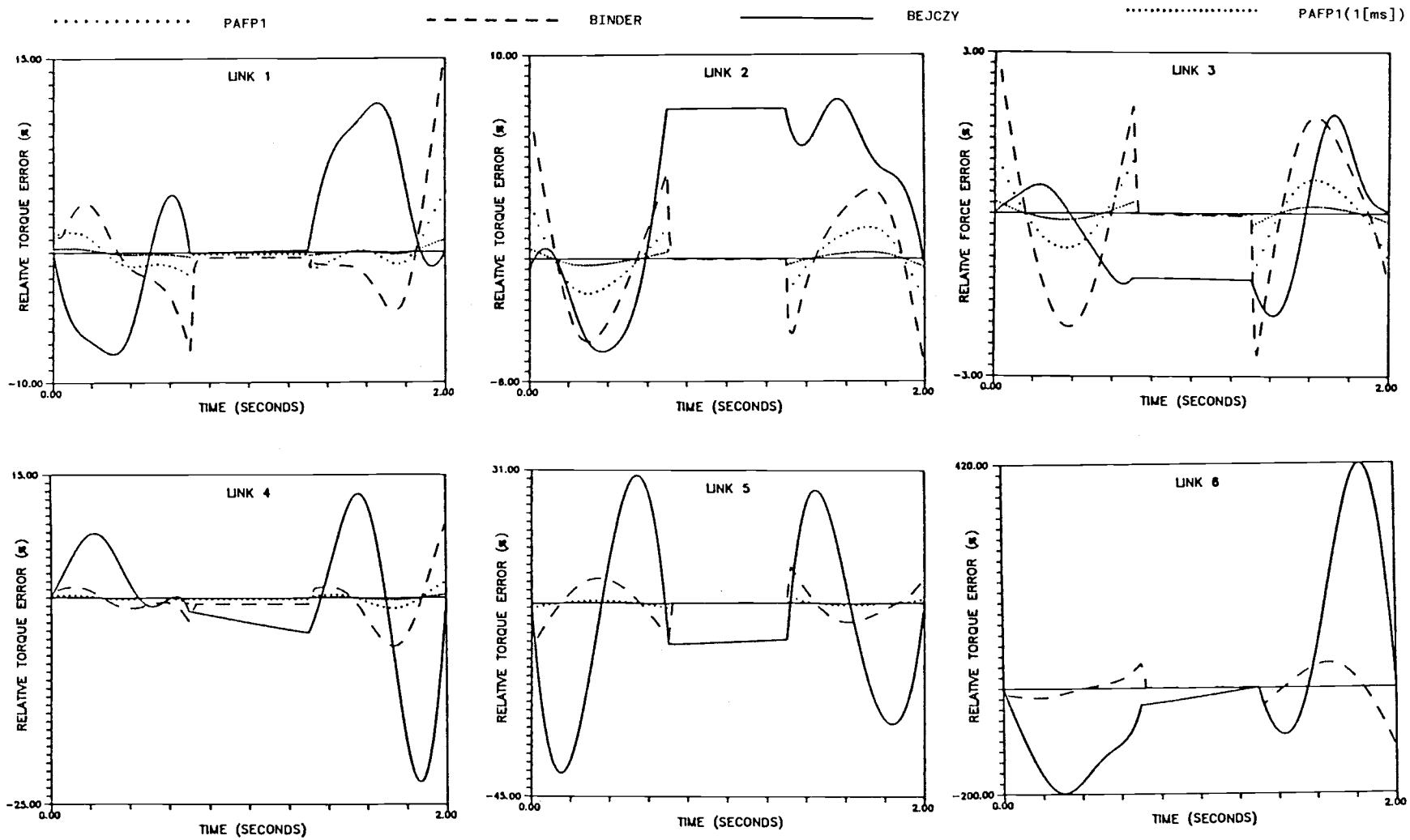

Figure 6.8 Comparison of Relative Errors,
Trajectory 1, 2-Second Duration.

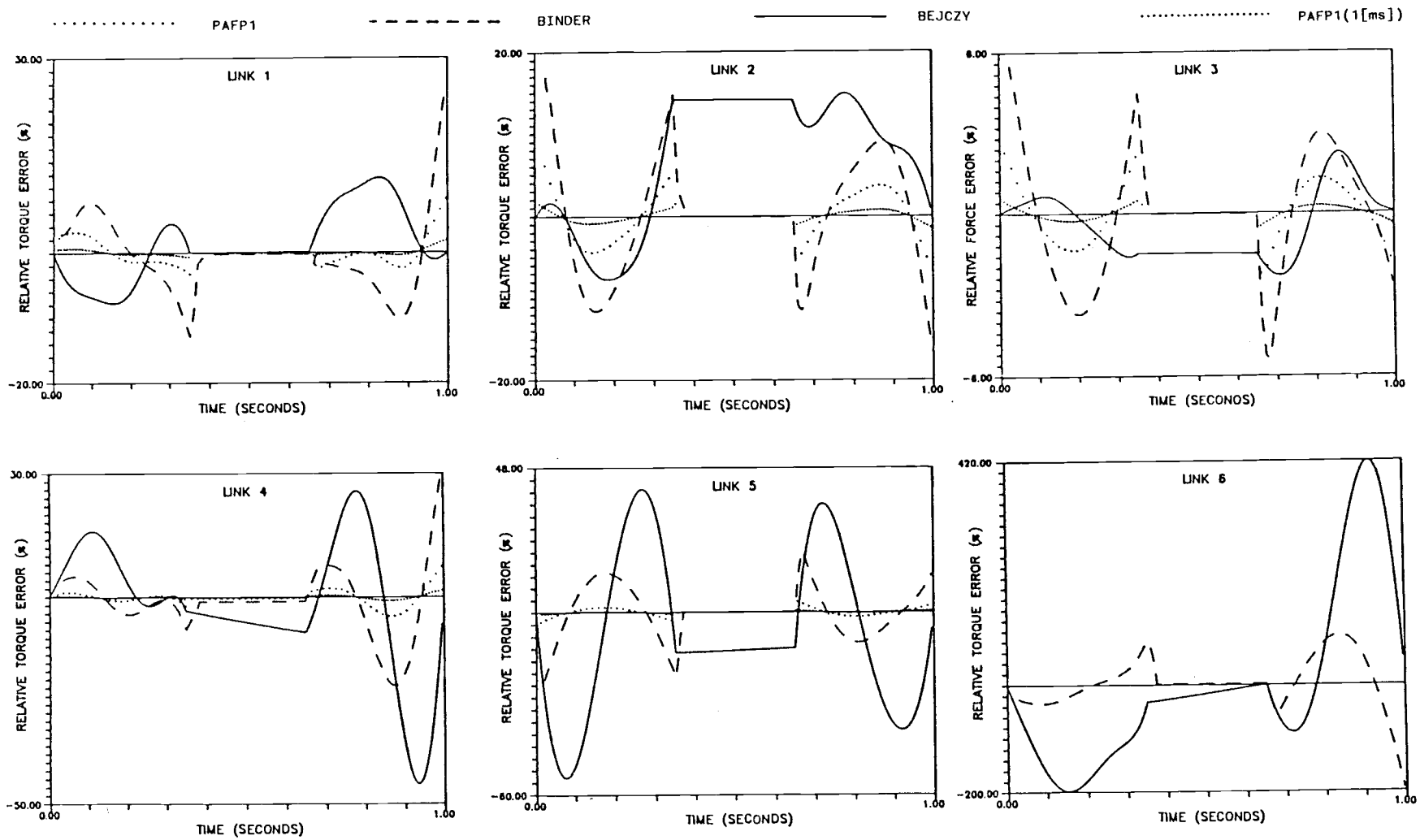

Figure 6.9 Comparison of Relative Errors,
Trajectory 1, 1-Second Duration.

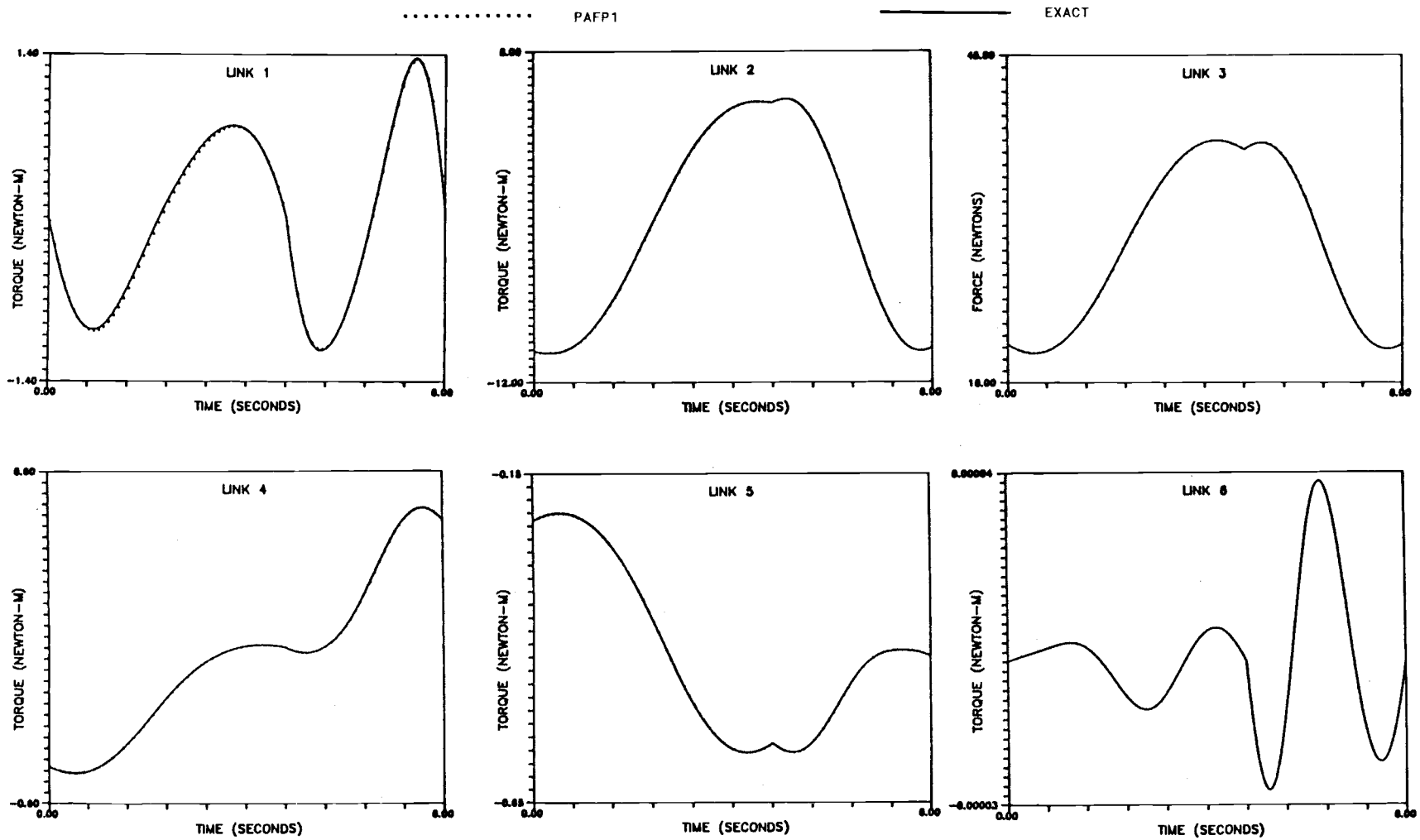

Figure 6.10 Comparison of Trajectory 2 Torque, 6-Second Duration.

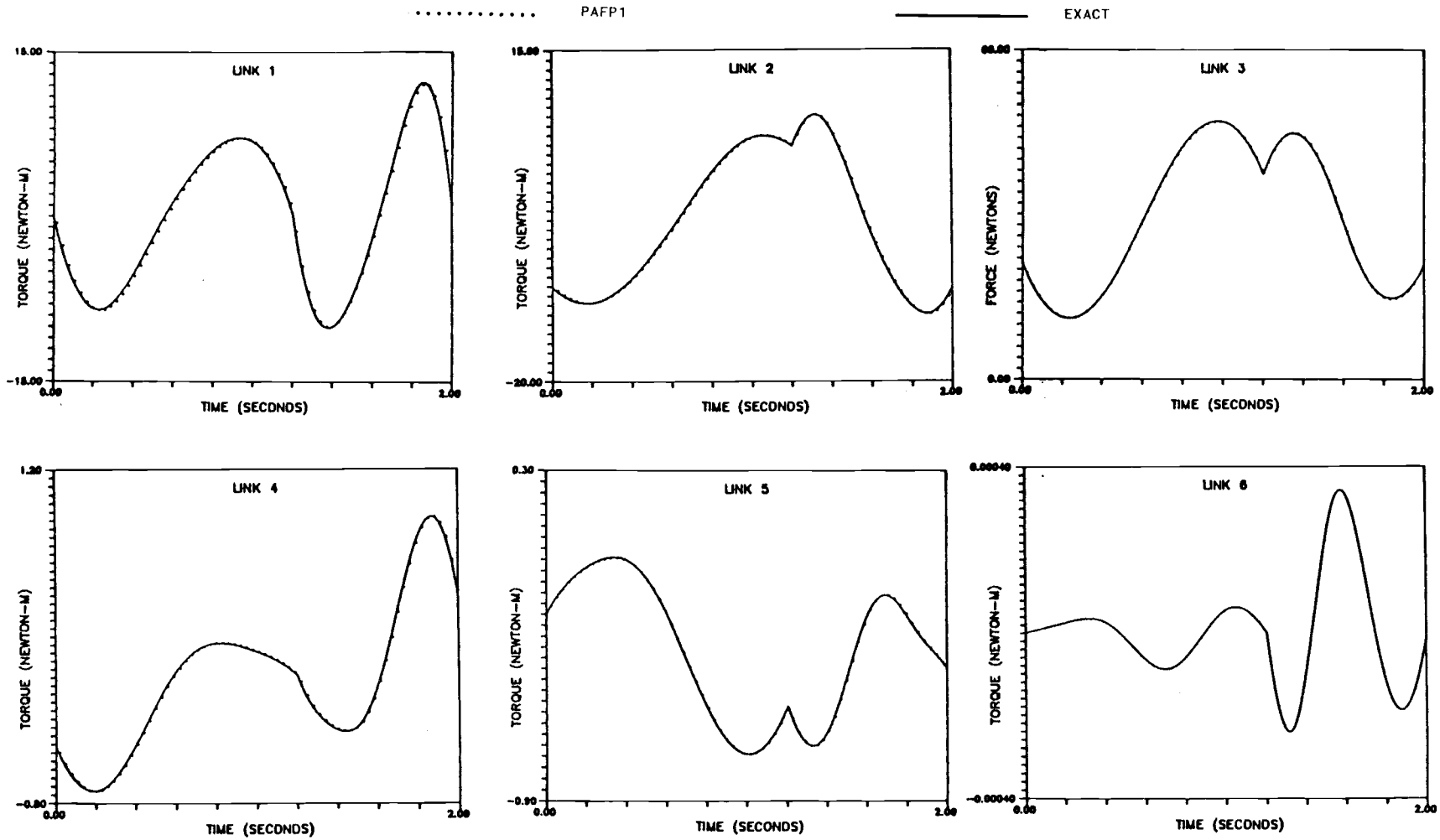

Figure 6.11 Comparison of Trajectory 2 Torque, 2-Second Duration.

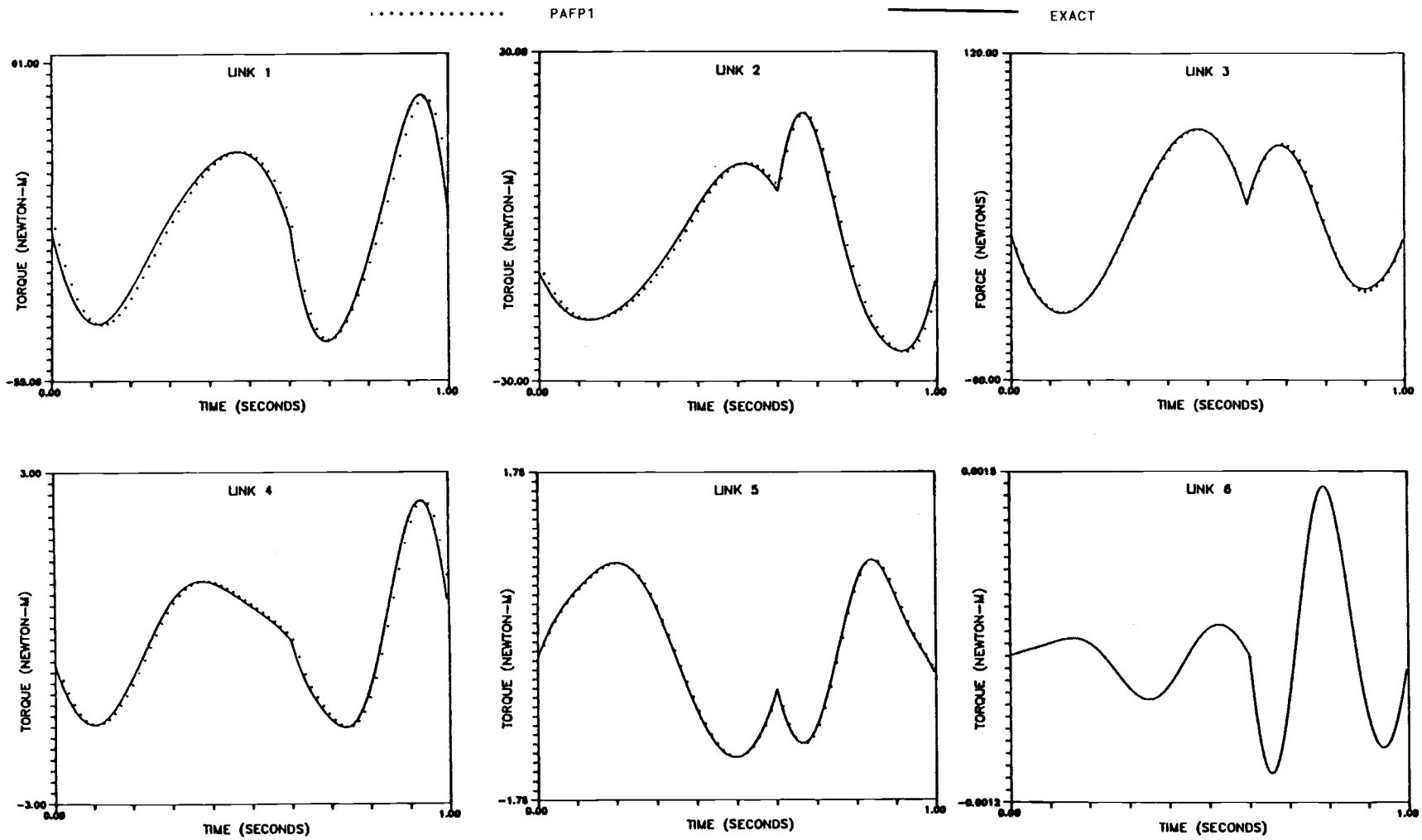

Figure 6.12 Comparison of Trajectory 2 Torque, 1-Second Duration.

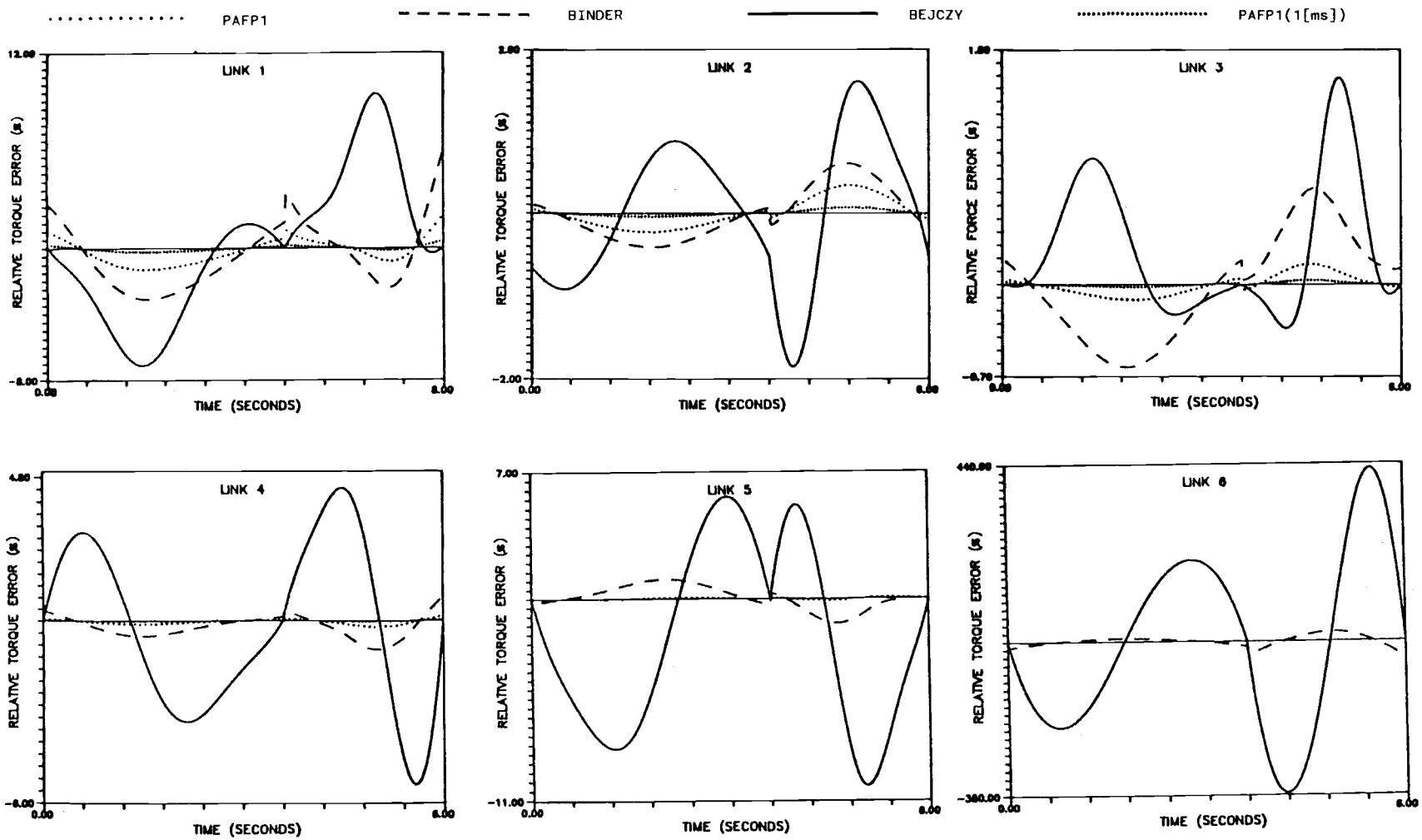

Figure 6.13 Comparison of Relative Errors,
Trajectory 2, 6-Second Duration.

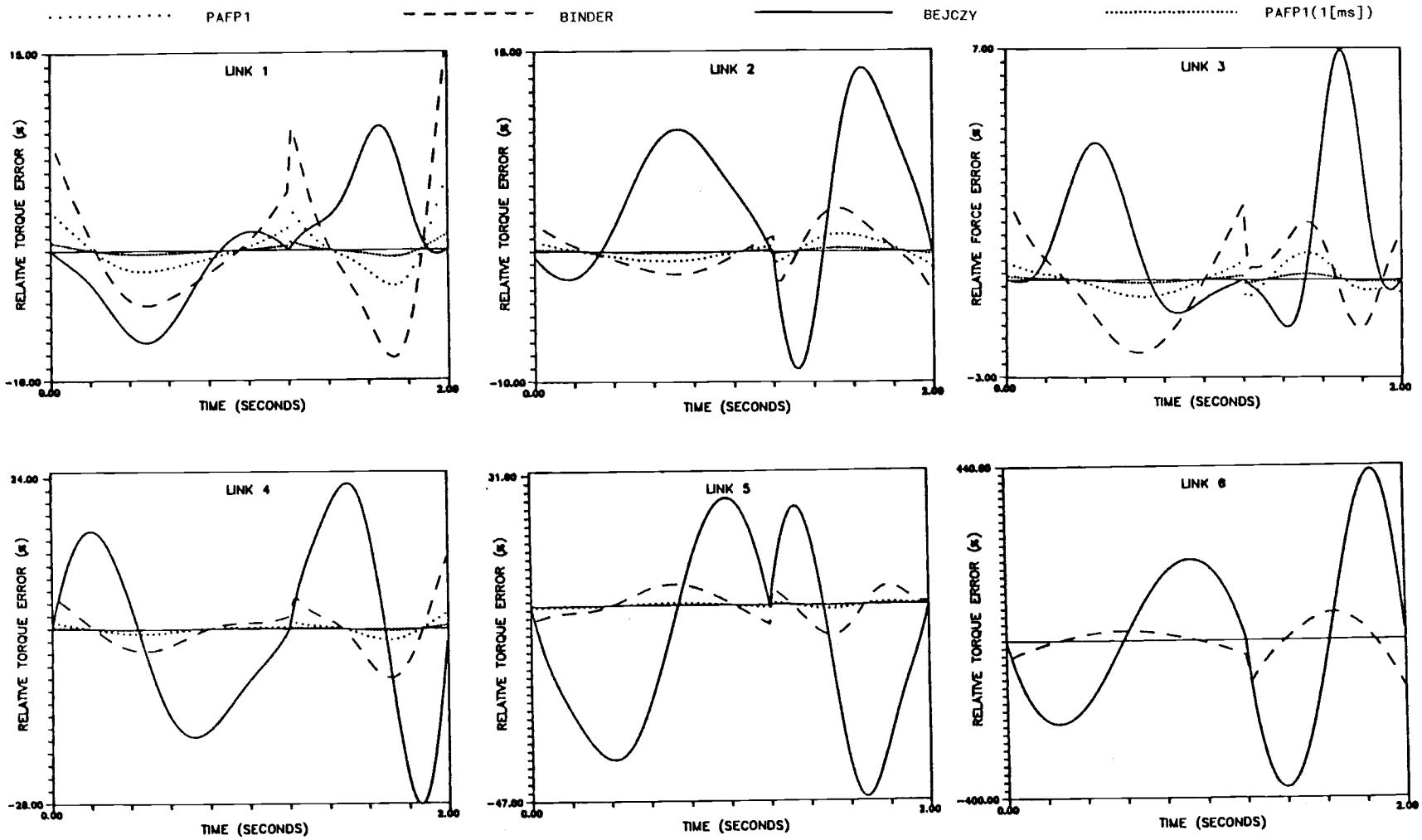

Figure 6.14 Comparison of Relative Errors, Trajectory 2, 2-Second Duration.

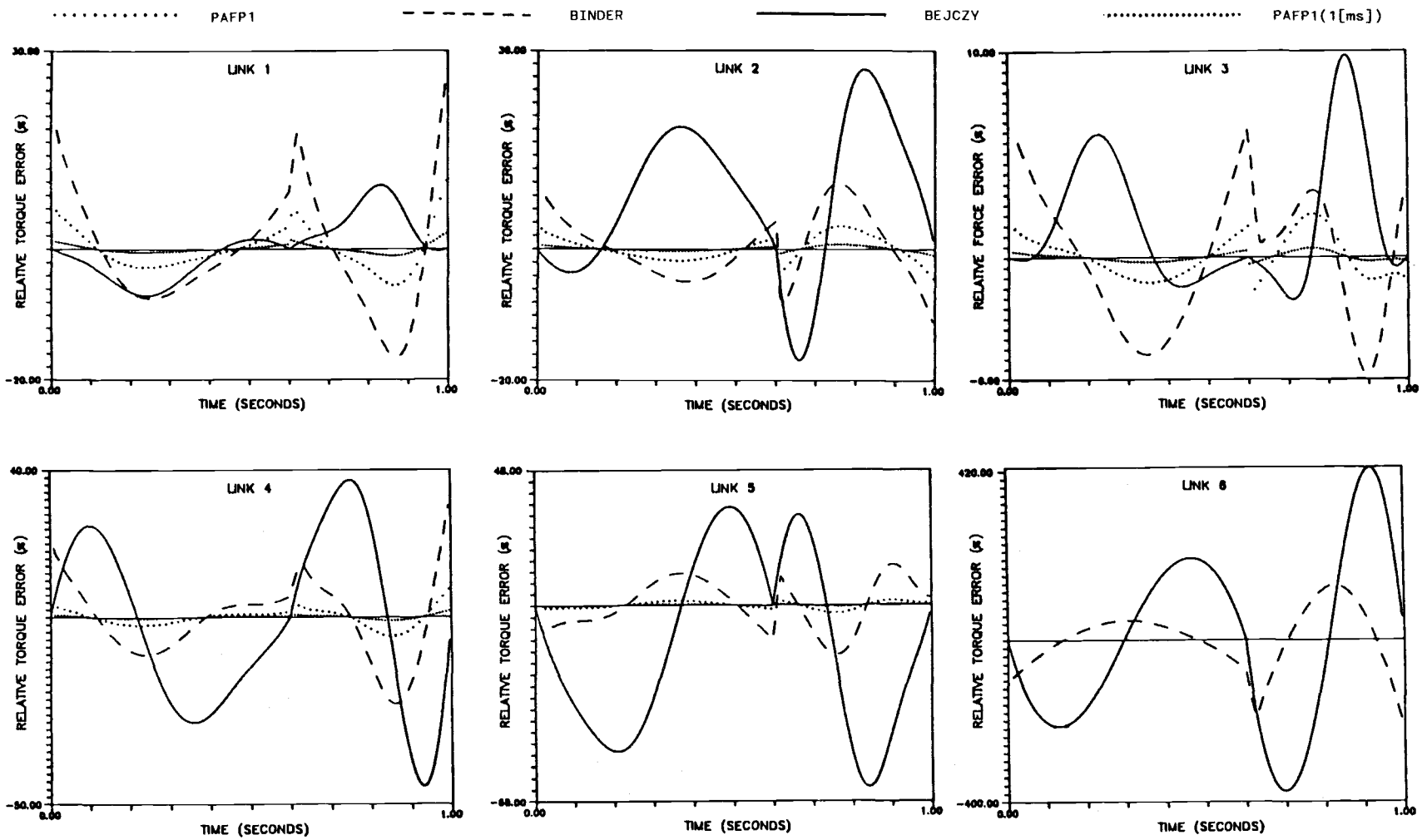

Figure 6.15 Comparison of Relative Errors,
Trajectory 2, 1-Second Duration.

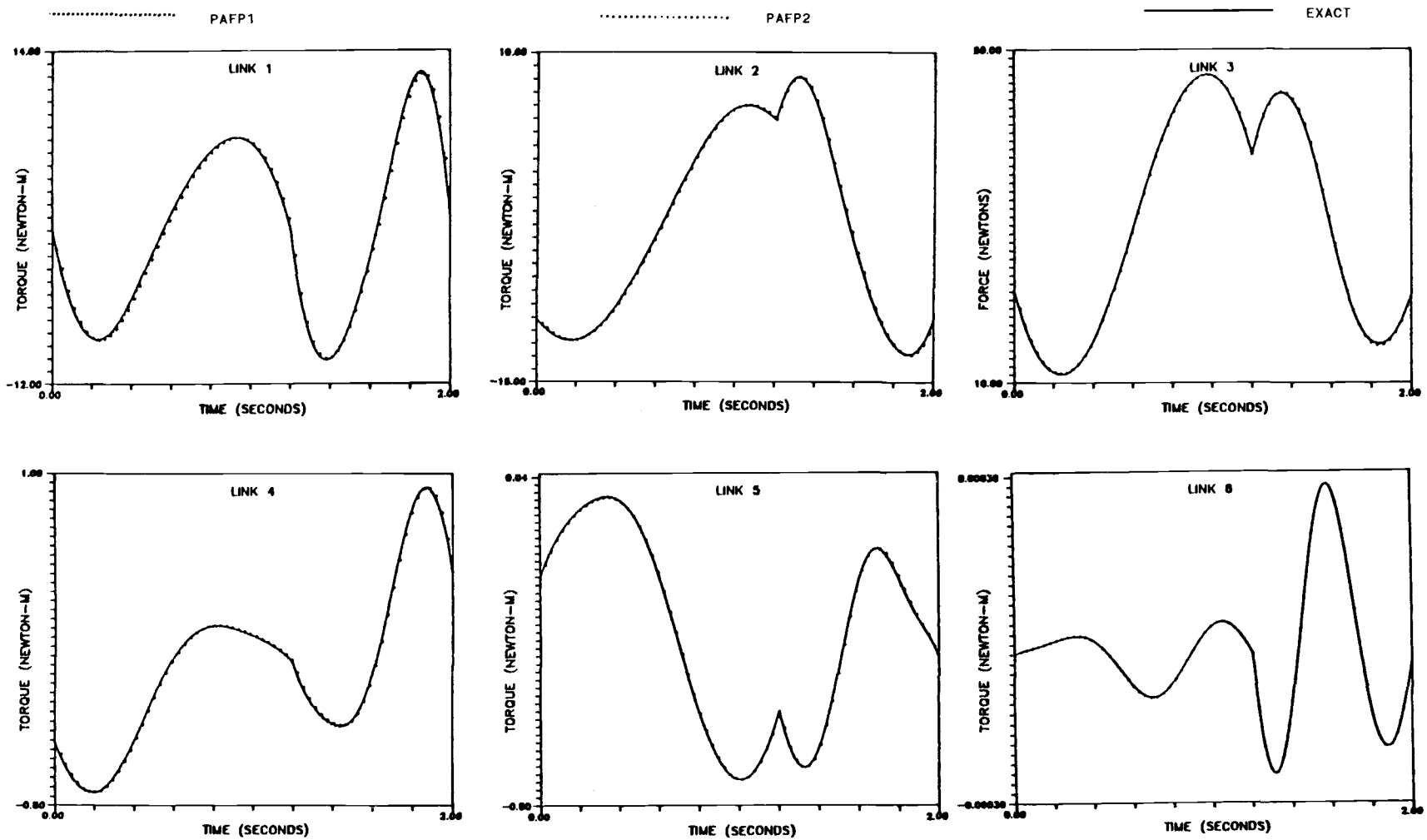

Figure 6.16 Comparison of Trajectory 2 Torque
PAFP2, 2-Second Duration.

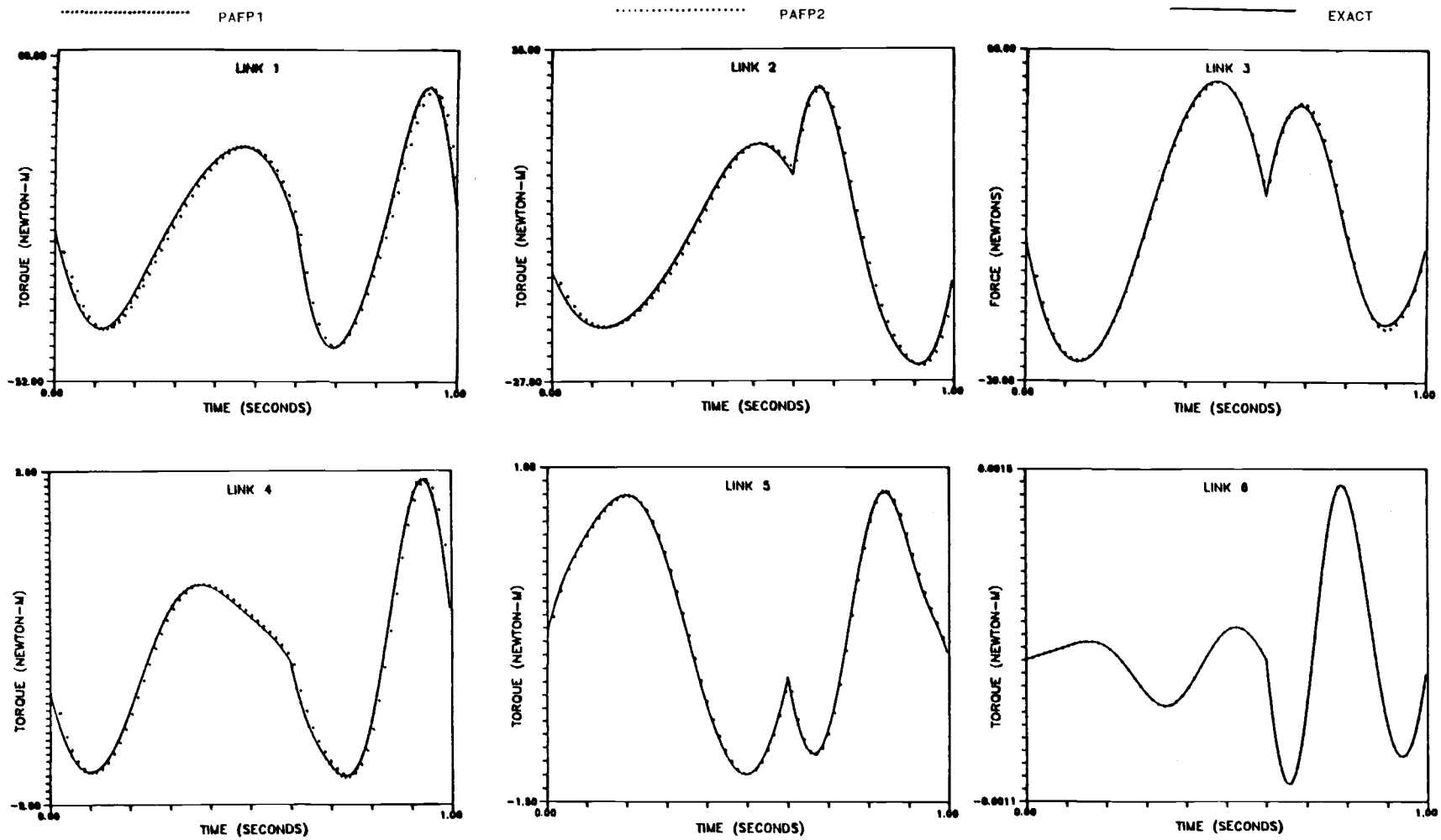

Figure 6.17 Comparison of Trajectory 2 Torque
PAFP2, 1-Second Duration.

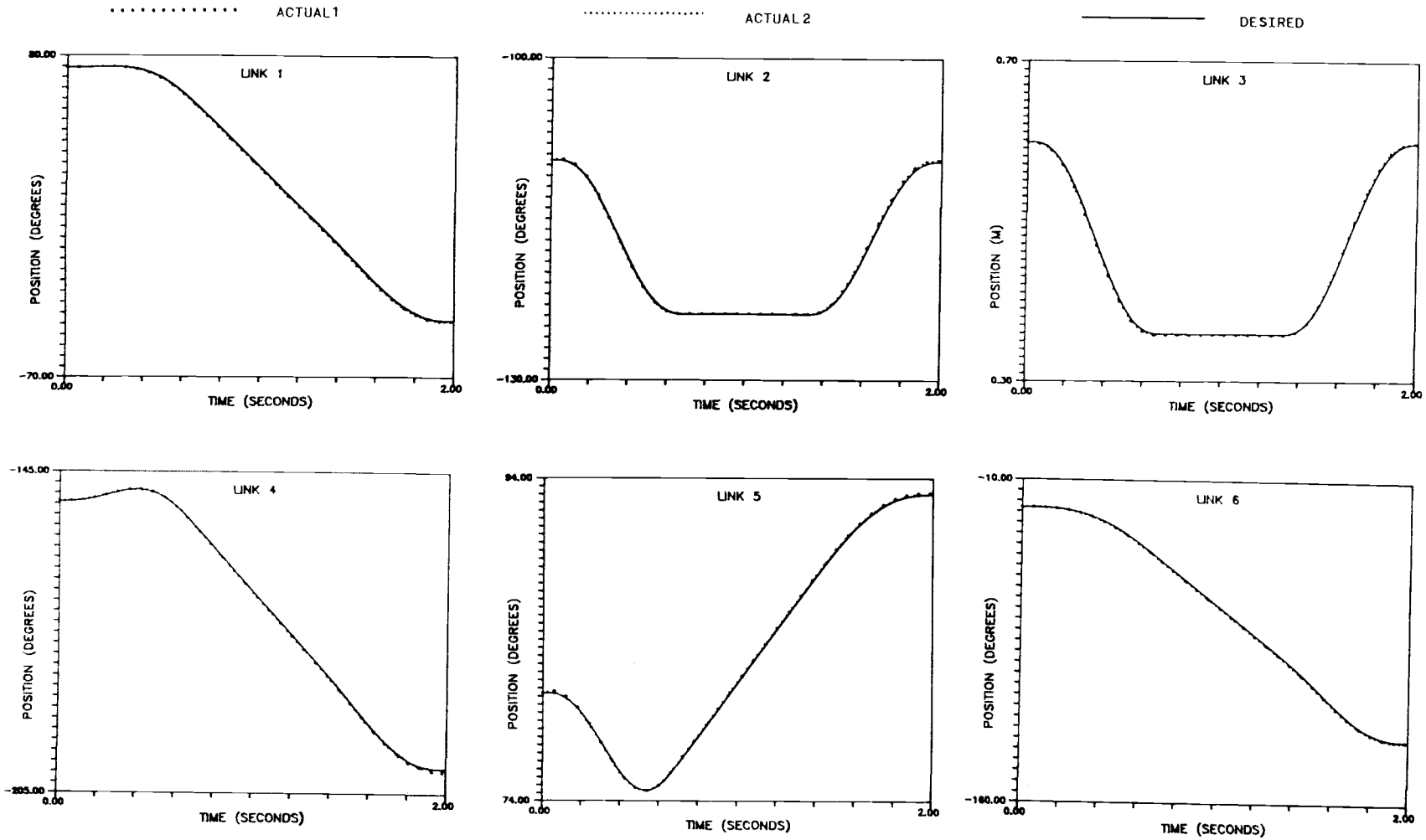

Figure 6.18 Desired and Actual Trajectories, Closed Loop Simulation

Table 6.1 Joint Positions for the Interpolation of the 2-Second Trajectory 1.

Joint Position	q_1 [deg]	q_2 [deg]	q_3 [deg]	q_4 [deg]	q_5 [deg]	q_6 [deg]
$J_s(t_0)$	74.4	-109.6	0.6	-150.6	80.7	-23.2
$J_1(t_1)$	53.9	-123.9	0.4	-155.4	76.4	-46.3
$J_2(t_2)$	0.8	-123.9	0.4	-179.6	86.3	-89.4
$J_f(t_3)$	-43.7	-109.6	0.6	-200.4	93.0	-130.7

Table 6.2 Parameters of 5th Order Polynomial for the Interpolation of the 2-Second Trajectory 1.

		Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6
Piont T_s	$q(0)$	74.4	-109.6	0.6	-150.6	80.7	-23.2
	$\dot{q}(0)$	0	0	0	0	0	0
Path Segment 1	P_0	74.4	-109.6	0.6	-150.6	80.7	-23.2
	P_1	0	0	0	0	0	0
	P_2	0	0	0	0	0	0
	P_3	126.6	-416.6	-6.9	194.6	-261.2	-90.5
	P_4	-529.6	892.8	14.8	-535.2	608.1	-15.1
	P_5	376.4	-510.1	-8.5	339.6	-361.3	-68.4
Piont T_1	$q(1)$	53.9	-123.9	0.4	-155.3	76.4	-46.3
	$\dot{q}(1)$	-88.6	0	0	-40.5	16.6	-71.7
Path Segment 2	P_0	115.9	-123.9	0.4	-126.9	64.8	3.9
	P_1	-88.6	0	0	-40.5	16.6	-71.7
	P_2	0	0	0	0	0	0
	P_3	0	0	0	0	0	0
	P_4	0	0	0	0	0	0
	P_5	0	0	0	0	0	0
Piont T_2	$q(2)$	0.8	-123.9	0.4	-179.6	86.3	-89.4
	$\dot{q}(2)$	-88.6	0	0	-40.5	16.6	-71.7
Path Segment 3	P_0	4412	-5483	-89	1985	-133	5476
	P_1	-14324	17242	287	-7027	732	-18013
	P_2	18658	-21885	-364	9137	-1018	23249
	P_3	-12070	13680	228	-5897	712	-14862
	P_4	3846	-4208	-70	1875	-244	4680
	P_5	-482	510	8	-234	33	-580
Piont T_f	$q(3)$	-43.7	-109.6	0.6	-200.4	93.0	-130.7
	$\dot{q}(3)$	0	0	0	0	0	0

Table 6.3 Comparison of Relative Root Mean Squared Errors, Trajectory 1, 6-Second Duration.

Sampling Rates	Algorithm	Link 1 [%]	Link 2 [%]	Link 3 [%]	Link 4 [%]	Link 5 [%]	Link 6 [%]
5 [ms]	Bejczy	5.411	Ø.718	Ø.351	1.213	4.943	162.64
	Binder	2.Ø34	Ø.372	Ø.457	Ø.315	Ø.674	8.98
	PAFP1	Ø.9Ø5	Ø.2Ø2	Ø.Ø93	Ø.Ø7Ø	Ø.Ø58	Ø
	PAFP2	Ø.42Ø	Ø.Ø84	Ø.Ø21	Ø.Ø38	Ø.ØØ5	Ø
1 [ms]	PAFP1	Ø.182	Ø.Ø4Ø	Ø.Ø19	Ø.Ø15	Ø.Ø12	Ø
	PAFP2	Ø.Ø8Ø	Ø.Ø15	Ø.ØØ4	Ø.ØØ8	Ø.ØØØ1	Ø

Table 6.4 Comparison of Relative Root Mean Squared Errors, Trajectory 1, 2-Second Duration.

Sampling Rates	Algorithm	Link 1 [%]	Link 2 [%]	Link 3 [%]	Link 4 [%]	Link 5 [%]	Link 6 [%]
5 [ms]	Bejczy	5.411	5.546	1.Ø82	7.757	19.456	162.22
	Binder	3.316	2.2Ø5	1.139	2.324	3.331	25.67
	PAFP1	1.Ø72	Ø.911	Ø.416	Ø.523	Ø.447	Ø
	PAFP2	Ø.52Ø	Ø.271	Ø.Ø78	Ø.124	Ø.Ø9Ø	Ø
1 [ms]	PAFP1	Ø.225	Ø.189	Ø.Ø85	Ø.1Ø6	Ø.Ø91	Ø
	PAFP2	Ø.1Ø2	Ø.Ø51	Ø.ØØ7	Ø.Ø21	Ø.ØØ1	Ø

Table 6.5 Comparison of Relative Root Mean Squared Errors, Trajectory 1, 1-Second Duration.

Sampling Rates	Algorithm	Link 1 [%]	Link 2 [%]	Link 3 [%]	Link 4 [%]	Link 5 [%]	Link 6 [%]
5 [ms]	Bejczy	5.415	10.591	1.331	15.705	26.774	162.35
	Binder	4.009	6.481	2.178	8.134	8.022	31.33
	PAFP1	1.951	2.474	0.907	1.901	1.150	0
	PAFP2	1.043	0.817	0.271	0.443	0.358	0
1 [ms]	PAFP1	0.431	0.542	0.190	0.402	0.237	0
	PAFP2	0.191	0.136	0.027	0.043	0.061	0

Table 6.6 Comparison of Relative Root Mean Squared Errors, Trajectory 2, 1-Second Duration.

Sampling Rates	Algorithm	Link 1 [%]	Link 2 [%]	Link 3 [%]	Link 4 [%]	Link 5 [%]	Link 6 [%]
5 [ms]	Bejczy	4.325	13.434	3.557	23.189	34.059	217.63
	Binder	8.534	4.842	3.118	10.178	8.541	73.15
	PAFP1	3.030	1.749	0.977	2.325	1.264	0
	PAFP2	1.176	0.469	0.259	0.378	0.290	0
1 [ms]	PAFP1	0.650	0.365	0.199	0.485	0.252	0
	PAFP2	0.218	0.070	0.024	0.021	0.005	0

Table 6.7 Comparison of Relative Position Errors, Trajectory 1.

Sampling Rates	Motion	Link 1 [%]	Link 2 [%]	Link 3 [%]	Link 4 [%]	Link 5 [%]	Link 6 [%]
5	Fast	Ø.3Ø	Ø.74	Ø.74	Ø.25	Ø.34	Ø.29
[ms]	Very Fast	Ø.6Ø	1.46	1.48	Ø.63	Ø.67	Ø.57
1	Fast	Ø.Ø6	Ø.15	Ø.15	Ø.Ø5	Ø.Ø7	Ø.Ø6
[ms]	Very Fast	Ø.12	Ø.29	Ø.3Ø	Ø.13	Ø.14	Ø.11

Chapter 7

Summary, Conclusions and Recommendations

The objectives of this research were to design a multiprocessor computing system with efficient computing algorithms for real-time robot control. Robot system control schemes for fast and precise robot motion require complete utilization of a robot's dynamic equations and the ability to evaluate these equations in real-time. The computing time for the dynamic equations is the principal problem for practical implementations due to the limitations of current computational power.

In response to this problem, three major research objectives were of principal concern. The first was to design a multiprocessor computing system based upon three integrated components: A system supervisor computer, a pipelined n-processor, and a microprocessor-based individual joint servo-controller. The intent of this system was to provide concurrent pipelined parallel computations, utilizing all of the n-processors for computation of dynamic equations and thereby increasing controller processing speed and CPU utilization in order to achieve real-time control.

The second research objective was to develop efficient computing algorithms for concurrent pipelined parallel computations, based upon the proposed computing system. The concurrent performance of pipelined parallel computations of the dynamic equations for the proposed system is based upon consideration of the sequential dependencies of the Newton-Euler dynamic equations, which are conducive to pipelining, and decomposition of the dynamic backward equations for fast parallel computation. The decomposition of the backward equations is based on computational simplification techniques. These computationally efficient simplification techniques, which were applied to the backward variables of dynamic formulations, were determined from consideration of four possible decomposition schemes for the dynamic formulations, each of which was further considered with respect to three computational methods. Following this procedure, an analysis of errors introduced into the proposed algorithms for the system was performed.

As a final step, motion simulations were performed and were compared with a related class of algorithms in order to evaluate the proposed algorithms. The results of the study indicated that the errors introduced into the algorithms were relatively small and that the proposed technique's performance levels were superior to

those of techniques involving a number of other methods of computation simplification.

In conclusion, use of the proposed computer architecture and computing algorithms, based upon the use of multiple low-cost microprocessors and the application of feedback, allow for the practical implementation of a highly parallel structure capable of achieving real-time robot control with high sampling rates. As a result of achieving a higher sampling rate, the robot system can be more robust with respect to the effect of unknown disturbances on trajectory tracking performance.

It is recommended that an investigation of a special architecture for the processor, suitable for the proposed computing architecture, be undertaken. For improved system performance, hardware parallelism should also be considered for the processor architecture. When the dynamic formulations are reviewed, it is clear that all of the operations involve 3×3 matrix, 3-dimensional vectors. Therefore, provision of hardware parallelism and pipelining to exploit potentially concurrent low-level vector/matrix operations, could result in the achievement of even greater speed.

An additional area of investigation would be the examination of different interpolation methods at trajectory generation in order to reduce the errors introduced into the proposed method. From the results of

the current study, it is apparent that the errors resulted from the effect of calculating the backward equation of link i due to differences in force and/or moment exerted on link i+1 by link i between adjacent sampling periods, with respect to the coordinate system of link i+1. In turn, a main source for the difference in force and/or moment exerted on link i+1 by link i resulted from the difference of input joint acceleration between adjacent sampling periods, which is determined by an interpolation method. A better method of interpolation, which could serve to reduce this difference, would yield better results.

References

AMIN 87 M. Amin-Javaheri and D.E. Orin. "A Systolic Architecture for Computation of the Manipulator Inertia Matrix." *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, Raleigh, NC, pp. 647-653, March 1987.

AN 87 C. H. An and C. G. Atteson. "Experimental Evaluation of Feedforward and Computed Torque Control." *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, Raleigh, NC, pp. 165-174, March 1987.

ARMS 79 W. W. Armstrong. "Recursive Solution to the Equations of Motion of an n-Link Manipulator." *Proceedings 5th World Congress on Theory of Machines and Mechanisms*, Montreal, Canada, pp. 1343-1346, July 1979.

BEJC 74 A. K. Bejczy. *Robot Arm Dynamics and Control.* Jet Propulsion Laboratory, California Institute of Technology, CA, TM: 33-669, February 1974.

BEJC 79 A. K. Bejczy. *Dynamics Model and Control Equations for Manipulators.* Jet Propulsion Laboratory, California Institute of Technology, CA, TM: 715-19, November 1979.

BEJC 81 A. K. Bejczy and R.P. Paul. "Simplified Robot Arm Dynamics for Control." *Proceedings of the 20nd IEEE Conference on Decision and Control*, San Diego, CA, pp. 261-262, December 1981.

BEJC 83 A. K. Bejczy and S. Lee. "Robot Arm Dynamic Model Reduction for Control." *Proceedings of the 22nd IEEE Conference on Decision and Control*, San Antonio, TX, pp. 1466-1476, December 1983.

BIND 86 E. E. Binder and J. H. Herzog. "Distributed Computer Architecture and Fast Parallel Algorithm in Real-Time Robot Control." *IEEE Transactions on System, Man and Cybernetics*, Vol. SMC-16, No. 4, pp. 543-549, July/August 1986.

CART 84 R. H. Cartesian and R. P. Paul."On-Line Dynamic Trajectory Generator." the International *Journal of Robotic Research*, Vol. 3, No. 1, pp. 68-72, Spring 1984.

CHEN 86 J. B. Chen, R. S. Fearing, B. B. Armstrong and J. W. Burdick. "NYMPH: A Multiprocessor for Manipulation Application." *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, San Francisco, CA, pp. 1731-1736, March 1986.

CRAI 86 J. J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Co., Reading, Massachusetts, 1986.

DESI 87 C. W. Desilve and J. V. Wissen. "Least Squares Adaptive Control for Trajectory Following Robots." *Transactions of ASME, Journal of Dynamic Systems, Measurement and Control*, Vol. 109, No. 2, pp. 104-110, June 1987.

HOLL 80 J. M. Hollerbach. "A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity." *IEEE Transactions on System, Man and Cybernetics*, Vol. SMC-10, No. 11, pp. 730-736, November 1980.

HOLL 84 J. M. Hollerbach and G. Sahara. "Wrist Partitioned Inverse Kinematic Acceleration and Manipulator Dynamics." *Proceedings of the International Conference on Robotics*, Atlanta, GA, pp. 152-161, March 1984.

HWAN 84 K. Hwang and F. Briggs. *Computer Architecture and Parallel Processing*. McGraw Hill Inc., New York, New York, 1984.

KANA 84 T. Kanade, P. K. Khosla and N. Tanaka. "Real-Time Control of the CMU Direct Drive Arm II Using Customized Inverse Dynamics." *Proceedings of the 23nd IEEE Conference on Decision and Control*, Las Vegas, NV, pp. 1345-1352, December 1984.

KASA 85 H. Kasahara and S. Narita. "Parallel Processing of Robot-Arm Control Computation on a Multiprocessor System." *Journal of Robotics and Automation*, Vol. RA-1, No. 2, pp. 104-113, June 1985.

KHOS 87 P. K. Khosla. "Choosing Sampling Rates for Robot Control." *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, Raleigh, NC, pp. 169-174, March 1987.

KHOS88a P. K. Khosla. "Some Experimental Results on Model -Based Control Schemes." *Proceedings of the 1988 IEEE International Conference on Robotics and Automation*, Philadelphia, PA, pp. 1380-1385, April 1988.

KHOS 86 P. K. Khosla and T. Kanade. "Real-Time Implementation and Evaluation of Model-Based Controls CMU Direct Drive Arm II." *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, San Francisco, CA, pp. 1546-1555, April 1986.

KHOS 85 P. K. Khosla and C. P. Newman. "Computational Requirements of Customized Newton-Euler Algorithm." *Journal of Robotic Systems*, Vol. 2, No. 3, pp. 309-327, Fall 1985.

KHOS88b P. K. Khosla and S. Ramos. "A Comparative Analysis of the Hardware Requirements for the Lagrange-Euler and Newton-Euler Dynamics Formulations." *Proceedings of the 1988 IEEE International Conference on Robotics and Automation*, Philadelphia, PA, pp. 291-296, April 1988.

KOGG 81 P. M. Kogge. *The Architecture of Pipelined Computers*. New York, New York, McGraw-Hill, 1981.

LATH 85 R. H. Lathrop. "Parallelism in Manipulator Dynamics." *International Journal of Robotics Research*, Vol. 4, No. 2, pp.80-102, Summer 1985.

LEE 82a C. S. Lee. "Robot Arm Kinematics, Dynamics, and Control." *IEEE Computer*, Vol. 15, No. 12, pp. 62-79, December 1982.

LEE 86a C. S. Lee. "Robot Arm Kinematics." *IEEE Computer Society*, Tutorial on Robotics, pp. 47-65, 1986.

LEE 86b C. S. Lee and P. R. Chang. "Efficient Parallel Algorithm for Robot Inverse Dynamics Computation." *IEEE Transactions on System, Man and Cybernetics*, Vol. SMC-16, No. 4, pp. 532-542, July/August 1986.

LEE 84 C. S. Lee.and B. H. Lee. "Resolved Motion Adaptive Control for Mechanical Manipulators." Transactions of ASME, *Journal of Dynamic Systems, Measurement and Control*, Vol. 106, No. 2, pp. 134-142, June 1984.

LEE 82b C. S. Lee, T. N. Mudge and J. L. Turney. "Hierarchical Control Structure Using Special Purpose Processors for Control of Robot Arms." *Proceedings of the Pattern Recognition and Image Processing Conference*, Las Vegas, NV, pp. 634-640, June 1982.

LEUN 87 S. S. Leung and M. A. Shanblatt. "Real-Time DKS on a Single Chip." *IEEE Journal of Robotics and Automation*, Vol. RA-3, No. 4, PP. 281-290, August 1987.

LEWI 74 R. A. Lewis. *Autonomous Manipulation on a Robot: Summary of Manipulator Software Functions*. Jet Propulsion Laboratory, California Institute of Technology, CA, TM: 33-679, March 1974.

LIN 83a C. F. Lin. "Microprocessor-Based Controller for the Intelligent Robot System." *Proceedings of the 13th International Symposium on Industrial Robots*, Chicago, IL, pp. 127-138, 1983.

LIN 83b C. G. Lin, P. R. Chang and J. Y. Luh. "Formulation and Optimization of Cubic Polynomial Joint Trajectories for Industrial Robots." *IEEE Transactions on Automatic Control*, Vol. AC-28, No. 12, pp. 1066-1073, 1983.

LING 88 Y. L. Ling, P. Sodayappan, K. W. Ison and D. E. Orin. "A VLSI Robotics Vector Processor for Real-Time Control." *Proceedings of the 1988 IEEE International Conference on Robotics and Automation*, Philadelphia, PA, pp. 303-308, April 1988.

LUH 81 J. Y. Luh and C. S. Lin. "Automatic Generation of Dynamic Equations for Mechanical Manipulators." *Proceedings of the 1981 Conference on Joint Automatic Control*, Charlottesville, VA, TA-2D, June 1981.

LUH 82 J. Y. Luh and C. S. Lin. "Scheduling of Parallel Computation for a Computer-Controlled Mechanical Manipulator." *IEEE Transactions on System, Man and Cybernetics*, Vol. SMC-12, No. 2, pp. 214-234, March/April 1982.

LUH 80 J. Y. Luh, M. W. Walker and R. P. Paul. "On Line Computational Scheme for Mechanical Manipulators." *Transactions of ASME, Journal of Dynamic Systems, Measurement and Control*, Vol. 102, No. 2, pp. 69-76, June 1980.

NIGA 85 R. Nigam and C. S. Lee. "A Multiprocessor-Based Controller for the Control of Mechanical Manipulators." *IEEE Journal of Robotics and Automation*, Vol. RA-1, No. 4, PP. 173-182, December 1985.

ORIN 79 D. E. Orin, et al. "Kinematic and Kinetic Analysis of Open-Chain Linkage Utilizing Newton-Euler Methods." *Mathematical Biosciences*, Vol. 43, pp.1343-1346, February 1979.

ORIN 84 D. E. Orin. "Pipelined Approach to Inverse Plant Plus Jacobian Control of Robot Manipulators." *IEEE International Conference on Robotics*, Atlanta, GA, pp. 169-175, 1984.

ORIN 87 D. E. Orin, K. W. Olson and H. H. Chao. "Systolic Architecture for Computation of the Jacobian for Robot Manipulators," in *Computer Architecture for Robotics and Automation*, pp. 39-67, Edited by J. H. Graham. Gordon and Breach Science Publishers, New York, 1987.

OZGU 85 F. Ozguner and M. L. Kao. "A Reconfigurable Multiprocessor Architecture for Reliable Control of Robotics System." *Proceedings of the 1988 IEEE International Conference on Robotics and Automation*, St. Louis, MO, pp.802-806, March 1985.

PAUL 79 R. P. Paul. "Manipulator Cartesian Path Control." *IEEE Transactions on System, Man and Cybernetics*, Vol. SMC-9, No. 11, pp. 702-711, November 1979.

PAUL 84 R. P. Paul. *Robot Manipulators: Mathematics, Programing, and Control*. MIT Press, Cambridge, MA, 1984.

SAUE 81 C. H. Sauer and K. M. Chandy. *Computer Systems Performance Modeling*. Prentice-Hall Inc., Englewood Cliffs, NJ, 1981.

SHAN 88 M. A. Shanblatt and S. S. Leung. "A Conceptual Framework for Designing Robotic Computational Hardware with ASIC Technology." *Proceedings of*

the 1988 IEEE International Conference on Robotics and Automation, Philadelphia, PA, pp. 461-464, April 1988.

SHIN 87 K. G. Shin. "Special Issues on Real-Time System." *IEEE Transactions on Computers*, Vol. C-36, No. 8, pp. 901-903, August 1987.

STAN 88 J. A. Stankovic. "Misconceptions about Real-Time Computing: A Serious Problem for Next-Generation Systems." *IEEE Computer*, Vol. 21, No. 10, pp. 10-19, October 1988.

TAYL 79 R. H. Taylor. "Planning and Execution of Straight Line Manipulator Trajectories." *IBM Journal of Research and Development*, Vol. 23, No. 4, pp. 424-436, July 1979.

TURN 81 J. L. Turney and T. N. Mudge. "VLSI Implementation of a Numerical Processor for Robotics." *Proceedings of the 27th International Symposium*, Indianapolis, IN, pp. 169-175, April 1981.

TURN 84 T. L. Turner, J. J. Craig and W. A. Gruver. "A Microprocessor Architecture for Advanced Robot Control." *Proceedings of the 14th International Symposium on Industrial Robots*, Gothenburg, Sweden, pp. 407-416, 1984.

WAH 87 B. W. Wah. "Systolic Array from Concept to Implementation." *IEEE Computer*, Vol. 20, No. 7, pp. 12-17, July 1987.

WALK 82 M. W. Walker and D. E. Orin. "Efficient Dynamics Computer Simulation of Robotic Mechanisms," in *Robot Motion*, Edited by Brady. MIT Press, Cambridge, MA, pp. 107-125, 1982.

WANA 86 W. Wanatabe, et al. "Improvement in the Computing Time of Robot Manipulator Using Multiprocessors." *Transactions of ASME, Journal of Dynamic Systems, Measurement and Control*, Vol. 108, No. 2, pp. 190-197, June 1986.

ZHEN 86 Y. F. Zheng and H. Hemami. "Computation of Multibody System Dynamics by Multiprocessor Schemes." *IEEE Transactions on System, Man and Cybernetics*, Vol. SMC-16, No. 1, pp. 102-110, January/February 1986.